VNIVERSIDAD NACIONAL AVTNOMA DE MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN

ASTEC: Diseño y simulación de un robot autónomo móvil para búsqueda y rescate

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN TELECOMUNICACIONES, SISTEMAS Y ELECTRÓNICA.

PRESENTA:
BRIAN GARCÍA SARMINA

ASESOR:
Fis. José de Jesús Cruz Guzmán

CUAUTITLÁN IZCALLI, ESTADO DE MÉXICO, 2020

VNIVERSIDAD NACIONAL AVÓNOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN
SECRETARÍA GENERAL
DEPARTAMENTO DE EXÁMENES PROFESIONALES

U. N. A. M.
FACULTAD DE ESTUDIOS SUPERIORES-CUAUTITLAN

ASUNTO: VOTO APROBATORIO

M. en C. JORGE ALFREDO CUÉLLAR ORDAZ
DIRECTOR DE LA FES CUAUTITLAN
PRESENTE

ATN: I.A. LAURA MARGARITA CORTAZAR FIGUEROA
Jefa del Departamento de Exámenes Profesionales
de la FES Cuautitlán.

Con base en el Reglamento General de Exámenes, y la Dirección de la Facultad, nos permitimos comunicar a usted que revisamos el: **Trabajo de Tesis**

ASTEC: Diseño y simulación de un robot autónomo móvil para búsqueda y rescate

Que presenta el pasante: **BRIAN GARCÍA SARMINA**
Con número de cuenta: **31165660-6** para obtener el Título de la carrera: **Ingeniería en Telecomunicaciones, Sistemas y Electrónica**

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el **EXAMEN PROFESIONAL** correspondiente, otorgamos nuestro **VOTO APROBATORIO**.

ATENTAMENTE
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Cuautitlán Izcalli, Méx. a 26 de Noviembre de 2019.

PROFESORES QUE INTEGRAN EL JURADO

	NOMBRE	FIRMA
PRESIDENTE	Lic. José de Jesús Cruz Guzmán	
VOCAL	Mtro. Jorge Buendía Gómez	
SECRETARIO	Mtro. Leopoldo Martín del Campo Ramírez	
1er. SUPLENTE	Ing. Nidia Mendoza Andrade	
2do. SUPLENTE	Dr. David Tinoco Varela	

NOTA: los sinodales suplentes están obligados a presentarse el día y hora del Examen Profesional (art. 127).

LMCF/ntm*

Dedicatoria

Este trabajo de tesis es dedicada para mi familia, a mis abuelos, tíos y primos, y en especial a mi padre José Tonatiu García Adan y a mi madre Lidia Alejandra Sarmina Specia, con quienes a pesar de tener muchas diferencias, siempre me han apoyado, porque fueron ustedes quienes madrugaron y sudaron para proveerme las herramientas necesarias para alcanzar el éxito, me han dado comida, techo y una educación. Además, me han enseñado a disfrutar de los pequeños momentos que tenemos con las personas que amamos, reír aunque las cosas no resulten como quisiéramos, levantar la cabeza y seguir avanzando sin importar lo que pase, me han enseñado muchas cosas más, muchas que sigo aprendiendo. Esta tesis no solo es el resultado de mi trabajo, sino también del suyo, porque sin ustedes, sin las risas y sin las lágrimas, no seria quien soy ahora, por eso y por tanto más, muchas gracias.

También quiero dedicarle este trabajo a Mariana Ortega Martínez, mi pareja, mi sol, mi compañera y sobretodo mi amiga, con quien he pasado muchas cosas, muchas buenas y unas cuantas malas, con quien he caído y con quien me he levantado, con quien he soñado y con quien he aprendido, con quien he llorado de felicidad y de tristeza, pero al final, quien siempre ha estado ahí. Gracias por apoyarme y creer en mi, por exigirme y ayudarme a ser mejor persona. Se que juntos llegaremos más y más alto, hasta alcanzar el ultimo de nuestros sueños.

Por ultimo, dedico este logro a todas las personas que forman parte de mi vida, las que siguen en este mundo y las que ya forman parte de otro, les dedico este trabajo, el esfuerzo, la tenacidad y la dedicación, porque cada uno de ustedes forman parte de mi, y se que sentirán muy orgullosos, como yo me siento orgulloso de cada uno de ustedes, este solo es un logro de muchos que vendrán, ya lo verán.

Agradecimientos

Agradezco al Fis. José de Jesús Cruz Guzmán, por todo el tiempo que hemos compartido juntos, todos los trabajos que hemos realizado, y en especial este trabajo de tesis. Gracias por todos los comentarios y las observaciones que ha tenido conmigo a lo largo de los años, me ha ayudado a formar una opinión mas amplia y clara de las cosas, a entender que no se puede predisponer una respuesta antes de observar, a informarme antes de decir, y aceptar a desconocer para poder empezar a entender.

Agradezco también al Ing. José Luis Barbosa Pacheco, quien además de ser mi profesor, es un amigo, con quien puedo platicar honestamente y escuchar una respuesta honesta, con quien comparto puntos de vista pero también algunas diferencias, con quien he aprendido muchas cosas académicas y personales.

También agradezco al Dr. Zbigniew Oziewicz, a quien considero una de las personas mas apasionadas en su trabajo, con quien he aprendido que a veces es mejor decir "no se" en lugar de decir algo que desconozco. Quien también me ha enseñado a ampliar mi perspectiva a nuevos conceptos y nuevos enfoques, que me ha enseñado a cuestionar lo establecido, no importando de donde o de quien venga.

Finalmente agradezco a los profesores: Ing. Raúl Marcos Bogard Sierra, Ing. David Alejo Torres y Dr. Ángel López Gómez, que me ayudaron en distintas etapas de la carrera, con la disposición de resolver mis preguntas, asesorándome, y dándome la oportunidad de aprender algo nuevo.

Agradecimiento especial para la Universidad Nacional Autónoma de México, proyecto PAPIME PE109619 por el apoyo a través de la beca de titulación por tesis.

Resumen

En el presente trabajo se describe el diseño y simulación de un robot autónomo móvil para búsqueda y rescate, tomando como base los aspectos de modularidad, generalidad y simplicidad, se destacan las características que conforman al hardware del diseño del prototipo (estableciendo la modularidad de las unidades que lo conforman), los algoritmos usados (utilizando la generalidad, para cubrir un mayor espectro de escenarios posibles), los modelos matemáticos (teniendo en cuenta la simplicidad para su entendimiento y aplicación) y los enfoques para solucionar las tareas de "exploración y trazado de ruta", "localización y mapeo simultaneo" y la percepción del entorno a través de los sensores del robot.

En particular, el diseño de un algoritmo capaz de resolver el problema de exploración y el problema de trazado de ruta, es explicado y resuelto a través de un propuesta diseñada e implementada de nombre "algoritmo QPA*" que incorpora subdivisión de cuadrantes para procesos de establecimiento de metas pseudo-aleatorias (protocolo de exploración), incluyendo campos de potencial binario para los objetos (paredes, obstáculos, etc) como un método para evitar las colisiones, y por ultimo, añadiendo una versión modificada de un algoritmo A* con ordenamiento de pila (para el vector de nodos de "frente") y un costo de "penalización" o "costo extra" hacia nodos visitados en rutas previas. Además, se destaca la incorporación del enfoque "Fast SLAM" para dar solución a las tareas de localización y mapeo del robot (de manera simultanea).

Cada algoritmo propuesto se lleva a la sección de simulaciones y resultados para poder obtener conclusiones acerca del desempeño de los distintos enfoques, y comprobar si estos tienen la capacidad de resolver los objetivos planteados para el diseño.

Finalmente, se aborda la etapa de conclusiones con base a las simulaciones generadas y los resultados obtenidos, donde se sintetizan los aspectos mas importantes y se explica como cada elemento planteado cumple con alguno de los objetivos generales o específicos de este trabajo de tesis.

Índice general

Capítulo 1

Introducción

1.1. Objetivo

Objetivo General

Diseñar y simular los algoritmos de un sistema robótico móvil autónomo inteligente para las tareas de búsqueda y localización de personas en una zona de desastre urbano.

Objetivos Específicos

- Crear distintos métodos para percibir el entorno, que puedan ser aplicados en el diseño de ASTEC.
- Diseñar un método de exploración y trazado de ruta.
- Realizar una comparativa entre dos enfoques del algoritmo A* y el algoritmo QPA*.
- Implementar un método de localización y mapeo simultáneo.

1.2. Hipótesis

El diseño robótico ASTEC podrá desempeñar en simulaciones, múltiples tareas necesarias en un robot de búsqueda y rescate, estas tareas se enfocaran a la exploración, trazado de ruta, localización y mapeo.

1.3. Antecedentes

México es un país con alta diversidad geográfica, diversidad de flora y fauna, diversidad climática, etc. Sin embargo, esto conlleva que sea también un país propenso a diversos fenómenos naturales y debido a que se encuentra en una zona de alta actividad sísmica, los sismos y sus consecuencias son uno de los principales problemas que afectan a México. Dicho esto, se menciona que existen varios precedentes de escenarios de

desastre ocasionados por sismos en México, los sismos que se presentaron en 19 de Septiembre son los más populares en la historia sismológica de México, por ello surge la propuesta del autor para presentar un método de búsqueda (directa) y rescate (indirecto) de personas en una zona de desastre urbana, donde existan edificios parcial o completamente colapsados, calles colapsadas, casas con daños arquitectónicos o derrumbadas, y en general donde se encuentren zonas de desastre urbano inaccesibles o peligrosas para personal humano o canino.

1.4. Robótica Móvil

Los robots móviles, en general son sistemas robóticos con movimiento autónomo, que tienen como tarea llevar acabo tareas programadas o dirigidas remotamente, donde la característica mas importante de estos robots es su locomoción [32] . Existe una gran variedad de robots móviles que son usados en distintos propósitos, que van desde lo educativo hasta lo militar, donde se buscan resolver problemas dinámicos o donde exista la necesidad de tener un agente que pueda moverse en un cierto entorno. En la actualidad existen muchas ramas de la robótica móvil que se plantean para aplicaciones en misiones peligrosas, ambientes complejos y terrenos impredecibles, así como exploración de planetas, exploración y reconocimiento de zonas de difícil acceso, anti-terrorismo, etc. [7]

1.5. Robótica de rescate, robótica de intervención y exploración

La "Robótica de rescate" se ocupa de desplegar sistemas robóticos con la finalidad de responder y actuar (en tareas necesarias) a distancia de la zona de desastre o del incidente extremo. En general, la robótica de rescate es un campo relativamente pequeño, que cuenta con pocos círculos de investigación de empresas y escuelas.[9]

Ahora bien, existe otra rama en la robótica que es conocida como "Robótica de intervención", esta rama implementa robots que operan usualmente en entornos parcialmente conocidos, donde existe el impedimento del ingreso de seres humanos (difícil acceso, condiciones ambientales peligrosas, radiación, etc.). Este tipo de robots usualmente son ubicados en el exterior de edificios donde se conocen a priori la ubicación exacta de los obstáculos que deberá de sortear o superar el robot.[6]

Los robots móviles usados para tareas de rescate o intervención pueden ayudar a investigar una situación peligrosa y/o escenario de desastre, además de tener la capacidad de buscar víctimas en la zona. Los robots mejoran la seguridad de los rescatistas al estar diseñados para entrar en zonas peligrosas sin la necesidad de usar

personal humano [12], ya que actualmente, en lo que se refiere a escenarios de desastre urbano, la mayoría de las tareas de búsqueda y rescate son llevadas a cabo por humanos y perros entrenados.[11]

Entonces se pude asumir que tanto la robótica para rescate como la robótica de intervención tratan de resolver problemas suscitados en lugares de desastre (con mayor énfasis en los desastres urbanos), zonas de difícil acceso o con peligros potenciales para elementos humanos y/o caninos. Para los propósitos de este trabajo, se contemplarán ideas de ambas ramas de la robótica, ya que, aunque la robótica para rescate presente puntos en común con los deseados para este proyecto, también se encuentran características compartidas en la robótica de intervención. Una característica fundamental que se expresa a lo largo de la literatura consultada, independiente de la rama de la robótica planteada (rescate, intervención o exploración) es que estos sistemas deben disponer de sistemas de locomoción que permitan operar en terrenos irregulares y/o sortear obstáculos de gran tamaño.[5]

Ahora, en otra parte se tienen los robots exploradores que se despliegan en entornos donde posiblemente no sea recuperable el robot, su función principal es recabar y enviar información durante el tiempo de operación. El ambiente puede ser parcialmente conocido o completamente desconocido, este tipo de robots usualmente cuentan con un sistema de locomoción de patas o híbrido que les permitan superar obstáculos muy difíciles.[3] [6]

Sus aplicaciones engloban: exploración de planetas, exploración de fondos marinos, exploración de volcanes, etc.[1]

Aunque los robots exploradores sean principalmente usados para tareas donde se pretende recabar información de escenarios en su mayoría desconocidos (asumiendo que se conocen ciertas características del entorno que inspiran el diseño del robot), se pueden rescatar muchas características que comparten con la robótica de rescate y la robótica de intervención, siendo que para el propósito de esta tesis existen ciertos factores que aborda la robótica para exploración que se pretenden incorporar en el prototipo ASTEC. La capacidad de recabar información y la posibilidad de perder el equipo durante el despliegue son algunas de las principales características buscadas.

Entonces, contemplando que existen tres vertientes de la robótica móvil que pueden ser implementadas, se propone realizar una combinación de características de la robótica de rescate o robótica para desastre, robótica de intervención y robótica de exploración, sin embargo, se cataloga el sistema ASTEC dentro de los robots para rescate o robots para desastre, debido a que la principal tarea del prototipo ASTEC será llevar acabo tareas de búsqueda y rescate, que son tareas que se abordan netamente en la robótica para desastre o robótica de rescate.

1.6. Breve contexto histórico de la robótica de rescate

Las investigaciones académicas sobre la robótica de rescate empezaron en 1995 con dos grupos importantes, el grupo a cargo de *Satoshi Takokoro* en la Universidad Kobe en Japón, que fue inspirado por el terremoto en *Hanshin-Awaji*, posteriormente este grupo fundo el *Instituto Internacional del Sistema de Rescate* (International Rescue System Institute, IRS). El segundo grupo importante a cargo de *Robin Murphy* en la Escuela de Minas en Colorado, la creación de este grupo fue motivada por el ataque con bomba al edificio federal "Alfred P. Murrah" en la ciudad de Oklahoma.

El *Dr. Red Whitaker* en Carnegie Mellon University lidero el primer equipo que construyo robots para entrar y explorar en "Three Mile Island", otro despliegue realizado fue en el desastre de Chernobyl en 1986; cabe destacar que estos robots fueron construidos de manera rápida, incorporando protección contra la radiación y con ello se hicieron bastante pesados, largos y muy lentos (este tipo de desastres no representaron un problema para las dimensiones de los robots, ya que se describen como escala humana). [9]

Posteriormente la tecnología implementada de los robots en desastres nucleares se llevó a la comunidad del escuadrón anti-bombas, donde como en los casos previos la velocidad no representaba un factor importante, y se ponderaba con mayor énfasis un buen sistema de protección del robot.

El parte aguas de la tecnología de los robots de rescate fue en el incidente de Oklahoma en 1995 antes mencionado, ya que los robots anti-bombas usados con anterioridad demostraron ser ineficientes por su tamaño y peso, generando la posibilidad de colapsos posteriores al mover escombros o pilares del edificio. La perspectiva de los robots de rescate cambio radicalmente a partir del incidente anterior, ya que ahora los aspectos que se buscan en los robots de rescate son: dimensiones pequeñas, agilidad y autonomía (con o sin supervisión humana).[9] Existe también una rama de los robots de rescate que se enfocan a los desastres en minas subterráneas, dichos robots exploradores de minas subterráneas aún se encuentran en una etapa temprana y necesitan ser investigados más a fondo para que puedan ser ocupados con regularidad en este tipo de escenarios.

A pesar de la urgencia de las situaciones de derrumbes en una mina, los robots de rescate son desplegados en un promedio de 6.5 días después del evento de desastre. Esta problemática afecta directamente la posibilidad de localizar sobrevivientes. *Miller* et al, mencionan en "Miller's Anesthesia E-Book" que la tasa de supervivencia comienza a decrecer (para el caso de víctimas atrapadas en la zona de desastre) a partir de las primeras 24 a 48 horas, dicha relación se encuentra presente en la mayoría de los escenarios de desastre urbanos, como: derrumbes parciales o completos de edificios, derrumbes de puentes o carreteras, minas colapsadas y en general situaciones donde se

puedan encontrar víctimas atrapadas en ambientes aislados y peligrosos generados después de un fenómeno natural o humano. Los robots que requieren mayores costes de producción no son recomendables para ser desplegados en un escenario de desastre, se precisan de robots más simples y baratos para realizar este tipo de tareas.

Los escenarios peligrosos típicos de los desastres en minas subterráneas generan varios retos para los robots de rescate de propósito general, el terreno puede contener explosivos, gases tóxicos, humo muy denso, superficies inestables, rieles dispersos, piedras, peñascos, grava, fluidos líquidos, pendientes y escaleras, e inclusive redes eléctricas colapsadas.[8]

La primera intervención de robots de rescate en minas subterráneas fue en 2001 en la mina no. 5 de "Jim Walter", desde entonces los robots de rescate se han desplegado en nueve de once desastres a gran escala en minas subterráneas, teniendo éxito solo en tres misiones; seis de las misiones fracasaron debido a fallas graves en los robots al tratar de sortear terrenos extremadamente dañados.

1.7. Características de los robots de rescate

Los robots de rescate entran en la categoría de los robots móviles, estos robots están construidos para sensar y actuar en un determinado escenario, en términos de inteligencia artificial este tipo de robots son llamados "agentes físicamente situados". Tienen que tener la capacidad de moverse por un "mundo impredecible", deben de estar conscientes del entorno y no solo en los aspectos de ubicación y navegación, si no también deben de prevenir el ocasionar colapsos secundarios, mover o recoger escombros (a menos de ser completamente necesario) y no perturbar evidencia forense.

En base a la literatura encontrada existen tres aspectos básicos que deben tenerse en cuenta en el diseño y fabricación de un robot de rescate:

- Operación en terrenos extremos, las condiciones de operación son afectadas por el tamaño del robot, el desempeño de los sensores, y la capacidad de supervivencia general del robot.
- Habilidad de funcionar en condiciones donde el terreno o el entorno no permitan una buena comunicación GPS o inalámbrica.
- Un sistema de control y adquisición de datos apropiados.

Las comunicaciones inalámbricas pueden ser mejoradas proporcionando una buena potencia, pero conlleva generar un robot con mayores dimensiones y más pesado, para poder llevar los métodos de alimentación para satisfacer esta comunicación.

La inteligencia artificial está enfocada en hacer al robot completamente autónomo, este método ha demostrado ser bastante eficiente.

Robin R. Murphy menciona también que los robots de rescate buscan ser sistemas tácticos, orgánicos y sin tripulación, que permiten a personas especializadas en emergencias percibir y actuar a distancia en tiempo real.[9]

Existen tres categorías básicas donde se pueden encontrar este tipo de sistemas (hablando de los robots de rescate) y son:

- Para despliegues en tierra, son llamados "vehículos terrestres no tripulados" (unmanned ground vehicles, UGVs).
- Para despliegues acuáticos, la mayoría son contenidos en la categoría de "vehículos marinos no tripulados" (unmanned marine vehicles, UMVs).
- Para despliegues aéreos, los vehículos se encuentran en varias categorías de las cuales incluyen "vehículos aéreos no tripulados" (unmanned aerial vehicles, UAVs) siendo estos uno de los términos más famosos, aunque también existen los "sistemas aéreos no tripulados" (unmanned aerial systems, UASs).

Cada desastre presenta características diferentes y por ello no se puede generar un modelo lo suficientemente fiable para todos los tipos de escenarios de desastre urbano posibles. Es por ello que el robot debe de ser dotado de "inteligencia" para poder superar la mayor cantidad de problemas que pudiesen presentarse en el escenario, y cumplir su tarea de localizar víctimas en las zonas donde sean desplegados.

Los robots de rescate ayudan a las partes interesadas a prevenir, preparar para, responder a, y recuperarse de los desastres urbanos en eventos extremos, estos robots proveen la habilidad de acceder a el área de interés. Si ocurre un desastre, puede suceder que sea físicamente imposible, demasiado peligroso, o ineficiente para que rescatistas sean desplegados en la zona de interés. Los robots no buscan remplazar a las personas o perros entrenados para estos eventos, sino más bien fungir como un complemento para las habilidades de humanos y caninos, y también ayudar a mitigar el riesgo que se crea en un despliegue humano en una zona de desastre.

1.8. Estadísticas posteriores a un desastre urbano o incidente extremo

Los desastres generan un gran impacto en la vida de una ciudad, la calidad de vida, economía, y el medio ambiente, y si no es posible prevenirlos, una respuesta rápida y adecuada representa una menor pérdida de vidas y menores problemas a largo plazo para la zona afectada. También favorece a una recuperación más rápida de la economía y del ambiente.

La Agencia Federal del Manejo de Emergencias de Estados Unidos establece muchas definiciones para el termino desastre, una de ellas es: "Un evento que requiere recursos y capacidad más allá de una comunidad y requiere la respuesta de múltiples agencias". Existe una gran variedad desastres, entre ellos, los eventos en lugares urbanizados son los más significativos en términos de muertes e impacto económico.

En el Informe Mundial de Desastres 2015 generado por la "Federación Internacional de Sociedades de la Cruz Roja y de la Media Luna Roja" (IFRC por sus siglas en inglés) señala que entre 2005 a 2014 se contabilizaron 1,254 desastres relacionados con terremotos, donde América ha padecido en el mismo periodo 1,242 desastres (que engloban inundaciones, terremotos, tormentas, olas de calor y sequías).[10]

Las estadísticas creadas desde el terremoto en la Ciudad de México en 1985 y subsecuentes desastres, establecen que solo una pequeña cantidad de víctimas atrapadas sobreviven en un desastre de tipo urbano, el ochenta por ciento de los sobrevivientes a los desastres urbanos son "víctimas en superficie", este tipo de víctimas se encuentran en zonas abiertas en la superficie donde son visibles para el rescatista. Solo el 20 % restante de las víctimas son personas que se encuentran al interior de edificios, casas, departamentos, etc. Estas víctimas pueden encontrarse principalmente en 2 casos: con daños físicos mínimos (lesiones pequeñas o relativamente pequeñas) y enterrados o atrapados debajo de escombros donde sí se encontrasen en el segundo caso suelen necesitar de asistencia médica urgente.

Por ultimo, tenemos que el índice de mortalidad de las personas enterradas o atrapadas después de un desastre urbano incrementan y llegan a su máximo después de las primeras 48 horas, esto conlleva que los sobrevivientes que no son rescatados dentro del rango de las 48 horas después del evento tienen pocas posibilidades de sobrevivir aun después de ser rescatadas y llevadas a un hospital. Estas estadísticas sugieren que, si los robots pueden penetrar al interior de los escombros para realizar una búsqueda rápida, rescatar e intervenir podrían ayudar a reducir el numero de las víctimas mortales posteriores al desastre urbano.

Capítulo 2

Marco teórico

El prototipo ASTEC se considera como una sola unidad, dicha unidad representa todo el robot, que debe contar con las características vistas en la bibliografía, algunas de ellas son: capacidad de percibir su entorno (sensores, cámaras, etc), capacidad de comunicación (inalámbrica para el envió de datos de la situación de rescate durante el despliegue) y capacidad de autonomía (métodos de búsqueda, capacidad de superar o evadir obstáculos, independencia energética, etc).

2.1. Diagrama general del robot prototipo ASTEC

Para el diagrama general del robot ASTEC se plantea basado en los conceptos básicos revisados en las referencias [1] - [13].

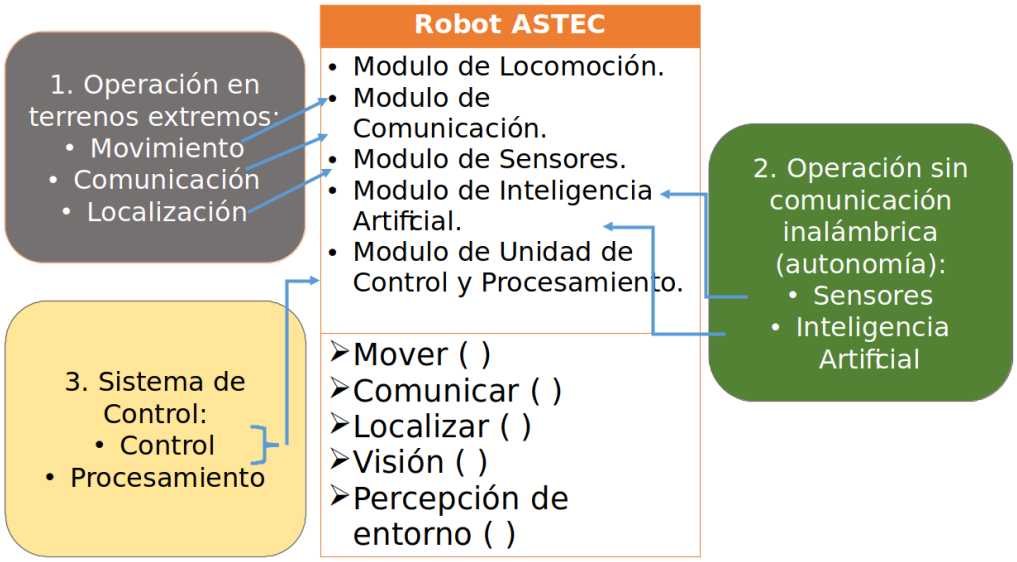


Diagrama 2.1.1. Características básicas (como robot de rescate) del prototipo ASTEC.

Observando el diagrama (2.1.1), se tienen de manera explicita las características básicas necesarias en un robot para rescate o desastre. El robot ASTEC debe contar con

tres características básicas, donde la "Operación en diferentes terrenos" es asumido por los módulos de locomoción y comunicación, esta característica contempla los aspectos necesarios para realizar una operación en terrenos de difícil acceso, irregulares, que presenten obstáculos, etc.

Después, se tiene la característica de "Operación sin comunicación inalámbrica (autonomía)", en esta característica con respecto a las referencias, se observó que existen dos vertientes principales, estas vertientes se basan cumplir un despliegue donde exista la posibilidad de perder comunicación inalámbrica, entonces, algunos investigadores optan por dotar al robot con un método de comunicación alámbrica (par trenzado, fibra óptica, etc.) para garantizar el control del robot, así como su capacidad de enviar información durante el despliegue; sin embargo, esto representa una gran cantidad de problemas para el robot durante el despliegue y la posterior recuperación del mismo (si pudiese darse el caso), dado que el hecho de poseer un cable en el robot puede aumentar considerablemente el peso del robot, la distancia efectiva de despliegue, entre otras cosas. Por ello, existe una segunda vertiente en la que el autor de la tesis considera una forma mas efectiva para cumplir esta característica, este método se basa en dotar al robot con autonomía, donde en el diseño de ASTEC, los módulos encargados de dar las condiciones para cumplir este aspecto serian: el modulo de sensores y el modulo de inteligencia artificial (donde se contemplan las técnicas de localización, mapeo y trazado de ruta).

Por ultimo, se tiene la característica respecto al "Sistema de control", donde se establece el control de cada modulo mencionado de las otras dos características, además, este modulo también se encarga de las tareas de procesamiento y tratamiento de la información adquirida, completando un método de adquisición de datos, el cual presenta elementos bastante importantes, ya que este método altera capacidad de la "inteligencia artificial" del diseño, dando información que permita corregir los métodos de búsqueda, establecer nociones del entorno para establecer las rutas de búsqueda efectivas, entre otros aspectos fundamentales para el éxito del despliegue.

En el diagrama (2.1.2) se realiza una separación modular para establecer los aspectos más relevantes para esta tesis, en función de las características básicas encontradas en la literatura que se refieren a un robot para rescate o desastre (además de los robots de exploración e intervención), y como serán revisados a profundidad a lo largo del marco teórico. La finalidad del diseño de ASTEC es generar un sistema integral capaz de realizar tareas de búsqueda y localización de manera autónoma. Además, en el diagrama (2.1.2) se muestran los módulos básicos (sensores, locomoción, comunicación y control/procesamiento) y un modulo propuesta (inteligencia artificial), el modulo de "inteligencia artificial" será el que tomara el rol mas importante a lo largo de la tesis.

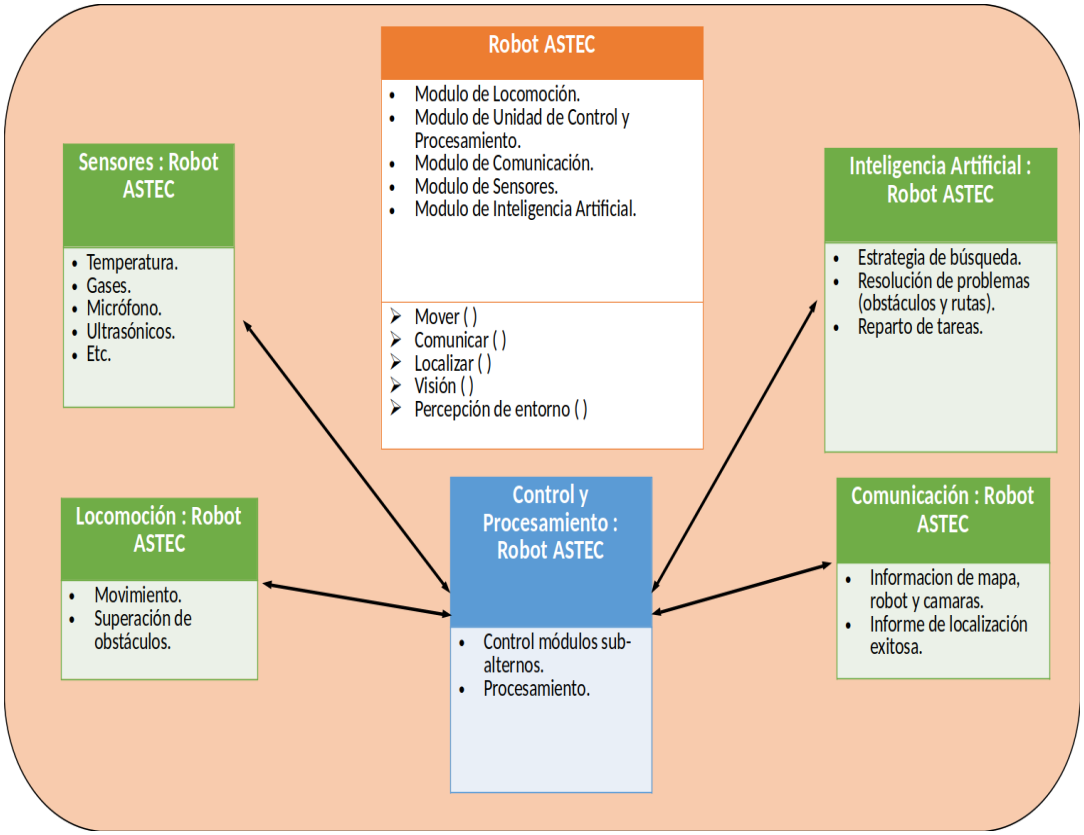


Diagrama 2.1.2. Módulos del robot ASTEC.

En resumen, las características más relevantes que se buscan en el diseño de ASTEC son:

- Capacidad de moverse por lugares estrechos y sortear escombros.
- Contemplar la incorporación de al menos una vía de comunicación, de corto alcance o mediano alcance.
- Sensores para percibir obstáculos cercanos, características del entorno, detección de víctimas y posibles peligros potenciales.
- Control del sistema basado en una placa de desarrollo (preferentemente una microcomputadora), con la finalidad de realizar las tareas de control de actuadores y procesamiento de la información obtenida de los sensores.
- Implementación de algoritmos de IA para mejorar los procesos de búsqueda, detectar características de condiciones que indiquen la ubicación de una posible víctima.

2.2. Separación modular del robot ASTEC

Usando el diagrama (2.1.2) se realiza la separación del robot ASTEC en módulos, esto permite tener una mayor profundización en los aspectos necesarios de conocimiento que

conforman al diseño, con esto se busca generar una investigación sistematizada para cada módulo propuesto, con ello se tendrá información que permita determinar cuál o cuáles son las herramientas tecnológicas disponibles y costeables que sean más adecuadas para generar un diseño adecuado a las tareas que se plantean para ASTEC.

Los módulos planteados para ASTEC son los siguientes:

- Módulo de Locomoción.
- Módulo de Unidad de Control y Procesamiento.
- Módulo de Comunicación.
- Módulo de Alimentación.
- Módulo de Sensores.
    - Sensores para búsqueda y percepción de condiciones del entorno.
- Módulo de Inteligencia Artificial.
    - Algoritmo de exploración.
    - Algoritmo de trazado de ruta.
    - Algoritmo de localización y mapeo.

2.2.1. Módulo de Locomoción

El módulo de locomoción, o en general los aspectos mecánicos del movimiento de los robots móviles y en específico de los robots para desastres, intervención y exploración, son sumamente importantes al abordar escenarios complejos de operación. *Robin R. Murphy* señala que el éxito de las misiones en zonas de desastre depende en gran medida de la locomoción de los robots móviles implementados [9], siendo que las características para superar obstáculos, escombros, subir escaleras, desplazarse por terrenos irregulares y/o blandos, etc. están directamente relacionados con la locomoción del robot.

En la revisión bibliografía que se realizó sobre los métodos de locomoción existentes en robots para desastres, intervención y exploración; se encontraron los siguientes tipos de locomoción:

- Locomoción mediante ruedas.
- Locomoción mediante rodamiento por oruga.
- Locomoción mediante patas.
- Locomoción Híbrida (esta engloba usualmente la combinación de al menos dos tipos de locomoción como: ruedas, oruga y patas).

Locomoción mediante ruedas

La locomoción mediante ruedas es el método más común de desplazamiento para una maquina terrestre [6], existen varios tipos de locomoción que incluyen el uso de ruedas como lo son:

- Ruedas motrices o de tracción: Ligadas a ejes motrices donde el movimiento se provee mediante un actuador, pueden usarse motores de CC o CA dependiendo de la aplicación. Los métodos de control para estos sistemas son bastante fáciles.
- Ruedas direccionales: Están unidas a ejes no motrices, su orientación depende del plano paralelo al desplazamiento, la mecánica resulta bastante compleja y por ello no es muy usual su implementación en la robótica.
- Ruedas libres o auxiliares: Este tipo de ruedas gira libremente sobre su eje, su punto de apoyo puede ser libre u oscilante.
- Ruedas omnidireccionales: Debido a su configuración de rodillos permite giro libre en cualquier dirección.

Existen 3 inconvenientes básicos asociados a los vehículos de locomoción con ruedas: limitación de obstáculos que puede sortear el robot [4], limitación de terrenos de operación, y limitación en los movimientos disponibles del robot. [1]

Los sistemas de locomoción a ruedas tienen un consumo de energía relativamente bajo [6] y permiten alcanzar altas velocidades en superficies regulares, sin embargo, es necesario que un sistema de locomoción a ruedas posea ruedas de un diámetro largo para ser capaces de sortear terrenos discontinuos. [8]

En resumen, las características más importantes se presentan a continuación:

- Sencillez en su mecánica y control. [1] [6] [7]
- Alta velocidad en superficies regulares, y baja velocidad de operación en superficies irregulares. [8]
- Limitación de movimientos disponibles. [1]
- Las ruedas “libres” pueden facilitar el movimiento general del robot, sin embargo, reducen considerablemente su operación efectiva en terrenos discontinuos e irregulares. [6]
- Modesta eficiencia energética debido a la poca perdida de energía por rozamiento y bajo consumo de energía. [6] [8]

- Dificultad al superar obstáculos cuyas dimensiones son similares o superiores al diámetro de las ruedas. [1] [4] [6] [8]
- La mayoría de aplicaciones son como robots de servicio o aplicaciones en superficies regulares, también pueden encontrarse en exteriores pero en terrenos no tan complejos. [1] [6]
- Problemas para desplazarse en terrenos irregulares. [4] [7]

Locomoción mediante rodamiento por oruga

Los sistemas de locomoción mediante rodamiento por oruga constan principalmente de uno o varios actuadores que son activados para rotar una banda dentada que permite el movimiento del chasis del robot. Los robots móviles basados en carros o plataformas y dotados de un sistema locomotor de tipo rodante, son robots con grandes capacidades de desplazamiento. [3]

Cuando la superficie del terreno es "blanda", movediza o incluso pedregosa, resulta conveniente emplear vehículos con sistemas de locomoción de tipo oruga, debido a su distribución de peso, tienen una mayor eficiencia en el agarre [1]. Incluso en terrenos abruptos, irregulares o deslizantes y con grandes obstáculos, los sistemas de locomoción basados en orugas son más eficientes que los basados en locomoción mediante ruedas. Con estos sistemas se consigue una mayor tracción y mantiene la sencillez de la mecánica y control de los sistemas basados en ruedas. [6]

Los robots que usan sistemas de locomoción mediante rodamiento por oruga han demostrado gran estabilidad, poca presión sobre terrenos y un control bastante simple, es por ello que este tipo de robots se han implementado ampliamente para aplicaciones en terrenos irregulares [7].

Las características más importantes de este tipo de locomoción son:

- Aplicación para terrenos irregulares o abruptos (superficies blandas, superficies movedizas, superficies pedregosas, grandes obstáculos). [1] [5] - [8] [12]
- Ofrecen una velocidad moderada en terrenos irregulares. [8]
- Los métodos de odometría no son fiables para la auto-localización del robot. [6]
- Sencillez en la mecánica y control del sistema. [6] [7]
- Grandes capacidades de desplazamiento. [3] [12]
- Buena distribución de peso, mejora la tracción y el agarre. [1] [5] - [7]
- Problemas relacionados con la eficiencia energética (perdidas por rozamiento). [6] [8]

Locomoción mediante patas

El método de locomoción mediante patas es el más común en la naturaleza, ofrecen gran maniobrabilidad y posibilidad de operar en terrenos irregulares, y también presentan gran eficiencia energética [6].

Una cualidad importante que permite el sistema de locomoción mediante patas es el que menciona *Molyneaux*, et al. en la memoria de conferencia publicada en el año 2015, donde los grados de libertad que permite un sistema de locomoción a patas permite superar los obstáculos más difíciles (en comparación con otros sistemas simples de locomoción), pero la complejidad, el costo, el consumo de energía y la velocidad limitada son imprácticas cuando se aplican para situaciones de desastre. [8]

Los movimientos básicos de avance y giro en el caso de los robots con locomoción mediante patas, necesitan de al menos dos grados de libertad por cada pata, donde pueden aplicarse actuadores individuales para cada grado de libertad. [6]

A continuación, se destacan elementos importantes de este tipo de locomoción:

- Complejidad para el control, debido al número de actuadores. [6] - [8]
- Capacidad de operar en terrenos muy irregulares y con obstáculos. [2] [3] [6] - [8]
- Operación en baja velocidad en terrenos irregulares. [7] [8]
- Costo elevado en comparación con sus similares a ruedas y oruga. [8]
- Gran consumo de energía, debido al número de actuadores usados, sin embargo, mantienen gran eficiencia energética (pocas perdidas por rozamiento). [6] [8]
- Permite tener cuatro o más grados de libertad. [1] [8]
- Poca capacidad de operación autónoma. [1]
- Problemas de estabilidad estática y dinámica. [6] [7]

Locomoción híbrida

Los sistemas basados en locomoción híbrida tienen muchas variantes en cuanto a su posible clasificación, debido a ello, en esta tesis se consideró que si existe la combinación de dos sistemas de locomoción clásicos (ruedas, oruga o patas) se asumirá como un sistema de locomoción híbrida. Los sistemas de locomoción híbrida más comunes son los sistemas basados en una combinación de patas y ruedas, o patas y orugas.

La locomoción híbrida permite adquirir las ventajas de cualquier tipo de locomoción clásico (ruedas, orugas o patas), con la finalidad de reducir los padecimientos que todo

tipo de locomoción tiene. Estos sistemas de locomoción se implementan principalmente para resolver despliegues en escenarios donde se precisa de flexibilidad en la locomoción. [6]

Existen también sistemas de locomoción híbridos que incorporan sistemas de sub-pistas que también permiten al robot subir escaleras, y con ello tener acceso a pisos superiores cuando son desplegados en una casa u edificio que posea mas de un nivel. Dos trabajos donde se puede apreciar la aplicación de las sub-pistas son: el robot de *K. Nagatani*, et al. en la memoria de conferencia publicada en 2011, el robot "Quince" incorpora un sistema de locomoción híbrido con cuatro actuadores que cuentan con locomoción por rodamiento de oruga y la capacidad de rotar dichos actuadores para superar una gran variedad de obstáculos. [24]

Los aspectos más relevantes de la locomoción híbrida son listados a continuación:

- Presentan una complejidad intermedia en lo que respecta a la mecánica y control. [6]
- Tienen la mejor capacidad para desplazarse en terrenos irregulares y con escombros. [2] [4] [6] [12] [21]
- Facilidad para superar distintos obstáculos. [2] [4] [6] [12] [21]
- Pueden alcanzar altas velocidades en terrenos discontinuos, en comparación con sistemas de locomoción convencionales (ruedas, patas u orugas). [2]
- Usualmente son de tamaño más grande, esto genera que tengan un consumo de energía mas alto que los sistemas de locomoción mediante ruedas u orugas. [6] [12]

2.2.1.1. Comparativa de locomociones

Con la finalidad de elegir el tipo de locomoción mas adecuado para el diseño de ASTEC, se presentan una serie de tablas donde se evalúan los distintos métodos de locomoción expuestos en la tesis. Primero se mostrará la tabla de definiciones (Tabla 2.2.1.1.1) que hace referencia a las características que se abordan en cada criterio de evaluación, posteriormente vendrá la tabla comparativa de las locomociones (Tabla 2.2.1.1.2). Después, se muestra la escala de colores (Tabla 2.2.1.1.3) usada en la tabla comparativa de las locomociones, la siguiente tabla se refiere a las escalas de puntuaciones para los aspectos evaluados (Tabla 2.2.1.1.4) de la tabla comparativa, y por ultimo, se muestra la tabla de puntuaciones finales (Tabla 2.2.1.1.5) de cada locomoción que servirá para determinar el método mas conveniente para el diseño del robot ASTEC.

Definiciones
Superación de Obstáculos (SdO). Capacidad de superar (puntuación de 0 a 3): escalones, piedras o escombros, y obstáculos de tipo arquitectónico.
Operación en Diferentes Terrenos (OeDT). Se contempla la capacidad de moverse en terreno (puntuación de 0 a 3): blando o movedizo, pedregoso o escabroso (superficies muy accidentadas) y terreno discontinuo o irregular.
Complejidad de Control (CdC). Se clasificará en una escala cualitativa (poca, media o alta), tomando en cuenta los aspectos de la complejidad de la mecánica y control del sistema de locomoción.
Velocidad en Terrenos Irregulares (VeTI). Se clasificará en una escala cualitativa (poca, media o alta), donde se evalúa la velocidad de operación en terrenos irregulares o discontinuos.
Consumo de Energía (CdE). Se usará una escala cualitativa (bajo, medio-bajo, medio-alto o alto), según sea el consumo de energía promedio de cada locomoción señalado en las referencias.
Costo. Se clasificará en una escala cualitativa (bajo, medio o alto). Se toma la media del precio del chasis que incorpora el tipo de locomoción basados en la consulta de algunas tiendas en línea.

Tabla 2.2.1.1.1. Definiciones para la tabla comparativa de las locomociones investigadas.

En la Tabla (2.2.1.1.1) se establece en forma breve las definiciones que se basan en la interpretación de lo dicho por los autores de las referencias usadas en la sección del módulo de locomoción, tratando de generar un estándar que permita comparar cada locomoción.

Locomoción	SdO	OeDT	CdC	VeTI	CdE	Costo
Ruedas	0	0	Poca	Baja	Bajo	Bajo
Oruga	2	2	Poca	Media	Medio-Bajo	Medio
Patas	3	2	Alta	Media	Medio-Alto	Medio
Híbridos	3	3	Media	Alta	Medio-Alto	Alto

Tabla 2.2.1.1.2. Comparativa de locomoción de robots para rescate, intervención y exploración.

Los métodos de evaluación y consideraciones sobre de el Tabla (2.2.1.1.2) son los siguientes:

- Evaluación mediante puntaje: Se puntuará con 1 punto por cada característica que la locomoción posea referente al apartado de las definiciones (SdO y OeDT), donde se podrá tener un mínimo de 0 y un máximo de 3.
- La evaluación de escala cualitativa (CdC, VeTI, CdE y Costo)se realiza recopilando los aspectos que señalan los autores de las referencias usadas para la sección de locomoción.

- La evaluación de escala en colores (Tabla 2.2.1.1.3) se establece para marcar los aspectos relevantes.

Escala de Colores
Insuficiente (0 puntos)
Aceptable (1 punto)
Adecuado (2 puntos)
Óptimo (3 puntos)

Tabla 2.2.1.1.3. Escala de colores, usada para clasificar aspectos del cuadro comparativo de locomoción.

- Se realiza una evaluación de todos los aspectos de la locomoción, basados en la escala de colores propuesta (Tabla 2.2.1.1.3), y utilizando la siguiente tabla de valores de las características más relevantes para el sistema (Tabla 2.2.1.1.4). [1]

Aspecto Evaluado	Puntuación
Superación de Obstáculos (SdO)	0 a 3
Operación en Diferentes Terrenos (OeDT)	0 a 3
Complejidad de Control (CdC)	0 a 2
Velocidad en Terrenos Irregulares (VeTI)	0 a 2
Consumo de Energía (CdE)	0 a 3
Costo	0 a 3

Tabla 2.2.1.1.4. Puntuaciones en función a la prioridad de las características evaluadas en el Tabla 2.2.1.1.2.

- Consideraciones para locomoción híbrida: Ya que la locomoción híbrida está basada en una combinación de dos o tres tipos de locomociones básica (ruedas, oruga, o patas) se contemplan ciertos aspectos evaluados como una combinación de las características individuales de estas locomociones, ya que no se encontró suficiente información que establezca estas comparaciones de manera explícita.

Entonces, usando la escala de colores (Tabla 2.2.1.1.3) y las puntuaciones en función de prioridad (Tabla 2.2.1.1.4), se puntuaron las locomociones del cuadro comparativo con la finalidad de establecer de manera clara cual es el método de locomoción mas adecuado para el prototipo ASTEC.

[1] Nota: Se aclara que la puntuación se propone contemplando los aspectos que se consideran prioritarios o no prioritarios para este trabajo.

Locomoción	Puntuación General
Ruedas	8 puntos
Oruga	10 puntos
Patas	8 puntos
Híbrida	10 puntos

Tabla 2.2.1.1.5. Puntuaciones generales de cada tipo de locomoción investigado.

Usando el Tabla (2.2.1.1.5), podemos decir que al priorizar ciertas características que se busca que el robot ASTEC posea (en su diseño teórico y simulado), resulta más conveniente incorporar la locomoción por rodamiento de oruga o la locomoción híbrida. En la evaluación general tanto la locomoción basada en orugas y la locomoción híbrida empatan en puntuación, sin embargo, debido a las características que se proponen para ASTEC, la superación de obstáculos y la capacidad de operar en diferentes terrenos son de mayor prioridad, por ello, se usará la locomoción híbrida para este robot.

Por ultimo, se señala que los otros métodos de locomoción (ruedas y patas) son también efectivos, dependiendo de las características buscadas para el prototipo, pueden o no ser una mejor opción para un robot de búsqueda y rescate.

El Tabla (2.2.1.1.5) será usada para fundamentar la elección de la locomoción en el siguiente capítulo de la tesis.

2.2.2. Módulo de Unidad de Control y Procesamiento

Una plataforma universal en cuanto a robots móviles inteligentes concierne (hablando de robots para rescate, intervención o exploradores), debe contar con las siguientes cualidades:

- Confiabilidad, robustez mecánica y un chasis simple.
- Construcción modular.
- Unidad de Control y Procesamiento que permitan su programación (preferiblemente en lenguajes de alto nivel, debido a la complejidad de las tareas a desarrollar).
- Posibilidad para agregar o desarrollar nuevos módulos para el sistema, como: subsistemas de censado, cámaras, actuadores, dispositivos de comunicación inalámbricos, etc.

Entonces, hablando del módulo de la “unidad de control y procesamiento” es el encargado, como su nombre lo dice, de las tareas de control y procesamiento del robot. Se plantea como un único módulo con la finalidad de que las etapas de control y

procesamiento trabajen de manera conjunta para realizar las tareas en el despliegue, donde se controlan los aspectos de movimiento, comunicación y obtención de información mediante sensores, donde también se procesan las señales obtenidas por la cámara, sensores más complejos y plantear las estrategias de búsqueda del robot.

En base a lo investigado, actualmente existen dos vertientes principales cuando se habla de plataformas de desarrollo (refiriéndose al hardware *OpenSource* en específico) de bajo costo, la primera está representada por microcontroladores (en su mayoría de 32-bits) que se separan en el rango de características que posea cada microcontrolador implementado en la placa, así como los periféricos disponibles de la misma.

De las vertientes encontradas, las plataformas:

- Arduino (basado en microcontrolador).
- Raspberry (microcomputadora).

Estas plataformas son las más destacadas, principalmente, en el hecho de ser *Open Source*, que presentan una gran flexibilidad de proyectos a realizar, esto genera que exista una amplia comunidad de desarrollo y soporte.

Ryan Krauss señala que un aspecto importante que se debe tomar en cuenta cuando se experimenta con un sistema de control para un robot móvil inteligente, es escoger el hardware y software correcto, ya que la implementación del control y la eficiencia del prototipo se basaran principalmente en el "cerebro" del sistema, además, de los dispositivos mecánicos y de censado externos a la unidad de control. [18]

Microcontrolador

En lo que respecta a plataformas de desarrollo que incorporan un solo microcontrolador, se pueden encontrar una gran variedad de soluciones. Donde las más populares se encuentran:

- Arduino (en el entendido de que estos dispositivos implementan un microcontrolador Atmel en su mayoría)
- TI LauchPad
- ARM mbed

Los microcontroladores y placas de desarrollo basadas en los mismos (PIC, Atmel, Arduino, LauchPad, etc.) son una opción razonable para la implementación de soluciones HIL (Hardware-in-loop) para el prototipado de control en tiempo-real, que se basa principalmente en control de sensores y actuadores que demanden pocos recursos.

Además, la mayoría de estas placas pueden ser programadas en C o C++, y cuentan con la capacidad de ejecutar tareas en intervalos duros de tiempo real usando interrupciones del TIMER. [14] [15]

La mayoría de las placas de desarrollo basadas en un micro-controlador comparten algunas características, cuentan con un solo procesador con memoria flash que puede ser programada mediante un programador especial, esto dependerá de la plataforma elegida, además, estas plataformas cuentan con muchos tipos diferentes de periféricos, como: convertidores analógico-digitales, puertos de entrada y salida digitales, 2 o más buses de I2C, puerto serial UART, etc.

Podemos decir que la mayor diferencia entre la lista de plataformas mencionadas son las herramientas de programación y debug (como se mencionó anteriormente), además de la comunidad de soporte, donde se pueden encontrar ejemplos, bibliotecas, guías, etc. Para una plataforma en específico, esto genera uno de los principales criterios para preferir una plataforma de desarrollo sobre otra. [20]

Microcomputadora

Las plataformas más populares que integran una microcomputadora son:

- Raspberry Pi
- Beagle Board/Bone

La principal ventaja de estas plataformas radica en la capacidad de correr distribuciones de sistemas operativos Linux, esto permite al usuario desarrollar algoritmos experimentales desde una PC o laptop. La versión final del sistema o del programa, puede ser implementada sin cambios importantes, trasladando directamente el programa a estas plataformas. Esto genera un factor considerable de reducción en el tiempo de desarrollo, y también reduce el conocimiento previo de dicha plataforma.

Raspberry Pi (y en general, cualquier placa o plataforma de desarrollo basada en una microcomputadora) permite programarse en lenguajes de alto nivel y usar diferentes sistemas operativos, con ello permite generar proyectos más robustos y aplicaciones que demanden tareas de gran procesamiento. Comparado con plataformas de desarrollo basadas en microcontroladores, las plataformas que integran microcomputadoras tienen mayor capacidad en el manejo y procesamiento de información que se obtiene en despliegues en campo. El poder computacional de una microcomputadora moderna permite la implementación de varios módulos alternos, módulos de sensores, cámaras y módulos de comunicación, e inclusive implementar programas y algoritmos que demanden gran cantidad de recursos. [15]

También se menciona que una microcomputadora permite incrementar las capacidades del sistema (debido a las características técnicas de las placas) embebido del robot móvil, sin la necesidad de agregar mas unidades de control y procesamiento. Se ha observado en varios proyectos de exploración, rescate o desastre, los desarrolladores optan por el uso de la Raspberry Pi en prototipos de robots móviles de alto nivel.

Elección de la unidad de control y procesamiento del robot ASTEC

Para la "unidad de control y procesamiento" del diseño ASTEC, se plantea implementar una plataforma de desarrollo basada en una microcomputadora. [17] [18] [25]

Esta unidad puede considerarse como la segunda mas importante de ASTEC, después de la unida de locomoción, dicho esto, esta unidad usará una plataforma Raspberry Pi 3 B para conformar la "Unidad de Control y Procesamiento".

Debido a las características tanto de hardware como de software mencionadas, la placa que desarrollo que se usará será la Raspberry pi que integra una microcomputadora, debido a que permite generar protocolos y características mucho más complejas que son necesarias para cumplir con los objetivos planteados de esta tesis.

Raspberry Pi, cuenta con una comunidad muy amplia de desarrollo y soporte, ya que esta plataforma también es considerada Hardware y Software *Open Source*. También, se menciona que la Raspberry Pi será programada a través de Python, ya que, con base a las referencias encontradas, muchos sistemas que incorporan "inteligencia artificial" o "machine learning" pueden ser programados en Python, además de contar con una gran variedad de bibliotecas, ejemplos de programas, y ser un intérprete muy intuitivo y fácil de programar en problemas más complejos, donde se destaca que para el diseño ASTEC en especifico.

La Raspberry Pi se encargará de operar los módulos de "Locomoción", "Sensores", "Comunicación", e "Inteligencia Artificial", donde se menciona que este último (módulo) es el más demandante en cuestiones de recursos.

En la Diagrama 2.2.2.1, se expresa de manera general el funcionamiento deseado de la "Unidad de Control y Procesamiento", así como su relación con los módulos que integran en ASTEC, a lo largo de la tesis se continuará abordando de manera mas profunda cada uno de estos aspectos, como se ha hecho con la locomoción, control y procesamiento del sistema.

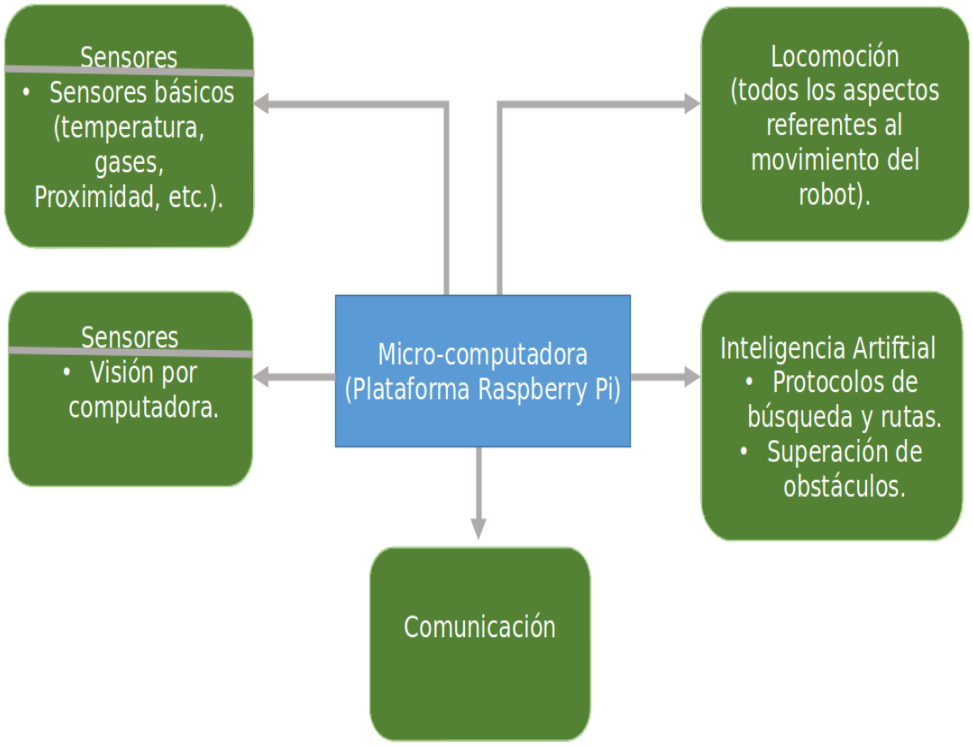


Diagrama 2.2.2.1. Módulos del robot ASTEC.

2.2.3. Módulo de Comunicación

Para el módulo de comunicación, existen dos métodos principales en los cuales se pueden catalogar los robots para rescate. Estos métodos son:

- Comunicación alámbrica
- Comunicación inalámbrica

La comunicación alámbrica e inalámbrica son combinadas en un gran número de robots de rescate o exploración, esto se debe al factor de redundancia, la idea de mantener comunicación sea alámbrica o inalámbrica, es indispensable en la mayoría de los proyectos investigados por el autor, dado que la mayoría carecen de una verdadera autonomía, y dependen en su mayoría de los comandos de un operador humano para las tareas de control e interpretación del entorno.

Ahora bien, la mayoría de los robots investigados que incorporan comunicación alámbrica, utilizan fibra óptica como el método "ideal", sin embargo, existen muchos inconvenientes con el uso de los sistemas alámbricos, algunos de estos son:

- Distancia de operación limitada a la longitud del cable.

- Problemas de atascamiento del cable debido a esquinas, escombros, etc.
- La fractura del cable representa una perdida total de comunicación.

Otro aspecto que es importante considerar al implementar la comunicación alámbrica, es el aumento de peso y el método de despliegue del cable para evitar los problemas presentados con anterioridad, sin embargo, al presentar un problema de peso y dimensiones, se decidió para los propósitos del desarrollo de ASTEC, seria mejor optar por un método de comunicación inalámbrico.

En lo que respecta a la implementación de comunicación inalámbrica en robots de rescate, existen varios métodos que pueden ser incorporados, la mayoría de ellos usan algún módulo de radiofrecuencia como el método de comunicación, esto se debe a la gran variedad de frecuencias y dispositivos que existen y que pueden ser usados para estas tareas.

En algunos proyectos se ha encontrado que en base a la elección de plataformas Hardware *Open Source* como: Raspberry, Arduino, BeagleBone, etcétera, existe una tendencia de implementación de comunicación inalámbrica tipo 2.4GHz, a través de módulos de comunicación de Wi-Fi o XBee, esto se debe a que este ancho de banda (que corresponde a los 2.4GHz) es usado principalmente para aplicaciones científicas, industriales y médicas [30], y es por ello que existen actualmente muchos protocolos y módulos de comunicación generados para esta frecuencia de "uso libre". Estos métodos se pueden observar en los proyectos de: "Robot Explorador UDA" usa Wi-Fi [3], "Diseño y construcción de un robot explorador de terreno" que usa transreceptores RMB-CM12111 (431Mhz a 478Mhz) [3], "Autonomous Stair-Climbing With Miniature Jumpling Robots" usa radiofrecuencia de 900Mhz (comentan que la señal tiene una mayor facilidad de penetrar objetos a esa frecuencia) [4], "HADES" usa radiofrecuencia a 2.4GHZ con módulos XBee-Pro [8], etc.

En el caso de la comunicación inalámbrica también existen varios inconvenientes, en la literatura investigada los mas mencionados son:

- Distancia efectiva de envió y recepción de la señal.
- Consumo de energía vs el alcance y potencia de la señal.
- Perdida de información debido a las condiciones del entorno.

Elección del módulo de comunicación

En base a lo investigado, la elección mas adecuada para los propósitos de la tesis, se basara en el uso de un modulo Digi Xbee, estos módulos son ampliamente usados en

aplicaciones para interiores, desde sistemas domóticos hasta aplicaciones para instrumentación remota. La frecuencia de operación del módulo se observa principalmente en dos anchos de banda.

- Frecuencia de operación a 900Mhz - 915Mhz
- Frecuencia de operación a 2.4Ghz

En el artículo publicado por *Scott Keller* [31], fundador y CEO de "SignalFire Wireless Telemetry", hace una comparativa entre estos dos tipos de frecuencias para aplicaciones de monitoreo y sistemas de control, en el artículo se experimenta con la relación entre la potencia de señal y el número de pisos de cobertura (en una edificación), donde se comparan ambos rangos de frecuencia y se concluye que para aplicaciones en interiores es mejor la frecuencia de 2.4Ghz, dado que presenta una menor atenuación entre pisos y una menor perdida de información, y para aplicaciones en exteriores se recomienda el uso de frecuencias de 900 - 915Mhz, ya que permiten una menor perdida de potencia en relación a la distancia entre emisor y receptor.

- Digi Xbee Zigbee de 2.4Ghz
- Digi Xbee 802.15.4 (2.4Ghz)

Dado que ambos productos pueden operar en este ancho de banda, la elección de alguno de estos módulos se podrá realizar en base a su precio y disponibilidad, además, se menciona que pueden existir otros módulos de comunicación inalámbrica que cuenten con estas características, dado que el ancho de banda para 2.4Ghz es muy común para distintas aplicaciones en interiores, por lo que también se propone elegir un módulo genérico de comunicación considerando su precio en relación a los Digi Xbee.

2.2.4. Módulo de Sensores

El módulo de sensores contempla los elementos de hardware necesarios para percibir características del entorno del robot durante el despliegue, un gran número de robots de rescate, servicio o exploración son dotados de un gran número de sensores, de los cuales la mayoría son sensores para medición de distancias (ultrasónicos, láser, etc), sensores de temperatura, acelerómetros, etc. [26]

La mayoría de los sensores pueden ser clasificados en cuatro características [29] (cuando hablamos de robots móviles), estas características son:

- Percepción interna o externa:

    - Propiocertivo: Se refiere a sensores que perciben información interna del robot (encoders, carga de las ruedas, ángulos de actuadores, estado de componentes del sistema, estado de batería, etc.)
    - Exteroceptivo: Se refiere a sensores que captan características e información del entorno (medidores de distancia, luminosidad, gases, temperatura, sonido, etc.)

- Tipo de captación, pasiva o activa:

    - Pasivos: Son sensores que captan la energía emitida del entorno (sondas de temperatura, micrófonos, cámaras térmicas, etc.)
    - Activos: Los sensores activos, emiten energía al entorno y miden la reacción del entorno a esta interacción (sensores ultrasónicos, sensores láser, etc.)

Existen algunos sensores que se consideran fundamentales durante las tareas de búsqueda y rescate, los que se contemplan para el diseño de ASTEC, son los siguientes:

- Sensores para medir distancias, en su mayoría se encuentran en 2 vertientes: ultrasónicos y/o láser. [15]
- Sensor de temperatura (preferentemente, cámara térmica).
- Cámara USB.
- Sensor de estado y carga de batería.

Tipos de sensores más comunes para búsqueda y rescate

El robot SALVOR [11] es un robot semiautónomo dotado con varios sensores, dichos sensores tienen la tarea de captar información del entorno del despliegue para realizar un número de tareas pre-programadas y para servir de guía para el usuario que controla al robot de manera remota, este robot cuenta con un sensor de gas y un sensor de sonido, como sensores básicos para percibir algunos peligros potenciales (como fugas de gas), y también percibir sonido que pueda indicar la localización de alguna víctima. SALVOR también posee cámara térmica y una cámara CMOS para ayudar en la detección de víctimas y obstáculos del entorno

En el artículo publicado por *Alberto Marroquin*, et. al, [32] su robot explorador incorpora un sistema dual de control y procesamiento, con una Raspberry Pi y un Arduino nano, con la incorporación de una cámara y un sensor de temperatura y humedad del aire, la información de los sensores y el vídeo de la cámara son enviados a una aplicación web, por donde también es controlado dicho robot.

En el proyecto llamado “Voice controlled Humanoid Robot with artificial vision” [33], a pesar de tratarse de un robot humanoide, cuenta con sensores similares a los encontrados en otros tipos de robots como SALVOR o CALEB-2, donde en se incorpora el módulo con cámara para Raspberry Pi (5 Mpx) que incluye también un sensor Omnivision OV5647, esta cámara se usa como detector de rostros usando la biblioteca OpenCV. Además, el robot humanoide pose una placa de reconocimiento de voz “EasyVR” que contiene un micrófono y un parlante.

Elección de sensores para el prototipo ASTEC

Tomando en cuenta estos ejemplos y en general la bibliografía encontrada, los sensores que se consideran más importantes para el diseño de la tesis, son:

- Cámaras: Se considera incorporar una cámara monofocal (por cuestiones de disponibilidad y costo) que sea utilizada para crear un registro en imagen (esto resultará fundamental para que el receptor pueda valorar e interpretar la información) de un lugar donde se sospeche la presencia de una o más víctimas.
- Sensor de temperatura: Se propone integrar una cámara térmica, ya que tienen la capacidad de captar un espectro de calor en un área más amplia (además, este tipo de sensor no necesita estar en contacto físico con el objeto a medir) en comparación con otro tipo de sensores térmicos.
- Sensores de proximidad: Para poder percibir las distancias a los objetos y/o víctimas, se utilizará un sensor LIDAR, debido a que permite tener una mayor confiabilidad en las mediciones, junto con una mayor distancia de medición (aproximadamente 12 - 14 metros según lo investigado).
- Sensores de control: Son aquellos que recaban información interna del robot, como puede ser velocidad, aceleración, estado de la batería, etc.
- Sensores de sonido: Los sensores de sonido tienen como función principal reconocer patrones de voz de víctimas en el escenario de despliegue de los robots.

Tomando el listado anterior, se mencionan dos características relevantes para los propósitos de este trabajo:

- Existe una ventaja técnica, al usar un sensor de proximidad (sensor de para medir distancias) para las tareas de localización y mapeo tanto del robot como de los objetos, es un consumo considerablemente menor a otros algoritmos de 3D o que incluso pueden usar imágenes para realizar el mapa de búsqueda, trazado de ruta,

detección de objetos, etcétera. Y considerando los recursos limitados que se establecen para el diseño, es un sensor de suma importancia para poder realizar las tareas de localización y mapeo, trazado de ruta y exploración de la zona de despliegue.

- La cámara térmica que permitirá tomar muestras de calor bajo ciertos protocolos, esto también funcionará como información de apoyo además de las imágenes tomadas por la cámara web, ya que al no realizar procesamiento de imagen (usando la cámara monofocal) en el lugar, el robot utilizará la información obtenida de la cámara térmica con un factor predominante para determinar la localización de un víctima.

2.2.5. Alimentación

La alimentación o módulo de alimentación para el diseño de ASTEC, se contempla básicamente como la "batería" del robot, esta batería debe proporcionar la energía necesaria para activar el robot, alimentando la unidad de control y procesamiento, el módulo de comunicación y el módulo de sensores.

En las distintas áreas de la robótica móvil, como lo son los robots para búsqueda y rescate, operaciones militares, minería y exploración planetaria, el módulo de alimentación (batería) es muy importante para poder permitir a los robots poseer algún grado de autonomía, en muchos casos el desempeño de estos robots se relaciona de manera directa con las fuentes de poder que estos tengan, ya que en la mayoría de los ambientes (sobre todo ambientes complejos y de difícil acceso como se plantea para el diseño de ASTEC) no existe la posibilidad de recargar estas fuentes de poder (batería), además es importante que la posea uno o varios métodos para ser monitorizada, donde se pueda conocer el "porcentaje de carga" de la batería, con la finalidad de permitir operarla en rangos adecuados de carga, prolongar la vida útil de la batería, evitar sobrecargas, etc. [56]

En general, existen dos tipos de baterías (que pueden englobarse como "celdas primarias" o "celdas secundarias"), las del primer tipo son:

Las baterías alcalinas son sensibles a la temperatura, donde su rango mas efectivo de desempeño se encuentra entre los $-20C$ a $40C$, estas baterías en promedio pueden proporcionar (de forma individual) un voltaje aproximado de 1.2V a 1.6V dependiendo el tipo de componente activo, además, estas baterías tienen un bajo nivel de descarga, que en el caso de aplicaciones que demanden poca corriente [57] (y voltaje) durante un periodo prolongado de tiempo, pueden ser muy útiles. Sin embargo, este tipo de baterías no ofrecen una solución confiable para problemas de robótica móvil a largo plazo, como se pretende en el diseño de ASTEC.

El otro tipo de batería que se encuentra presente en muchos proyectos de robótica móvil, son las "baterías de litio", estas baterías son bastante ligeras comparadas con las alcalinas, debido al tipo de metal que ocupan para generar esta energía, además, estas baterías poseen una alta capacidad de almacenaje de energía y también altas capacidades de descarga [57]. En general, las baterías de litio se encuentran en dos tipos, el primer tipo es de Ion-Litio (Lithium-Ion), estas baterías poseen un rango de voltaje por celda de 3.05V a 4.5V, con una energía especifica de 80 - 180 Wh/kg, estas baterías son algo costosas pero permiten una buena resistencia del ciclos de vida y en la cantidad de energía (corriente y voltaje) que pueden suministrar. Las de segundo tipo son las baterías Li-po (Lithium-Polymer) estas baterías poseen características muy similares a las de Ion-Litio [57], sin embargo, estas baterías poseen un rango de descarga (cuanta corriente pueden suministrar). Debido a la aplicación que se plantea en el diseño de ASTEC, para el módulo de alimentación se propone el uso de una batería de Ion-Litio como mejor opción.

Usando las hojas de datos técnicos de los sensores, unidad de control y procesamiento y módulo de comunicación propuestos para el diseño, se genero el Tabla 2.2.5.1, donde se establece una aproximación del consumo energético que se plantea para ASTEC, donde se considera un voltaje de operación de 5V y una corriente de operación promedio de 2.5A, en el capitulo siguiente se abordarán de forma más especifica los elementos del diseño.

Componente	Consumo promedio [W]	Consumo máximo [W]
Motor Izq	2.5	4.1
Motor Der	2.5	4.1
Cámara WEB	0.5	1
Lidar	1.2	1.5
Unidad de Control y Procesamiento	3.8	5.5
Comunicación	1.5	1.8
Sensores	1	1.7
Total	13	19.7
Total (corriente) a 5V	2.6 A	3.94 A

Tabla 2.2.5.1. Consumo energético tentativo para prototipo ASTEC.

El tiempo de autonomía propuesto para ASTEC sera de 150 a 250 minutos aproximadamente (dependiendo del consumo por componente), donde el tipo de batería

elegido para el diseño será de Ion-Litio con una capacidad de carga de 10,000mAh y con un voltaje de operación de 5V.

2.2.6. Módulo de Inteligencia Artificial

Lo que corresponde al módulo de inteligencia artificial, responde a las técnicas implementadas para los algoritmos de localización, mapeo y trazado de ruta del robot, así como la ubicación de víctimas en la zona de despliegue.

En el ámbito de la inteligencia artificial, los sistemas que describen un comportamiento o un actuar determinado en un ambiente son llamados "agentes", un agente puede ser cualquier cosa (dispositivo, un robot, un androide, etcétera) que sea capaz de percibir su entorno mediante sensores y realizar tareas o acciones en dicho ambiente mediante actuadores. [34]

Hablando de un "agente", existen dos componentes importantes que lo conforman:

- Función del agente: Esta característica genera el mapa de acción de cualquier precepto (información obtenida del ambiente) hacia una determinada acción. [34]
- Programa del agente: Esto se refiere a la "programación interna" de la "función del agente", que se puede entender como una implementación concreta del agente a través de un sistema físico. [34]

Russell Stuart y *Norvig Peter* en el libro "Artificial Intelligence: A Modern Approach", se refieren a un "agente racional" como aquel que realiza la acción correcta (con base a la secuencia de percepto obtenida y la función del agente).

"*Para cada secuencia de perceptos, el agente racional debe seleccionar la acción que maximice su 'medición de rendimiento' (performance measurement)* [2] *tomando a consideración la evidencia recolectada por la secuencia de perceptos y el conocimiento que el agente tenga.*"

Dado estos principios, un agente que pueda actuar de manera "racional", cumpliendo con las características de la definición anterior, puede también ser considerado como "inteligente".

Ahora, *M. Tim Jones* en su libro titulado "Artificial Intelligence: A Systems Approach" [35], da una definición propia sobre lo que se considera como inteligencia y es:

[2]Un método que permite al agente saber si sus acciones son correctas o no, es a través de una 'medida de rendimiento' (performance measurement), esta medición captura que tan 'atractivo' es el resultado observado en el ambiente debido a las acciones del agente. [34]

"*La inteligencia puede ser definida en la perspectiva de un humano, como una serie de propiedades de la mente, donde, si tratamos de darle una perspectiva mas general, se conjuntan habilidades como: planificación, resolución de problemas, es decir, razonamiento.*"

Una definición más genérica de la inteligencia es: "La habilidad de realizar la decisión correcta, dada una serie de datos entrantes y una variedad de acciones posibles." [35]

Características básicas de un agente racional

Existen muchas características básicas que puede contener un agente racional o un agente inteligente, sin embargo, para propósitos del prototipo ASTEC, existen al menos cuatro que se consideran importantes resaltar.

- Recopilación de información.
- Exploración.
- Aprendizaje.
- Autonomía.

En lo que respecta a la "recopilación de información" se llevara acabo a través de los sensores planteados en la subsección anterior, estos sensores se plantearon con el propósito de facilitar y maximizar la recopilación de información más importante del entorno, tomando en cuenta que ASTEC al ser un robot de diseño reducido, debe considerarse la eficiencia de sus recursos; los datos obtenidos del módulo de sensores serán principalmente enfocados al estudio del lugar (detección de objetos) a través de la cámara térmica y el sensor LIDAR, y donde la cámara térmica se usará como elemento principal para la detección de posibles víctimas, donde esta detección sera considerada como la "medida de rendimiento" del robot.

Después se tiene la característica de "exploración", este concepto es fundamental ya que ASTEC debe inherentemente realizar una exploración del escenario de despliegue en busca de posibles víctimas, esta etapa va de la mano con la etapa de "recopilación de información", ya que si no existe una etapa de exploración, no se puede establecer el éxito de la misión.

La característica de "aprendizaje" (también conocida como autonomía), resulta crucial en los métodos de localización y mapeo de ASTEC, e incluso en los algoritmos para el trazado de ruta, ya que como se explicará en breve, estos algoritmos dependen de la recopilación de información y de la exploración del agente, ya que con base a la secuencia de perceptos recabados por ASTEC, este tendrá la capacidad de corregir la noción de su

localización y la localización de las víctimas y objetos de su ambiente (mapa), y también ayudaran para realizar, generar o corregir rutas planteadas por el robot.

Por ultimo, la característica de autonomía se refiere a la necesidad de los agentes de generar sus propias decisiones tomando en cuenta el conocimiento adquirido, es decir, que debe tener la capacidad de corregir sus conceptos y funcionamientos durante el despliegue, ya que si este necesitara de un operador humano, se vería comprometida una de las ideas básicas de la tesis planteada, o también si ASTEC solo se desempeñara el conocimiento pre-programado (datos ingresados en la memoria del robot antes del despliegue), este carecería de una verdadera autonomía, ya que la información y el conocimiento que el robot adquiera durante el despliegue seria inútil, por ello, ASTEC deberá poseer los sensores, actuadores, algoritmos y energía necesarios para llevar acabo las tareas de búsqueda por si solo.

Como se observa en las características mencionadas, existen muchas similitudes con los elementos fundamentales mencionados en el capítulo anterior sobre los robots de rescate, esto se debe a que estos robots son también catalogados como agentes inteligentes en varios textos revisados en la bibliografía, aunque algunos autores no hablen explícitamente de que su prototipo posea inteligencia artificial [7] [11] se puede decir que estos robots poseen un grado de inteligencia.

Tipos de Ambientes y Tipos de Agentes

Cuando se habla de tipos de ambientes y tipos de agentes, existen algunas características que pueden ser encontradas ampliamente en la bibliografía sobre "inteligencia artificial". [34][35]

En la información encontrada, pueden clasificarse a los ambientes y agentes en al menos diez tipos diferentes, sin embargo, se consideran solo los siguientes siete aspectos como los más importantes para la clasificación de ASTEC y definir cuales serán las problemáticas principales al plantear a este robot como un agente racional para tareas de búsqueda y rescate.

1. Observabilidad
    - Completamente Observable: Cuando el agente es capaz de recabar toda la información "relevante" del escenario de despliegue, con la cual pueda tomar una acción correcta.
    - Parcialmente Observable: Si el agente no es capaz de obtener toda la información relevante del entorno, debido a incertidumbre en los sensores, actuadores, etc. En la mayoría de los casos es necesario mantener un registro del "estado interno".
2. Número de agentes

- Mono-agente: Se basa en un solo agente que se desenvuelve en un entorno, donde busca maximizar su rendimiento, pueden existir otros objetos dentro de su entorno que podrían parecer "agentes", pero en este planteamiento, en lo que al mono-agente se refiere todos los elementos de su entorno son tratados como objetos y no como agentes.
- Multi-agente: Cuando en el ambiente existe más de un agente, dentro de esta clasificación también se habla de los "ambientes competitivos" donde existe una batalla entre los agentes para maximizar su desempeño, con el entendido de que esto también afecta a los demás y de manera inversa, también existen los "ambientes cooperativos" que permiten a los agentes trabajar en conjunto para aumentar las posibilidades de éxito de la misión y con ello su desempeño.

3. Modelo de Enfoque

   - Determinista: Cuando los estados de evolución del entorno solo dependen del estado actual del agente y la acción que este ejecutará para llegar a ellos, es decir, se conocen todos los posibles escenarios resultantes antes de que ellos ocurran.
   - Estocástico: Cuando existe una incertidumbre asociada al estado actual, evolución de estados, obtención de información, acciones, etc. En este caso se tiene que no se puede saber con absoluta certeza el resultado de alguna acción (medición o movimiento) del agente, entonces los estados son descritos como distribuciones de probabilidades, así como las mediciones y acciones posibles del agente.

4. Método de acción

   - Episódico: Un ambiente episódico se describe como el proceso de un agente cuando este recibe algún percepto y posterior a ello realiza una sola acción, donde se realiza especial énfasis al hecho de que el siguiente episodio (futuro) ya no dependerá de las acciones tomadas en episodios previos.
   - Secuencial: Cuando la decisión actual puede o no afectar las decisiones futuras del agente.

5. Tipos de objetos (mapa)

   - Estático: Cuando el ambiente no cambia en el despliegue del agente.
   - Dinámico: Cuando existe una evolución del entorno del agente, puede darse una reubicación de objetos, objetos en movimiento, etc.

6. Descripción temporal

- Discreto: Se establece cuando existe una "secuencia finita" de estados posibles del agente, y en la mayoría de los casos se habla de que los perceptos y acciones se realizan de manera discreta también.
- Continuo: Si los estados del ambiente y/o del agente son descritos de manera continua, con una serie infinita de estados posibles, donde también pueden encontrarse los perceptos y las acciones del agente como funciones continuas.

7. Conocimiento del ambiente

   - Conocido: Se trata principalmente del "estado del agente" donde se conoce a priori los resultados de todas las acciones que le son dadas al agente, se dice que el agente tiene conocimiento completo y preconcebido de la relación entre sus acciones y los resultados producidos en el entorno.
   - Desconocido: Cuando no se conoce completamente los resultados de una acción determinada del agente, en este caso el agente tiene que "aprender" el funcionamiento para con ello realizar una acción correcta.

Ahora, contemplando los aspectos mencionados a continuación, se realizaron dos tablas del robot ASTEC (Tabla 2.2.6.1.a y Tabla 2.2.6.1.b), para describir el tipo de agente y el ambiente en el que el se plantea desempeñar.

Ambiente / Agente	Observabilidad	Numero	Enfoque
Zona de desastre urbano / Robot ASTEC	Parcial	Mono	Estocástico

Tabla 2.2.6.1.a. Tabla de agente y ambiente para ASTEC.

Acción	Mapa	Tiempo	Conocimiento
Secuencial	Dinámico	Discreto	Desconocido

Tabla 2.2.6.1.b. Tabla de agente y ambiente para ASTEC.

Identificación de problemas y la incorporación de inteligencia artificial en ASTEC

En general, cuando se habla de robótica para rescate, se han encontrado cuatro problemáticas principales que debe resolver el prototipo ASTEC.

- Exploración

- Localización de víctimas
- Localización del robot
- Generación del mapa de exploración

"La exploración, es la tarea de guiar un vehículo de tal manera que pueda cubrir el medio ambiente con sus sensores" [36], para ASTEC, es la etapa en la cual el robot debe de ser capaz de generar protocolos de búsqueda en un entorno determinado con la finalidad de ubicar víctimas dentro de la zona de desastre.

La generación de mapas de exploración es una de las principales tareas y problemas de la robótica móvil, la calidad y fiabilidad de los mapas dependen de muchos factores [36]. La estimación de la posición del robot durante la generación del mapa es uno de los más importantes, ya que existe una dependencia directa entre la posición del robot y la posición de los objetos en el mapa (esto se explicará con más detalle en la parte de localización del robot y generación del mapa de exploración), la calidad de las mediciones de los sensores (y su incertidumbre asociada) y el desempeño de los actuadores del robot (donde también existe una incertidumbre en sus movimientos).

En esta etapa, se han encontrado distintos enfoques que buscan resolver este problema. Los enfoques más utilizados en la actualidad son basados en heurísticas y algoritmos bio-inspirados, hasta la fecha no existe algún método o estrategia que garantice realizar la mejor exploración posible y esto se debe a muchos factores: el tipo de escenario de despliegue, las características del robot, la capacidad de percepción del robot, ángulos de visualización, consumo de energía, etc. Existen acercamientos al problema de la exploración en robots móviles, como lo son: los algoritmos para camino más corto (punto de inicio y un punto de meta), algoritmos de trazado de ruta, etcétera, pero no deben ser confundidos con el problema de "exploración".

El problema de exploración se considera en el robot ASTEC debido a la posibilidad de que el robot no conoce la "localización de las víctimas" en la zona de desastre, por ello, el robot debe ser capaz de realizar rutinas de exploración que le permitan recabar información del entorno y generar un mapa de las zonas exploradas, en el momento que éste detecte una "posible víctima" el problema deja de ser uno de "exploración" y pasa a convertirse en uno de trazado de ruta.

En el artículo "Uncertainty Constrained Robotic Exploration: An Integrated Exploration Planner" por *Alexander Ivanov* y *Mark Campbell*, se encuentra planteado de manera explícita la problemática que se enfrentaría el prototipo ASTEC [37], y es que la exploración en zonas de desastre urbano con información global limitada es un problema fundamental para los robots móviles, incluida la robótica para rescate, debido a que esto conlleva una relación directa con el grado de autonomía e independencia del robot.

Además, ya que el robot ASTEC debe operar en zonas de desastre urbano, es importante que el robot posea la capacidad de tener cierto conocimiento respecto a su entorno y lo que lo rodea, para llevar acabo las tareas de exploración, localización, mapeo, y trazado de rutas para rescatar el mayor número de víctimas posibles.

En inteligencia artificial, el trazado de ruta se basa en generar una serie de acciones (movimientos y mediciones) que se usan para transformar el estado del robot en un estado meta deseado. [45]

"El problema de exploración en un área desconocida mientras se trata de completar una misión de alta prioridad, es conocido como: exploración integrada" [37], la idea fundamental de la "exploración integrada" es la capacidad de recopilar información del entorno, y al mismo tiempo ser capaz de mantener una noción de la localización o ser capaz de recalcular la posición del robot en el momento.

En lo que respecta al robot ASTEC, se plantea como solución a esto implementar tres técnicas de manera conjunta, tratando de simplificar el problema, descomponiéndolo en tres enfoques principales.

- Generación de un algoritmo de exploración básico para 2D.
- Implementación de un algoritmo para trazado de ruta.
- Implementación de una estrategia de localización y mapeo.

2.2.6.1. Algoritmo de exploración y trazado de ruta para 2D

En lo que corresponde al "Algoritmo de exploración" que se pretende para este diseño, se busca implementar una estrategia conjunta con el algoritmo de trazado de ruta, esto es, que el algoritmo de exploración se interpreta como un protocolo que permite al robot generar un proceso de exploración de su entorno.

Además, en esta sección se plantea el funcionamiento general del algoritmo para la generación de la ruta "tentativa" del robot, es importante señalar que esta ruta se considera "tentativa" ya que aunque la ruta esta basada en la información obtenida en la sección de exploración, estos datos tienen una incertidumbre asociada [38] y es de hecho la robótica, en general, una forma de aplicación de la tecnología que presenta muchos problemas con el ruido de sensores, actuadores, incertidumbre en posición (del mismo sistema y de los objetos dentro del sistema), problemas en la conversión de fenómenos analógicos a mediciones digitales procesables por una computadora, etcétera. Los sensores poseen una capacidad limitada para percibir el entorno físico en el que son inmersos [38] [6], estos problemas van desde la capacidad de resolución, rango, método de conversión, y ruido en general, es por ello se considera que la información que se

obtiene por medio de la sección de exploración “no es exacta”, por ello la ruta que se generaría en esta sección “no sería exacta”. Sin embargo, con base a lo investigado existen métodos que permiten reducir estos errores e incertidumbres de las mediciones obtenidas, pero estas correcciones se realizarán en la sección de “implementación de alguna estrategia de localización y mapeo”.

Además, es necesario agregar que también los métodos de movimiento (actuadores) con énfasis en los sistemas usados en la robótica móvil, presentan errores por ruido en control, fallas mecánicas, variación en las dimensiones de los elementos mecánicos, etc. Y al igual que en la información de los sensores, pueden reducirse estos errores a través de filtros con enfoque probabilístico, donde podría parecer contradictorio plantear un método de exploración y un método de trazado de ruta determinista, esto se hace con la finalidad de reducir el tiempo de diseño del robot, y tratar de simplificar las problemáticas inherentes de los robots para rescate, es una solución simple y puede ser mejorada posteriormente con técnicas que consideran errores aleatorios de los sistemas robóticos. [38] [6] [9]

Ahora bien, considerando los factores mencionados, existen algunos aspectos que deben considerarse para el planteamiento del método de solución para esta sección, estos son:

- Contar con información de posición de inicio y posición de meta (principal o secundaria) para simplificar los algoritmos y el costo computacional de los mismos.
- Considerar los objetos detectados en la parte de exploración.
- Generar, a través del plano del algoritmo para el trazado de ruta, información que sirva para controlar el movimiento de los actuadores del robot.

En la actualidad existen un gran número de algoritmos que pueden ser utilizados para realizar rutas y aplicarlas en la robótica móvil, estos algoritmos pueden garantizar o no soluciones óptimas locales o globales, donde también si se opta por soluciones óptimas globales usualmente se habla de algoritmos que consumen una mayor cantidad de recursos computacionales en comparación con los algoritmos de ruta óptimos de manera local, sin embargo, estos algoritmos de aplicación local pueden estancarse en ciertas circunstancias debido a su naturaleza simple, y también debe mencionarse, que existen algoritmos híbridos que utilizan métodos de soluciones globales para un inicio en el problema y posteriormente atacarlos con métodos de optimización local para mejorar el tiempo de ejecución del programa.

Algunos de los algoritmos encontrados más frecuentemente en la literatura son:

- Greedy Algorithms (Algoritmos del codicioso), donde se considera la aplicación del Bug Algorithm (Algoritmo de gusano).

- Dijkstra's Algorithm (Algoritmo de Dijkstra)
- A * Algorithm (Algoritmo de A *)
- D * Algorithm (Algoritmo de D *)
- Potencial Fields Algorithms (Algoritmos de Campo de Pontencias)
- Algunos más actuales: RRT, LQRPlanner, etc.

Algoritmos del codicioso (Greedy Algorithms)

Una de las formas mas básicas de implementar un método de optimización o búsqueda básica es a través de los "algoritmos del codicioso" (Greedy algorithms), estos algoritmos son relativamente sencillos de implementar, pero también pueden ser difíciles de diseñar y analizar, debido a que estos algoritmos implementan diversas técnicas. [39] La tarea más difícil para implementar alguno de estos algoritmos es analizar el problema y determinar cual de ellos es el más adecuado, estos algoritmos buscan principalmente maximizar o minimizar alguna y/o algunas características cuantitativas sujetas a ciertos criterios.

Características:

- Maximizar o minimizar el número de eventos generados, y con la restricción de no incurrir en eventos ya generados (repetición).
- Maximizar o minimizar el número de "saltos"para realizar un recorrido en un escenario, contando con un inicio y una meta.
- Maximizar o minimizar el costo del camino seleccionado en un grafo, sin generar discontinuidades a lo largo del mismo.

En el caso de exploración para robots móviles en dos dimensiones, existe una estrategia básica para un algoritmo del codicioso, que de hecho sirve como base para otros algoritmos, y esta estrategia es: la distancia euclidiana entre dos puntos en un plano.

Algoritmo del Codicioso usando distancia euclidiana

En este caso se contempla la "distancia euclidiana" como el método para calcular la distancia no negativa entre dos puntos: A, B.

- Sea: $A = (a_1, a_2, ..., a_n)$
- Sea: $B = (b_1, b_2, ..., b_n)$

Donde $a_1, a_2, ..., a_n$ y $b_1, b_2, ..., b_n$ son magnitudes correspondientes a cada una de las dimensiones del espacio.

Para un espacio de "n" dimensiones se tiene:

(2.2.1)

$$d(A,B) = \sqrt{\sum_{i=1}^{N} (b_i - a_i)^2} =$$

(2.2.2)

$$\sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2 + ... + (b_N - a_N)^2}$$

Ahora bien, para el caso de dos dimensiones, es decir, en el plano, se tiene:

(2.2.3)

$$d(A,B) = \sqrt{\sum_{i=1}^{2} (b_i - a_i)^2} =$$

(2.2.4)

$$\sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2}$$

Algunas propiedades del cálculo de la distancia euclidiana son las siguientes:

- La distancia entre ambos puntos: $d(A,B) \geq 0$
- Propiedad simétrica: $d(A,B) = d(B,A)$
- Distancia entre A y A: $d(A,A) = 0$
- Propiedad de desigualdad triangular: $d(A,B) \leq d(A,C) + d(C,B)$
- Si $d(A,B) = 0$ entonces $A = B$

Algoritmo de Dijkstra (Dijkstra's Algorithm)

El algoritmo Dijkstra es un algoritmo de búsqueda por grafos, donde se basa en la aplicación de nodos vecinos y en encontrar la solución óptima en un camino (no negativo) entre nodos. Este algoritmo usualmente es usado para generar la ruta más corta a un destino. Actualmente, el algoritmo Dijkstra puede encontrarse aplicado como kernel de algoritmos de trazado de ruta o como subrutinas de programas de búsqueda en problemas de grafos, este algoritmo, aunque garantiza la solución óptima, puede ser muy costoso en procesamiento computacional, donde la forma de exploración así como su complejidad están dados por el número de grafos y las conexiones que existen en los mismos. [44]

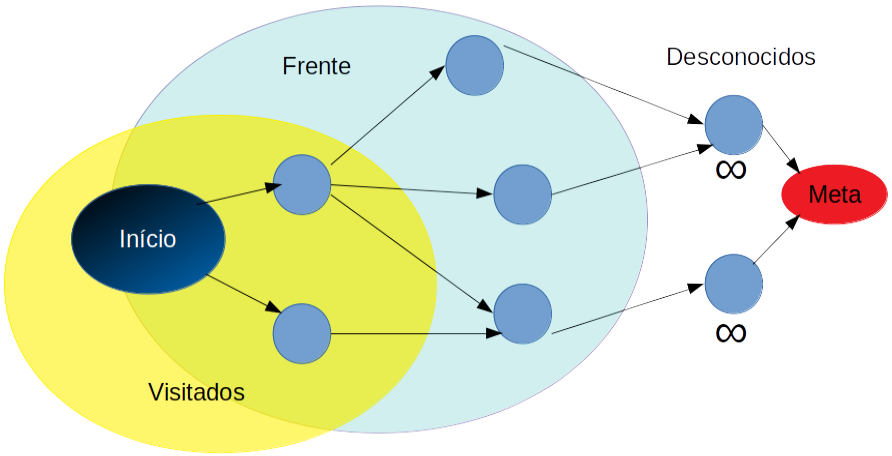


Ilustración 2.2.6.1.1. Funcionamiento de algoritmo Dijkstra.

En la ilustración (2.2.6.1.1), se observa el funcionamiento básico del algoritmo Dijkstra. Las características principales de este algoritmo son las siguientes:

- Un nodo de inicio, y un nodo de meta.
- Un conjunto de nodos "visitados" donde se conoce el costo final correcto.
- Un conjunto de nodos llamado "frente" donde se esta explorando para encontrar un nodo mínimo.
- Los nodos que aun no se encuentren en "frente", se consideran desconocidos y se les asigna un valor de ∞ (en la implementación práctica asignar un valor de ∞ no es posible, y se opta por asignar un valor muy alto de costo para representar su valor desconocido).

Sea $G = (E, V)$ un grafo con peso y dirección asociados, donde: E corresponde a las conexiones (bordes) y V al numero de vértices (nodos) del grafo. Todos los nodos fuente s cumplen $s \in V$ y todos los nodos (vértices) v cumplen $v \in V$. [44]

Entonces se tiene que:

(2.2.5)

$$d(v) = min\{d(u) + c(u, v)\}$$

Donde:

- $d\,(v)$ corresponde al "nodo activo" (el nodo que se esta explorando y modificando si es el caso).
- $d\,(u)$ es la distancia mínima del nodo más cercano proveniente del nodo anterior.
- $c\,(u, v)$ es la distancia de u a v.

Ahora bien, cuando se habla del uso del algoritmo de Dijkstra para el trazado de ruta en robots móviles, usualmente se genera una malla de nodos (como en el caso de los algoritmos del codicioso), estos nodos tienen un costo asociado a sus movimientos y la forma de la exploración esta condicionada a los movimientos posibles entre los nodos de las mallas.

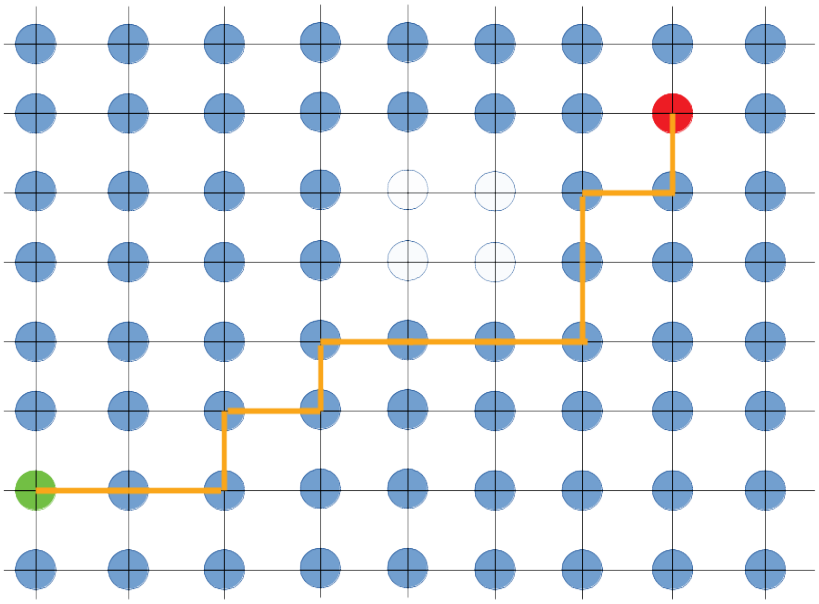


Ilustración 2.2.6.1.2. Malla de grafos.

En la ilustración (2.2.6.1.2) se puede observar la idea de la generación de una malla de grafos, esta malla representa los estados posibles del robot en un mapa determinado (inherentemente determinista), donde las características más importantes están dadas por el costo de traslado nodo a nodo (estado a estado), y el número de conexiones o puentes que existen entre los mismos.

En el algoritmo Dijkstra clásico cada nodo tiene cuatro movimientos posibles (arriba, abajo, izquierda, derecha) pero pueden agregarse más conexiones para mejorar la trayectoria de la ruta del inicio a la meta (esto también incrementa los cálculos que realiza el algoritmo para generar la ruta óptima, ya que tiene un mayor numero de nodos que pueden o no ser los nodos mínimos para el camino deseado), y por último, otro factor importante a considerar es; en el caso de un robot de móvil en dos dimensiones, es necesario establecer las ubicaciones de objetos (obstáculos, paredes, etc.) donde la

técnica más usada al menos para los algoritmos de Dijkstra y A*, es agregar al vector de nodos visitados estos nodos que representan la posición y dimensión de los objetos del mapa, con ello se pretende garantizar que el robot no cruce por esas zonas (se realiza una restricción hacia dichos nodos), que en la aplicación en campo seria imposible.

Algoritmo A* (A "star" Algorithm)

El algoritmo de A* ("estrella") es uno de los algoritmos para trazado de ruta más conocidos, este algoritmo puede ser usado en diferentes tipos de espacios de configuración, usualmente métricos o topológicos. [45]

El algoritmo se basa en la implementación de dos técnicas ya mencionadas en este trabajo, una heurística; como es el caso del algoritmo del codicioso en función de la distancia euclidiana entre inicio y meta, y un algoritmo de búsqueda de camino más corto, como el algoritmo de Dijkstra, A* busca generar una solución óptima a un problema, en el caso de ASTEC, para encontrar la ruta más corta. El algoritmo A* utiliza el algoritmo de Dijkstra como la base para la exploración de los nodos de la malla del mapa e implementando la heurística de la distancia euclidiana más corta, con el fin de "dirigir" la búsqueda en función a la actualización de los costos de los nodos que se visitaran y su relación con la distancia a la meta.

Para todos los elementos de $i\,(1, 2, 3, ..., n)$ que se encuentren en el vector de nodos f ("frente") para un momento t (dicho vector es generado por el algoritmo Dijkstra), se tiene que:

(2.2.6)

$$a_i^t\,(f) = g_i\,(f) + d_i\,(f)$$

Donde:

- $g_i\,(f)$ Corresponde a la distancia calculada del nodo $i\,(1, 2, 3, ..., n)$ a m (meta), usando la heurística del algoritmo del codicioso (euclidiana, Chebyshev, Manhattan, etc.).
- $d_i\,(f)$ Corresponde a la distancia del camino (o cualquier parámetro establecido para el algoritmo) desde el estado inicial hasta el estado actual, a través de una serie de nodos seleccionados.
- $a_i\,(f)$ Es el total de la suma de las dos técnicas implementadas.

Una vez evaluado cada uno de los nodos presentes en "frente", se establece que el siguiente nodo a explorar, para un momento $t + 1$, será:

(2.2.7)

$$a^{t+1}(f) = min\{\, a_1(f), a_2(f), a_3(f), ..., a_n(f) \,\}$$

Contemplando lo mencionado, ahora se puede observar la ilustración (2.2.6.1.3).

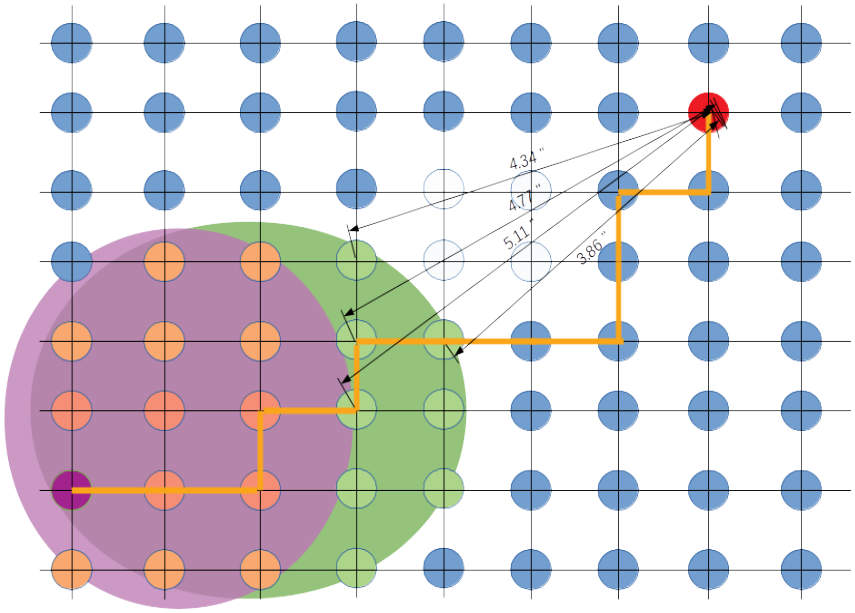


Ilustración 2.2.6.1.3. Funcionamiento Algoritmo A "estrella".

La ilustración (2.2.6.1.3) plasma la idea general de un Algoritmo de A* "estrella", donde se muestran las características del algoritmo Dijkstra, donde existe en color violeta, el vector de nodos visitados; después se tiene en color verde el conjunto de nodos que pertenecen al "frente" que son los que están siendo explorados en un instante t, donde la diferencia notable con respecto a Dijkstra es el hecho del cálculo de la distancia euclidiana de cada uno de los nodos de frente hacia la meta, además, de que se suele incorporar este valor a la suma del costo de cada nodo visitado, es decir:

Actualización de costo para nodo visitado:

(2.2.8)

$$costo_i = costo_{t-1} + \Delta m_j$$

Donde:

- $costo_i$ son los valores los nodos visitados $(0, 1, 2, ..., i)$, contemplando el camino recorrido y el costo del movimiento, usando el principio de Dijkstra.
- $costo_n^{t-1}$ corresponde al valor correspondiente hasta el instante $t-1$ (valor del camino mínimo generado hasta ese momento).

- Δm_j es el valor establecido por el "costo" del movimiento, dentro de los movimientos disponibles $(0, 1, 2, ..., n)$

Finalmente, a $costo_i$, se aplica el principio del Algoritmo del Codicioso (con distancia Euclidiana).

(2.2.9)

$$costofinal_i = costo_i + \sqrt{(y_m - y_i)^2 + (x_m - x_i)^2}$$

Donde:

- $costofinal$ es el valor final actualizado de cada nodo visitado.
- $costo_i$ valor del costo calculado con el principio de Dijkstra.
- (x_m, y_m) posición de la meta.
- (x_i, y_i) posición del nodo del cual se calcula la distancia a la meta.

Algoritmo de Campos de Potencial (Potencial Fields Algorithm)

El "Algoritmo de Campos de Potencial", implementa un campo de potencial físico que obedece la ecuación de Laplace, existen muchos ejemplos de estos "campos de potencial", que van desde las curvas de nivel en mapas topográficos, campos electromagnéticos, campos gravitacionales, etc. Cuando se habla de la implementación de este enfoque en algoritmos de trazado de ruta para robots móviles, se genera un campo de potencial "artificial" con la finalidad de "regular" al robot y los objetos que pueden encontrarse en algún mapa, la palabra "regular" se entiende como una manera de generar una ruta efectiva y sin colisiones usando este algoritmo. [47]

Los campos artificiales de potencial son generados a partir de una función de potencial de los obstáculos, estructuras en el mapa, el agente y la meta. [47] [48]

Al igual que en los otros algoritmos ya explicados, se asume que el mapa es una malla de nodos, donde tanto el robot como los nodos que se establecen como obstáculos se les asigna un campo de potencial artificial de ellos (robot y objetos) hacia los nodos vecinos en función de una ecuación de campo de potencial. La mayoría de estos algoritmos se caracterizan por usar al robot y los objetos del mapa, como los focos de generación de estos campos de potencial, donde las zonas con un potencial más alto se encuentran de manera proporcional a la cercanía de estos focos (se debe aclarar, que también se puede realizar el proceso inverso), donde la función genera una especie de "repulsión" hacia estos focos y genera además (en su versión original) una fuerza de atracción hacia la meta en el

mapa, ya que como en los otros algoritmos, la meta principal es generar el camino mínimo de un inicio a una meta, evitando las "zonas mas costosas" del mapa. [47]

El algoritmo clásico de campos de potencial, genera un criterio similar al algoritmo del codicioso (basado en distancia euclidiana, Manhattan, Chevysheb, etc.), donde la fuerza de "atracción", que se describirá a continuación, depende de manera proporcional a la distancia entre el punto de posición actual a la meta.

La descripción matemática del funcionamiento de este algoritmo es la siguiente:

Primero se calculan las fuerzas de atracción (de nodo a meta), (x_n, y_n) posición del nodo actual, (x_m, y_m) posición de la meta y repulsión (de obstáculo a nodo), con (x_{obs}, y_{obs}) posición del obstáculo.

Fuerza de atracción

(2.2.10)
$$F_a = K_a \sqrt{(y_m - y_n)^2 + (x_m - x_n)^2}$$

Donde:

- F_a corresponde al valor asignado de atracción, para un nodo determinado.
- K_a es una constante arbitraria de atracción.

De igual forma se calcula,

Fuerza de repulsión

(2.2.11)
$$F_r = K_r \sqrt{(y_n - y_{obs})^2 + (x_n - x_{obs})^2}$$

Donde:

- F_r la fuerza de repulsión del nodo actual a un obstáculo en el mapa (pueden existir múltiples obstáculos).
- K_r constante de repulsión arbitraria.

Contemplando estas dos fuerzas, entonces se calcula la "fuerza total" del nodo, con la siguiente ecuación:

(2.2.12)
$$F_n(x, y) = F_a(x_n, y_n) - \sum_{i=1}^{obs} F_r(x_n, y_n)$$

La ecuación calcula la fuerza de atracción, a esa fuerza se le resta el sumatorio de todas las fuerzas de repulsión de los objetos con respecto del nodo, el numero de obstáculos esta dado por $(0, 1, 2, .., obs)$.

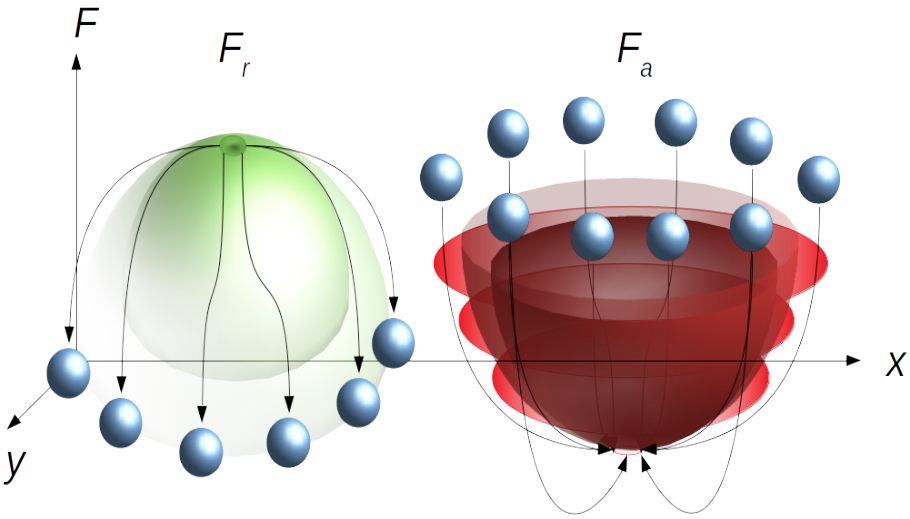


Ilustración 2.2.6.1.4. Algoritmo de Campos de Potencial.

En la Ilustración (2.2.6.1.4) se observa la idea del funcionamiento básico de Algoritmo de Campos de Potencial, donde existen dos tipos de fuerzas características que existen dado este enfoque, en verde se tiene la fuerza de repulsión (forma en color verde), dicha fuerza es característica de los obstáculos presentes en el mapa, y generan un desplazamiento hacia valores "negativos" como se observa en la ecuación anterior; después se tiene la fuerza de atracción que es generada por el punto de meta (forma en color rojo), esta fuerza se considera de tipo "positiva" en la ecuación anterior, con la finalidad de generar una atracción de la ruta hacia la meta.

Cada nodo presente en el mapa (excluyendo al nodo meta, y los nodos obstáculo) se les asigna un valor correspondiente a la ecuación anterior, donde se busca repeler los objetos que son los obstáculos y a traer la meta, donde pueden existir nodos con un costo negativo, donde se supondría serian los más costos, y los nodos con valor positivo serian los menos costos (dependerá netamente del planteamiento de la formula anterior), y también puede darse el caso de nodos nulos (nodos donde su costo sea "cero" cuando se este en equilibrio de fuerzas).

2.2.6.2. Algoritmo para localización y mapeo

El primer paso para plantear el algoritmo para localización y mapeo del robot, es necesario plantear un modelo que permita generar el movimiento del robot "modelo de

movimiento", y un método para obtener la información de los sensores planteados en secciones anteriores "modelo de observación".

Modelo de movimiento del robot

Los modelos de movimiento de los robots móviles son muy variados, pero las aproximaciones a estos problemas se pueden enfocar principalmente en dos tipos:

- Modelos deterministas
- Modelos probabilistas

Los modelos deterministas, tienen la diferencia en comparación con los probabilistas, que estos no contemplan en sus ecuaciones los errores aleatorios de un sistema [38] [49], donde los errores modelados en los enfoques deterministas son de tipo sistemático (también pueden usarse distintos parámetros de calibración), sin embargo, cualquier robot presenta incertidumbres asociadas a distintos factores, como pueden ser: actuadores, ruido en sensores, inexactitud de piezas mecánicas, etc. [50] Esto genera que los enfoques deterministas necesiten modelarse con mayor precisión y contemplando cualquier error presente en el robot o que pueda presentarse en su entorno (lo cual, en la opinión del autor de la tesis, no es factible), en contraparte, los modelos probabilistas, sobre todo en robótica móvil, presentan varias alternativas para lidiar con estas incertidumbres de cualquier sistema (de robots), dado que los modelos que se implementan consideran los errores no modelados directamente en las ecuaciones de movimiento o de percepción de la información (sensores), como incertidumbres, y estas incertidumbres pueden ser anexadas en los modelos del robot. Un aspecto importante a señalar, es que también existen modelos de tipo "híbrido", aunque en la bibliografía no se exprese de manera explicita, son modelos que en las ecuaciones del movimiento o de los sensores del robot, se tratan de manera determinista, pero tienen la diferencia de que estos modelos son tratados con filtros probabilísticos, que permiten contemplar las incertidumbres no descritas directamente en el modelo matemático, pero que si afectan al robot, aunque los modelos probabilísticos se integran de una mejor manera con estas técnicas de filtrado, resultan más complejos y computacionalmente más demandantes que los híbridos o deterministas.

Después, se tiene que los modelos pueden encontrarse principalmente en dos subtipos, los modelos cinemáticos y los modelos dinámicos. En el modelo cinemático del robot, solo se contemplan los efectos de los comandos de control sobre la configuración o estado del robot [38]. En los modelos dinámicos, se considera la "trayectoria" del robot, donde se tiene el camino (generado por algún algoritmo) y alguna especificación de tiempo hasta que la siguiente configuración en la trayectoria es alcanzada; estos modelos deben encontrar

trayectorias para el sistema que cumplan con la dinámica de masas e inercias del sistema, los limites de los actuadores y las fuerzas como gravedad y fricción. [49]

Como puede inferirse, la aproximación por modelos dinámicos de movimiento, se caracteriza por ser más compleja, y para propósitos de la tesis se abordará un modelo cinemático de movimiento para el robot.

En las referencias consultadas [38] [6] [9] [12], se han encontrado dos formas para platear el modelo de movimiento cinemático de un robot móvil, estas son:

- Modelo de movimiento por velocidad.
- Modelo de movimiento por odometría.

El modelo de movimiento por velocidad u_t, se basa en la capacidad de control del robot a través de dos velocidades: velocidad rotacional denotada por w_t y velocidad de traslación denotada por v_t.

Entonces, el control del robot en un instante t se expresa como:

(2.2.13)
$$u_t = (v_t, w_t)^T$$

Para el modelo cinemático de movimiento basado en odometría, usa información obtenida de los codificadores de los motores, en lugar de datos de "control" [38]. Sin embargo, esto puede suplantarse por comandos PWM para los motores, ya que la mayoría de estos modelos se enfocan en las variaciones entre las revoluciones de los motores de un instante a otro.

Debido a la información adquirida en las lecturas de "SLAM Lectures" por *Claus Brenner* [50], el autor de la tesis continuará con el modelo de movimiento por odometría, por ello debe establecerse como se expresará la posición del robot para un tiempo t.

El vector de que corresponde a la posición del robot puede ser expresado de manera general de la siguiente forma:

(2.2.14)
$$x_t = \begin{bmatrix} x \\ y \\ z \\ \theta_x \\ \theta_y \\ \theta_z \end{bmatrix}$$

Donde:

- x_t corresponde a la posición del robot en un tiempo t.
- x, y, z son las coordenadas en $3D$.
- $\theta_x, \theta_y, \theta_z$ son los ángulos de las coordenadas para x, y, z.

Para propósitos de la tesis, el sistema de coordenadas de la localización del robot y sus referencias del mapa se considerarán solo en $2D$. La ecuación anterior, para el caso de $2D$ puede ser expresada de la siguiente forma:

(2.2.15)
$$x_t = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

Donde:

- x_t es la posición del robot en el plano.
- x, y son las coordenadas del robot en el plano.
- θ es el ángulo de viraje (este ángulo, corresponde al ángulo del frente del robot, con respecto a alguno de sus ejes x o y).

Ahora, contemplando el vector de posición del robot x_t expresado en la ecuación (2.2.15), se puede retomar el modelo de movimiento por odometría. La idea de este modelo es establecer la posición actual x_t del robot a partir de la posición anterior del robot x_{t-1}, este modelo usa la "información relativa de movimiento", en el intervalo de $t-1$ a t.

El vector de control para este caso, se puede expresar de la siguiente forma:

(2.2.16)
$$u_t = \begin{bmatrix} \bar{x}_{t-1} \\ \bar{x}_t \end{bmatrix}$$

Donde:

- u_t es el comando de control de transición entre el instante $t-1$ a t.
- $\bar{x}_{t-1}$ corresponde a la posición relativa anterior $(\bar{x}, \bar{y}, \bar{\theta})^T$.
- $\bar{x}_t$ corresponde a la posición relativa actual $\left(\bar{x}', \bar{y}', \bar{\theta}'\right)^T$.

Modelo de observación del robot

Un "modelo de observación" o "modelo de medición" tiene como propósito interpretar la información obtenida de los sensores (principalmente exteroceptivos) y generar un modelo que permita expresar los diferentes elementos presentes en un escenario para un robot móvil.

Este modelo (desde un enfoque de probabilidad) se establece como:

(2.2.17)

$$p(z_t|x_t, m)$$

Donde:

- $p(z_t)$ es la probabilidad de realizar la medición dado un estado x_t y un mapa.
- x_t representa el estado (o posición) del robot.
- m se denota como el mapa.

El enfoque que plantea la ecuación (2.2.17) permite considerar, en cierta medida, las distintas incertidumbres (errores aleatorios) que pueden presentarse en el proceso de adquisición e interpretación de los datos de los sensores. [38]

Contemplando la información de la subsección (2.2.4) referente al Módulo de Sensores, existen principalmente dos sensores que aportan información sobre el entorno del robot (donde, la cámara térmica no se contempla debido a que se plantea como un indicador de posible víctima, y no como un método de percepción de obstáculos del mapa), estos sensores son la cámara WEB y el sensor láser omnidireccional LIDAR.

Para el caso del sensor LIDAR (y otros sensores de su tipo) se dice que este tipo de sensores generan "barridos" de un área determinada, generando mediciones de las distintas distancias relativas a la ubicación del sensor y a su ángulo en algún momento del "barrido". Esto conlleva a interpretar las mediciones de la siguiente manera:

(2.2.18)

$$z_t = \left\{z_t^1, z_t^2, z_t^3, ..., z_t^K\right\}$$

Donde:

- z_t es el vector de "barrido" del sensor en un instante t.
- K representa al número de elemento interno del vector del barrido de medición.

Las cámaras usadas para detección de objetos de un mapa, son ampliamente usadas en la actualidad [8] [27]. Los métodos que usan estas cámaras para poder transformar una representación física del entorno en información útil para el robot, se basan principalmente en técnicas de proyección geométrica [38], sin embargo, en el caso de los proyectos y propuestas investigados, las cámaras implementadas son estereoscópicas o multifocales, ya que con una cámara monofocal, como la propuesta para este trabajo, no se puede establecer una relación de la distancia de los objetos y la relación de dimensiones de los mismos, entonces, la opción que se plantea más adecuada para el diseño de ASTEC es usar el sensor LIDAR. Estos sensores láser son modelados de forma "similar" a los sensores ultrasónicos, en el caso del sensor láser, este emite un rayo de luz y registra el eco del rayo (de manera similar al sensor ultrasónico). Los sensores láser proporcionan una mayor cantidad de haces enfocados que los sensores ultrasónicos, en general, los modelos de observación basados en sensores de distancia láser, usan la medición del "tiempo de vuelo", que consiste básicamente en la diferencia de tiempo de la emisión del rayo y el tiempo que tarda en captar el eco de la señal.

Entonces, en lo que respecta al algoritmo para realizar la localización y el mapeo debe pensarse en dos términos:

- Localización del Robot
- Generación del mapa

Localización

El problema de la localización de un robot [38] se define usualmente como el problema de generar la posición (pose) del robot dado un mapa determinado, a este problema suele llamarse "estimación de posición".

Cuando se habla de robótica, el problema de la localización se considera como uno de los problemas fundamentales de la robótica móvil, en general, cualquier sistema robótico debe poseer algún conocimiento de su localización, para llevar acabo tareas predeterminadas en un cierto ambiente. De manera que, el problema de localización se describe como un problema de transformación de coordenadas, donde el mapa se describe como un sistema global de objetos y sus posiciones en el mismo que son independientes de la posición del robot, con base a esta información (ubicación de los objetos del mapa) el robot debe tener la capacidad de generar su posición relativa, en el sistema de coordenadas locales del robot. Además, el sistema de coordenadas del robot posee la ubicación de los objetos relativas en este sistema.

Cuando la localización del robot no puede ser obtenida de manera directa (usualmente se habla de ambientes de GPS denegado) [25], la posición del robot se obtiene a través de

sensores que permiten obtener la ubicación de las "referencias del mapa", que permiten posteriormente generar la posición relativa del robot con base a estas referencias, entonces entran en juego las aproximaciones o estimaciones de localización de manera indirecta.

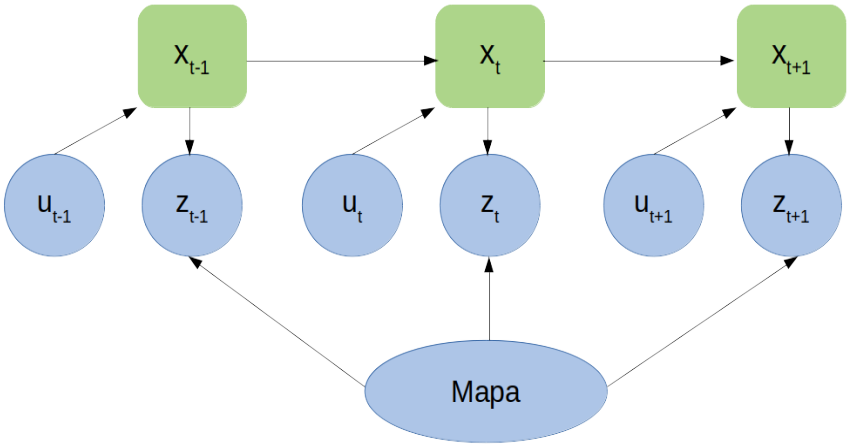


Ilustración 2.2.6.2.1. Modelo Gráfico de Localización (Localización de Markov).

Debe tomarse en cuenta que el problema (solamente) de localización, establece la noción del mapa "a priori", como su principal característica.

En la ilustración 2.2.6.2.1, se expresa en forma de "red de Bayes dinámica" el problema de la localización, donde el objetivo principal es establecer la posición X_t del robot, usando como datos de entrada el mapa m, los comandos de control $u_{1:t}$ y las observaciones o mediciones $z_{1:t}$. La aproximación probabilística para la localización (en robótica móvil) es una aplicación del filtro de Bayes, en especifico, el uso de este filtro en localización se conoce como "Localización de Markov". Contemplando estos aspectos, el problema de localización se define en su enfoque probabilístico de la siguiente manera:

(2.2.19)

$$p(x_t|u_{1:t}, z_{1:t}, m)$$

Como se ha considerado la información respecto a los enfoques deterministas y probabilistas, lo que corresponde a la parte de localización y mapeo del diseño para ASTEC será contemplado como un enfoque probabilístico, donde la idea de hacer esta aproximación mediante probabilidad (e inherentemente procesos estocásticos) es el manejo de las incertidumbres generadas en una aplicación en campo de este tipo de sistemas. Tanto el algoritmo de exploración y el correspondiente algoritmo para trazado de ruta, contienen enfoques en su mayoría deterministas, donde las condiciones de operación, obstáculos, mediciones y acciones mecánicas son planteadas de manera ideal

(con el fin de simplificar un poco la aplicación de estas tareas), es por ello, que para tener un diseño mas realista para el caso de ASTEC, es necesario establecer métodos de corrección y filtrado de información (para localización y mapeo) donde se contemplen algunos errores aleatorios e incertidumbres asociadas a este tipo de problemas.

La taxonomía planteada para el problema de localización de un robot móvil, por *Sebastian Thrun*, *Wolfram Burgard* y *Dieter Fox* en el libro "Probabilistic Robotics" [38], es la siguiente:

- Tipo de Localización
  - Seguimiento de posición: Donde se conoce el "estado inicial" o la "posición inicial" del robot, los errores de posición se consideran pequeños, usualmente la incertidumbre se asocia con distribuciones unimodales, es por ello que se cataloga como un problema "local".
  - Localización global: El principal factor es el desconocimiento de la "posición inicial" del robot, carente de conocimiento de su entorno inicial, y no puede aproximarse a distribuciones unimodales, estos problemas son de tipo global y necesitan distribuciones multimodales.
    - Problema del robot secuestrado: En general, se considera una variante del problema de localización global, la idea de este problema es que el robot puede encontrarse en escenarios donde pueda perder de manera total, la noción de su localización, donde, si este problema es resuelto, el robot es capaz de sobreponerse a perdidas totales de conocimiento relativo de su localización, y reubicarse de manera efectiva en el mapa.
- Tipo de Escenario
  - Ambientes estáticos: Donde la única variable que se tiene es la "posición del robot", todos los demás elementos del mapa y del ambiente se consideran estáticos.
  - Ambientes dinámicos: Donde los elementos del ambiente (objetos del mapa) y la posición del robot cambian en función del tiempo, donde, se hace énfasis en los cambios que se realizan de manera persistente (más de una vez) en un tiempo determinado.
- Tipos de Enfoque
  - Localización pasiva: No establece una correlación entre los comandos de control $u_{1:t}$ y la localización del robot.

- Localización activa: Establece una relación entre la localización del robot y los comandos de control u_{1_t} del mismo.

- Número de Agentes

  - Localización mono-robot (mono-agente): La localización solo depende de un robot, donde toda la información del escenario es adquirida por un solo agente.
  - Localización multi-robot (multi-agente): La localización es realizada por múltiples agentes, cada robot inicialmente genera su propia localización, pero pueden aportarse perspectivas de las localizaciones de cada agente relativas a cada uno de ellos.

Mapeo

El problema del "mapeo", para robótica móvil, contempla el hecho de que no en todos los escenarios donde un robot explorador, robot de intervención, robot de servicio, o un robot de búsqueda y rescate (como en el caso de ASTEC) es colocado, se cuenta con información exacta o relativamente exacta del mapa, es decir, que el robot debe de generar aprendizaje y conocimiento de su entorno (inicialmente desconocido). "*De hecho, el mapeo es una de las competencias centrales de los robots que son verdaderamente autónomos*". [38]

La idea del mapeo presenta muchos retos, dado que la idea de la aproximación de los mapas por el uso de "Filtros de Bayes" resulta poco eficiente, debido a que el espacio de mapas posibles es muy grande, donde aun teniendo una aproximación discretizada como la que se plantea para el diseño de ASTEC las variables de un mapa se pueden encontrar en el orden de 10^5 [38]. Otro aspecto también importante, es el hecho de la generación del mapa "relativo", donde el hecho de la generación de un posible mapa es dado a la localización relativa del robot en dicho mapa, entonces, es cuando se habla que el problema de "Localización y Mapeo Simultaneo" que se interpreta como un problema del "huevo y la gallina".

La taxonomía usada por *Sebastian Thrun*, *Wolfram Burgard* y *Dieter Fox* en el libro "Probabilistic Robotics" [38], es la siguiente:

- Tamaño: El tamaño del mapa se relaciona de manera directa con la dificultad de adquisición y representación del mismo (desde la perspectiva del robot).

- Ruido

  - Ruido en percepción: La relación que existe entre el ruido que reciben o interpretan los sensores, y la dificultad de hacer una aproximación más exacta y real del mapa.

- Ruido en actuadores: Se refiere a los errores inherentes a los movimientos mecánicos y piezas mecánicas de los actuadores del robot, que se traducen en un aumento de la incertidumbre tanto en localización como en el mapeo del robot (y con ello, aumenta la dificultad de las tareas de localización y mapeo del robot).

- Ambigüedad perceptual: Relaciona la correspondencia o falta de la misma con lugares "parecidos" en distintos instantes de tiempo.

- Ciclos: Se relaciona con la repetición de lugares o trayectorias que generen una acumulación de los errores de odometría del robot.

Hablando netamente del problema de Mapeo, se tiene que establecer la premisa de que la posición "exacta" del robot es conocida en todo momento, donde la generación del mapa aun siendo relativa a la posición del robot, no genera el problema de asociar una incertidumbre a la localización, ya que esta se conoce *a priori*.

En la ilustración 2.2.6.2.2, se plantea el problema de la generación del mapa, donde los "estados" o posiciones del robot $x_{1:t}$ son conocidos, junto con los comandos de control $u_{1:t}$ y la información de los sensores $z_{1:t}$, y se busca generar el "mapa" relativo al robot m.

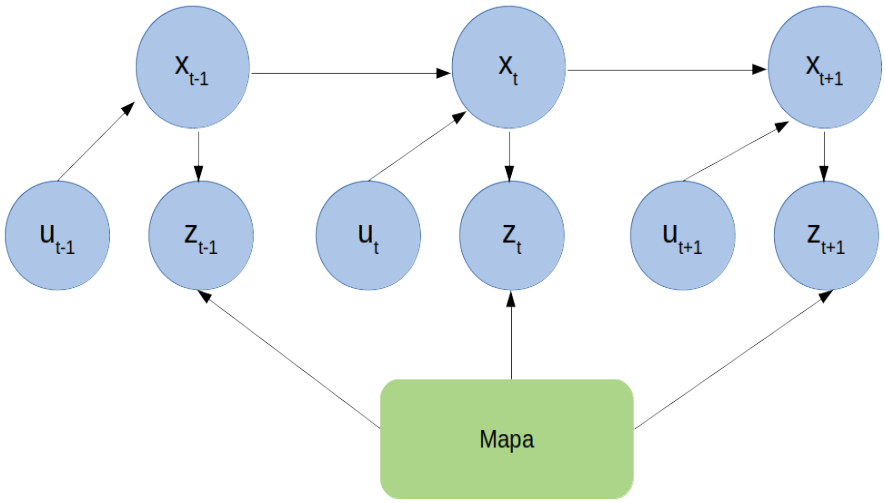


Ilustración 2.2.6.2.2. Modelo Gráfico de Mapeo.

Dado que se conocen "todos" los estados o posiciones del robot, los comandos de control usualmente son descartados, entonces el problema se traduce como:

(2.2.20)

$$p(m|z_{1:t}, x_{1:t})$$

Localización y Mapeo Simultaneo

Una vez explicados los conceptos por separado de la "Localización" y el "Mapeo", se plantea la problemática real para el diseño de ASTEC, donde tanto la localización como el mapa, son desconocidos para el robot, este problema es conocido como "Localización y Mapeo Simultaneo" - SLAM (Simultaneous Localization and Mapping), o como "Localización y Mapeo Concurrente" - CML (Concurrent Mapping and Localization).

Como se ha comentado con anterioridad, es un problema catalogado como del huevo y la gallina, donde el problema consiste en generar un mapa relativo a la ubicación del robot y la localización del robot relativa al mapa generado de manera "simultanea". [51]

Los problemas de SLAM, se catalogan en dos vertientes:

- Online SLAM
- Full SLAM

En problema de "Online SLAM" se basa en la obtención del "posterior" en un instante determinado (pose momentánea) y el mapa (de igual manera, relativo hasta ese instante).

(2.2.21)

$$p(x_t, m|z_{1:t}, u_{1:t})$$

Donde:

- x_t es la posición en el instante relativo t (también conocido como "posterior").
- m que representa el mapa.
- $z_{1:t}$ son las mediciones de los sensores (observaciones) generadas hasta el tiempo t.
- $u_{1:t}$ representan los comandos de control o información de odometría generada hasta el instante t.

El planteamiento de "Online SLAM" solo realiza la estimación de las variables que se mantienen hasta t, se considera que este enfoque descarta la información de los sensores y control después de su procesamiento e integración a la estimación deseada (es un problema de tipo "incremental").

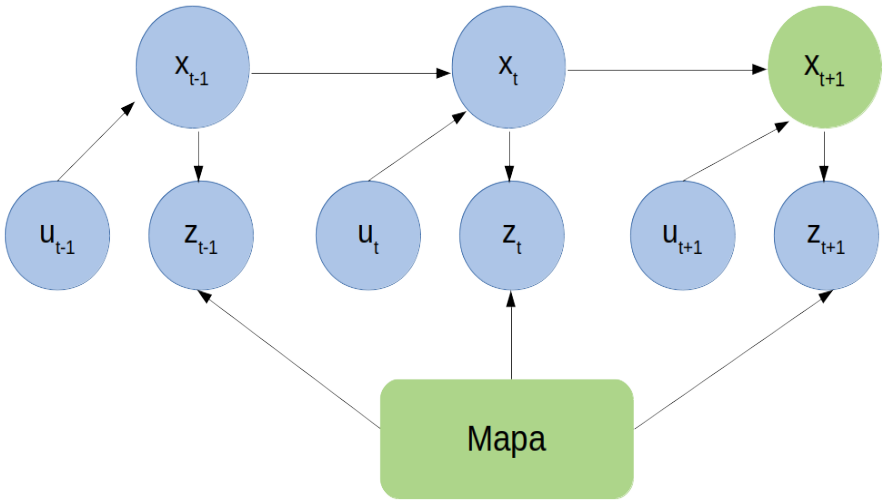


Ilustración 2.2.6.2.3. Modelo Gráfico de Localización y Mapeo Simultaneo (Online SLAM).

Ahora, en el caso de "Full SLAM" el problema consiste en la obtención de todos los estados (calculo del "posterior") a lo largo de todo el camino $x_{1:t}$ y de manera simultánea generar el mapa.

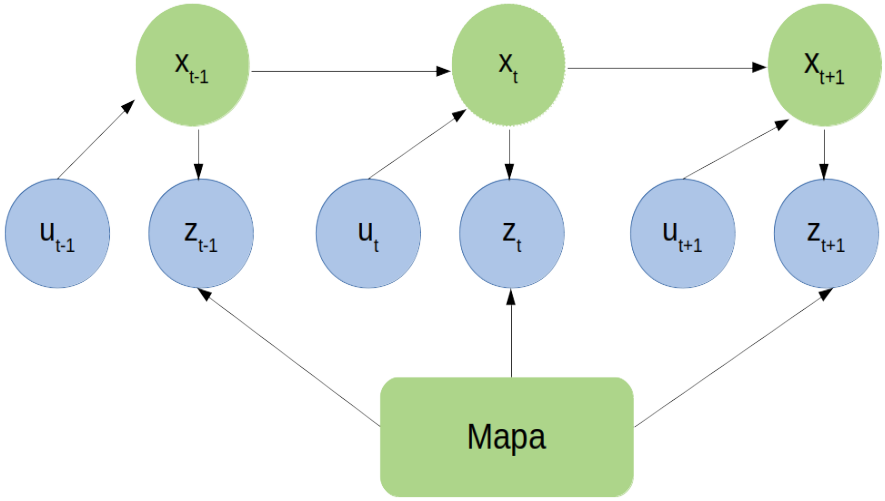


Ilustración 2.2.6.2.4. Modelo Gráfico de Localización y Mapeo Simultaneo (Full SLAM).

Usando la ilustración (2.2.6.2.4), la expresión matemática para el problema de "Full SLAM" es la siguiente:

(2.2.22)

$$p(x_{1:t}, m|u_{1:t}, z_{1:t})$$

Donde m, $u_{1:t}$ y $z_{1:t}$ tienen el mismo significado que para la ecuación (2.2.21). La diferencia principal entre Online SLAM y Full SLAM, es que en Online SLAM los cambios que generan los movimientos de control y las correcciones que se presentan después de integrar las mediciones en el recorrido se realizan "una a la vez", es decir, que en cada instante después de un movimiento u_t y una medición z_t, en Online SLAM se realizan las correcciones (integrándolas) a la posición actual del robot (posterior) y el mapa correspondiente hasta ese momento, mientras que para el caso de Full SLAM el problema tiene una mayor complejidad, ya que este enfoque pretende mantener una noción o registro de los estados $x_{1:t}$ durante todo el recorrido, y no solo durante un instante determinado como en Online SLAM.

En la teoría se dice que el problema de "SLAM", es un híbrido temporal, ya que contiene una parte continua en lo que respecta a la posición y el mapa, y una parte discreta debido a la correspondencia de las marcas en el mapa (es una variable que establece la relación o no de las marcas del mapa). [38]

Para el caso de Online SLAM:

(2.2.23)
$$p(x_t, m, c_t | u_{1:t}, z_{1:t})$$

Y para Full SLAM:

(2.2.24)
$$p(x_{1:t}, m, c_{1:t} | u_{1:t}, z_{1:t})$$

Donde c_t relaciona las correspondencias de las marcas del mapa para el instante t, y $c_{1:t}$ relaciona la correspondencia de las marcas del mapa a lo cargo de todo el camino recorrido.

Métodos y Enfoques para resolver el problema de SLAM

En la actualidad existen muchos enfoques que tratan de resolver el problema de la "Localización y el Mapeo Simultaneo" ("SLAM", por sus siglas en ingles), donde en algunos métodos se contempla la noción de las correspondencias c_t (para Online SLAM) o $c_{1:t}$ (para Full SLAM), pero en general, en la bibliografía consultada se han encontrado tres formas principales para resolver el problema de SLAM (Online o Full). [34] [35] [38] [50] [51]

En el caso de SLAM, el estado (posterior) del sistema se denota como "estado" $x_{slam} = (x_t, m)^T = (x, y, \theta, m_{1,x}, m_{1,y}, s_1, m_{2,x}, m_{2,y}, s_2, ..., m_{N,x}, m_{N,y}, s_N)^T$, donde se incluye el estado del robot $x_t = (x, y, \theta)^T$, el mapa de exploración del robot dado como

$m = (m_{1,x}, m_{1,y}, s_1, ..., m_{N,x}, m_{N,y}, s_N)^T$ cuya dimensión expresada por N dependerá del número de referencias en el mapa (para el caso de los "mapas basados en características"), donde este mapa contiene la ubicación de las marcas en (x, y) para dos dimensiones, y un elemento llamado s_N que es una variable que relaciona alguna característica específica de la marca (color, textura, forma, dimensión, etc.), para el diseño de ASTEC esta variable s_N no se dispone.

Para Online SLAM:

- Filtro Extendido de Kalman - SLAM (Extended Kalman Filter SLAM, EKF-SLAM).
- Fast - SLAM (usa conjuntamente EKF y Filtro de partículas).

Para Full SLAM:

- Fast - SLAM (resuelve tambien problema de Full SLAM).
- Graph - SLAM (utiliza un enfoque basado en Teoría de Grafos).

Usando como base lo visto en el curso "SLAM Lectures"de *Claus Brenner* [50], se implementara en especifico el método de "Fast SLAM" para los propósitos del algoritmo de localización y mapeo del diseño de ASTEC, sin embargo, no se establece que sea la única alternativa para resolver este problema.

La aproximación llamada "Fast SLAM" utiliza como su kernel dos técnicas que por separado pueden resolver los problemas de "localización" o "mapeo", pero en conjunto presentan una solución bastante aceptable para resolver los problemas tanto de Online SLAM como de Full SLAM. La primer técnica de este enfoque usa Filtro Extendido de Kalman (Extended Kalman Filter - EKF) y la segunda técnica ocupa un Filtro de Partículas (Particle Filter - PF).

Un "Filtro Extendido de Kalman" es la aplicación "práctica" del "Filtro de Bayes".

El Filtro de Bayes, a grandes rasgos, contiene dos conceptos fundamentales. El primer concepto se le conoce como "creencia" (antes de incorporar la medición z_t), la "creencia" se relaciona con la "probabilidad de transición de estado", que define como "estado del ambiente" (estado del robot) x_t evoluciona como consecuencia de los comandos de control del robot u_t, es decir, la relación entre la transición del estado dado un movimiento del robot.

(2.2.25)

$$p(x_t|x_{t-1}, u_t)$$

Para el segundo concepto del Filtro de Bayes se tiene que después de la generación de la "creencia", como lo mostrado en las ilustraciones 2.2.6.2.3 y 2.2.6.2.4, existe un elemento de medición z_t que se relaciona con el concepto de "corrección" o "actualización", y esta relacionada con el termino de "probabilidad de medición", esta probabilidad se entiende como la medición de un cierto instante t dado el estado x_t en que se encuentre el sistema (o el robot).

(2.2.26)

$$p(z_t|x_t)$$

Usando la idea de la "probabilidad de transición de estado" y la "probabilidad de medición", se establecen las siguientes "Distribuciones de Creencias" [38]. Con el concepto de probabilidad de transición de estado, se genera el concepto de "creencia" antes de la incorporación de la medición z_t, esta "creencia anterior" se denota como $\bar{bel}()$.

(2.2.27)

$$\bar{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t})$$

Donde, x_t corresponde al estado de "predicción" (antes de la medición), $z_{1:t-1}$ son todas las mediciones hechas hasta $t-1$, y $u_{1:t}$ son todos los comandos de control realizados hasta el instante t.

Después, se usa la "probabilidad de medición", al integrar la medición z_t que permite realizar la "corrección" o "actualización" de la predicción generada en $\bar{bel}()$, esta corrección usa la medición hecha en el instante t posterior al comando de control u_t, esta creencia se denota como $bel()$.

(2.2.28)

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t})$$

Para la ecuación (2.2.28) es necesario aplicar el Teorema de Bayes [3] para generar el inverso de la probabilidad condicional.

En la ilustración (2.2.6.2.5) se observa la relación entre el "Modelo de Movimiento" (en color rojo) y "Modelo de Observación" (en color verde) con el problema de Full SLAM. La probabilidad de transición de estado interactúa con el modelo de movimiento, es decir, que la aplicación del modelo de movimiento u_t en el estado x_t genera la probabilidad de transición o la "creencia anterior" (antes de la medición), y la probabilidad de medición se relaciona con el modelo de observación, que realiza la "corrección" o "actualización" de la creencia anterior para generar la "nueva creencia" del estado x_t posterior a la integración de la medición z_t.

[3] El Teorema de Bayes relaciona la probabilidad condicional $p(a|b)$ con su inverso $p(b|a)$, disponible en el Apéndice A.1.

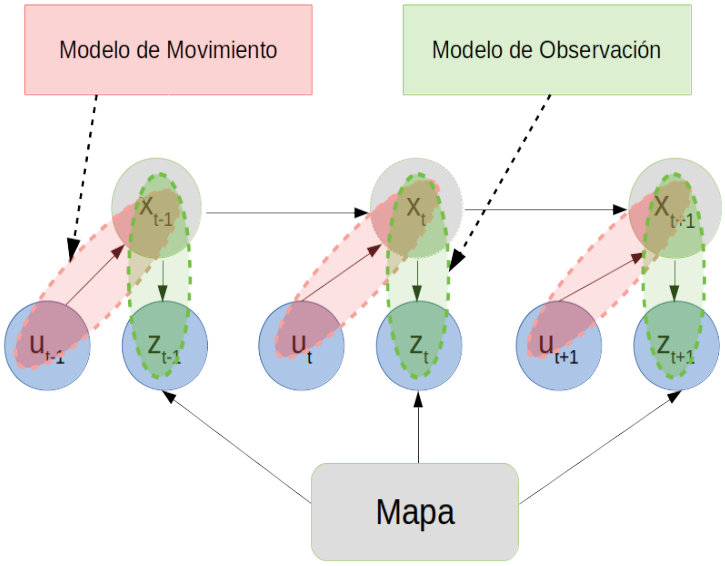


Ilustración 2.2.6.2.5. Modelo Gráfico (Full SLAM) incorporando Modelo de Movimiento y Modelo de Observación.

El principio del Algoritmo de Bayes, como se comento antes, se basa en dos etapas:

- Etapa de Predicción.
- Etapa de Corrección.

La idea de la "Etapa de Predicción" del Algoritmo de Bayes se presenta en la ilustración 2.2.6.2.6.

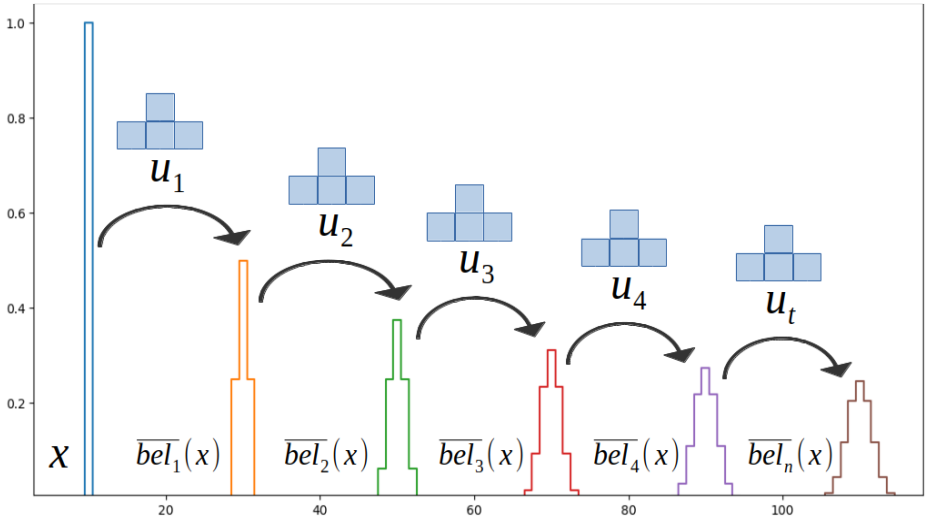


Ilustración 2.2.6.2.6. Propagación de los comandos $u_{1:t}$ en un estado x.

En los cursos tomados de "SLAM Lectures" [50], la representación de la transición de estado, generación de la "creencia anterior" o "etapa de predicción" se puede entender como la convolución de la distribución de un estado x_t (en el caso de la ilustración 2.2.6.2.6, el estado es x) y la distribución que representa al modelo de movimiento $u_{1:t}$ (es piramidal) en color azul (arriba de los comandos de control), como puede apreciarse, al aplicar los comandos de control a un estado unitario x el estado tiende a "extender" su distribución (aumenta la incertidumbre), es decir, que al aplicar el movimiento como una "convolución" sobre el estado, la incertidumbre de la posición del robot crece.

Para todos los elementos que conforman al estado x_t se realiza:

(2.2.29)

$$\overline{bel}(x_t) = \sum p(x_t|u_t, x_{t-1})\ bel(x_{t-1})$$

Donde:

- $\overline{bel}(x_t)$ es la "creencia previa" (previo de la medición).
- u_t son los comandos de control.
- x_{t-1} es el estado anterior.
- $bel(x_{t-1})$ la "creencia" (posterior a la medición z_{t-1}) del estado anterior.

Una vez que se genera la "transición de estado", "creencia previa" o la "etapa de predicción", el filtro de Bayes genera la "corrección" de la "predicción", donde las mediciones $z_{1:t}$ (dependiendo del instante en el que se encuentre) son integradas a la etapa de predicción del filtro.

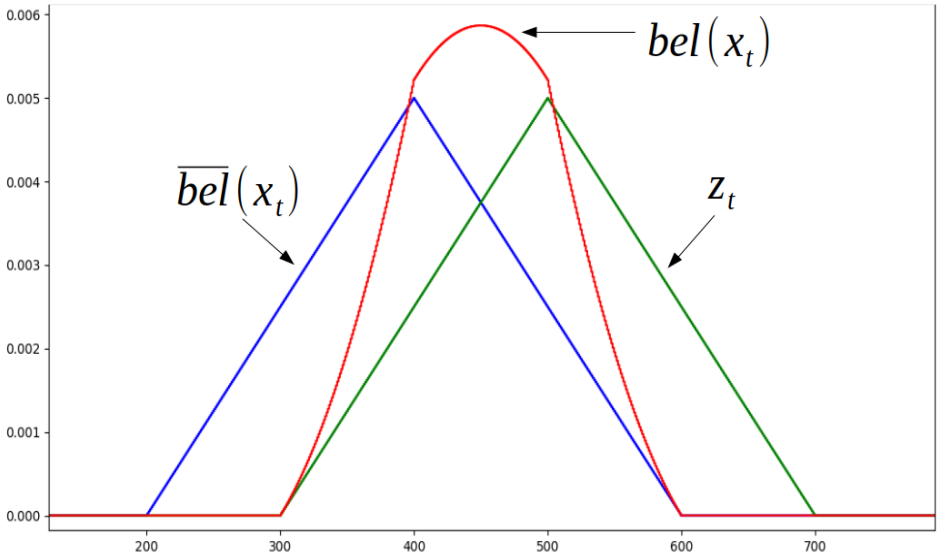


Ilustración 2.2.6.2.7. Etapa de Corrección del Filtro de Bayes.

En la ilustración 2.2.6.2.7 se plantea la representación gráfica de la "Etapa de Corrección" del algoritmo de Bayes, donde de color azul se tiene una "creencia previa" $\bar{bel}(x_t)$ generada por la "Etapa de Predicción", en color verde se representa la medición generada z_t, que dicha medición es relativa a la posición del robot posterior al movimiento, es una medición realizada después de generar $\bar{bel}(x_t)$, posteriormente, en color rojo se establece la interacción entre la creencia previa y la medición z_t. Como puede observarse, la idea de la etapa de corrección es básicamente "multiplicar" la distribución que se tiene como $\bar{bel}(x_t)$ y la medición realizada z_t, al contrario de la etapa de predicción donde al aplicar los movimientos o comandos de control al estado (generando una transición de estado usando el modelo de movimiento) que expande o incrementa la incertidumbre asociada a la posición del robot, la corrección busca "concentrar" (y de alguna forma podría decirse que "mediar") la distribución de la "creencia posterior" $bel(x_t)$.

La ecuación para la "Etapa de Corrección" en el Filtro de Bayes se define como:

(2.2.30)

$$bel(x_t) = \eta p(z_t|x_t)\ \bar{bel}(x_t)$$

Donde:

- $bel(x_t)$ corresponde a la "creencia posterior" a la integración de la medición z_t.
- $p(z_t|x_t)$ la probabilidad de la medición z_t dado que se encuentre en el estado x_t.
- $\bar{bel}(x_t)$ es la "creencia previa" posterior al movimiento u_t y previo a la medición z_t.
- η factor de normalización.

Finalmente, existe un concepto llamado la "Suposición de Markov" o la "Asunción de Markov", en la cual postula que no existe (es independiente) relación entre el pasado y el futuro si el estado x_t es conocido. Sin embargo, en la mayoría de los casos prácticos esto no es posible, debido a un gran número de factores asociados, como lo son: objetos en movimiento no modelados, errores en los modelos probabilistas propuestos, errores de aproximaciones (linealización de distribuciones de movimiento, medición, etc.), etc.

Por ello, se plantea que para el diseño de esta tesis, se puede decir que existe una violación de la "Suposición de Markov", ya que crear un modelo que sea completamente exacto (es decir, que no viole el postulado de Markov) demanda muchos más recursos computacionales, y tiempo de modelado para el movimiento y medición del diseño de ASTEC. [38]

Contemplando los conceptos explicados en relación al "Filtro de Bayes" [4], ahora se abordará el "Filtro de Kalman" como una aplicación de practica del "Filtro de Bayes".

Filtro de Kalman n dimensional

El "Filtro de Kalman" se puede considerar dentro de los "Filtros Gaussianos" [38], donde dichos filtros gaussianos son una aplicación del "Filtro de Bayes" para problemas en espacios continuos, estos filtro se consideran de tipo paramétrico [50], donde se establece una forma de abordar los problemas de filtrado para procesos estocásticos con sensores (y actuadores) ruidosos [52], los filtros de Kalman contienen la idea del "Filtro de Bayes" al generar las etapas de predicción y corrección, con la diferencia que para los filtros de Kalman estas predicciones y correcciones son expresadas en distribuciones Gaussianas (Normales) que pueden ser uni-variantes o multi-variantes. Estos filtros (de Kalman) se consideran paramétricos por el hecho de que solo es necesario conocer el primer y segundo momento de la distribución para poder ser expresados, donde el primer momento corresponde a la "media" y el segundo momento corresponde a la "varianza". La distribución de probabilidad Normal o Gaussiana (multi-variantes) se representa como :

(2.2.31)
$$p(x) = det(2\pi\Sigma)^{-1/2} \exp^{(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))}$$

Donde:

- $p(x)$ es la distribución de probabilidad de x.
- μ es la media (tiene la misma dimensión que el vector de estado x para distribuciones multi-variantes).
- Σ representa a la covarianza, que es simétrica y semidefinida-positiva (su dimensión es n x n, donde n es la dimensión del vector de estado.

Para el caso de modelos de transición de estado (modelo de movimiento) y modelos de observación lineales, primero se tiene la forma en como se expresa el estado x_t del robot como:

(2.2.32)
$$x_t = A_t x_{t-1} + B_t u_t + \psi_t$$

[4] La deducción y comprobación matemática del "Filtro de Bayes", puede consultarse en [38] paginas 31 - 33.

Donde:

- $x_t = \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ . \\ . \\ . \\ x_{n,t} \end{bmatrix}$ es un vector correspondiente a los estados (posiciones) del robot.
- $u_t = \begin{bmatrix} u_{1,t} \\ u_{2,t} \\ . \\ . \\ . \\ u_{m,t} \end{bmatrix}$ es el vector correspondiente a los comandos de control.
- A_t es una matriz n x n, n corresponde a la dimensión del vector de estado.
- B_t es una matriz n x m, m corresponde a la dimensión del vector de comandos de control.
- ψ_t es una variable aleatoria que representa el "ruido" o la "incertidumbre" asociada a la transición de estado, la media de esta variable es *cero* y la covarianza se representa por R_t

Ahora, para la transición de estado $(x_t|u_t, x_{t-1})$ se usa la ecuación (2.2.31) y la ecuación (2.2.32).

(2.2.33)
$$p(x_t|u_t, x_{t-1}) = det(2\pi R_t)^{-1/2} \exp^{(-\frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T R_t^{-1}(x_t - A_t x_{t-1} - B_t u_t))}$$

Las ecuaciones para calcular los parámetros necesarios para el "Filtro de Kalman" se pueden dividir de forma similar al "Filtro de Bayes", entonces, para la etapa de predicción (relaciona al estado anterior x_{t-1} con la aplicación de un comando de control u_t) se tiene:

(2.2.34)
$$\bar{\mu_t} = A_t \mu_{t-1} + B_t u_t$$

(2.2.35)
$$\bar{\Sigma_t} = A_t \Sigma_{t-1} A_t^T + R_t$$

Donde:

- $\bar{\mu_t}$ representa la media de la predicción (también se puede interpretar como la media del estado del robot x_t).

- A_t y B_t son las matrices para la media del estado y los comandos de control respectivamente.
- μ_{t-1} corresponde a la media del estado en $t-1$ (estado anterior).
- $\bar{\Sigma_t}$ es la matriz de covarianza, que se refiere a la incertidumbre en la posición de la "predicción" $\bar{\mu_t}$.
- Σ_{t-1} corresponde a la covarianza (incertidumbre) en la posición anterior, es decir, en $t-1$.
- R_t se le conoce como "Ruido del Movimiento", estos datos usualmente son obtenidos de hojas de datos técnicos (datasheets) donde se expresan las varianzas correspondientes a los movimiento mecánicos de cierto actuador, motor, etc.

Contemplando lo mencionado respecto a la etapa de "Predicción" presente en el "Filtro de Kalman" debido a su kernel basado en el algoritmo de Bayes, se dice que:

(2.2.36)
$$\bar{Bel}(t) = \begin{cases} \bar{\mu_t} \\ \bar{\Sigma_t} \end{cases}$$

Donde para propósitos prácticos μ_t representa la aproximación del estado x_t, y su incertidumbre asociada en posición Σ_t.

Después usando la "Predicción" de la transición de estado generada, se plantea la etapa de "Corrección" de filtro de Kalman, donde se calcula la "Ganancia de Kalman".

(2.2.37)
$$K_t = \bar{\Sigma_t} C_t^T (C_t \bar{\Sigma_t} C_t^T + Q_t)^{-1}$$

Donde:

- K_t corresponde a la "Ganancia del Filtro de Kalman" o "Ganancia de Kalman", que genera la relación entre el modelo de observación lineal C_t y la interacción entre la incertidumbre de la predicción y la incertidumbre en la medición.
- C_t es una matriz k x n, donde k es la dimensión del vector correspondiente a las mediciones. Esta matriz se asocia con el modelo de observación del robot, y para este caso dicho modelo debe ser lineal.
- Q_t es la matriz de covarianza asociada a la incertidumbre del proceso de medición, actúa de manera similar a R_t.

Una cuestión interesante respecto a la ecuación (2.2.37) es la parte de $C_t\bar{\Sigma}_t C_t^T$ que consiste en propagar la incertidumbre a través de C_t y posteriormente sumarlo a la incertidumbre (errores aleatorios) de las mediciones Q_t.

Una vez generada la "Ganancia de Kalman", se pueden establecer las ecuaciones para la etapa de "Corrección" del filtro.

(2.2.38)

$$\mu_t = \bar{\mu}_t + K_t(z_t - C_t\bar{\mu}_t)$$

(2.2.39)

$$\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$$

Donde:

- μ_t corresponde a la media o estado del sistema posterior a la incorporación de la medición.
- K_t es la ganancia de Kalman.
- $z_t = C_t\mu_t + \delta_t$ representa a la medición en función de la forma determinista del modelo de observación C_t, la posición μ_t (o x_t) y un parámetro de ruido Gaussiano δ_t.
- Σ_t es la covarianza (relacionada con la incertidumbre) de la posición μ_t posterior a la "Corrección" (incorporación de la medición).
- I es la matriz identidad.

Finalmente se dice que $Bel(t)$, es decir, la creencia posterior a la etapa de "Corrección" esta representado por:

(2.2.40)

$$Bel(t) = \begin{cases} \mu_t \\ \Sigma_t \end{cases}$$

En general, la idea del "Filtro de Kalman" [5] n dimensional (lineal) puede usarse en muchas aplicaciones donde se precise de un modelo que pueda incorporar incertidumbres o errores aleatorios a los movimientos y mediciones de un sistema, para el diseño de ASTEC se usara una versión de estos filtros llamada "Filtro de Kalman Extendido" que se explicará y aplicará en el próximo capítulo como una parte elemental del enfoque de "FAST-SLAM".

[5] La deducción y comprobación matemática del Filtro de Kalman para n dimensiones, puede consultarse en [38] paginas 45 - 54, y en [52].

Filtro de Kalman Extendido

El "Filtro Extendido de Kalman" tiene como propósito lidiar con el problema de las transiciones lineales necesarias para implementar el "Filtro de Kalman" [38] [50], en general el "Filtro Extendido de Kalman" es aplicado para sistemas no lineales [53].

Los filtros de Kalman son ampliamente usados para responder a problemas de filtrado con funciones de movimiento y medición que presentan algún grado de incertidumbre o errores aleatorios presentes en cualquier dispositivo robótico. Sin embargo, estos filtros al ser perimétricos, presentan problemas al poseer modelos de transición de estado o modelos de corrección no lineales [38], dado que representar la etapa de "predicción" y "corrección" en una forma Gaussiana, son fundamentales para el buen funcionamiento de estos filtros. Es por ello que el problema de la linealización de funciones para la aplicación de estos filtros es necesaria.

Para trabajar con modelos no lineales existe un tipo de filtro (dentro de la familia de los filtros de Kalman) llamado "Filtro Extendido de Kalman", este filtro tiene como característica principal la representación de los modelos de transición de estado (modelo de movimiento) y el modelo de corrección (modelo de medición) a través de funciones no lineales.

(2.2.41)
$$x_t = g(u_t, x_{t-1}) + \eta_t$$

(2.2.42)
$$z_t = h(x_t) + \rho_t$$

Como puede observarse en la ecuación (2.2.41), a diferencia de la representación usada en el "Filtro de Kalman n dimensional", la función de transición de estado $g()$ (modelo de movimiento) se dice que puede ser de tipo no lineal más un parámetro de error aleatorio denotado por η_t asociado con la incertidumbre en las transiciones de estado del robot (para el caso lineal del filtro de Kalman, esta transición estaba denotada por las matrices A_t y B_t). También, la ecuación (2.2.42) se dice que la función $h()$ que representa al modelo de corrección (modelo de medición) puede ser de tipo no lineal más un parámetro de incertidumbre denotado por ρ_t que representa al error aleatorio asociado a los procesos de corrección (medición) del filtro.

Las ecuaciones necesarias para implementar el "Filtro Extendido de Kalman" son muy similares a las usadas para el caso lineal en "Filtro de Kalman n dimensional", sin embargo presentan algunas modificaciones como se explicará a continuación.

Para la etapa de "predicción" se tiene:

(2.2.43)

$$\bar{\mu_t} = g(u_t, \mu_{t-1})$$

(2.2.44)

$$\bar{\Sigma_t} = G_t \Sigma_{t-1} G_t^T + R_t$$

Donde $\bar{\mu_t}$ es la representación de la media (estado) de la predicción del filtro, la función no lineal $g()$ se aplica al estado anterior μ_{t-1} y el comando de control u_t, esta función $g()$ en el caso del diseño de ASTEC, es generada por el "Modelo de Movimiento" explicado en subsecciones anteriores, después se tiene el cálculo de la matriz de covarianza $\bar{\Sigma_t}$ que es la incertidumbre asociada a μ_t, esta covarianza se calcula usando una matriz G_t que es una "matriz Jacobiana" [6] de la función $g()$ con respecto al estado $\frac{\partial g}{\partial estado}$. Esta matriz Jacobiana G_t utiliza un método llamado "Expansión por Series de Taylor" (de primer orden), que básicamente contiene las derivadas parciales de la función $g(x, y, \theta, der, izq)$ (x, y y $theta$ son las variables del estado del robot, y der e izq son los valores de los comandos de control u_t) con respecto a las variables del estado (x, y, θ). Por ultimo, en la ecuación (2.2.44) tenemos a R_t que se puede descomponer como:

(2.2.45)

$$R_t = V_t \Sigma_{control} V_t^T$$

Donde:

- V_t es una matriz Jacobiana de la función $g(x, y, \theta, der, izq)$ con respecto de los comandos de control (der, izq) de μ_t, es decir, $\frac{\partial g}{\partial control}$.
- $\Sigma_{control}$ es $\begin{bmatrix} \sigma_{izq}^2 & 0 \\ 0 & \sigma_{der}^2 \end{bmatrix}$ que son los valores de las desviaciones estándar para los actuadores izquierdo y derecho elevadas al cuadrado (estos datos se pueden modelar u obtener de hojas de datos técnicos de los actuadores).

Para la etapa "corrección" para el "Filtro Extendido de Kalman" se tiene:

(2.2.46)

$$K_t = \bar{\Sigma_t} H_t^T (H_t \bar{\Sigma_t} H_t^T + Q)^{-1}$$

(2.2.47)

$$\mu_t = \bar{\mu_t} + K_t(z_t - h(\bar{\mu_t}))$$

(2.2.48)

$$\Sigma_t = \bar{\Sigma_t}(I - K_t H_t)$$

[6] En Apéndice A.2 se encuentra una explicación mas amplia de lo que es una matriz Jacobiana

Donde:

- μ_t representa a la posición (estado) del robot posterior a la etapa de "corrección".
- K_t es la ganancia del filtro extendido de Kalman.
- $h()$ es el modelo de observación (no lineal) aplicado al estado de predicción $\bar{\mu}_t$.
- H_t es una matriz Jacobiana del modelo de medición $h()$ con respecto del estado (x, y, θ).
- Q es la matriz de covarianza de la incertidumbre asociada a la medición $Q = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\alpha^2 \end{bmatrix}$, donde r y α son las variables del modelo de movimiento que se explicaran en el siguiente capítulo.

La etapa de "corrección" del "Filtro Extendido de Kalman" [7] funciona de manera similar al "Filtro de Kalman n dimensional", sin embargo, para el caso del filtro extendido, la función $h()$ se considera no lineal, esta función se puede interpretar como el "Modelo de Observación" para el diseño de ASTEC. Además, la matriz C_t presente en la etapa de "corrección" del filtro de Kalman (ecuaciones (2.2.37) y (2.2.38)) es suplantada por una matriz Jacobiana H_t que relaciona $\frac{\partial h}{\partial estado}$, es decir, que genera la relación del modelo de observación y su transformación con respecto del estado (x, y, θ). De manera similar al filtro de Kalman n dimensional, la etapa de "innovación" esta presente en $(z_t - h(\bar{\mu}_t))$, esta "innovación" establece la relación entre lo que el sensor detecta z_t y lo que "debería detectar", cuando se dice lo que "debería detectar" se habla de la aplicación del modelo de observación $h()$ al estado de predicción $\bar{\mu}_t$.

Filtro de Partículas

La ultima parte del enfoque de "FAST-SLAM" es el "Filtro de Partículas" que utiliza una técnica de aproximaciones discretas para expresar $\bar{Bel}(x)$ y $Bel(x)$ (etapa de predicción y corrección), el "Filtro de Partículas" (FP) busca encontrar la mínima varianza en un grupo muestra de partículas [53], además, se dice que el grupo muestra es seleccionado de manera "aleatoria", con la finalidad de poder representar a través de dicha muestra, una densidad de probabilidad del estado del sistema. El "Filtro de Partículas" se considera un filtro "no paramétrico" [8] [38] [50], los filtros no paramétricos (Filtros de Histograma, Filtro de Partículas, etc.) tienen la característica de no necesitar representar el "posterior" $(Bel(x))$ en una forma estricta, es decir, a diferencia de los

[7] La deducción y comprobación matemática del Filtro Extendido de Kalman, puede consultarse en [38] paginas 59 - 61.

[8] Los "filtros no paramétricos" no dependen de una forma funcional fija del posterior.

"Filtro de Kalman" que necesitan representar su posterior como una forma de distribución Gaussiana o Normal, los filtro de partículas no paramétricos no. Para el caso del "Filtro de Partículas" este representa su posterior $Bel(x)$ en función de muestras aleatorias del estado del mismo posterior $Bel(x)$. El "Filtro de Partículas" es un método secuencial de "Monte Carlo", donde las variables latentes se encuentran conectadas dentro de la "Cadena de Markov". [53]

En general, los filtros de partículas poseen dos ventajas con respecto a otros métodos de filtros probabilísticos, la primer ventaja radica en el hecho de poder representar un espectro más grande de densidades de probabilidad (a diferencia de los "Filtro de Kalman", que están restringidos a su forma Gaussiana), es decir, que pueden representar distribuciones multi-modales, y la otra ventaja se presenta en la capacidad de modelar transformaciones (transiciones) no lineales de variables aleatorias (estados del robot). [38] [50]

Sin embargo, la precisión que se pretenda de este tipo de filtros genera una relación directa con el costo computacional de los cálculos necesarios. [50]

La representación del "posterior" en el "Filtro de Partículas" es la siguiente:

(2.2.49)
$$X_t := x_t^1, x_t^2, ..., x_t^M$$

Donde:

- X_t es el "posterior" en un instante t, algo similar a $Bel(x)$.
- $\left\{x_t^1, x_t^2, ..., x_t^M\right\}$ corresponde a las "Partículas", cada partícula funciona como una posibilidad o hipótesis del estado X_t, donde las partículas que representan al posterior pueden ser desde 1 hasta M, y M es el número total de partículas del posterior, usualmente se relaciona con un número grande $M = 1,000$ o mayor.

Además, estas partículas contienen una "probabilidad" o "posibilidad" asociada para cada partícula, que debe ser proporcional al posterior del "Filtro de Bayes" $Bel(x_t)$.

(2.2.50)
$$x_t^{[m]} \sim p(x_t|z_{1:t}, u_{1:t})$$

La ecuación (2.2.50) tiene como finalidad generar conjuntos de sub-regiones de partículas con una mayor "densidad", y esto conlleva a que el "verdadero estado" (del robot) tenga una mayor probabilidad de encontrarse dentro de estas sub-regiones.

Para el cálculo de etapa de "predicción" del "Filtro de Partículas" $\bar{x}^{[}m]_t$, se establece que para cada partícula desde $m = 1$ hasta M (en el entendido de que en su estado inicial

χ_{t-1} esta denotado por algún arreglo de partículas con alguna distribución a priori, puede ser Gaussiana u otra).

Entonces, como primer paso se generan muestras para los comandos de control izq y der.

(2.2.51)
$$sample\ i'_t \sim N(i_t, \sigma^2_{i_t})$$

(2.2.52)
$$sample\ d'_t \sim N(d_t, \sigma^2_{d_t})$$

Donde:

- *sample* representa a alguna función de muestreo "aleatoria" para generar las muestras de la distribución de los comandos de control.
- i'_t es la muestra del comando izq del robot.
- d'_t es la muestra del comando der del robot.
- $N(i_t, sigma^2_{i_t})$ corresponde a una distribución Normal (N) o Gaussiana para el valor izq de control, donde el primer momento (media) esta dado por el valor mismo i_t (valor del comando izquierdo) y su segundo momento (varianza) esta dado por $sigma^2_{i_t}$.
- $N(d_t, sigma^2_{d_t})$ corresponde a una distribución Normal (N) o Gaussiana para el valor der de control, donde el primer momento (media) esta dado por el valor mismo d_t (valor del comando derecho) y su segundo momento (varianza) esta dado por $sigma^2_{d_t}$.

Una vez que se tiene la muestra "aleatoria" para el comando izq (i'_t) y der (d'_t), se aplica la función de transición de estado $g()$ (puede ser lineal o no lineal) y se agrega cada partícula $\bar{x}^{[m]}_t$ y son reemplazadas por los arreglos de partículas anteriores en χ_{t-1} generando el nuevo estado de predicción $B\bar{e}l_t$ o $\bar{\chi}_t$.

(2.2.53)
$$\bar{x}^{[m]}_t = g(x^{[m]}_{t-1}, \begin{bmatrix} i'_t \\ d'_t \end{bmatrix})$$

Entonces, el estado de "predicción" queda denotado por:

(2.2.54)
$$\bar{\chi}_t = \left\{ \bar{x}^{[1]}_t, \bar{x}^{[2]}_t, ..., \bar{x}^{[m]}_t \right\}$$

Posterior a la generación de las "Partículas de Predicción", el filtro de partículas [9] contiene también una etapa de "corrección" similar a la del "Filtro de Bayes", en esta etapa de "corrección" se genera el estado corregido $Bel(t)$ en función a la medición generada z_t. Esta etapa de corrección consiste principalmente en dos pasos, el primer paso de la generación de "pesos" o mejor conocida como "factor de importancia" [38] [50] [53] denotado por $W_t^{[m]}$, para este paso se tiene que por cada partícula desde $m = 1$ hasta M del estado de predicción $\bar{\chi}_t$ se aplica la ecuación:

(2.2.55)
$$W_t^{[m]} = p(z_t|\bar{\chi}_t^{[m]})$$

Donde:

- $W_t^{[m]}$ es el peso por partícula que relaciona la probabilidad de la medición z_t dado el estado de hipótesis generado por alguna partícula $\bar{x}_t^{[m]}$.
- z_t es la medición "real" generada por el sensor.
- $\bar{\chi}_t^{[m]}$) es el conjuto de particulas generadas en el estado de predicción $\bar{Bel}(x)$.

El segundo paso es realizar un procedimiento llamado "Muestreo de Importancia", este proceso consiste en generar un nuevo conjuto de particulas proporcional al peso generado por partícula en la ecuación (2.2.55).

(2.2.56)
$$mapear\ i \propto w_t^{[i]}$$

Donde:

- *mapear* se define como el proceso de generar una "probabilidad" asociada al índice i de una partícula perteneciente al conjunto de partículas de la etapa de predicción.
- i corresponde al índice individual de cada partícula proporcional a su peso $w_t^{[i]}$.
- $w_t^{[i]}$ es el peso individual de cada partícula.

[9] La deducción y comprobación matemática del Filtro de Partículas, puede consultarse en [38] paginas 103 - 104.

Fast SLAM

Como se menciono en subsecciones anteriores, el enfoque para resolver el problema de la "Localización" y el "Mapeo" para el diseño de ASTEC, se conoce como "Fast SLAM" [38] [50], Fast SLAM implementa conjuntamente la idea del "Filtro Extendido de Kalman" (para el caso de ASTEC), y el "Filtro de Partículas".

Fast SLAM implementa una versión de los filtros de partículas llamado "Filtros de Partículas Rao-Blackwellized", esta versión permite representar el posterior (el estado generado después de la etapa de corrección) $Bel(x)$ a través de partículas, donde además se ocupan distribuciones paramétricas como Gaussiana (u otras) para representar todas las demás variables [38] [50]. Ya que el estado del sistema se contempla como partículas, cada error es condicionalmente independiente entre partículas, es decir, que cada partícula tiene su incertidumbre en transición y en medición asociada.

Cada marca de mapa m_N se le asocia (en el caso de ASTEC) una posición $(m_{1,x}, m_{1,y}, ..., m_{N,x}, m_{N,y})$, Fast SLAM usa la idea de separar cada marca de mapa como un Filtro Extendido de Kalman independiente, esto permite generar procesos de filtrado independientes por marca y de baja dimensionalidad. Esta estrategia presenta diversas ventajas y desventajas con respecto a otras aproximaciones como "EKF - SLAM" que genera un solo Gaussiano para representar el estado del sistema, lo que puede ocasionar muchos problemas al usar un gran número de marcas de mapa, debido a que cada marca de mapa aumenta en un factor de $3n$ la dimensión del filtro.

Una de las ventajas más marcadas de Fast SLAM (FS) con respecto de otros enfoques para resolver el problema de "SLAM", es el hecho de que cada "hipótesis" de estado se genera de manera individual para cada partícula, esto genera que Fast SLAM genere una aproximación del posterior, y no solo una "máxima asociación de probabilidad de los datos" [38]. La implementación de un filtro de partículas de manera interna para FS permite también lidiar con los problemas de no linealidad de los modelos de movimiento y observación, además de problemas de incertidumbre elevada.

En el siguiente capítulo para explicarán los elementos propuestos para el diseño de ASTEC, como lo son: elementos de hardware contemplados para el diseño, la programación de los algoritmos, además, se mostraran algunas simulaciones y gráficas obtenidas.

Capítulo 3

Materiales y Diseño

3.1. Materiales

Ya que éste trabajo de tesis se plantea como un diseño teórico y simulado de un robot de rescate (ASTEC), haciendo uso de la información planteada en el marco teórico, en esta sección se colocarán los elementos de hardware de elección final para el diseño, aunque no se pretende abordar una etapa de implementación física del prototipo, se busca dar una idea firme de lo que se busca pueda desempeñar el diseño ASTEC.

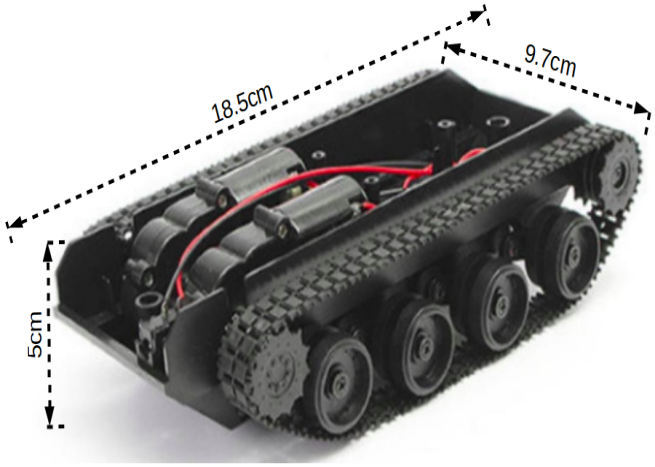
3.1.1. Hardware de Módulo de Locomoción

Entonces, el primer elemento contemplado para el diseño es la "Unidad de Locomoción", esta unidad es propiamente el chasis del diseño del robot, donde en base a lo explicado anteriormente, las dos opciones principales son: un chasis de rodamiento por oruga o un chasis de locomoción híbrida.

Debido al aspecto del costo del chasis, y con la idea de mantener el diseño general de ASTEC, en un costo relativamente bajo, para propósitos de este trabajo, se pretende incorporar un chasis de rodamiento por oruga.

En la ilustración (3.1.1) se pude observar el chasis de rodamiento por oruga planteado para el diseño de ASTEC, las dimensiones del chasis son: 18.5cm de largo, 9.7cm de ancho y 5cm de alto; el costo promedio de este modelo de chasis esta entre $300.00 a $400.00 MXN (incluyendo gastos de envió).

El chasis esta hecho en su mayoría de plástico ABS, es por ello que el chasis no se menciona de "alta confiabilidad", sin embargo, este chasis es un elemento que sirve como base para realizar el planteamiento del diseño teórico de ASTEC, en caso de una implementación física se pueden explorar otros chasis (que contengan un sistema de locomoción similar o un sistema de locomoción híbrido) que estén fabricados con materiales mas resistentes.



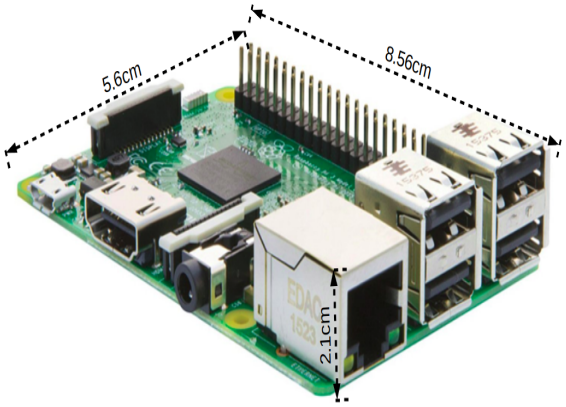
Ilustracion 3.1.1. Chasis de rodamiento por oruga (imagen original obtenida de: `https://www.amazon.com/SCENERY-Tracked-Platform-Chassis-Arduino/dp/B0798D8RD6/ref=sr_1_12?keywords=robot+chassis&qid=1555958717&s=gateway&sr=8-12`.)

Las características de operación mas relevantes del chasis son:

- 2 motores de DC, con voltaje de operación de 4.5V a 7.0V (aproximadamente), y una corriente de consumo (con carga) de 250mA (max).
- Cuenta con un deposito para 4 baterías AA (estas no serán las que que se pretenden para el diseño), además, cuenta con un conector de alimentación (o carga) en la parte posterior.
- Peso por motor: 17.5gr (solo contemplando el motor), y el peso total del chasis es de: 221.126gr (según los datos de la pagina del vendedor).
- Capacidad de amortiguación del rodamiento de aproximadamente 0.7cm.
- Área de superficie superior $162cm^2$ (aproximadamente), esta zona esta destinada para la colocación de los componentes del robot.

3.1.2. Hardware de Unidad de Control y Procesamiento

La "Unidad de Control y Procesamiento" que se pretende para ASTEC (como se menciono en el capítulo anterior) esta basada en una microcomputadora, en especifico, una Raspberry Pi 3 B, por su bajo costo y buena disponibilidad (además de su gran comunidad de soporte y desarrollo). El costo aproximado de esta plataforma de desarrollo es de $750.00 MXN (tomando el precio de `www.amazon.com.mx`).



Ilustracion 3.1.2. Unidad de Control y Procesamiento con Raspberry Pi 3 B (imagen original obtenida de: `https://www.amazon.com/Raspberry-Pi-MS-004-00000024-Model-Board/dp/B01LPLPBS8/ref=sr_1_4?crid=1XW8GNQ9EEZO2&keywords=raspberry+pi+3+b&qid=1555962735&s=gateway&sprefix=rasp\%2Caps\%2C237&sr=8-4`.)

Consultando la hoja de datos técnicos del fabricante [58], las características más relevantes de la placa son:

- Procesador: Broadcom BCM2387 chipset, 1.2Ghz Quad-Core ARM Cortex-A53
- GPU: Dual Core VideoCore IV, 24GFLOPs
- Memoria: 1GB LPDDR2 RAM, Memoria ROM (SD Card) de 8 - 32 GB.
- Alimentación recomendada: 2.5A a 5V.
- Dimensiones: 8.5cm de alto, 5.6cm de largo, y aproximadamente 2.1cm de alto.
- Conectividad: 802.11 b/g/n Wireless LAN y Bluetooth 4.1 (Classic y LE)
- Puertos: 4 x USB 2.0, 1 x Ethernet (10 / 100), 1 x HDMI, 1 x CSI Camera Connector, 1 x Micro USB Connector (Alimentación), 40 GPIOs (UART, 2 x SPI, 2 x TWI, 26 I / O, 4 x PWM, Alimentación externa: 5V, 3.3V, GND, Vin, etc) voltaje y corriente de operación recomendado para los GPIOs es de 3.3V con una corriente máxima de descarga de 54mA.

La Raspberry Pi 3 B se usara con el SO “Raspbian” y se utilizará Python 3 para la programación de los algoritmos de ASTEC.

3.1.3. Alimentación

La unidad de alimentación se propone con un paquete de baterías con conexión USB, esto tomando en cuenta la compatibilidad del puerto de alimentación de la Raspberry, dicho paquete de baterías son de Ion-Litio. La idea del paquete de baterías (también se conocen como Power Banks) es que este tipo de dispositivos pueden almacenar una gran cantidad de energía (desde 4,000mAh hasta 20,000mAh) dependiendo del modelo y el tipo de dispositivos a los cuales alimenta, además de esto, este tipo de fuentes ofrecen una salida constante de 1A a 3A (dependiendo del modelo) de corriente; como se planteo en el Cuadro 2.6, el diseño ASTEC consumirá en promedio 2.5A (con un máximo de 5A). El paquete de baterías es el siguiente:

Ilustracion 3.1.3. Unidad de Alimentación para ASTEC (imagen original obtenida de: `https://www.amazon.com/dp/B00Z9QVE4Q/ref=emc_b_5_i`.)

En la descripción de la batería "Anker PowerCore 13000", [59] usando los datos de la pagina del vendedor, las características más importantes de la batería son:

- Carga máxima de 13,000mAh.
- Dos salidas de 5V con descarga promedio de 2.5A (con un máximo de 5.4A).
- Peso de 240grms.
- Dimensiones: 9.652cm alto, 7.874cm ancho y 2.286cm de alto.

3.1.4. Hardware de Módulo de Comunicación

El modulo de comunicación tiene como propósito proporcional el método de transmisión de información del despliegue del diseño ASTEC, el modulo de comunicación se propone con un ancho de banda de operación de 2.4Ghz, en el marco

teórico se planteo la implementación de un par de módulos Digi Xbee de 2.4Ghz, sin embargo, consultado en páginas de venta en línea el costo de estos productos es demasiado alto, al rededor de los $1100.00 a $1500.00 MXN por el par (sin contemplar los adaptadores para alguno de los puertos de comunicación de la Raspberry Pi). Debido a que el diseño ASTEC se pretende sea de bajo costo, se plantea como opción más viable un par de transreceptores E01-ML01DP5, que usan un ancho de banda similar, y cuentan con una biblioteca para Python 3, usando el estándar SPI de comunicación.

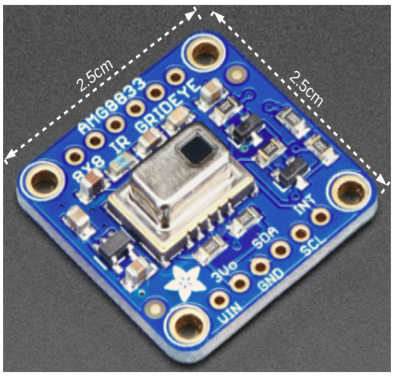
Ilustracion 3.1.4. Módulo de Comunicación para ASTEC (imagen original obtenida de: `https://www.amazon.com/MakerFocus-nRF24L01P-Transmission-Interface-Anti-Interference`.)

El par de transreceptores de la Ilustración 3.1.4 tienen un costo aproximado de $350.00 MXN (incluyendo envió), un transreceptor se integra como el método de comunicación en físico (para ASTEC) y el otro transreceptor se establece como el receptor físico de un usuario en campo, cada transmisor cuenta con una antena SMA. Las características más importantes de estos transreceptores son [60]:

- Ancho de banda de 2.4Ghz a 2.5Ghz.
- Potencia máxima de 100mW (20dBm).
- Velocidad de transmisión hasta 2Mbps.
- Distancia efectiva de 2km (en los datos del vendedor, no se contempla información de uso en interiores o exteriores).
- Cuentan con escudo anti-interferencia.

3.1.5. Hardware de Módulo de Sensores

El primer sensor que se pretende para el diseño es la "cámara térmica", el nombre de la cámara térmica es "Adafruit AMG8833 IR Thermal Camera Breakout", en particular este sensor tiene el propósito de extraer las lecturas térmicas de ciertas áreas determinadas, estas lecturas se obtienen como una matriz de espectros de calor generados por la cámara térmica, la matriz es de $8x8$ donde cada elemento es un sensor IR térmico (diseñado por Panasonic) con un rango efectivo de captación de señales de calor humanas de $7m$ (según datos de Adafruit).



Ilustracion 3.1.5. Cámara Térmica (imagen original obtenida de: `https://www.adafruit.com/product/3538`.)

Las características más relevantes de la cámara térmica AMG8833 [61] son:

- Temperaturas de captura de 0 a 80 grados centígrados.
- Distancia efectiva de captura (rango de temperatura humana) 7m.
- Frame rate de 10Hz.
- Bus I2C.
- Matriz de 8x8 píxeles (64 lecturas individuales).

La cámara térmica AMG8833 tiene bibliotecas compatibles con Python 3 y Python 2 para poder implementar los protocolos de comunicación y control por I2C, además también pueden ocuparse herramientas de la biblioteca SciPy (de Python 3 o Python 2) para hacer procesamiento de la información y generar imágenes con los espectros de calor detectados por la cámara, esto con la finalidad de obtener información que se pueda interpretar con un mayor facilidad para el usuario, y también proporcionar formas más adecuadas para establecer la identificación de posibles víctimas por ASTEC.

El segundo sensor propuesto es el sensor láser omnidireccional LIDAR, este sensor generaría las mediciones de las distancias a los objetos a través de un envió de un pulso del láser y posterior recepción del pulso, el sensor LIDAR pensado para el diseño de ASTEC es el "Slamtec RPLIDAR A1", este sensor esta diseñado para procesos de "Localización y Mapeo Simultáneo" como los que se proponen para el diseño ASTEC [62], además este sensor cuenta con una biblioteca de control para Python 3 y un adaptador para USB, lo cual permite controlarlo usando uno de los puertos USB de la Raspberry Pi 3 B, sin la necesidad de ocupar los GPIOs de la placa.

Ilustracion 3.1.6. Sensor LIDAR (Láser Omnidireccional) (imagen original obtenida de: `https://www.adafruit.com/product/4010`.)

Las características técnicas del RPLIDAR A1, según la hoja de datos técnicos [62] son:

- Rango de barrido de 360 grados.
- Rango de captura de 6m (para objetos blancos) y 12m (objetos de colores diversos).
- Frecuencia de escaneo de 5.5hz a 10hz.
- Comunicación serial por puerto USB.

Para el diseño de ASTEC, el sensor RPLIDAR A1 se pretende sea el sensor que proporcione la mayor cantidad de información para procesamiento, ya que con la información de las distancias a los objetos se busca establecer las estrategias de exploración, trazado de ruta, localización y mapeo, estos sensores (en aplicaciones de robots móviles) usualmente es colocado en el centro del robot, en una zona donde no existan elementos mecánicos propios del robot móvil que puedan interferir con los procesos de medición del sensor láser.

Ilustracion 3.1.7. Camara WEB de 5mpx con base sujetadora (imagen original obtenida de: `https://www.amazon.com.mx/Logitech-C170-Webcam-Pixeles-Recortar/dp/B009E47ZDI.`)

Como último elemento del "módulo de sensores" se plantea el uso de una sensor de visión, este sensor sera una "Cámara WEB de 5Mpx" (ilustración (3.1.7)), esta cámara tendría la finalidad de proporcionar información gráfica respecto a las zonas donde se presuma la localización de una víctima, esta información no sera procesada directamente por ASTEC, ya que las tareas de procesamiento de imágenes usualmente demandan una gran cantidad de tiempo y recursos, así que para propósitos de este diseño, la cámara WEB solo obtendrá imágenes bajo ciertos protocolos que se explicaran en las siguientes subsecciones, y estas imágenes serán enviadas a través del módulo de comunicación a un usuario humano, para poder ser interpretadas y ayudar a la efectividad de la búsqueda de víctimas en la zona de despliegue.

Las características mas importantes de la cámara WEB son:

- Sistema de monofocal.
- Resolución de 5Mpx.
- Comunicación serial vía USB.
- Base sujetadora incorporada.

3.2. Diseño

En esta sección se abordará como los elementos de hardware de ASTEC serán integrados en el diseño. Se explicara de manera general el funcionamiento y comportamiento del hardware y el software de este diseño, posterior a ello, se hablara de la interacción de los componentes y protocolos de ASTEC así como su programa

asociado a las tareas que se plantean para el diseño, asiendo énfasis que este trabajo de tesis, plantea una aproximación teórica y simulada de un robot para búsqueda y rescate autónomo, que pueda ser llevado en un proyecto a futuro, a una etapa de implementación física.

3.2.1. Diagramas de ASTEC

Debido a que este trabajo de tesis se aborda desde el punto de vista del diseño, programación y simulación de un robot de búsqueda y rescate, donde no se contempla la implementación física del prototipo, es necesario plantear las ideas fundamentales que se desean en el funcionamiento del robot, para ello, el uso de diagramas para explicar los elementos, su interacción, comunicación y procesado de información resulta fundamental en este trabajo de tesis, los diagramas que se explicaran y expondrán en esta subsección de la tesis están basados en los componentes elegidos para el diseño, así como las estrategias y características básicas necesarias para implementar un robot de búsqueda y rescate, y dichas características fueron cubiertas en el marco teórico de esta tesis.

Como primer diagrama de "ensamble" para ASTEC se tiene la conexión entre el chasis, la alimentación (batería) y la unidad de control.

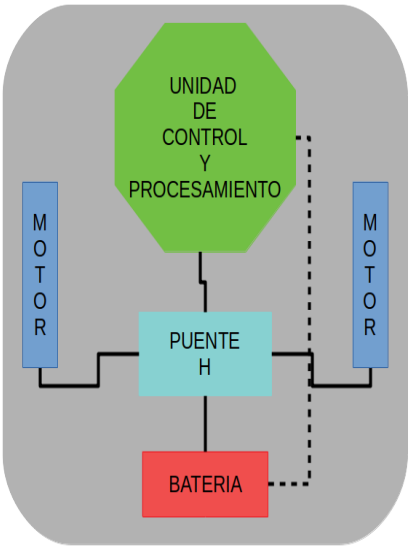


Diagrama 3.2.1.1. Diagrama de conexión de chasis, batería, unidad de control y procesamiento.

En el diagrama 3.2.1.1 se integran los elementos básicos del chasis del diseño (color azul), con la unidad de control y procesamiento (color verde), así como la batería (color rojo), debido a que el chasis no cuenta con un método de control para los motores (color azul) de los rodamientos por oruga izquierdo y derecho, la idea de contemplar un "puente H" (color azul cielo) para alimentar y controlar los motores resulta más adecuada, ya que

si bien el control y la alimentación pueden generarse directamente desde la Raspberry Pi 3 B, esta placa recomienda no drenar mucha corriente de los puertos, aproximadamente 50mA máximo, dado que los motores tienen un consumo (con carga) de 250mA.

Es necesario proteger la unidad de control y solo controlar el puente H a través de PWM (Pulse Width Modulation), este puente H puede también usarse para alimentar a la Raspberry, sin embargo, no se ve tan necesario debido a que las salidas de la alimentación (PowerBank) son de tipo USB, lo cual permite fácilmente conectarla a la unidad de control con un cable conversión de USB a Micro USB.

En el diagrama 3.2.1.2 se muestra la conexión entre los sensores (color azul) con la unidad de control y procesamiento (color verde), también se muestra la conexión al modulo de comunicación (color violeta).

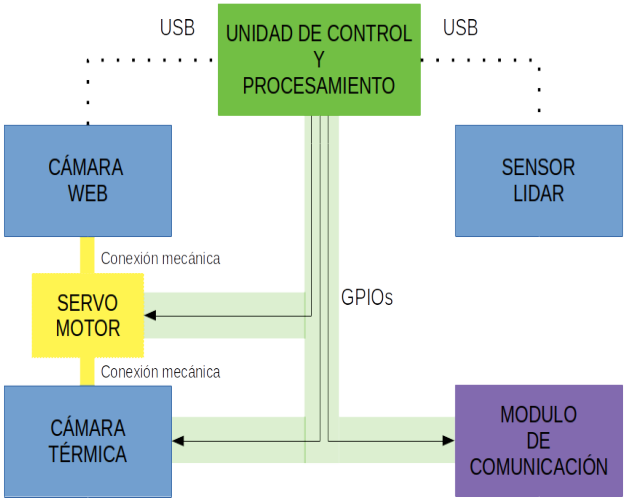


Diagrama 3.2.1.2. Diagrama de conexión de sensores, comunicación y control.

También, se muestra (en el diagrama 3.2.1.2) la conexión (mecánica) entre la cámara térmica y la cámara web a un servo motor (color amarillo), este ensamble entre las cámaras y el servo se realiza para poder realizar un barrido, al igual que en el sensor LIDAR, para tomar muestras de un área alrededor del robot, estas mediciones se deben relacionar con las mediciones del sensor LIDAR para establecer la ubicación aproximada de las lectoras térmicas y las imágenes obtenidas.

Las conexiones de los sensores y el módulo de comunicación hacia la unidad de control y procesamiento se dan de la siguiente manera:

- Conexión I2C entre “cámara térmica” y unidad de control y procesamiento (UCP).
- Conexión serial vía puerto USB entre “sensor LIDAR” y UCP.
- Conexión serial vía puerto USB entre “cámara WEB” y UCP.

- Conexión por SPI entre “modulo de comunicación” y UCP.
- Conexión por GPIOs generando PWM (propósito general) entre “servo motor” y UCP.

En la ilustración (3.2.1.1) se plantea un diseño tentativo de ensamble para la cámara térmica (AMG8833) y la cámara WEB al servo motor, el servo motor permite la rotación de las cámaras en diferentes ángulos, para poder obtener una mayor área de cobertura de lecturas, tanto térmicas como en imágenes, la imágenes que debe tomar la cámara WEB solo se darán si se cumple un criterio establecido entre la cámara térmica y la UCP, es decir, que si se obtiene un arreglo de lecturas térmicas que indique la presencia de una posible víctima, la cámara WEB se activara para tomar una imagen de dichas lecturas.

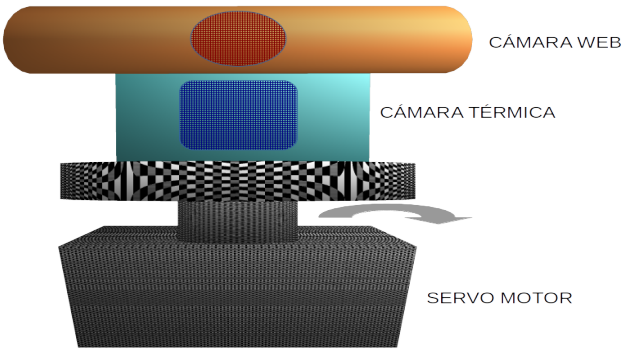


Ilustración 3.2.1.1. Ensamble de cámaras (WEB y térmica) al servo motor.

Tomando como base el chasis propuesto para el diseño de ASTEC, se realizó el siguiente un ensamble ilustrativo:

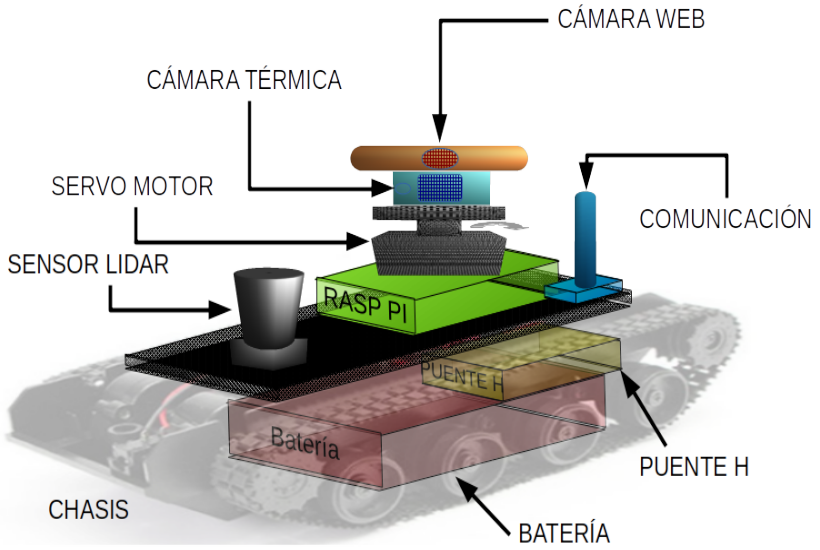


Ilustración 3.2.1.2. Ensamble completo de ASTEC.

En la ilustración 3.2.1.2 se muestra el ensamble (ilustrativo) completo del diseño ASTEC, donde se contempla la UCP (color verde), el módulo de comunicación (color azul, lado derecho), la cámara WEB (naranja) y la cámara térmica (azul cielo) con su integración al servo motor (color gris), el sensor LIDAR (gris metálico), la batería (color rojo) que se coloca al interior del chasis donde se colocaría en lugar del compartimento para 4 baterías AA del diseño original del chasis, y por último el puente H que estaría colocado al interior del chasis, en un espacio que existe entre los motores y el compartimento de baterías. En esta distribución de los elementos también se puede intercambiar la colocación del sensor LIDAR y las cámaras (junto con el servo), con la finalidad de proporcionar una zona de mayor libertad de medición para el sensor láser omnidireccional, la colocación del sensor LIDAR y las cámaras resulta de vital importancia, ya que ambos sensores (hablando del LIDAR y las dos cámaras), necesitan tener acceso de 360 grados de "visión", para con ello evitar "puntos ciegos" en los procesos de obtención de distancias, arreglos térmicos o imágenes para ASTEC.

Una vez teniendo la noción de los elementos que conformaran el diseño de ASTEC, su mecanismo de funcionamiento (de manera general) y su distribución, se puede abordar las siguientes subsecciones de la tesis, donde se explicaran los algoritmos y su interacción con el hardware de ASTEC, y como se pretende que ASTEC pueda desempeñar en tareas de búsqueda y localización.

Capítulo 4

Algoritmos y Programación

En esta sección se abordaran los aspectos relacionados con los algoritmos y la programación de ASTEC, estos algoritmos están basados en las características de los robots de búsqueda y rescate explicadas en el marco teórico, donde los modelos y algoritmos planteados de basarán en modelos de dos dimensiones.

Las tareas que debe realizar el diseño ASTEC se pueden dividir en los siguientes problemas:

- Algoritmo de Exploración.
- Algoritmo de Trazado de Ruta.
- Algoritmo de Localización y Mapeo.
- Captura de imágenes.
- Algoritmo para captura de espectro de calor.

4.1. Algoritmo de exploración y trazado de ruta: QPA*

En el caso del "Algoritmo de Exploración" y el "Algoritmo de Trazado de Ruta", se puede plantear en términos de un solo algoritmo, este algoritmo en particular busca usar estrategias investigadas en el marco teórico, con la finalidad de explotar las características de varios enfoques para resolver los problemas de exploración y trazado de ruta. Para el caso de estos algoritmos el enfoque que se implemento es de tipo determinista, donde el robot ASTEC es pensado como "punto masa", y el mapa es expresado usando una matriz de exploración de nodos, donde cada nodo puede o no ser un estado posible para ASTEC, esta matriz funciona como una representación del mapa "real" en dos dimensiones usando la información de los sensores del diseño ASTEC.

Este algoritmo es un diseño original del autor de la tesis, y recibe el nombre de QPA*, el nombre de QPA se da por las estrategias que el algoritmo implementa, la "Q" radica en la implementación de una estrategia llamada "Quadrant Grid Search" [1] (búsqueda por red de cuadrantes) que es el kernel de los procesos de exploración del algoritmo QPA*, la "P" se da por el enfoque de "Potencial Field Algorithm" (algoritmo de campos de potencial) este enfoque realiza el establecimiento de áreas de no acceso para ASTEC, donde se aclara que esta parte del algoritmo (que corresponde al algoritmo de campos de potencial) no implementa las ecuaciones propias de este enfoque, pero utiliza un principio similar para casos de ocupación de mallas binarias; por último se tiene la parte de "A*" que se coloca por el "A* (star) Algorithm" (algoritmo de A estrella), esta sección del programa se encarga de establecer el trazado de ruta tentativo para ASTEC, donde se ocupa una versión modificada del algoritmo A* hecha por el autor de este trabajo, donde la modificación establecida se buscó con la finalidad de mejorar la interacción con la etapa de exploración del algoritmo QPA*.

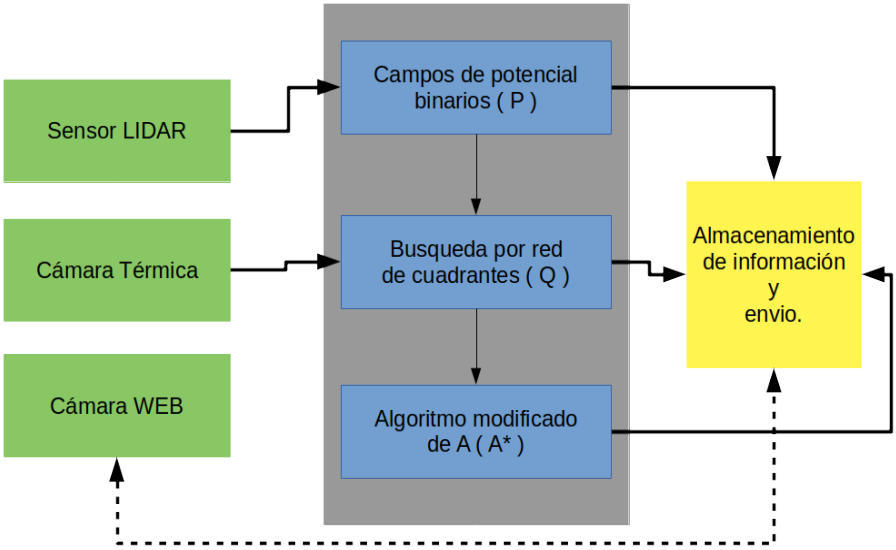


Diagrama 4.1.1. Funcionamiento general del algoritmo QPA*.

En el diagrama (4.1.1) se establece el funcionamiento del algoritmo QPA*, donde en el lado izquierdo de color verde, se tienen los sensores que necesita el algoritmo QPA*, como primera instancia el sensor LIDAR envía la información a la etapa de los "campos de potencial binarios", en esta etapa se generan las "zonas restringidas" y el mapa de exploración, después se llega a la etapa de "búsqueda por cuadrantes", usando la información que obtiene la cámara térmica en esta etapa se establece el protocolo de exploración (o búsqueda) o en caso de detectarse lecturas térmicas de una posible víctima, se estima una posición que se envía a la ultima etapa de QPA*, en la última

[1] Termino acuñado por el autor de la tesis, que representa la estrategia le la heurística de búsqueda a través de una sub-división del mapa de exploración.

etapa se accesa al algoritmo de trazado de ruta usando el "algoritmo modificado de A*" donde se realiza la ruta tentativa que posteriormente será enviada al "algoritmo de localización y mapeo". Cada etapa de QPA* genera información que es almacenada para su posterior envió (si es necesario), además, se explica que la cámara WEB se activa en el cumplimiento de una condición dada por la información de la cámara térmica, en caso de cumplirse dicha condición la cámara WEB toma un registro de imagen de esa zona y es almacenada para su posterior envió.

Teniendo la idea general del algoritmo QPA*, se explicará de manera más profunda el funcionamiento y la programación más relevante del algoritmo.

El protocolo de exploración del algoritmo QPA* se puede subdividir en dos casos. Estos casos se eligen en relación a las lecturas obtenidas por la cámara térmica, en un primer caso se obtiene una serie de lecturas en i y j (de la matriz generada por la cámara térmica) que no indican un rango de temperatura preestablecido (relacionado con captación de temperatura corporal humana), si este caso se cumple, el algoritmo cae en el protocolo de exploración (o búsqueda) en donde no se conoce la ubicación de alguna posible víctima, en este caso el algoritmo genera una meta "secundaria" de manera "pseudo-aleatoria", este protocolo tiene como finalidad explorar la zona de despliegue. El segundo caso es cuando la cámara térmica indica la presencia de una posible víctima, si este caso se cumple, se realiza una estimación de la zona donde pueda estar la posible víctima, se realiza un respaldo de la información de la cámara térmica, se realiza un registro en imagen por la cámara WEB y se realizan paquetes de información referentes al mapa y localización de objetos explorados hasta el momento por el robot ASTEC.

Dado que el trabajo de tesis se basa en la simulación de estos algoritmos, es necesario generar procesos que permitan "simular" entornos de zonas de despliegue que permitan probar estos algoritmos, es por ello que existe una etapa que sera abordada a mayor profundidad en el capítulo de Resultados, Simulación y Conclusiones y en el Apéndice A, este tema se refiere a un programa para la "generación de mapas aleatorios", estos mapas aleatorios tienen la finalidad de simular las lecturas obtenidas del sensor LIDAR.

La generación de mapas aleatorios envía la información de las ubicaciones de obstáculos en un plano, denotadas por $M_{osb} = (M^x_{obs}, M^y_{obs})$, estas ubicaciones son dadas en coordenadas (x, y), con estas ubicaciones se procede a aplicar la estrategia de los "campos de potencial binarios", esta estrategia se expresa en la ecuación (4.1.1) para casos de desplazamiento positivo, y para casos de desplazamientos negativos en la ecuación (4.1.2).

(4.1.1)

$$c_p(M_{obs}) = \begin{bmatrix} c^x_p \\ c^y_p \end{bmatrix} = \begin{bmatrix} M^{x+1}_{obs}, M^{x+2}_{obs}, ..., M^{x+n}_{obs} \\ M^{y+1}_{obs}, M^{y+2}_{obs}, ..., M^{y+n}_{obs} \end{bmatrix}$$

(4.1.2)

$$c_p(M_{obs}) = \begin{bmatrix} c_p^x \\ c_p^y \end{bmatrix} = \begin{bmatrix} M_{obs}^{x-1}, M_{obs}^{x-2}, ..., M_{obs}^{x-n} \\ M_{obs}^{y-1}, M_{obs}^{y-2}, ..., M_{obs}^{y-n} \end{bmatrix}$$

Donde:

- $c_p()$ representa a la aplicación de la función que genera los campos de potencial binarios para cada obstáculo detectado M_{osb}.
- M_{obs} representa a los obstáculos o objetos que el sensor LIDAR registró, donde pueden encontrarse elementos arquitectónicos (paredes, pilares, escaleras, etc), muebles de oficina, víctimas, etc.
- (c_p^x, c_p^y) representan a los nodos que serán contemplados como un campo de potencial binario, en x y y respectivamente.
- n es una constante establecida para la generación de los mapas, esta constante tiene una relación directa con las dimensiones del robot, ya que dadas las dimensiones del robot se busca establecer zonas de no acceso para el robot debido a sus dimensiones físicas (para el caso de ASTEC) $n = 10$.

El generador de mapas aleatorio, además de generar un mapa con obstáculos en el, también se encarga de recorrer el mapa generado al primer cuadrante en un plano cartesiano, este cuadrante establece que todos sus elementos son positivos, es decir, que el cuadrante donde operan los campos de potencial binarios es de tipo $(+,+)$, con ello, los campos artificiales generados al rededor de los obstáculos del mapa siempre se encontraran en coordenadas positivas para (x, y), la aplicación ecuaciones (4.1.1) y (4.1.2) se expresa en la siguiente ilustración.

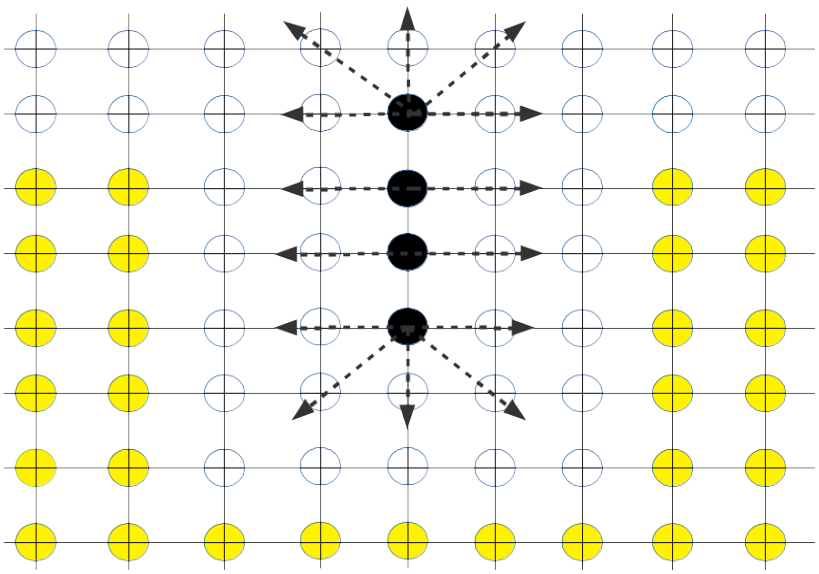


Ilustración 4.1.1. Ejemplo de generación de campos de potencial binarios.

En la ilustración (4.1.1) se establece un objeto (color negro), un campo artificial alrededor del objeto (nodos sin color) y una zona de posible exploración (en color amarillo).

En general, solo se contemplan dos condiciones fundamentales para la aplicación del campo artificial, la primer condición es que tomando el elemento (M^x_{obs}, M^y_{obs}) al aplicar el desplazamiento positivo $([x+1, x+2, ..., x+n], [y+1, y+2, ..., y+n])$ o desplazamiento negativo $([x-1, x-2, ..., x-n], [y-1, y-2, ..., y-n])$ la posición resultante (del nodo) este dentro de las dimensiones del mapa $(dimesion\_n - 1)$ generado aleatoriamente, ya que la $dimension\_n$ es tomada como la frontera del mapa, no es necesario generar campos artificiales fuera de esta frontera. La segunda condición se expresa en la siguiente ecuación:

(4.1.3)
$$\sum_{0}^{i=obs} (x'_i, y'_i) \notin \sum_{0}^{j=obs} (x_j, y_j)$$

Donde:

- (x'_i, y'_i) corresponde al arreglo de coordenadas de elementos posterior al corrimiento (o aplicación del campo artificial) positivo o negativo.
- (x_j, y_j) corresponde al arreglo de coordenadas de elementos originales, es decir, las coordenadas de los obstáculos generados por el "generados de mapas aleatorio".

La ecuación (4.1.3) establece que para poder generar un nodo de campo artificial, las coordenadas del nodo no pueden pertenecer a las coordenadas de un obstáculo generado por el generador de mapas aleatorio, es decir, que para poder ser un nodo de campo potencial binario, debe estar dentro de los limites de $(dimension\_n - 1)$ y además no estar previamente ocupado (ocupado se refiere a que no exista un obstáculo u objeto en esa coordenada) por un obstáculo de la matriz de mapa generada.

La aplicación del "campo de potencial binario" se muestra en el siguiente algoritmo, donde se escribió la sección más relevante del funcionamiento de los campos artificiales que genera el algoritmo QPA*, este algoritmo contiene comentarios para facilitar el entendimiento del mismo.

```
1 # CAMPOS DE POTENCIAL BINARIO
2
3 # Dado que cada elemento de la matriz "coordenadas_objetos" es
4 # un tuple es necesario iterar en cada elemento correspondiente
5 # a esta lista.
6
7 for i in range(len(coordenadas_objetos)):
8
9     # La variable "frontera" se refiere al valor maximo
```

```
10      # de propagacion del campo artificial de los objetos,
11      # esta variable se basa en las dimensiones fisicas
12      # del robot, donde ya que el algoritmo de trazado de
13      # ruta ( A* ) genera la ruta contemplando al robot
14      # como un "punto masa", sin embargo, esto no
15      # representa la realidad, entonces con este metodo
16      # podemos generar un "campo" en funcion a los
17      # objetos detectados, donde el algoritmo de busqueda
18      # no puede accesar.
19      frontera = 10
20
21      # Coordenadas de los objetos
22      y, x = coordenadas_objetos[i]
23
24      # Cada elemento que corresponda a un obstaculo[y][x]
25      # se le asigna "valor_objeto", un valor dado para
26      # identificar elementos especificos que
27      # corresponden a obstaculos en el mapa.
28      obstaculos[y][x] = valor_objeto
29
30      # Doble ciclo for para recorrer los elementos en
31      # "j" e "i" de la matriz "coordenadas_objetos".
32      for a in range(frontera):
33          for b in range(frontera):
34
35              # Valores de movimientos posibles para la
36              # generacion del campo artificial de los
37              # objetos.
38              v_y_pos = y + a
39              v_x_pos = x + b
40              v_y_neg = y - a
41              v_x_neg = x - b
42
43              # Campo en y (positiva)
44              if v_y_pos <= (dimension_n-1) and not \
45                  obstaculos[v_y_pos][x] == valor_objeto:
46                   obstaculos[v_y_pos][x] = valor_objeto
47
48              # Campo en y (negativa)
49              if 0 <= v_y_neg <= (dimension_n-1) and not \
50                  obstaculos[v_y_neg][x] == valor_objeto:
51                   obstaculos[v_y_neg][x] = valor_objeto
52
53              # Campo en x (positiva)
54              if v_x_pos <= (dimension_n-1) and not \
55                  obstaculos[y][v_x_pos] == valor_objeto:
56                   obstaculos[y][v_x_pos] = valor_objeto
57
58              # Campo en x (negativa)
59              if 0 <= v_x_neg <= (dimension_n-1) and not\
60                  obstaculos[y][v_x_neg] == valor_objeto:
61                   obstaculos[y][v_x_neg] = valor_objeto
62
63                              #
64                              #
65                              #
66                              #
```

Algoritmo 4.1.1.a. Implementación en Python 3 de los campos de potencial binarios.

Como primer paso en la generación de los campos en la línea (6) del algoritmo (4.1.1.a) se desarrolla un ciclo for sobre todos los "pares" de coordenadas que representan la posición de los objetos, después en la línea (18) se establece la "frontera" del campo, es decir, que tanto se pretende propagar el campo de potencial binario sobre los objetos del mapa. En las líneas (31) y (32) se realizan dos ciclos for anidados (matriz) para recorrer las posiciones de los objetos en x y y, el desplazamiento del campo se puede traducir en cuatro variables de propagación $(v\_y\_pos, v\_x\_pos, v\_y\_neg, v\_x\_neg)$, estas variables generan que en las iteraciones de los ciclos for exista un desplazamiento en cuatro direcciones posibles, sin embargo, a partir de la línea (43) a (85) se tienen las condiciones para asegurar que los desplazamientos del campo cumplan con las condiciones señaladas de estar dentro de los limites de $(dimension\_n - 1)$ y cumplir con el criterio de exclusión de la ecuación (4.1.3).

Una vez terminados los ciclos for anidados, para poder simular un inicio del robot dentro del mapa de exploración, en el archivo de los campos de potencial binario, se establece una posición pseudo-aleatoria de la posición de inicio para el robot (este criterio será explicado con mayor profundidad en el capítulo de simulación).

Posterior a la generación del "inicio pseudo-aleatorio", el programa termina con la ejecución de la función plasmada en el algoritmo (4.1.1.b).

```
1  # El programa regresa a la funcion pricipal,
2  # las coordenadas iniciales en (x, y), las
3  # coordenadas de los objetos generados por
4  # el "generador de mapas aleatorio", y los
5  # "obstaculos" que corresponden a los
6  # elementos generados por los campos de
7  # potencial binario.
8
9  return (coordenada_inicial_y, coordenada_inicial_x), \
10        coordenadas_objetos, obstaculos
```

Algoritmo 4.1.1.b. Implementación en Python 3 de los campos de potencial binarios.

La segunda parte del algoritmo QPA*, como lo muestra el diagrama (4.1.1), es la "búsqueda por red de cuadrantes (Q)", esta parte se encarga del protocolo de exploración y la interpretación de los datos obtenidos por la cámara térmica. La búsqueda por cuadrantes recopila los datos de la matriz que genera la cámara térmica, estos datos establecen los dos casos mencionados con anterioridad, el primer caso es cuando el indicador relaciona la existencia de una posible víctima, cuando las temperaturas registradas por la cámara térmica se encuentran por encima de un nivel establecido, si este caso se cumple, el protocolo de búsqueda per se, queda anulado, se establece (en esta etapa del desarrollo) la posición de la posible víctima y se envía esta

posición "hipotética" al algoritmo modificado de A*, el segundo caso es cuando la cámara térmica no indica valores térmicos por encima del nivel establecido, lo cual activa al protocolo de búsqueda. El protocolo de búsqueda establece una subdivisión del mapa de exploración a través de nueve cuadrantes de exploración $(0, 1, 2, ..., 8)$, una vez que estos cuadrantes se establecen, el protocolo elige de manera "pseudo-aleatoria", se elige una posición dentro del cuadrante (de manera pseudo-aleatoria igualmente) y se pasa esa posición al algoritmo modificado de A*, este proceso se repite de manera iterativa hasta la localización de una posible víctima (usando los datos de la cámara térmica).

El primer paso de la etapa de "búsqueda por red de cuadrantes", establece la aplicación de la siguiente ecuación:

(4.1.4)

$$\begin{cases} \text{a} = \text{if } (t^x_{i:n}, t^y_{j:n}) > valor\_gatillo \\ \text{b} = \text{if } (t^x_{i:n}, t^y_{j:n}) < valor\_gatillo \end{cases}$$

Donde:

- $(t^x_{i:n}, t^y_{j:n})$ son las lecturas de la matriz de temperatura, donde las lecturas sobre i se relacionan con la coordenada x, y las lecturas sobre j se relacionan con la coordenada y.
- n corresponde a la dimensión de la matriz de la cámara térmica, que en el caso de la cámara térmica (AMG8833) propuesta para el diseño ASTEC, es de dimensión 8 $(8x8)$.
- $valor\_gatillo$ se relaciona con el valor de nivel de temperatura, que establecerá o no el protocolo de búsqueda por red de cuadrantes, el valor establecido de esta constante sera de 23.

La ecuación (4.1.4) establece dos posibilidades, la posibilidad "a" cuando la temperatura se encuentra por encima del valor gatillo, si este es el caso se anula el protocolo de búsqueda, en caso de que la "temperatura" se encuentre menor al $valor\_gatillo$ entonces se continua con el protocolo de búsqueda.

La "búsqueda por red de cuadrantes" esta basada en una heurística planteada por el autor de la tesis, los pasos que sigue esta heurística se en listan a continuación.

(1) Se adquiere la posición pseudo-aleatoria inicial, generada por los campos de potencial binario.

(2) Se subdivide el mapa de búsqueda en nueve cuadrantes, como se muestra en la ilustración (4.1.2).

(3) Se establece una condición dual de acción, esta condición establece que si es la primera iteración de búsqueda del protocolo, el cuadrante elegido sera el cuadrante central (cuadrante 4), en caso de que el centro $c(x, y) = (\frac{dim_i}{2}, \frac{dim_j}{2})$ este libre, esa será la coordenada de meta secundaria "s", en caso de que no este libre (existe algún obstáculo o campo artificial), se elegirá un una coordenada dentro del cuadrante central. El otro caso es cuando no es la primera iteración de búsqueda, entonces se genera una selección pseudo-aleatoria de un cuadrante (usando la función *randint*() de Python 3).

(4) Posteriormente se procede a elegir de manera pseudo-aleatoria un nodo (dentro del cuadrante seleccionado del paso 3) $q_{pos}(x_{cuadrante}, y_{cuadrante})$ dentro de ese cuadrante, se usa la función *randint*().

(5) Cuando la meta secundaria $q_{pos}(x_{cuadrante}, y_{cuadrante})$ es alcanzada (usando el algoritmo de trazado de ruta), este punto meta se convierte en el punto de inicio para la siguiente iteración.

(6) Se actualizan los nodos visitados, los nodos visitados establecen zonas donde no pueden generarse metas aleatorias, con la finalidad de no re-visitar zonas ya exploradas, para la primera iteración, la matriz de "nodos visitados" contiene solo *ceros*.

(7) Se repiten los pasos (3) a (6), hasta que la cámara térmica indique una lectura sobre una posible víctima.

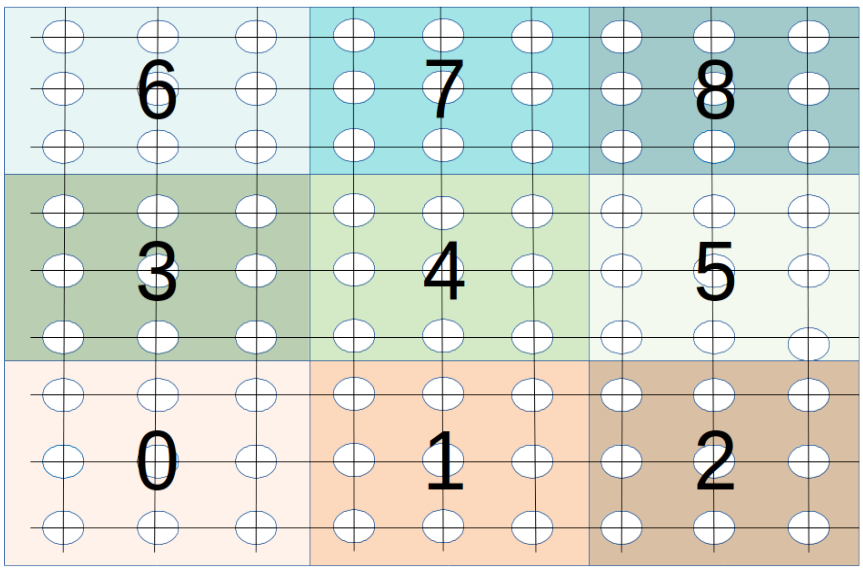


Ilustración 4.1.2. Subdivisión en nueve cuadrantes.

El programa implementado de la etapa de búsqueda por malla de cuadrantes, es el siguiente:

```
# BUSQUEDA POR MALLA DE CUADRANTES

def busca_victima(protocolo_inicio, cuadrante_anterior, \
                  dimension, busca, visitados, obstaculos):

    # Restringe la zona de exploracion
    cuadrantes = obstaculos.shape
    valor_cuadrante = int(dimension / 3)
    lista_cuadrantes = [0,1,2,3,4,5,6,7,8]

    # La variable de "nivel_exploracion" se usara para
    # modificadiones futuras del protocolo de "busqueda
    # por cuadrantes" de QPA*.
    nivel_exploracion = 0
    valor_nivel = 0
    empieza_arbol = True

    # El parametro pasado a la funcion "busca_victima"
    # llamado "visitados" se relaciona internamente para
    # la seccion de "busqueda por cuadrantes" con la
    # variable "nodos_visitados" que es un array copia
    # de los elementos presentes en "visitados".
    nodos_visitados = visitados


    # Se establece inicialemnte el tipo de meta como "s"
    # (secundario).
    tipo_meta = 's'

    # Se adquieren los valores de la camara termica,
    # para establecer si existe una posible victima o no.
    # Para indicar si existe una posible victima, los
    # valores del array en "j" e "i" de la camara,
    # tienen que tener elementos mayores a "23".

    dato_j_camara_termica = input("Ingresa valor para" + \
                                  "j de camara termica: ")
    dato_i_camara_termica =\
                          input("Ingresa valor para" + \
                                "i de camara termica: ")

    dato_j_camara_termica = int(dato_j_camara_termica)
    dato_i_camara_termica = int(dato_i_camara_termica)

    if dato_j_camara_termica > 23 and \
       dato_i_camara_termica > 23:

        # En caso de que la camara termica indique la
        # ubicacion de una posible victima, se ingresan
        # las "coordenadas de la victima", estas
        # coordenadas se podrian generar usando un metodo
        # para relacionar las lecturas de la camara termica
        # con las lecturas del sensor LIDAR.

        ubicacion_y = input("Ingresa" + \
                            "coordenada Y de la victima: ")
        ubicacion_x = input("Ingresa" + \
```

```
58                              "coordenada X de la victima: ")
59
60          ubicacion_y = int(ubicacion_y)
61          ubicacion_x = int(ubicacion_x)
62
63          # Se establece un tipo de meta "p" (prioritario),
64          # y se regresan los parametros de la ubicacion
65          # de la "victima" en (x, y), tipo de meta,
66          # nivel de exploracion y el protocolo de inicio.
67          tipo_meta = 'p'
68
69          return (ubicacion_y, ubicacion_x, tipo_meta,\
70                  nivel_exploracion, protocolo_inicio)
71
72      # La variable "porcentaje_de_exploracion" se usara
73      # para modificaciones futuras del protocolo de
74      # "busqueda por cuadrantes", para esta tesis esta
75      # variable no sera abordada.
76      porcentaje_de_exploracion = []
77
78      # Se realiza la eleccion aleatoria de algun cuadrante,
79      # de la "lista_de_cuadrantes".
80      cuadrante_explorar_lista = sample(lista_cuadrantes,1)
81
82      # El "cuadrante_a_explorar" es el cuadrante tomado de
83      # la lista de cuadrantes.
84      cuadrante_a_explorar = cuadrante_explorar_lista[0]
85
86
87      # Estas condiciones se relacionan con la repeticion
88      # o no repeticion del cuadrante que sera explorado,
89      # estas condiciones se relacionan con el nivel de
90      # exploracion, que no sera abordado en este trabajo
91      # de tesis.
92      if cuadrante_a_explorar == cuadrante_anterior:
93
94          valor_nivel = 0
95
96      if cuadrante_a_explorar != cuadrante_anterior:
97
98          valor_nivel = 1
99
100     cuadrante_anterior = cuadrante_a_explorar
101
102
103     # Inicio de la heuristica de busqueda por cuadrantes,
104     # ir al cuadrante central ("4"), despues se
105     # establecen condiciones similares para el resto de
106     # los cuadrantes posibles para explorar.
107     if (protocolo_inicio == True) or \
108        (cuadrante_a_explorar == 4):
109
110         # Generacion de coordenada al centro del mapa,
111         # o generacion de meta pseudo-aleatoria en el
112         # cuadrante central.
113
114         while True:
```

```
115
116             # Condicion si el centro del mapa se
117             # encuentra libre, no posee obstaculos o
118             # campos de potencial binario asociados a
119             # ese coordenada.
120             if (obstaculos[int(dimension/2)]\
121                 [int(dimension/2)] != 255):
122
123                 meta_cuadrante_cuatro_y = int(dimension/2)
124                 meta_cuadrante_cuatro_x = int(dimension/2)
125                 ubicacion_y = meta_cuadrante_cuatro_y
126                 ubicacion_x = meta_cuadrante_cuatro_x
127                 empieza_arbol = False
128                 protocolo_inicio = False
129                 nivel_exploracion = 0
130                 break
131
132             # En caso de que el centro se encuentre
133             # "ocupado", se genera una meta secundaria
134             # pseudo-aleatoria en el cuadrante 4.
135             else:
136
137                 meta_cuadrante_cuatro_y = randint\
138                                           (valor_cuadrante,\
139                                            2*valor_cuadrante)
140                 meta_cuadrante_cuatro_x = randint\
141                                           (valor_cuadrante,\
142                                            2*valor_cuadrante)
143
144                 if (obstaculos[meta_cuadrante_cuatro_y]\
145                     [meta_cuadrante_cuatro_x] != 255) and \
146                     (nodos_visitados[meta_cuadrante_cuatro_y]\
147                      [meta_cuadrante_cuatro_x] == 0):
148
149                     ubicacion_y = meta_cuadrante_cuatro_y
150                     ubicacion_x = meta_cuadrante_cuatro_x
151                     empieza_arbol = False
152                     protocolo_inicio = False
153                     nivel_exploracion = 0
154                     break
155
156     # Se repiten los pasos para los cuadrantes
157     # (0, 1, 2, 3, 5, 6, 7, 8), mientras el indicador
158     # de una posible victima no sea activado.
159
160     # Se envian los parametros de ubicacion en (x, y)
161     # que es la meta a la que se desea llegar en mapa,
162     # tipo_meta (puede ser primaria "p" o secundaria
163     # "s"), protocolo_inicio que sirve para saber si
164     # es la primera iteracion de exploracion de QPA*.
165     return (ubicacion_y, ubicacion_x, tipo_meta,\
166             nivel_exploracion, protocolo_inicio)
```

Algoritmo 4.1.2. Implementación en Python 3 de la etapa de búsqueda por malla de cuadrantes.

Como etapa final del algoritmo QPA*, se tiene la generación del trazado de ruta,

como se explico, el algoritmo QPA* es un algoritmo de enfoque determinista, donde las etapas previas se encargan de la generación de los campos artificiales (campos de potencial binario) alrededor de los objetos u obstáculos del mapa, posterior a ello se interpretan los datos obtenidos por la cámara térmica para determinar si es necesario o no, aplicar el protocolo de búsqueda (exploración) donde se ha de desplegar el robot, en esta etapa de exploración, en caso de que la cámara térmica obtenga lecturas térmicas por encima de un valor establecido, llama de manera directa al algoritmo de trazado de ruta, en caso contrario, genera una meta secundaria pseudo-aleatoria y esta meta es pasada como parámetro al algoritmo de trazado de ruta, para generar la ruta "tentativa". La ruta "tentativa" se expresa así debido a que el enfoque del modelo determinista de QPA* no contempla los errores aleatorios (o incertidumbres) asociadas a cualquier sistema robótico móvil, ya sea en su etapa de "trazado de ruta", "exploración", "localización" o "mapeo", esta etapa que corresponde a la ruta que deberá seguir ASTEC, es básicamente una secuencia de nodos que se prevén como la ruta mínima, ya sea a una meta principal "p" (cuando se cree que esta meta es una posible víctima) o una meta secundaria "s" (generada por el protocolo de exploración o búsqueda).

El trazado de ruta del algoritmo QPA*, se lleva acabo usando un "algoritmo de A* modificado". El "algoritmo de A* clásico" se explico en el marco teórico, el concepto "modificado" del algoritmo A* se da por dos principales razones:

- Utilizando el enfoque que propone *Claus Brenner* en el curso "SLAM Lectures" [50], se utiliza un algoritmo de ordenamiento llamado "ordenamiento de pila" (HEAP Queue Algorithm), esto mejora el proceso de búsqueda planteado en la ecuación (2.2.7).

- La segunda modificación del algoritmo "clásico" de A*, es una propuesta del autor de la tesis, donde se presenta un parámetro de "costo de penalización" a nodos ya visitados, este parámetro se incluye en la ecuación (2.2.6).

La primer modificación generada usando un algoritmo de ordenamiento de pila, es un proceso que establece la o las vertientes propias del algoritmo de A* modificado, este algoritmo de ordenamiento crea un "árbol binario" [2] de nodos mínimos, el algoritmo de ordenamiento de pila busca mejorar la búsqueda del nodo o nodos mínimos del algoritmo de A* clásico. La versión estándar de Python 3, contiene una función llamada "heapq()", esta función aplica un algoritmo de ordenamiento de pila, las pilas son arboles binarios donde cada "nodo padre" tiene un valor menor o igual al de sus hijos, es decir, que los "nodos padre" siempre tendrán los valores más bajos (que son los que se buscan en la aplicación de la ecuación (2.2.7)) del árbol binario.

[2]El concepto de "árbol binario" se encuentra disponible en el Apéndice A.4.

La función de “heapq()” cumple con las siguientes dos ecuaciones, para los propósitos de ordenamiento del algoritmo.

(4.1.5)

$$a[k] \leq a[2 * (k + 1)] \in k$$

(4.1.6)

$$a[k] \leq a[2 * (k + 2)] \in k$$

Donde:

- k corresponde a los “niveles” del árbol binario.
- $a[k]$ se relaciona con el concepto de “nodo padre” o “nodo madre”.

El algoritmo “heapq()” de Python 3, se considera de tipo dinámico, esto se debe a que los nodos que son examinados, son insertados en la posición que cumpla con las ecuaciones (4.1.5) y (4.1.6), esto genera que los nodos y la estructura del árbol cambien en presencia de nuevos datos. Otro aspecto interesante e útil del algoritmo de ordenamiento de pila, es que la raíz (cuando $k = 0$) siempre sera el mínimo.

La segunda estrategia esta relacionada con la modificación del algoritmo de A* clásico, se refiere a la incorporación de un “costo extra”, este costo extra es un parámetro relacionado con los “nodos visitados” que se generan en iteraciones pasadas $(t - n)$ donde n es el numero de iteraciones. Este costo extra tiene como finalidad decrementar la re-exploración de nodos ya visitados, esto se plantea debido a que el protocolo de búsqueda o exploración de ASTEC, solo genera una serie de metas secundarias “s” de manera pseudo-aleatoria en una cierta zona de despliegue (mapa de exploración), si bien, la parte de la “búsqueda por red de cuadrantes” no permite generar metas secundarias en coordenadas que se encuentren dentro de los elementos de la matriz de nodos visitas, esta sección de QPA* no contempla que los “nuevo nodos visitados” se repitan con respecto a los “nodos visitados pasados”. El valor de este parámetro (el costo extra), se elige en usando los costos de las “conexiones” o “movimientos” que conectan uno o más de los nodos previamente visitados p_{vis}^{t-n}, entonces, la modificación de la ecuación (2.2.6) del algoritmo de A* (clásico) se establece de la siguiente manera:

(4.1.7)

$$a_i^t(f) = g_i(f) + d_i(f) + p_{vis}^{t-n}$$

Donde:

- $a_i^t(f)$ es el costo por nodo de frente del algoritmo de A* modificado.

- $g_i(f)$ es el costo generado por el enfoque "codicioso" (función de distancia euclidiana, Manhattan, Minkowski, etc) por nodo de frente del algoritmo de A*.
- $d_i(f)$ es el costo generado por la parte de "Dijkstra" por nodo de frente del algoritmo de A*.
- p_{vis}^{t-n} es el "costo extra" o penalización al internar establecer una ruta tentativa a través de nodos previamente visitados.

El parámetro p_{vis}^{t-n} solo se aplica cuando es activado el protocolo de exploración de la búsqueda por red de cuadrantes, es decir, que si esta etapa de QPA* establece una meta secundaria "s", entonces este parámetro se agrega al costo total de los nodos del vector "frente", en caso de que se pase un indicador que relaciona a una posible víctima, el parámetro de "costo extra" es anulado, y solo se contempla la ecuación (2.2.6) en su forma original.

El programa del trazado de ruta usando el algoritmo de A* modificado, es el siguiente:

```
# ALGORITMO DE A* MODIFICADO

# Algoritmo para trazado de ruta, basado en un algoritmo de A*
# clasico, con modificaciones de ordenamiento (usando
# algoritmo de ordenamiento de pila), y protocolo de busqueda
# con costo extra para nodos previamente visitados.
def a_estrella_robot(ini, m, obstaculos, visitados_anteriores,\
                     activador):

    # Corresponde al vector de "frente" que contiene los
    # elementos que se estan explorando, frente se tiene
    # una estructura de lista con tuples internos, donde
    # como elemento frente[0] se tiene el calculo de la
    # distancia euclidiana, frente[1] con un costo de
    # nodo inicial relativamente bajo, frente[2] almacena
    # las posiciones del nodo explorado (en la primera
    # iteracion, este nodo son las coordenadas de "inicio"),
    # frente[3] se refiere a las posiciones de los nodos
    # minimos, que al final del programa seran el camino
    # minimo.
    frente = [((inicio + distancia(inicio, meta)), 0.001, inicio, None)]

    # Mientras exista elementos en el vector de "frente"
    # se realizara este ciclo.
    while frente:

        # El elemento minimo, se extrae del arbol generado
        # del vector frente.
        minimo = heappop(frente)
        # Se extraen los datos de ese "minimo".
        costo_total, costo, pos, pose_previa = minimo
        # Se extraen las coordenadas en (x, y) de la
        # posicion "pos".
        y, x = pos
```

```
35
36          # Si visitados en la posicion actual "pos" de
37          # (x, y) es mayor que 0, entonces se salta
38          # el resto del codigo y se reinicia con el
39          # while loop.
40          if visitados[int(y),int(x)] > 0:
41              continue
42
43          # Si el nodo fue menor o igual que 0, entonces
44          # ese nodo adquiere el valor de "costo" que
45          # se tenga hasta esa iteracion.
46          visitados[int(y),int(x)] = costo
47
48          # Si pos[0] y pos[1], es decir, (y, x) de la
49          # posicion actual "pos" es igual a la meta[0]
50          # y meta[1], meta(y, x), entonces se rompe
51          # el while loop de "frente".
52          if (pos[0] == meta[0] and pos[1] == meta[1]):
53              break
54
55          # Si la condicion anterior no se cumplio,
56          # entonces se exploran los nodos en relativos
57          # a la posicion actual "pos", usando los
58          # movimientos posibles, y los costos de cada
59          # uno de ellos.
60          for dx, dy, deltacosto in movimientos:
61
62              # Valores de (y, x) de pos.
63              y, x = pos
64              # Nueva posicion en (x, y) usando
65              # los valores de costo de conexion
66              # por movimiento.
67              nueva_x = x + dx
68              nueva_y = y + dy
69
70              # Si el movimiento no es nulo, es decir,
71              # es mayor que cero y esta dentro de los
72              # limites del mapa de exploracion.
73              if 0.0 <= nueva_x < limites[1] or\
74                 0.0 <= nueva_y < limites[0]:
75
76                  # Se genera la nueva posicion,
77                  # usando los nuevos valores de (x, y).
78                  nueva_pose = [nueva_y, nueva_x]
79
80                  ### Modificacion de costo extra a nodos
81                  ### previamente visitados, dependiendo
82                  ### del tipo de meta ###
83
84                  # Anexar un costo de nodos ya visitados,
85                  # con la finalidad de mejorar las zonas
86                  # de exploracion y tratar de no repetir
87                  # rutas a traves de nodos visitados, si
88                  # el tipo de meta es secundaria 's'.
89                  if m[2] == 's':
90                      if visitados_anteriores[int(nueva_y),\
91                                              int(nueva_x)] != 0:
```

```
92                      costo_visitado_anterior = deltacosto
93
94                  # Si existe una meta de tipo principal 'p',
95                  # el costo por nodo_visitado_anterior se
96                  # vuelve "0", con la finalidad de mejorar
97                  # la efectividad de la ruta y no de la
98                  # exploracion, como es el caso cuando la
99                  # meta es secundaria.
100                 if m[2] == 'p':
101                     costo_visitado_anterior = 0
102
103                 # Actualizacion del costo, contemplando el
104                 # costo del camino hasta el momento ("costo"),
105                 # el costo del movimiento ("deltacosto") y
106                 # el costo si dicho nodo a sido visitado con
107                 # anterioridad (y es tambien una ruta a una
108                 # meta secundaria).
109                 nuevo_costo = costo + deltacosto +\
110                               costo_visitado_anterior
111
112                 # Entonces, nuevo_costo_total es la suma
113                 # de nuevo_costo mas el valor calculado por
114                 # la heuristica del Greedy Algorithm (basado
115                 # en distancia euclididana) de la nueva pose
116                 # hacia la meta.
117                 nuevo_costo_total = nuevo_costo +\
118                                     distancia(nueva_pose, meta)
119
120                 if visitados[int(nueva_y), int(nueva_x)]\
121                    == 0 and not obstaculos[int(nueva_y),\
122                                            int(nueva_x)] == 255:
123
124                     # Agregamos el nuevo nodo usando la
125                     # funcion "heappush()" al arbol (vector)
126                     # de frente, con el nuevo costo total,
127                     # nuevo costo (del nodo), nueva pose
128                     # (pose actual), y la posicion anterior
129                     # (nodo minimo de donde viene).
130                     heappush(frente,(nuevo_costo_total,\
131                                      nuevo_costo,\
132                                      nueva_pose, pos))
133
134             # Si la posicion esta fuera de los limites,
135             # se repite el ciclo y se pueba otra posicion,
136             # se prueba otro movimiento.
137             else:
138                 continue
```

Algoritmo 4.1.3. Implementación en Python 3 del algoritmo A* modificado.

La etapa del algoritmo QPA* que se encarga de la exploración y trazado de ruta para el diseño de ASTEC, se generó la función “main()” donde se integran las diferentes etapas de QPA*, en el siguiente algoritmo se colocan solo los elementos mas relevantes de la función “main()”.

```
### Funcion main del algoritmo QPA* ###

if __name__ == '__main__':

    # Se establece la dimension de la zona de depliegue simulada.
    dimension_n = 600

    # Busqueda por cuadrantes
    protocolo_inicio = True
    # "cuadrante_anterior" puede ser cualquier valor entero que
    # no se encuentre listado dentro de los cuadrantes (0, 1,
    # 2, ..., 8).
    cuadrante_anterior = 15

    # Generacion de mapa, objetos y obstaculos
    ini, objetos, obstaculos =\
         generacion_mapa_exploracion(dimension_n)

    # Matriz de exploracion inicial
    activador = True
    limites = obstaculos.shape
    visitados_anteriores = np.zeros(limites,\
                                    dtype=np.float32)

### Inicio del while para exploracion y busqueda de victimas,
### metas principales y secundarias ###

# Ciclo de busqueda mientras la meta 's' se mantenga, meta 's'
# se refiere a una meta "secundaria", la meta 'p' (principal)
# se establece cuando se ha detectado una posible victima,
# mientras esto no ocurra el explorador debe generar metas
# "secundarias" que permitan continuar la exploracion.
    while True:

        # Busca se le asigna un valor de verdadero, para la
        # funcion "busca_victima".
        busca = True

        # Se llama la funcion busca_victima, que genera la
        # "busqueda por red de cuadrantes.
        m = busca_victima(protocolo_inicio,\
                          cuadrante_anterior, dimension_n,\
                          busca, visitados_anteriores,\
                          obstaculos)
        protocolo_inicio = m[4]

        # Se llama la funcion de "trazado de ruta", usando
        # la funcion "a_estrella_robot" que implementa el
        # algoritmo A* modificado.
        camino, visitados = \
                a_estrella_robot(ini, m, obstaculos,\
                                 visitados_anteriores,\
                                 activador)

        # Condicion si es encontrada una victima,
        # la meta tiene un parametro 'p' en m[2], esta
        # condicion se relaciona con las lecturas de la
```

```
58          # camara termica.
59          if m[2] == 'p' :
60
61              # Elementos para la simulacion grafica del
62              # algoritmo QPA*, para deteccion de victima.
63              plt.plot(m[1], m[0], 'ro')
64              for i in range(dimension_n):
65                  for j in range(dimension_n):
66
67                      if visitados[j][i] != 0.0:
68                          y = j
69                          x = i
70                          vis_y_p.append(y)
71                          vis_x_p.append(x)
72                      else:
73                          continue
74              for i in range(len(camino)):
75                  y, x = camino[i]
76                  ruta_y_p.append(y)
77                  ruta_x_p.append(x)
78
79              # Inicio final, es el "ini" de la ultima
80              # iteracion.
81              inicio_final = ini
82
83              # Se generan los comandos de control que seran
84              # usados en el algoritmo de localizacion y
85              # mapeo de ASTEC.
86              generar_rpm_motores(camino)
87
88              # Se termina el while loop.
89              break
90
91          # Tipo de meta "secundaria" 's', se relaciona con
92          # el protocolo de exploracion del algoritmo QPA*.
93          if m[2] == 's':
94
95              # Elementos para simulacion grafica del
96              # algoritmo QPA*, para protocolo de exploracion,
97              # o meta secundaria.
98              for i in range(dimension_n):
99                  for j in range(dimension_n):
100
101                     if visitados[j][i] != 0.0:
102                         y = j
103                         x = i
104                         vis_y_s.append(y)
105                         vis_x_s.append(x)
106                     else:
107                         continue
108             for i in range(len(camino)):
109                 y, x = camino[i]
110                 ruta_y_s.append(y)
111                 ruta_x_s.append(x)
112             plt.scatter(vis_x_s, vis_y_s, color='yellow')
113             inicio_secundario_y.append(ini[1])
114             inicio_secundario_x.append(ini[0])
```

```
115
116         # Se aumenta el contador de iteraciones de
117         # exploracion, el activador se cambia a "false",
118         # la matriz de "visitados_anteriores" toma los
119         # valores de la matriz de visitados generada por
120         # el algoritmo A* modificado.
121         contador_iteraciones += 1
122         activador = False
123         visitados_anteriores = visitados
124
125         # Nuevo inicio corresponde a la meta
126         # secundaria alcanzada.
127         ini = m[0:2]
128
129 ### Fin del while de busqueda a metas principales (metas a
130 ### posibles victimas) y secundarias (metas para explorar
131 ### mapa).
```

Algoritmo 4.1.4. Integración del algoritmo QPA*

En el algoritmo (4.1.4) están colocados los elementos más importantes del la función *main* de QPA*. En la línea 9 del código, se tiene la asignación a la variable "dimension_n", esta variable representa la dimensión de la zona de despliegue para ASTEC, esta variable sirve para generar los limites de las matrices necesarias para implementar el algoritmo de exploración y trazado de ruta, en la lineas 16 y 17 se llama a la primera parte del algoritmo QPA*, donde se genera el mapa pseudo-aleatorio de exploración junto con los campos de potencial binario (artificiales) del mapa y los obstáculos dentro de él, después, a partir de la línea 33 a 127, se realiza el ciclo principal de ejecución, dentro del *while*, en la línea 41 se llama a la función que se encarga de "buscar víctimas" a través de la búsqueda por red de cuadrantes, donde, dependiendo de las lecturas simuladas de la cámara térmica se puede realizar la trayectoria directa a una posible víctima o iniciar el protocolo de búsqueda si las lecturas térmicas se encuentran por debajo de los 23 grados.

En la línea 50 se encuentra la llamada a la función encargada del "trazado de ruta" de ASTEC, la función "a_estrella_robot" regresa el camino generado y los nodos visitados, en caso de que la "búsqueda por malla (o red) de cuadrantes" establezca un parámetro de meta "p" se cumple la condición de la linea 59 del código, lo que permite la creación de los "comandos de control" necesarios para la siguiente etapa de ASTEC, referente a la localización y el mapeo, usando la función "generar_rpm_motores" (linea 86) que genera los datos de control que serán interpretados en algoritmos posteriores, y en la linea 89 se termina el *while loop* con el comando *break*.

Si el tipo de meta corresponde a "s", entonces la condición que se cumple es la de la linea 93, esta condición no establece un paro para el *while loop*, y solo almacena datos para su posterior interpretación gráfica en las simulaciones, las líneas 121, 122 y 123 se encargan de modificar los datos referentes a las iteraciones de exploración del algoritmo

QPA*, donde en la línea 123 se tiene la asignación de los nodos visitados en ese ciclo, estos elementos de la matriz se convierten en los valores de la matriz de "visitados_anteriores" que ayudan al establecimiento del costo extra para nodos visitados en iteraciones previas de la ecuación (4.1.7), el último paso esta en la línea 127 donde la meta (secundaria) alcanzada se convierte en el "nuevo inicio" para la siguiente iteración (si es el caso).

4.2. Algoritmo de Localización y Mapeo

El algoritmo de localización y mapeo establecido para el diseño de ASTEC, será una versión de "Fast SLAM" vista en el curso impartido por *Claus Brenner* en "SLAM Lectures" [50], el enfoque de "Fast SLAM" fue explicado de manera general el capítulo anterior, ahora, se explicaŕa el funcionamiento en específico para el diseño de ASTEC.

Como primer elemento se hablará de la biblioteca de "funciones_apoyo", ésta biblioteca contiene las funciones de apoyo necesarias para implementar el algoritmo de "Fast SLAM", además de algunas funciones para facilitar la lectura de los archivos generados, escritura de archivos, y lectura de datos se los sensores y actuadores simulados.

4.2.1. Funciones de apoyo para procesamiento de información de sensor LIDAR

Estas funciones de apoyo para procesar la información cruda obtenida por el sensor LIDAR, permiten identificar una serie de objetos (para el caso de las simulaciones de ASTEC serán cilindros), estos objetos permiten establecer las referencias del mapa que serán usadas en el algoritmo de localización y mapeo de ASTEC, este tipo de "mapa" se conoce como "mapa basado en características", donde las características del mapa funcionan como referencias para establecer la localización y el mapeo del robot, este enfoque por lo general es mas económico en cuestión de recursos computacionales, sin embargo, no permite extraer mucha información de las características de la zona de despliegue, estas características se pueden establecer usando las esquinas de una habitación, pilares, muebles, etc. Para el caso de ASTEC, las referencias que serán usadas se plantean en un escenario ejemplo del curso de "SLAM Lectures" [50] que están conformadas por seis cilindros distribuidos en un mapa de 2,5m x 2,5m.

La primer función de apoyo, corresponde al calculo de la derivada numérica de los datos obtenidos por el sensor LIDAR, esta función calcula los "saltos" de las lecturas de distancias del sensor, estos saltos deben de ser mayores a una distancia mínima establecida como "min_dist".

```
# Biblioteca "funciones_apoyo" necesarias para implementar
# Fast SLAM, junto con funciones de escritura y lectura de
# archivos generados en la simulacion.

# Codigo original de Claus Brenner, de "SLAM Lectures", 27 ENE 2013
# generado para la version Python 2, version modificada por Brian
# Garcia Sarmina, 14 FEB 2019, para la version Python 3.

# La "compute_derivative" se encarga del procesamiento de los datos
# provenientes del sensor LIDAR "scan" calculando la derivada
# numerica de los datos, ademas, la funcion permite ignorar
# mediciones invalidas de los sensores.
def compute_derivative(scan, min_dist):

    # "jumps" es una lista que guarda los "saltos" generados en los
    # calculos de la derivada numerica.
    jumps = [ 0 ]

    # Se realiza un for loop para todos los datos que se encuentren
    # en el archivo de mediciones del sensor LIDAR.
    for i in range(1, len(scan) - 1):

        # Se establece el incio del "salto" con "l" y el final del
        # salgo como "r".
        l = scan[i-1]
        r = scan[i+1]

        # Si la distancia del salto de "l" y de "r" es mayor a la
        # distancia minima "min_dist", entonces se calcula la
        # derivada numerica del salto y se agrega a la lista de
        # saltos "jumps".
        if l > min_dist and r > min_dist:
            derivative = (r - l) / 2.0
            jumps.append(derivative)

        # Si se encuentra por debajo del valor minimo permitido,
        # se agrega un valor 0 para ese elemento.
        else:
            jumps.append(0)
    jumps.append(0)

    # Se regresa la lista de saltos (jumps) detectados, estos
    # saltos se relacionan con los cilindros (obstaculos) del
    # mapa.
    return jumps
```

Algoritmo 4.2.1.1.a. Biblioteca de “Funciones de Apoyo” para Fast SLAM “cálculo de derivada numérica”.

Cada salto que sea mayor a la distancia mínima (de profundidad) establecida, se agregará a la lista “jumps” para su posterior uso en otras funciones.

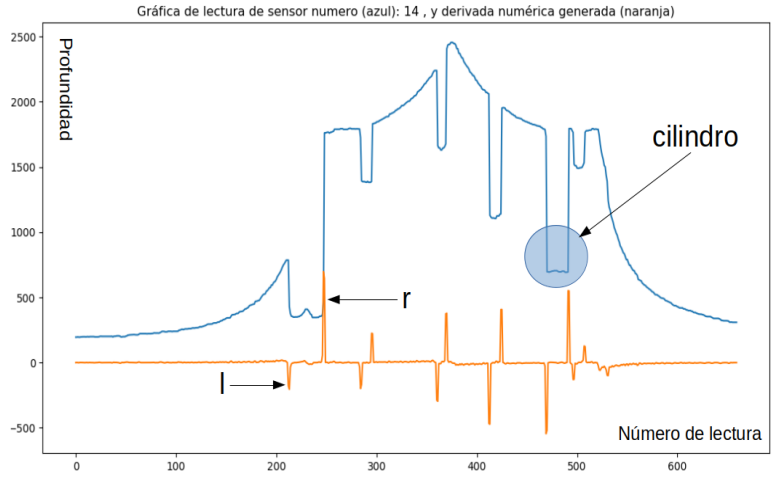


Ilustración 4.2.1.1. Cálculo de derivada numérica.

En la ilustración (4.2.1.1), se muestra el funcionamiento de la función "compute_derivative", el eje vertical corresponde a la profundidad de las medidas (en mm) y el eje horizontal corresponde al número de medición (cada ciclo de medición del sensor LIDAR se considera de 600 lecturas), en color azul se expresan las lecturas obtenidas por el sensor láser (LIDAR), los datos de color azul representan las lecturas crudas obtenidas por el sensor, donde las lineas que no presentan declives o subidas son consideradas como las paredes del mapa, y los declives pronunciados (circulo azul) son contemplados como los obstáculos del mapa (cilindros). De color naranja se presentan los datos posterior a la aplicación del calculo de la derivada numérica, donde existe un inicio denotado como "l" y un fin (del cilindro u obstáculo) denotado como "r".

La siguiente función de apoyo "find_cylinders", usa los datos obtenidos de la función de calculo de derivada numérica para "encontrar" los elementos de estas derivadas que correspondan a un cilindro (obstáculo) del mapa. La función, detecta dentro del arreglo de elementos de las derivadas, los inicios y finales de cada cilindro, para con ello establecer el centro de cada cilindro, usando un conteo de mediciones desde el inicio del cilindro "l" hasta el fin del cilindro "r", y el promedio de distancias detectadas en ese "valle" que corresponde al cilindro.

```
# Funcion para encontrar los cilindros, detectando el flanco de
# bajada y el flanco de subida de la grafica Se calcula el
# promedio muestras del censor (average_ray) y el promedio de
# muestras de profundidad (average_depth)
def find_cylinders(scan, scan_derivative, depth_jump, min_dist):

    cylinder_list = []
    on_cylinder = False
    sum_ray, sum_depth, rays = 0.0, 0.0, 0

# Se realiza un loop por cada valor de obtenido de las derivadas,
# Python por default considera que las listas (en este caso,
```

```
# scan_derivative[]) son ciclicas, se debe realizar un arreglo
# para poder contemplas todos los valores existentes de la lista,
# esto se hace creando un "adelanto" al inicio de la lista y
# posteriormente, creando un atrazo en el ultimo elemento de la
# lista.
    for i in range(1, len(scan_derivative) - 1):

        # En base a los datos obtenidos de la funcion de
        # derivacion (compute_derivative), se calculan los
        # valores actuales (scan_derivative[i-1]) y los valores
        # "siguientes" (scan_derivative[i]), en base a estos,
        # calculamos la diferencia y obtenemos el valor del
        # saldo del flanco de subida o bajada.
        up_edge = depth_jump
        down_edge = (-1) * depth_jump
        left_scan = scan_derivative[i-1]
        right_scan = scan_derivative[i]
        jump_value = left_scan - right_scan

        # Se crea la codicion de deteccion de flanco de subida
        # o bajada, mayor al valor minimo de salto 100.0mm
        # (depth_jump).
        if jump_value <= depth_jump or jump_value >= depth_jump:
            on_cylinder = True

        # Deteccion del flanco de bajada, empieza la cuenta de
        # numero de rayos del censor (rays), suma del valor de
        # los rayos en esa posicion (sum_ray) y suma de
        # profudidades de los valores de los rayos (sum_depth)

            if on_cylinder == True and jump_value <= down_edge:
                rays = rays + 1
                sum_ray = sum_ray + (i)
                sum_depth = sum_depth + scan[i]

        # Generamos la condicion de salida del detector de
        # cilindros, cuando existe un flanco de subida, esto
        # indica que el cilindro termina y obtenemos los promedios
        # del numero de rayos obtenidos (average_ray) y el
        # promedio del numero de profundidades obtenidos
        # (average_depth).
            if on_cylinder == True and up_edge <= jump_value\
               and rays != 0 and sum_ray != 0.0\
               and sum_depth != 0.0:
                average_ray = (sum_ray / rays)
                average_depth = (sum_depth / rays)
                cylinder_list.append((average_ray, average_depth))
                on_cylinder = False
                rays = 0
                sum_ray = 0.0
                sum_depth= 0.0

        # Esta condicion solo contempla cuando no existe una
        # variacion minima considerable del salto de
        # profundidad de los valores de las derivadas.
        else:
            on_cylider = False
```

```
70              rays = 0
71              sum_ray = 0.0
72              sum_depth = 0.0
73
74      # Se regresa la lista de cilindros detectados "cylinder_list".
75      return cylinder_list
```

Algoritmo 4.2.1.1.b. Biblioteca de "Funciones de Apoyo" para Fast SLAM "encontrar cilindros (obstáculos)".

La función "find_cylinders" regresa una lista de *tuples* [3], donde cada elemento de la lista contiene el valor promedio de las mediciones del cilindro "average_ray", y el valor promedio de distancias de estas lecturas "average_depth".

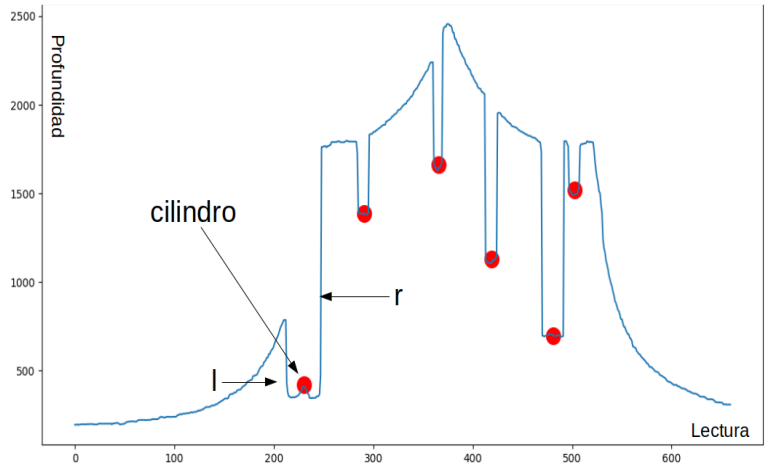


Ilustración 4.2.1.2. Encontrar cilindros a partir de derivada numérica.

En la ilustración (4.2.1.2), se muestran los cilindros detectados (color rojo) por la función "find_cylinders", estos cilindros detectados funcionan como un ejemplo de detección de obstáculos para los datos obtenidos por el sensor LIDAR, dado que esta aproximación se puede adaptar a distintos objetos e incluso ser más flexible para poder incluir posibles detecciones de "esquinas" estructurales, y otros elementos que sirvan como referencias para la localización y mapeo de ASTEC.

La siguiente función de la biblioteca de "funciones de apoyo" para el algoritmo de localización y mapeo, integra las funciones de "comput_derivative" y "find_cylinders" para generar una lista de pares de tuples, donde cada par contiene: primer tuple $(distancia\_promedio, angulo)$ (este elemento es calculado con la función anterior y analizando el número de medición del escáner que indica el ángulo de barrido del sensor LIDAR) y segundo tuple (x, y) coordenadas cartesianas en el sistema de coordenadas del sensor LIDAR.

[3]Un "tuple" es una secuencia de objetos inmutables usada en Python, a diferencia de las "listas" (arreglos o vectores) los tuples no pueden ser modificados o alterados

```
# La funcion "get_cylinders_from_scan" integra las funciones de
# "comput_derivative" y "find_cylinders" para calcular el angulo
# y coordenadas cartesianas de los cilindros detectados.
def get_cylinders_from_scan(scan, jump, min_dist, cylinder_offset):

    # Se calcula la derivada numerica por "corrida" del sensor LIDAR.
    der = compute_derivative(scan, min_dist)
    # Se encuentran los cilindros por "corrida" del sensor LIDAR.
    cylinders = find_cylinders(scan, der, jump, min_dist)

    # Se genera una lista "result" para almacenar los angulos y
    # coordenadas cartesianas que corresponden a cada cilindro
    # detectado.
    result = []

    # Se realiza un ciclo por cada elemento "c" en cilindros
    # "cylinders".
    for c in cylinders:

        # Se extrae el angulo que corresponde al numero de medicion
        # que se este analizando, en funcion al cilindro detectado.
        bearing = LegoLogfile.beam_index_to_angle(c[0])
        # El calculo de la distancia, se calcula usando la distancia
        # calculada de "find_cylinders" mas la distancia de desfase
        # del sensor al centro de ASTEC.
        distance = c[1] + cylinder_offset
        # Se calculan los valores para (x, y) del cilindro, y el
        # angulo que corresponden.
        x, y = distance*cos(bearing), distance*sin(bearing)
        result.append( (np.array([distance, bearing]), np.array([x, y])) )

    # Se regresa la lista de tuples, donde los tuples son arreglos
    # dobles (usando la funcion np.array() de numpy), donde se
    # almacenan las (distancias, angulos) y coordenadas (x, y)
    # (sistema de coordenadas del sensor LIDAR).
    return result
```

Algoritmo 4.2.1.1.c. Biblioteca de "Funciones de Apoyo" para Fast SLAM "calculo de coordenadas cartesianas".

La idea de esta biblioteca es generar datos que corresponden a las referencias (cilindros) del mapa, estas referencias se pueden expresar en las coordenadas cartesianas (x, y) que se relacionan con el sistema de coordenadas del sensor, y cuando se considera el ángulo y distancia, también se relaciona con el sistema de coordenadas del sensor, esto establece que los datos procesados por estas funciones, realmente contemplan al sensor como "el centro del robot" sin embargo, eso no es aplicable en la realidad, por ello, habrá funciones que convertirán estos datos del sistema de coordenadas del sensor, al sistema de coordenadas del robot (más precisamente, del centro del robot).

4.2.2. Funciones de apoyo para implementación de Fast SLAM

Estas funciones de apoyo, también se encuentran contenidas en el programa "funciones_apoyo", pero con la diferencia (con respecto a las explicadas anteriormente) de que estas funciones se ocupan directamente para llevar acabo tareas complementarias del enfoque Fast SLAM de ASTEC.

La primer función de apoyo para Fast SLAM, se basa en la extracción de las medias o promedios, de la ubicación (coordenadas x y y de las partículas) y la orientación (tomando el eje horizontal como referencia).

```
### La funcion "get_mean" funciona para extraer la media (o promedio)
### del conjunto de particulas (de la etapa del filtro de particulas
### de Fast SLAM, los datos relevantes que se extraen de este conjunto
### son la media de "posicion" y la media de "orientacion". ###
def get_mean(particles):

    # Las variables "mean_x" y "mean_y" se ocupan para almacenar la
    # suma de la ubicacion en (x, y) respectivamente, del conjunto
    # de particulas generado.
    mean_x, mean_y = 0.0, 0.0
    # Las variables "head_x" y "head_y" alacenan la suma de funciones
    # cos (para el caso de "x") y sen (para el caso de "y") sobre
    # todo el conjunto de particulas generado.
    head_x, head_y = 0.0, 0.0
    for p in particles:
        x, y, theta = p.pose
        mean_x += x
        mean_y += y
        head_x += cos(theta)
        head_y += sin(theta)

    n = max(1, len(particles)) # Numero de particulas

    # Se regresa el calculo de los promedios (o medias) de las
    # ubicaciones y orientaciones de las particulas como un
    # np.array (usando la biblioteca "numpy").
    return np.array([mean_x / n, mean_y / n, atan2(head_y, head_x)])
```

Algoritmo 4.2.2.1.d. Biblioteca de "Funciones de Apoyo" para Fast SLAM "calculo de promedios de partículas".

Una vez generadas las medias de "ubicación" y "orientación" del conjunto de partículas, se puede calcular la matriz de covarianza de xy y la varianza en la orientación del promedio (en ubicación y orientación) del conjunto de partículas.

```
### Funcion "get_error_ellipse_and_heading_variance" se usa para
### calcular la "elipse de error" y la "varianza de orientacion".
### La "elipse de error" se establece como el area de incertidumbre
### donde se presume estaria el robot, la "varianza de orientacion"
### se refiere como la incertidumbre en el angulo de orientacion del
```

```
### frente del robot, donde existe un area indica esta incertidumbre
### estos elementos seran observados en las simulaciones del
### siguiente capitulo. ###

def get_error_ellipse_and_heading_variance(particles, mean):

    # Se extraen los promedios (o medias) calculados por la funcion
    # "get_mean()" anterior.
    center_x, center_y, center_heading = mean
    n = len(particles)

    # Si el numero de particulas es menor a "2" este metodo no
    # puede ser aplicado de manera correcta, es necesario,
    # poseer minimo "2" particulas como referencia, del cambio
    # con respecto a su promedio (de las particulas).
    if n < 2:
        return (0.0, 0.0, 0.0, 0.0)

    # Se calcula la matriz de covarianza en xy, para calcular esta
    # covarianza es necesario calcular la suma de todos los
    # desplazamientos (desviacion estandar) de la ubicacion de
    # cada particula (x, y) con respecto de la media
    # (center_x, center_y), y calcular el cuadrado para la
    # covarianza en "x", "y" y "xy" (y dividir esos resultados
    # entre el numero total de particulas). Tambien se realiza
    # el calculo de la "varianza" en orientacion, en este caso
    # la varianza se calcula sobre una propagacion a una sola
    # dimension (la dimension que corresponde a la "orientacion").
    sxx, sxy, syy = 0.0, 0.0, 0.0
    var_heading = 0.0

    for p in particles:

        # Covarianza para ubicacion (x, y)
        x, y, theta = p.pose
        dx = x - center_x
        dy = y - center_y
        sxx += dx * dx
        sxy += dx * dy
        syy += dy * dy

        # Varianza de orientacion (theta)
        dh = (p.pose[2] - center_heading + pi) %(2*pi) - pi
        var_heading += dh * dh

    cov_xy = np.array([[sxx, sxy], [sxy, syy]]) / (n-1)
    var_heading = var_heading / (n-1)

    # Se realiza la conversion de xy, a una interpretacion
    # que pueda ser usada en el trazado de la elipse, este
    # metodo se realiza calculando los eigenvalores (valores
    # propios) y lo eigenvectores (vectores propios), usando
    # la funcion np.linalg.eig() de "numpy", esta funcion
    # solo calcula usando la matriz de covarianza (xy)
    eigenvals, eigenvects = np.linalg.eig(cov_xy)

    # Con los "eigenvals" y "eigenvects", se calcula el
```

```
63      # angulo de la elipse.
64      ellipse_angle = atan2(eigenvects[1,0], eigenvects[0,0])
65
66      # Se regresan los datos de: angulo de elipse, desviacion
67      # estandar en "x", desviacion estandar en "y", y desviacion
68      # estandar en "orientacion".
69      return (ellipse_angle, sqrt(abs(eigenvals[0])),
70              sqrt(abs(eigenvals[1])),
71              sqrt(var_heading))
```

Algoritmo 4.2.2.1.e. Biblioteca de "Funciones de Apoyo" para Fast SLAM "calculo de matriz de covarianza xy y varianza de orientación".

La función del algoritmo (4.2.2.1.e), permite hacer el cálculo de la elipse que será graficada en la GUI (en el siguiente capítulo del trabajo de tesis), este método puede ser entendido de manera general, como un método para simplificar la proyección de la etapa del "filtro extendido de Kalman" usado en Fast SLAM, donde en una proyección real de la incertidumbre asociada al filtro se daría en relación al número de variables de estado, tomando como ejemplo el caso de ASTEC en un instante $t = 0$, el estado del sistema esta descrito por (x, y, θ), pero a medida de que la interpretación de los datos de los sensores aumenta el número de marcas en el mapa, la dimensión del filtro va aumentando $(x, y, \theta, x_1^m, y_m^1, x_m^2, y_m^2, ..., x_m^n, y_m^n)$ (en el caso cuando solo se contemple la ubicación de las marcas en (x, y)), por ello es importante lograr obtener la "matriz de covarianza" de la ubicación (x, y) de todas las partículas del mapa, y obtener la varianza en el ángulo de orientación de las partículas, ya que si bien, las dimensiones (x, y) no representan todas las dimensiones del filtro, permiten establecer las distribuciones marginales, junto con el ángulo asociado a esta elipse. Además, esta representación de la incertidumbre asociada con (x, y) en una matriz de covarianza xy establece que x y y son correlacionadas. [50]

La biblioteca de "funciones_apoyo", contiene también algunas funciones para guardar los datos obtenidos en las simulaciones de los escenarios, estos datos son guardados con un encabezado que indica el tipo de información relacionado, y la información asociada a ese encabezado (por renglón), todos los datos son almacenados en archivos (.txt) para mantener el bajo consumo en el almacenamiento de los datos, y permitir un mejor envió de los mismos.

Antes de abordar el algoritmo de Fast SLAM, es necesario plantear el modelo de movimiento (encargado de la transición de estado) y el modelo de observación (encargado de la etapa de corrección) que serán implementados para ASTEC, ya que estos modelos son necesarios para realizar las etapas de predicción y corrección del enfoque Fast SLAM.

4.2.2.1. Modelo de Movimiento para ASTEC

Con los aspectos comentados en capítulo anterior, ahora se pueden expresar las ecuaciones para el modelo cinemático de movimiento por odometría, debe aclararse que este modelo cinemático esta basado en cinemática de cuerpo rígido, que se plantean en el diseño del robot ASTEC, estas ecuaciones son en su mayoría tomadas de [50] "SLAM Lectures" por *Claus Brenner* y del libro de "Probabilistic Robotics" de *Sebastian Thrun*, *Wolfram Burgard* y *Dieter Fox* [38] (se han encontrado que varios modelos de movimiento están basados en el libro de "Probabilistic Robotics" que es una referencia frecuente, usada en robótica móvil).

En la ilustración (4.2.2.1.1), se puede observar el diseño básico de como se abordará el modelo de movimiento del robot ASTEC, dado que se contempla la existencia de los rodamientos por oruga planteados como el chasis tentativo del diseño del robot (color gris obscuro), después se tienen los dos motores (motor izquierdo y motor derecho) que se encuentran como rectángulos color rojo, el cuerpo del robot planteado en color gris y por último, el centro del robot como un círculo rojo, donde además, dado este diseño, en el centro del robot irían colocados los elementos de percepción de los objetos (sensor LIDAR), la cámara WEB y la cámara térmica.

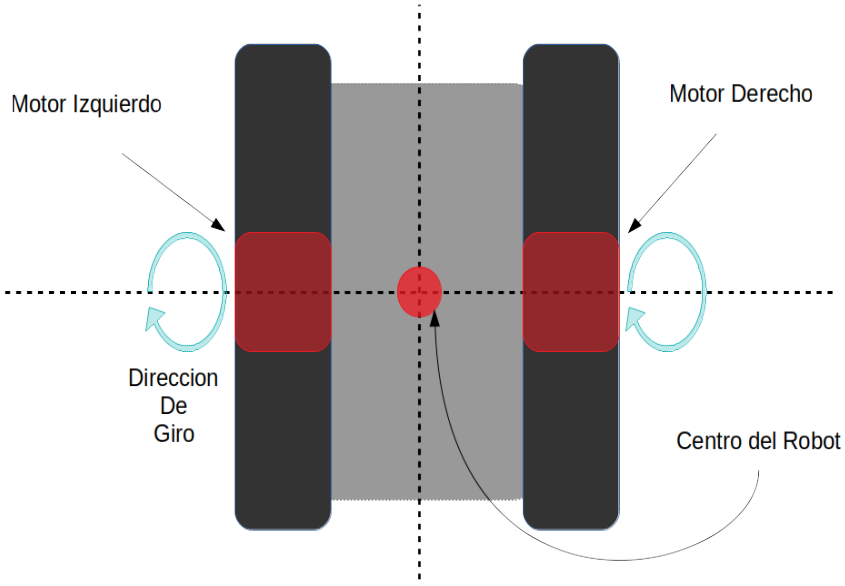


Ilustración 4.2.2.1.1. Representación del Robot ASTEC para el modelo de movimiento.

En el modelo de movimiento por odometría, solo se plantean dos casos posibles de funcionamiento de los motores del robot, donde, para simplificar las ilustraciones se contemplará al robot como un sistema de dos ruedas y un centro.

Los dos casos contemplados son:

- Cuando la distancia izquierda D_i es igual a la distancia derecha D_d, es decir, que no existe variación en el ángulo de viraje θ.
- Cuando distancia izquierda D_i no es igual a distancia derecha D_d, existe una variación en el ángulo de viraje θ (cambio de ángulo de orientación del robot).

Distancia Izquierda D_i es igual a Distancia Derecha D_d

Entonces para el primer caso se puede observar la ilustración (4.2.2.1.2), donde se ejemplifica el movimiento contemplado, donde no hay un cambio en el ángulo de viraje θ.

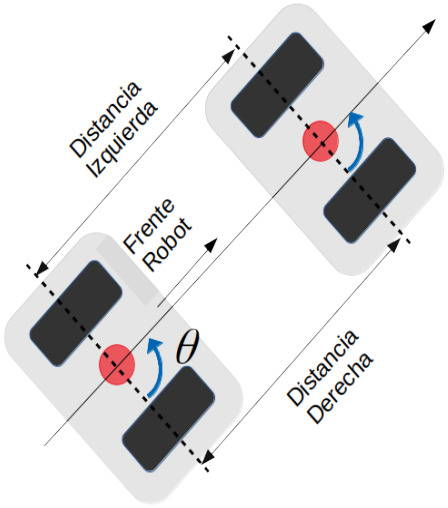


Ilustración 4.2.2.1.2. Movimiento sin cambio de ángulo de viraje.

Entonces, lo primero es generar las distancias para el lado izquierdo y derecho del robot (distancia izquierda y distancia derecha, respectivamente) D_i y D_d.

Para la D_i se tiene:

(4.2.2.1.1)

$$D_i = RPM_i * Cov_{mm}$$

Donde:

- RPM_i son las revoluciones medidas de dos instantes del codificador del motor izquierdo, usualmente para $t-1$ y t, esto se realiza así, debido a que si no existe variación en las mediciones de los codificadores, indica que el robot no esta realizando ningún movimiento, es decir, los motores no se encuentran en movimiento.

- Cov_{mm} es un parámetro de calibración que se establece en la implementación física del robot, y este indica la cantidad de milímetros mm que el robot avanza dado un "tic" del codificador del motor.

Para la D_d se tiene la aplicación de la ecuación (4.2.2.1.1), pero contemplando los valores de RPM_d.

(4.2.2.1.2)

$$D_d = RPM_d * Cov_{mm}$$

Donde:

- RPM_d corresponde al valor de diferencia del codificador del motor derecho del robot, contemplando los factores antes explicados.
- Cov_{mm} mismo factor de conversión usado para el calculo de D_i.

Ahora, se establecen las fórmulas de transición de estado del robot, cuando se cumple la condición de que $D_i = D_d$.

(4.2.2.1.3)

$$\theta' = \theta$$

(4.2.2.1.4)

$$x' = x + D_i * cos\,(\theta)$$

(4.2.2.1.5)

$$y' = y + D_i * sen\,(\theta)$$

Donde:

- θ' corresponde al nuevo ángulo de viraje, y θ es el ángulo de viraje anterior, sin embargo, en el caso $D_i = D_d$ el ángulo de viraje se mantiene.
- (x, y) es la posición del robot en el estado anterior en $t - 1$.
- (x', y') es la posición actual del robot en t.

Cuando se cumple el caso $D_i = D_d$ se puede observar que el modelo de movimiento del robot cumple con muchos criterios donde para el cálculo de la nueva posición del robot (x', y') puede usarse indiferentemente la D_i o D_d, en el caso planteado, se uso la distancia del motor izquierdo (observar ecuaciones (4.2.2.1.4) y (4.2.2.1.5)).

Además, de esta condición, el ángulo de viraje anterior en $t - 1$ se mantiene para la posición actual de robot en t, y el uso de la función *cos* y *sen* (para x', y', respectivamente) se establece dependiendo del eje de referencia para el ángulo de viraje.

Distancia Izquierda D_i no es igual a la Distancia Derecha D_d

En este caso se contempla la condición de $D_i \neq D_d$, es decir, que las distancias de los motores izquierdo y derecho son diferentes.

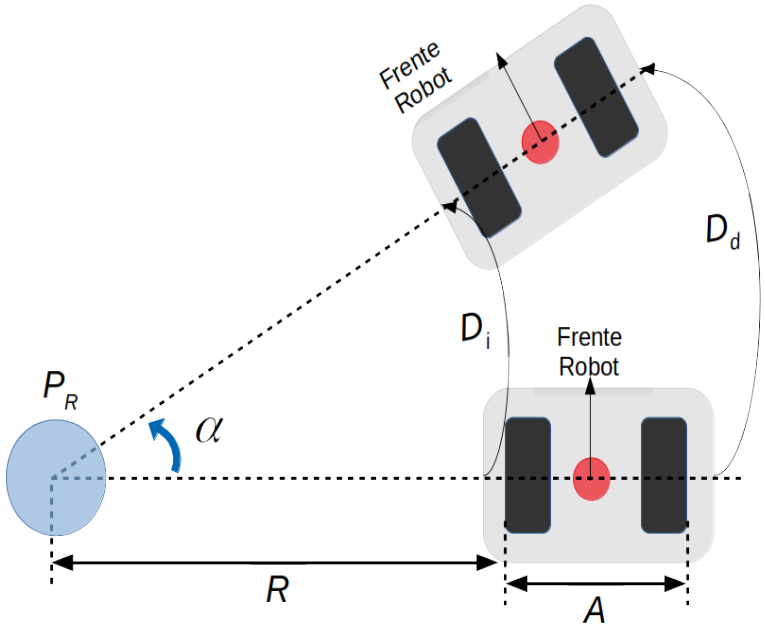


Ilustración 4.2.2.1.3 Movimiento con cambio de ángulo de viraje.

Observando la Ilustración (4.2.2.1.3), podemos establecer dos ecuaciones para las trayectorias de la "rueda derecha" y para la "rueda izquierda", contemplando el punto de referencia P_R (circulo azul).

Para la rueda derecha tenemos que la D_d esta dada por:

(4.2.2.1.6)

$$D_d = \alpha \cdot (R + A)$$

Donde:

- α corresponde al ángulo de la rotación (en referencia a P_R), del robot en el instante $t-1$ a t, donde el ángulo esta dado en radianes (para fines de los cálculos del programa).
- R es la distancia de la referencia (círculo azul) al robot en el instante $t-1$.
- A es el ancho del robot.

Para la rueda izquierda la D_i esta dada por:

(4.2.2.1.7)

$$D_i = \alpha \cdot R$$

Donde α y R tienen la misma definición que para la ecuación (4.2.2.1.6).

Si se resta (4.2.2.1.7) a (4.2.2.1.6), para obtener α se tiene:

(4.2.2.1.8)

$$D_d - D_i = \alpha \cdot (R + A) - \alpha \cdot R$$

(4.2.2.1.9)

$$\alpha = \frac{D_d - D_i}{A}$$

Y para obtener R (distancia del robot a P_R en $t-1$), se usa la ecuación (4.2.2.1.7).

(4.2.2.1.10)

$$R = \frac{D_i}{\alpha}$$

Ahora, obtenidos los valores para R y α, dados los valores de D_d y D_i (que pueden ser calculados), el ancho del robot A, y el punto de referencia P_R.

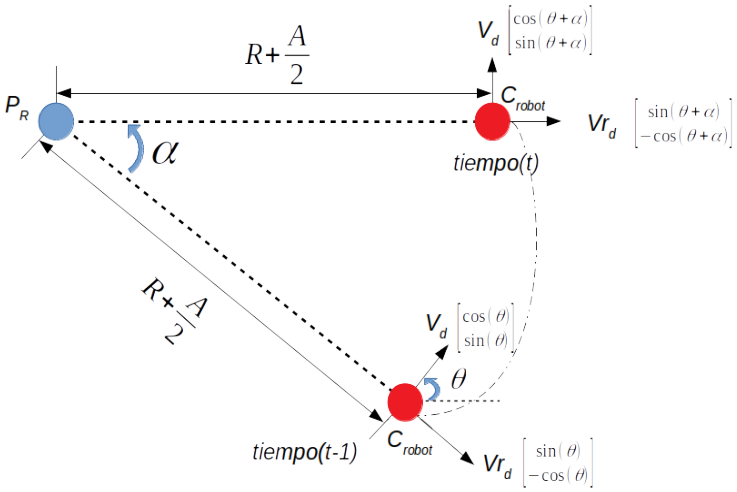


Ilustración 4.2.2.1.4. Translación en curva, análisis como punto masa.

Contemplando los valores obtenidos de R y α, ahora se calcula la posición del punto de referencia $P_R = \begin{bmatrix} x_R \\ y_R \end{bmatrix}$, la nueva posición del robot $\bar{x}_t = C_r = \begin{bmatrix} x' \\ y' \end{bmatrix}$, donde C_r corresponde al centro del robot en el tiempo t, y por último, se debe calcular el nuevo ángulo de viraje θ' (dirección de orientación actual del robot).

En la ilustración (4.2.2.1.4) se observa el análisis del movimiento del robot cuando $D_i \neq D_d$ (como punto masa), donde en color azul se tiene a P_R que es el punto de referencia de giro (punto pivote), después se tiene en color rojo, dos puntos que representan en "centro" del robot C_r para los instantes $t-1$ (en la parte inferior derecha) y t (parte superior derecha), cada C_r posee dos vectores asociados, el vector de dirección V_d y el vector rectangular al vector de dirección [4] como Vr_d. Para el instante $t-1$ el vector dirección es $V_d = \begin{bmatrix} cos(\theta) \\ sin(\theta) \end{bmatrix}$ y el vector rectangular de dirección es $Vr_d = \begin{bmatrix} sin(\theta) \\ -cos(\theta) \end{bmatrix}$, para la posición actual en t se tiene $V_d = \begin{bmatrix} cos(\theta+\alpha) \\ sin(\theta+\alpha) \end{bmatrix}$ como vector de dirección y $Vr_d = \begin{bmatrix} sin(\theta+\alpha) \\ -cos(\theta+\alpha) \end{bmatrix}$ como su vector rectangular. Estos vectores planteados se dan con forme al tipo de giro, que puede ser derecha ($-cos(\theta)$) e izquierda ($-sin(\theta)$), para este modelo se usa siempre el modelo de giro hacia la derecha con $-cos(\theta)$.

Para calcular el P_R (la referencia del pivote) se plantea la siguiente ecuación:

(4.2.2.1.11)
$$\begin{bmatrix} P_{rx} \\ P_{ry} \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \end{bmatrix} - (R + \frac{A}{2}) \begin{bmatrix} sin(\theta_{t-1}) \\ -cos(\theta_{t-1}) \end{bmatrix}$$

Donde:

- $\begin{bmatrix} P_{rx} \\ P_{ry} \end{bmatrix}$ corresponde a la posición en x P_{rx} y la posición en y P_{ry} de la referencia.
- $\begin{bmatrix} x_{t-1} \\ y_{t-1} \end{bmatrix}$ es la posición anterior al movimiento del robot.
- $\begin{bmatrix} sin(\theta_{t-1}) \\ -cos(\theta_{t-1}) \end{bmatrix}$ que corresponde al vector rectangular a la dirección en el ángulo anterior [5] al movimiento del robot.

Después, se calcula el nuevo ángulo de viraje θ' usando el antiguo ángulo de viraje θ_{t-1} y el ángulo de giro con respecto a la referencia, hacia la nueva posición del robot α.

(4.2.2.1.12)
$$\theta' = (\theta_{t-1} + \alpha) * mod(2\pi)$$

Donde el termino $mod(2\pi)$ se refiere a una operación de "módulo".

Usando la ecuación (4.2.2.1.11) y (4.2.2.1.12) ahora se calcula C_r el centro del robot con:

[4] La función sin() (para lenguajes de programación) es una equivalencia para la función sen() (trigonométrica), se usará indistintamente una y otra a lo largo de la tesis.

[5] El ángulo θ_{t-1} corresponde al ángulo θ de la ilustración (4.2.2.1.4), para la ecuación (4.2.2.1.11) se hace énfasis en que corresponde al ángulo de orientación del robot antes del movimiento.

(4.2.2.1.13)

$$\begin{bmatrix} C_{rx} \\ C_{ry} \end{bmatrix} = \begin{bmatrix} P_{rx} \\ P_{ry} \end{bmatrix} + (R + \frac{A}{2}) \begin{bmatrix} sin(\theta') \\ -cos(\theta') \end{bmatrix}$$

Con la ecuación (4.2.2.1.13) se calcula la posición para x C_{rx} y la posición en y C_{ry} del robot, usando la referencia P_r y el nuevo ángulo de viraje θ'.

4.2.2.2. Modelo de Observación para ASTEC

El modelo de observación o modelo de medición [38] consiste básicamente en como el robot es capaz de percibir su entorno a través de los sensores, y como es procesada y representada dicha información para el robot.

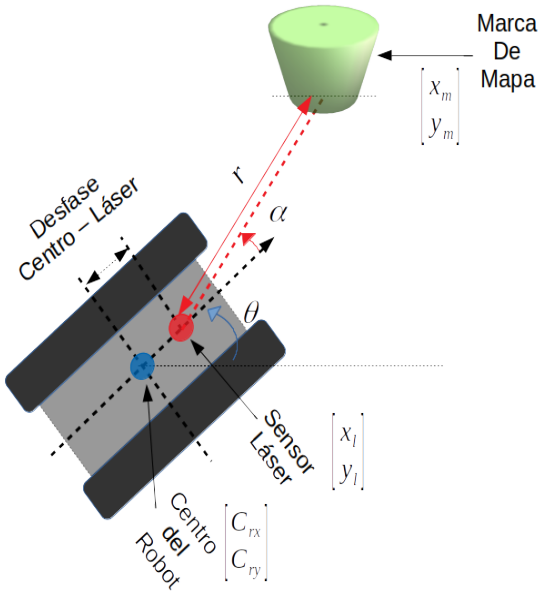


Ilustración 4.2.2.2.1. Representación del Robot ASTEC para el modelo de observación.

En la ilustración (4.2.2.2.1) se plasma el modelo de observación general para el diseño de este trabajo, donde se tiene la posición del centro del robot (usando la nomenclatura de la ecuaciones anteriores) $\begin{bmatrix} C_{rx} \\ C_{ry} \end{bmatrix}$, la posición del sensor $\begin{bmatrix} x_l \\ y_l \end{bmatrix}$, esta posición del sensor tiene un desfase con respecto al centro del robot, y dicho parámetro de calibración debe ser contemplado para una implementación física, dado que el no contemplar el desfase, genera un aumento en la incertidumbre del cálculo de la localización del robot y las marcas del mapa. Posteriormente se tiene la posición de las marcas del mapa $\begin{bmatrix} x_m \\ y_m \end{bmatrix}$, dado que el mapa de aplicación del diseño robótico es "estático" (el concepto de "estático" se aplica pensando en que las marcas del mapa no presentan muchos movimientos considerables, además de ser una forma para simplificar el planteamiento del modelo de observación), la posición de las marcas de mapa se vuelven constantes. El cálculo de la posición $\begin{bmatrix} x_m \\ y_m \end{bmatrix}$ de las marcas del mapa, se abordará en el capítulo siguiente.

El modelo de observación para el diseño de ASTEC, se basará en el cálculo de r que corresponde a la distancia entre la marca del mapa y la ubicación del sensor láser, y el cálculo de α, que es el ángulo de apertura entre la marca del mapa y el ángulo de viraje del robot para el instante t.

El cálculo de la posición del sensor láser, se plantea con la siguiente ecuación:

(4.2.2.2.1)

$$\begin{bmatrix} x_l \\ y_l \end{bmatrix} = \begin{bmatrix} C_{rx} \\ C_{ry} \end{bmatrix} + d_{rl} \begin{bmatrix} cos(\theta) \\ sen(\theta) \end{bmatrix}$$

Donde:

- d_{rl} es la distancia de desfase entre el centro del robot C_r al sensor láser.
- $\begin{bmatrix} cos(\theta) \\ sen(\theta) \end{bmatrix}$ son las componentes para x, y del desfase del sensor, con respecto al ángulo de dirección del robot θ.

Después de calcular la posición relativa del sensor láser, se calcula la distancia relativa del láser a la marca del mapa usando la función de distancia euclidiana entre dos puntos.

(4.2.2.2.2)

$$r = \sqrt{(x_m - x_l)^2 + (y_m - y_l)^2}$$

Donde r corresponde a la distancia relativa del láser $\begin{bmatrix} x_l \\ y_l \end{bmatrix}$ a la marca del mapa $\begin{bmatrix} x_m \\ y_m \end{bmatrix}$.

Y finalmente, se calcula el ángulo α usando la función $atan()$ y restando el ángulo de orientación del robot θ.

(4.2.2.2.3)

$$\alpha = atan(\frac{y_m - y_l}{x_m - y_l}) - \theta$$

Como se comentó anteriormente, el modelo de observación permite ejecutar la etapa de "corrección" del enfoque Fast SLAM, este modelo no debe de ser confundido con el método de procesamiento de los datos del sensor LIDAR visto en la biblioteca "funciones_apoyo", dado que esas funciones realizan el "pre-procesamiento" de la información, y posterior a ello es cuando el modelo de observación entra en función.

4.2.3. Aplicación de enfoque Fast SLAM

Para la aplicación del enfoque Fast SLAM para el diseño de ASTEC, se hará uso del modelo de movimiento, modelo de observación, teoría explicada en el marco teórico referente al "Filtro Extendido de Kalman" y el "Filtro de Partículas Rao-Blackwellised", y la biblioteca "funciones_apoyo".

El método de Fast SLAM permite realizar la factorización del "posterior" en el problema de "SLAM" (ecuaciones (2.2.22) y (2.2.24) dependiendo de que se establezcan las características de las marcas $c_{1:t}$) [50] [38]. En esta factorización es necesario establecer $y_{1:t} = (x_{1:t}, m_n)$ como el estado del sistema.

(4.2.3.1)

$$p(y_{1:t}|u_{1:t}, z_{1:t}, c_{1:t}) = p(x_{1:t}|u_{1:t}, z_{1:t}, c_{1:t}) \prod_{i=1}^{n} p(m_i|u_{1:t}, z_{1:t}, c_{1:t})$$

En la ecuación (4.3.2.1) la variable m representa el numero de partículas que se tiene para expresar el estado. Esta factorización del problema de SLAM $p(y_{1:t}|u_{1:t}, z_{1:t}, c_{1:t})$ no es una aproximación, es una representación exacta cuando se aplica el enfoque de Fast SLAM [50]. Analizando la ecuación (4.2.3.1) con más detalle, se tiene:

(4.2.3.2)

$$p(x_{1:t}|u_{1:t}, z_{1:t}, c_{1:t})$$

Teniendo la primer etapa de la factorización con la ecuación (4.2.3.2), esta etapa es la sección que corresponde a la aplicación del "filtro de partículas" [65], donde el estado $x_{1:t}$ es expresado a través de un conjunto de partículas. Esta primer sección corresponde a la localización (o estado) del robot, dado que el método de Rao-Blackwellised es aplicado, este método establece que la localización del robot es conocida (debido a la representación del conjunto de partículas).

(4.2.3.3)

$$p(x_{1:t}|u_{1:t}, z_{1:t}, c_{1:t}) \overset{PF}{\rightarrow} \sum_{i=1}^{m} \left\{x_{1:t}^{i}\right\}$$

Donde:

- $x_{1:t}^{i} = \begin{bmatrix} x^i \\ y^i \\ \theta^i \end{bmatrix}_{1:t}$ representa que cada partícula en el conjunto desde 1 hasta m contiene su propia secuencia de "estados $1:t$".

Posterior a ello, la siguiente etapa de la factorización:

(4.2.3.4)

$$p(m_i|u_{1:t}, z_{1:t}, c_{1:t})$$

En la segunda etapa de la factorización (ecuación (4.2.3.4)), corresponde a la etapa de la aplicación del "filtro extendido de Kalman", esta etapa se encarga de estimar la localización de las características (o marcas) del mapa [38] [50] [65]. En esta etapa existen dos elementos importantes de señalar, la primera es que dada la representación del estado del robot a través del filtro de partículas Rao-Blackwellised, permite que las marcas del mapa sean independientes de la localización del robot (esto se debe a que en la aproximación del filtro de partículas R-B, la "posición" del robot es "conocida" a lo largo de todo el camino) [65].

(4.2.3.5)

$$p(m_m|u_{1:t}, z_{1:t}, c_{1:t}) = \sum_{i=1}^{m} \left[(\mu_1, \Sigma_1)^i, (\mu_2, \Sigma_2)^i, ..., (\mu_n, \Sigma_n)^i\right]$$

Donde:

- m_m es el mapa correspondiente a cada partícula en el conjunto de partículas establecido m.
- μ^n es la "media" o "promedio" de la marca n del mapa.
- Σ^n es la matriz de covarianza de la marca n del mapa.
- m representa al numero de partículas, en el conjunto de partículas establecido.

La ecuación (4.2.3.5) implica la generación de filtros de Kalman individuales para cada marca de mapa, en donde el estado esta descrito de manera discreta con la localización del robot (x, y, θ) y una marca del mapa (x_i, y_i donde i es el número de referencia (marca) del mapa. Ésta estrategia permite reducir el problema de dimensionalidad que puede ser generada en un filtro de Kalman (extendido o estándar) multivariante, en el enfoque de EKF SLAM (donde solo se ocupan filtros de Kalman para realizar la localización y mapeo) existe un problema de la "maldición de la dimensionalidad" donde el estado del robot contiene todas las ubicaciones de las marcas del mapa hasta un instante t, lo que genera un problema de alta complejidad computacional, es por ello que el enfoque de Fast SLAM usualmente es menos robusto en términos de recursos (memoria y procesamiento) y permite ser implementado en aplicaciones de campo con más facilidad (como en el caso de ASTEC).

La factorización del enfoque de Fast SLAM, se expresa en la siguiente tabla:

$x_{1:t}^{[1]}$	$\mu_1^{[1]}, \Sigma_1^{[1]}$	$\mu_2^{[1]}, \Sigma_2^{[1]}$	. . .	$\mu_N^{[1]}, \Sigma_N^{[1]}$
$x_{1:t}^{[2]}$	$\mu_1^{[2]}, \Sigma_1^{[2]}$	$\mu_2^{[2]}, \Sigma_2^{[2]}$	. . .	$\mu_N^{[2]}, \Sigma_N^{[2]}$
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
$x_{1:t}^{[M]}$	$\mu_1^{[M]}, \Sigma_1^{[M]}$	$\mu_2^{[M]}, \Sigma_2^{[M]}$	. . .	$\mu_N^{[M]}, \Sigma_N^{[M]}$

Tabla 4.2.3.1. Enfoque Fast SLAM.

En la tabla (4.2.3.1) se tiene en color naranja, la etapa que corresponde a la aplicación del "filtro de partículas" y en color azul se tiene la aplicación de los "filtros de Kalman extendidos" individuales por marca.

Como se mencionó en el marco teórico, el enfoque de Fast SLAM permite resolver el problema de "Full SLAM" y "Online SLAM", dado que en este enfoque cada partícula mantiene su propia relación de marcas y trayectoria este método se considera mucho más robusto que otros enfoques para resolver SLAM, para propósitos del diseño de ASTEC el enfoque de Fast SLAM se usará como un "filtro", es decir, que funcionará para resolver el problema de Online SLAM.

Para un mejor entendimiento de la aplicación del enfoque (Fast SLAM), se dividirá el algoritmo total de localización y mapeo, en dos secciones, la primera sección corresponderá a la "etapa de predicción" (similar a la idea de los filtros de Bayes) y la segunda etapa corresponderá a la "etapa de corrección" (de manera similar a los filtros de Bayes también).

Etapa de Predicción de Fast SLAM

Ahora, contemplando lo mencionado, a continuación se comenzara a desglosar el programa implementando para la sección de localización y mapeo de ASTEC, usando el enfoque de Fast SLAM.

```
# Aplicacion de enfoque Fast SLAM, localizacion , mapeo y conteo
# de marcas de mapa.

# Codigo original de Claus Brenner, de "SLAM Lectures", 20 FEB 2013
# generado para la version Python 2, version modificada por Brian
# Garcia Sarmina, 14 FEB 2019, para la version Python 3.

### Definicion de Clase Particula (Particle) ###

### La clase "Particle" (particula) contiene la mayoria de las
### funciones y metodos necesarios para la aplicacion del enfoque
### Fast SLAM, algunas de las funciones y metodos mas importantes que
### contiene esta clase son: la funcion de movimiento g(), funcion
### de medicion h(), calculo de matrices jacobianas, probabilidad de
### correspondencia, inicializacion de nuevas marcas de mapa,
### actualizacion de marcas de mapa, etc. ###
```

```
17 class Particle:
18
19     # Se establecen los elementos pertenecientes a cada particula,
20     # pose (posicion robot), marcas de mapa (posicion, matrices de
21     # covarianza, y contadores de marcas).
22
23     def __init__(self, pose):
24         self.pose = pose
25         self.landmark_positions = []
26         self.landmark_covariances = []
27         self.landmark_counters = []
28
29     # Funcion para retornar el numero de marcas pertenecientes a
30     # cada particula, se establece como una funcion para evitar
31     # su confucion con otros datos.
32     def number_of_landmarks(self):
33         return len(self.landmark_positions)
34
35     # Metodo Estatico #1
36     # Corresponde a la aplicacion de la modelo de transicion de
37     # estado, esta etapa se relaciona con la prediccion del
38     # movimiento aplicado a cada particula, esta funcion regresa
39     # un array que consite en los cambios en (x, y), y angulo de
40     # orientacion.
41     @staticmethod
42     def g(estado, control, w):
43         x, y, theta = estado
44         izq, der = control
45         if der != izq:
46             alpha = (der - izq) / w
47             rad = izq/alpha
48             g1 = x + (rad + w/2.)*(sin(theta+alpha) - sin(theta))
49             g2 = y + (rad + w/2.)*(-cos(theta+alpha) + cos(theta))
50             g3 = (theta + alpha + pi) %(2*pi) - pi
51         else:
52             g1 = x + izq * cos(theta)
53             g2 = y + izq * sin(theta)
54             g3 = theta
55         return np.array([g1, g2, g3])
56
57     # Funcion para enviar los datos de control (movimientos) a
58     # la funcion g().
59     def move(self, izq, der, w):
60         self.pose = self.g(self.pose, (izq, der), w)
```

Algoritmo 4.2.3.1.a. Etapa de predicción de Fast SLAM, inicio de clase “partícula” y función modelo de movimiento.

El algoritmo (4.2.3.1.a) muestra el inicio del programa de Fast SLAM para ASTEC, en la línea 7 comienza la clase “Particle” (partícula), esta clase contiene la mayoría de los métodos y funciones necesarios para implementar el enfoque de Fast SLAM, donde la otra clase que contendrá este programa será la clase FastSLAM. Al inicio de la clase en las líneas 23 a 27 se definen los elementos pertenecientes a esta clase, donde existe el elemento “self.pose” que relaciona la ubicación del robot (x, y, θ) y los demás parámetros

de este objeto son características de las marcas del mapa, donde se almacena su posición, covarianzas y contadores.

Después se establece el primer método estático (staticmethod), de la línea 41 a 55, la idea de usar un método estático es que estos métodos (funciones) están restringidos a su ámbito, es decir, no tienen acceso inherente a los atributos del objeto. Este método estático contiene la función "g()" que representa la aplicación del modelo de movimiento (o función de transición de estado) visto en las ecuaciones (4.2.2.1.3), (4.2.2.1.4) y (4.2.2.1.5) (líneas 51 a 54) que corresponden al movimiento en linea recta, cuando no existe variación de las distancias de la rueda izquierda o derecha, en caso de que exista variación en las distancias de la rueda izquierda y derecha se usa el caso de las líneas 45 a 50, donde se aplican las formulas (4.2.2.1.9) y (4.2.2.1.10) para el cálculo del ángulo (alfa) de rotación (línea 46) y el radio (distancia con respecto al pivote (linea 47), después se aplican las ecuaciones (4.2.2.1.11) con (4.2.2.1.13) en una versión reducida combinando ambas ecuaciones (líneas 48 y 49), y por último, la ecuación (4.2.2.1.12) para el cálculo del nuevo ángulo de orientación en la linea 50. Esta función regresa un *array* que contiene a $g1$, $g2$ y $g3$ que corresponden a (x, y, θ) respectivamente.

La función $g(estado, control, w)$ al ser un método estático, no puede hacer uso de los atributos de la clase "partícula", es por ello que se crea la función "move()", que se encarga de enviar los parámetros y llamar a la función de transición de estado, los parámetros que son enviados a esta función son; estado de la partícula (self.pose), las distancias recorridas por la rueda izquierda y derecha (izq, der) y el ancho del robot (w), esta función además de llamar a la función donde se aplica el modelo de movimiento, modifica el estado anterior de la partícula (self.pose) para generar la etapa de "predicción". La función "move()" no tiene necesidad de regresar ningún dato, dado que esta función modifica directamente el atributo de la "posición de la partícula" (self.pose).

```
1  ### Clase del Filtro Extendido de Kalman (FastSLAM) ###
2
3  ### Esta clase puede interpretarse como el "main()" del
4  ### enfoque de Fast SLAM, donde en esta clase son llamados los
5  ### metodos y funciones de la clase "Particula" (particle), la unica
6  ### funcion extra que se encuentra en esta clase, es la funcion que
7  ### se encarga de realizar el remuestreo de las particulas. ###
8
9  class FastSLAM:
10
11     # Se definen los parametros del filtro, las particulas iniciales
12     # y algunos parametros de calibracion necesarios.
13     def __init__(self, initial_particles,
14                  robot_width, scanner_displacement,
15                  control_motion_factor, control_turn_factor,
16                  measurement_distance_stddev, measurement_angle_stddev,
17                  minimum_correspondence_likelihood):
18
19         # Particulas
```

```
20          self.particles = initial_particles
21
22          ### Constantes de calibracion ###
23
24          # Ancho del robot.
25          self.robot_width = robot_width
26          # Desfase del sensor LIDAR.
27          self.scanner_displacement = scanner_displacement
28          # Error en desplazamiento en linea recta.
29          self.control_motion_factor = control_motion_factor
30          # Error en desplazamiento en giro (un poco mayor al de
31          # linea recta).
32          self.control_turn_factor = control_turn_factor
33          # Error en medicion de distancia (r).
34          self.measurement_distance_stddev = measurement_distance_stddev
35          # Error en angulo de medicion (alpha).
36          self.measurement_angle_stddev = measurement_angle_stddev
37          # Distancia minima de error para correspondencia de marcas
38          # del mapa
39          self.minimum_correspondence_likelihood = \
40              minimum_correspondence_likelihood
41
42      # Paso de prediccion, se calcula la desviacion estantar de
43      # izquierda y derecha, despues, se realiza una distribucion
44      # "random" (de movimiento izquierda y derecha) que es aplicada
45      # a cada particula a traves de la funcion "p.move()" que
46      # pertenece a la clase "Particle".
47      def predict(self, control):
48
49          izq, der = control
50          izq_estandar  = sqrt((self.control_motion_factor * izq)**2 +\
51                               (self.control_turn_factor * (izq - der))**2)
52          der_estandar = sqrt((self.control_motion_factor * der)**2 +\
53                              (self.control_turn_factor * (izq - der))**2)
54
55          # Modifica la lista de particulas, estableciendo un movimiento
56          # pseudo-aleatorio usando una distriucion Gaussiana con
57          # "random.gauss" para cada una de ellas, donde la media se
58          # define como (izq) o (der), mas su desviacion estandar
59          # (izq_estandar) o (der_estandar).
60          for p in self.particles:
61              d_i = random.gauss(izq, izq_estandar)
62              d_d = random.gauss(der, der_estandar)
63              p.move(d_i, d_d, self.robot_width)
```

Algoritmo 4.2.3.1.b. Etapa de predicción de "Fast SLAM", inicio de clase "FastSLAM" y aplicación de función de transición de estado $g()$.

El algoritmo (4.2.3.1.b), contiene el inicio de la segunda clase llamada "FastSLAM", esta clase se implementa para poder llamar los métodos de la clase "partícula" (particle), donde puede entenderse como una especie de función "main()", también debe señalarse que esta clase (FastSLAM) contiene la función para realizar el "re-muestreo" de las partículas (esta función se explicará más adelante en la "etapa de corrección". Ahora, de la línea 24 a la línea 40 se colocan algunas constantes de calibración necesarias para operar la etapa

de la clase “FastSLAM”, donde dichas constantes son necesarias de manera explicita para el funcionamiento de los “filtro extendidos de Kalman” que se aplican a las marcas del mapa. En la línea 47 se tiene la función “predict”, esta función se encarga de la “etapa de predicción”, donde se toman los datos referentes al control, que están a su vez compuestos por las distancias izq y der, después se genera la desviación estándar para izq y der usando la siguiente formula:

(4.2.3.6)
$$\sigma_i = \sqrt{(e_m * izq)^2 + (e_r * (izq - der))^2}$$

(4.2.3.7)
$$\sigma_d = \sqrt{(e_m * der)^2 + (e_r * (izq - der))^2}$$

Donde:

- σ_i es la desviación estándar para la distancia izquierda.
- σ_d es la desviación estándar para la distancia derecha.
- e_m error en movimiento en línea recta “self.control_motion_factor”.
- e_r error en la rotación (movimiento en curva) “self.control_turn_factor”.

Después, en la línea 60 (del algoritmo (4.2.3.1.b)) se aplica la transición de estado para cada una de las partículas en “self.particles”, donde a cada partícula se le asigna una distancia recorrida de la rueda izquierda y derecha (d_i y d_d) de manera “pseudo-aleatoria”, con esas distancias recorridas asignadas se aplica la función “move(d_i, d_d, $self.robot\_width$)” de la clase “particle” (p) para cada partícula existente.

Etapa de Corrección de Fast SLAM

Para la “etapa de corrección” de “Fast SLAM” se tienen los métodos y funciones para realizar las correcciones de la localización de las partículas y las marcas del mapa.

La corrección del enfoque Fast SLAM puede ser dividida en dos secciones, la primer sección se encarga de la actualización y cálculo de los “pesos”, y la función de “re-muestreo” de las partículas.

```
1 # Funcion de correccion, llama a las funciones de calculo de los pesos
2 # y remuestreo
3
4 def correct(self, cilindros):
5
6     pesos = self.update_and_compute_weights(cilindros)
7     self.particles = self.resample(pesos)
```

Algoritmo 4.2.3.1.c. Etapa de corrección de “Fast SLAM”, función “correct” (corrección).

En el algoritmo (4.2.3.1.c) se llaman a las dos secciones para la etapa de corrección de Fast SLAM, donde primero se calculan los “pesos” por partícula usando el método “self.update_and_compute_weights” por medio de los datos obtenidos de los cilindros (empleando la biblioteca de “funciones_apoyo”), posteriormente se realiza la “depuración” de las partículas con el método “self.resample” utilizando la lista de pesos (por partícula) generados.

```
1  # Funcion que realiza la actualizacion y calculo de los pesos,
2  # basicamente llama a las funciones de la clase "Particle",
3  # por cada una de las particulas, despues se realiza un loop
4  # sobre todas las mediciones del sensor LIDAR y mediciones en
5  # el sistema de coordenas del sensor (que fueron detectados en
6  # el vector de "cilindros"), se calcula el peso global por
7  # particula y lo agregamos a un arreglo de pesos, despues
8  # llamamos a la funcion para remover las marcas del mapa
9  # erroneas si es el caso.
10 def update_and_compute_weights(self, cilindros):
11
12     Qt_covarianza_medicion = \
13         np.diag([self.measurement_distance_stddev**2,
14                  self.measurement_angle_stddev**2])
15     pesos = []
16     for p in self.particles:
17
18         p.decrement_visible_landmark_counters(\
19             self.scanner_displacement)
20
21         numero_marcas = p.number_of_landmarks()
22         peso = 1.0
23
24         for medicion, medicion_sistema_lidar in cilindros:
25             peso *= p.update_particle(
26                 medicion, medicion_sistema_lidar,
27                 numero_marcas,
28                 self.minimum_correspondence_likelihood,
29                 Qt_covarianza_medicion, desplazamiento_sensor)
30
31         pesos.append(peso)
32         p.remove_spurious_landmarks()
33
34     return pesos
```

Algoritmo 4.2.3.1.d. Etapa de corrección de “Fast SLAM”, función de actualización y calculo de “pesos” por partícula.

El algoritmo (4.2.3.1.d) contiene, de manera general, la etapa de actualización del enfoque Fast SLAM que corresponde a la aplicación de los “filtro extendidos de Kalman”. En la línea 12 se calcula la matriz de covarianza de la medición, esta matriz contiene la desviación estándar (error o incertidumbre) en la distancia de medición y la

desviación estándar con respecto al ángulo de la medición (del sensor LIDAR). Después, en la línea 16 se realiza un ciclo por cada partícula en "self.particles" donde por cada partícula (en la linea 18) se decrementa el contador para las marcas visibles "p.decrement_visible_landmark_counters", después del decremento de los contadores de las marcas, en la línea 24 se genera otro ciclo sobre todas las marcas detectadas (cilindros en el mapa), dentro de este ciclo se llama al método (de la clase particle) "p.update_particle" donde se envían los datos de la medición (distancia y ángulo), medición_sistema_lidar (coordenadas x, y en el sistema de coordenadas del sensor láser), numero_marcas que representa al número de cilindros detectados en ese "ciclo de medición", la matriz de covarianza Q_t y el desplazamiento del sensor LIDAR (scanner_displacement). Finalmente, se agregan los pesos calculados por "p.update_particle" a la lista pesos y se llama al método "p.remove_spurious_landmarks" que depura las posibles marcas de mapa erróneas que fueron detectadas.

```
1  # Clase "Particle"
2  # Funcion para decrementar el contador de las marcas que estan
3  # dentro de el rango de vision porpuesto en esa posicion para
4  # el escaner, en caso de la marca "antes vista" no este dentro
5  # del rango, la funcion no altera el valor de su contador.
6  def decrement_visible_landmark_counters(self, desplazamiento_escaner):
7
8      ang_min, ang_max = LegoLogfile.min_max_bearing()
9      conta = self.landmark_counters
10
11     for i in range(len(self.landmark_counters)):
12
13         dist, ang_marca = self.h_expected_measurement_for_landmark(i,\
14                                                 desplazamiento_escaner)
15
16         if ang_min < ang_marca < ang_max:
17             self.landmark_counters[i] = self.landmark_counters[i] - 1
18         else:
19             self.landmark_counters[i] = self.landmark_counters[i]
20
21     return self.landmark_counters
```

Algoritmo 4.2.3.1.e. Etapa de corrección de "Fast SLAM", función de decremento de contadores de marcas de mapa.

Como se mostró en algoritmo (4.2.3.1.d), la primer función que es ejecutada es la "actualización de pesos" (update_and_compute_weights) es la función de "decrement_visible_landmark_counters", esta función se encuentra en el algoritmo (4.2.3.1.e). La función tiene como propósito decrementar los contadores referentes a las marcas del mapa, estos contadores tienen la finalidad de remover posibles "marcas falsas" que se hayan podido mal interpretar en la detección de cilindros u otras funciones, como primer paso en la linea 8 se obtienen los ángulos máximo y mínimo

(ang_min, ang_max) del sensor LIDAR, si bien el sensor LIDAR que se pretende para ASTEC cuenta con un rango de mediciones de 360 grados, existen zonas que se prevén no accesibles para el sensor debido a la colocación de hardware para otras tareas, es por ello que este factor de ángulo máximo y mínimo se debe contemplar (estos valores pueden también omitirse en caso de que se tenga una visión sin obstáculos del robot hacia el mapa), una vez que se tienen los parámetros de "visibilidad" del robot se realiza un *loop* sobre todos los contadores de marcas de mapa (este número, corresponde al número de marcas en el mapa), en la línea 13 se realiza el cálculo de la "distancia esperada" (usando la función "self.h_expected_measurement_for_landmark"), entonces en la línea 15 esta la condición que si dicha marca se encuentra dentro del rango de visibilidad del sensor se decrementa (-1) al contador de dicha marca, y si no se cumple esta condición, en la línea 17, el contador de la marca no se ve afectado.

```
1  # Funcion de "update_particle" (actualizacion de particula), basicamente
2  # hace el proceso de integrar las funciones de correccion (para cada
3  # particula) y verifica si el vector de "probabilidad de
4  # correspondencia" establece si alguna marca debe ser actualizada o
5  # debe agregarse una nueva marca (cuando esta "probabilidad de
6  # correspondencia" se encuentra por debajo de un valor minimo).
7
8  def update_particle(self, medicion, medicion_sistema_lidar,\
9                      numero_marcas, prob_min_correspondecia,\
10                     Qt_covarianza_medicion, desplazamiento_sensor):
11
12     prob_correspondecia = self.compute_correspondence_likelihoods(\
13         medicion, numero_marcas, Qt_covarianza_medicion,\
14         desplazamiento_sensor)
15
16     if not prob_correspondecia or max(prob_correspondecia) <
       prob_min_correspondecia:
17
18         self.initialize_new_landmark(medicion_sistema_lidar,\
19                                      Qt_covarianza_medicion,\
20                                      desplazamiento_sensor)
21         extension = [1]
22         self.landmark_counters.extend(extension)
23
24         return prob_min_correspondecia
25
26     # Si no se genero nueva marca, se calcula un "peso" (w) que
27     # corresponde al valor maximo encontrado en el vector de
28     # probabilidades de correspondecia (prob_correspondecia),
29     # y se actualiza el contador de las marcas para el "indice"
30     # que corresponde al valor maximo encontrado en las pc
31     # (probabilidades de correspondecia).
32     else:
33
34         w = max(prob_correspondecia)
35         indice_max = prob_correspondecia.index(w)
36         self.update_landmark(indice_max, medicion,\
37                              Qt_covarianza_medicion,\
38                              desplazamiento_sensor)
```

```
39
40          self.landmark_counters[indice_max] = self.landmark_counters[\
41              indice_max] + 1
42
43          return w
```

Algoritmo 4.2.3.1.f. Etapa de corrección de "Fast SLAM", función "update_particle" (actualización de partícula).

La línea 25 del algoritmo (4.2.3.1.d) establece el llamado a la función "update_particle" implementada en el algoritmo (4.2.3.1.f), esta función realiza la actualización de las marcas y contadores de las marcas (por partícula), donde en la linea 12 se llamada a la función "self.compute_correspondence_likelihoods" donde se calculan las probabilidades de que las mediciones realizadas correspondan o no a alguna marca del mapa.

En la línea 16 (del algoritmo (4.2.3.1.f)) se tiene la condición para inicializar una nueva marca en caso de que el elemento con el valor máximo de correspondencia ($max(prob\_correspondencia)$) en el vector de probabilidades de correspondencia "prob_correspondencia" sea mayor que el valor mínimo de correspondencia, es decir, que no se encuentra dentro del área de incertidumbre asociada para dicha marca del mapa.

En caso de que la condición de la línea 16 no se cumpla, en la línea 32 se condiciona a que el elemento máximo en el vector de probabilidades de correspondencia (que se refiere a la posible localización de alguna marca en el mapa) actualiza dicha marca (usando el índice) con la función "self.update_landmark" y posterior a ello aumentar el contador de esa marca +2.

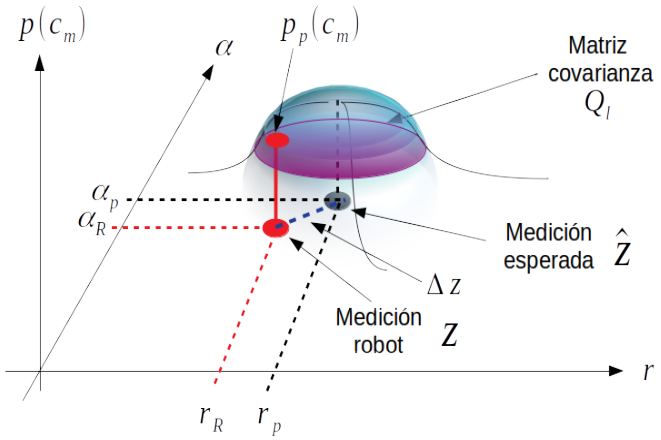


Ilustración 4.2.3.1. Cálculo de probabilidad de correspondencia $p(c_m)$ a marca de mapa m.

En la ilustración 4.2.3.1 se ejemplifica el cálculo de la "probabilidad de correspondencia" (ésta imagen será usada para explicar algunos algoritmos a continuación), esta probabilidad de correspondencia $p_p(c_m)$ (punto rojo "superior") se refiere a la distancia de que la proyección de la probabilidad de la medición "real" (de una marca del mapa) realizada por el robot (medición robot Z) se encuentre dentro de un valor mínimo de correspondencia establecido "prob_min_correspondecia" hacia la medición "esperada" (por partícula) $\hat{Z}$.

```
# Esta funcion calcula para cada medicion dada una lista de
# correpondencias entre todas las marcas de mapa detectadas "a priori",
# generanado un loop (sobre todas las marcas detectadas) para
# calcular la "probabilidad de correspondecia" hacia las marcas.

def compute_correspondence_likelihoods(self, medicion,\
                                       numero_de_marcas,\
                                       Qt_covarianza_medicion,\
                                       desplazamiento_sensor):

    probabilidades_cor = []
    for i in range(numero_de_marcas):
        probabilidades_cor.append(self.wl_likelihood_of_correspondence\
                            (medicion, i, Qt_covarianza_medicion,\
                             desplazamiento_sensor))
    return probabilidades_cor

# En esta funcion calculamos la probabilidad de correspondencia de que
# la medicion hecha corresponda a una marca de mapa (usamos la funcion
# que cacula la distancia y angulo de prediccion), esta funcion regresa
# un valor que es una probabilidad de correspondecia entre la medicion
# y la medicion predicha a la marca de mapa.

def wl_likelihood_of_correspondence(self, medicion,\
                                    indice_marca,\
                                    Qt_covarianza_medicion,\
                                    desplazamiento_sensor):

    cal_r, cal_alpha = medicion
    es_r, es_alpha = self.h_expected_measurement_for_landmark\
                     (indice_marca, desplazamiento_sensor)

    delta_z = np.array([(cal_r - es_r), (cal_alpha - es_alpha)])

    H, Ql = self.H_Ql_jacobian_and_measurement_covariance_for_landmark\
            (indice_marca, Qt_covarianza_medicion,\
             desplazamiento_sensor)

    term_1 = (1 / ((2*pi) * (sqrt(np.linalg.det(Ql)))))
    term_2 = exp((-1 / 2) * np.dot(delta_z.T, np.dot(np.linalg.inv(Ql),
    delta_z)))

    probabilidad_correspondencia = term_1 * term_2

    return probabilidad_correspondencia
```

Algoritmo 4.2.3.1.g. Etapa de corrección de "Fast SLAM", cálculo de probabilidades de correspondencia.

En el algoritmo (4.2.3.1.g) se expresan las dos funciones aplicadas para el cálculo de las "probabilidades de correspondencia", en estas funciones se aplican las siguientes ecuaciones:

(4.2.3.8)

$$\hat{Z} = h((x_p, y_p, \theta_p), m_n)$$

(4.2.3.9)

$$H = \frac{\partial h}{\partial m_n}\Big|_{(x_p, y_p, \theta_p, m_n)}$$

(4.2.3.10)

$$Q_l = H\Sigma_n H^T + Q_t$$

- $h()$ es el modelo de observación implementado para ASTEC.
- (x_p, y_p, θ_p) es el "estado" o "posición" de la partícula p que se esta actualizando.
- H es la matriz jacobiana (que contiene las derivadas parciales) del modelo de observación y la marca de mapa n, la matriz es de dimensiones (2 x 2).
- Q_l corresponde a la covarianza de la medición de la "marca de mapa" n, esta matriz en su termino $H\Sigma_n H^T$ sufre una propagación de varianza, donde Σ_n es la varianza que posee la marca n, pero al multiplicarla por H y H^T se dice que se genera una "varianza de medición" dada por la "varianza de marca" [50], y finalmente, se le suma la "varianza" (o incertidumbre) asociada al proceso de medición Q_t.
- Σ_n es la covarianza de la "marca de mapa" en (x, y) .
- Q_t es la matriz de covarianza de la medición (puede también interpretarse como la incertidumbre asociada al proceso de medición del sensor LIDAR u otro sensor), donde puede depender del tiempo t, sin embargo, para este enfoque se tomará como una matriz constante $\begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\alpha^2 \end{bmatrix}$.

En la línea 6 se tiene la función "compute_correspondence_likelihoods" (cálculo de probabilidades de correspondencia), esta función realiza un ciclo sobre todas las marcas de mapa detectadas hasta ese instante (línea 12) donde por cada marca de mapa i se realiza el cálculo la probabilidad de correspondencia con "self.wl_likelihood_of_correspondence". Después, en la línea 24 se tiene (propiamente) la función para el cálculo de correspondencia "self.wl_likelihood_of_correspondence" (por numero de marca), donde en la línea 29 se

colocan los datos "reales" de la medición (ilustración (4.2.3.1) "medición robot Z"), en las líneas 30 y 31 se realiza el cálculo de la "medición esperada" con la ecuación (4.2.3.8) (usando el modelo de medición u observación de ASTEC) que puede verse en la ilustración (4.2.3.1) como la medición esperada $\hat{Z}$, luego se calcula la diferencia entre la medición y la medición esperada (línea 33) en "delta_z" (ecuación (4.2.3.11)), y posteriormente se calcula la matriz jacobiana H y la matriz de covarianza Q_l (que corresponde al círculo de color morado en la ilustración anterior) en las líneas 35 a 37. Por último, de la línea 39 a 42 se calcula la "probabilidad de correspondencia" l_{cor} usando las ecuaciones (4.2.3.9), (4.2.3.10) y (4.2.3.12) (en la ilustración (4.2.3.1) la probabilidad de correspondencia estaría representada por $p_p(c_m)$).

(4.2.3.11)

$$\Delta Z = Z - \hat{Z}$$

(4.2.3.12)

$$l_{cor} = \frac{1}{2\pi\sqrt{det(Q_l)}}\, e^{-\frac{1}{2}\Delta Z^T Q_l^{-1} \Delta Z}$$

- ΔZ corresponde a la diferencia de la medición "real" Z y la medición esperada (aplicando el modelo de observación) $\hat{Z}$.
- l_{cor} es el valor de la proyección de la "probabilidad de correspondencia" en la matriz de covarianza Q_l.

En las líneas 30 y 31 del algoritmo (4.2.3.1.g) se llama a "medición esperada de marca" (self.h_expected_measurement_for_landmark), esta función se implementa en el algoritmo (4.2.3.1.h).

```
1  # Metodo Estatico #2
2  # Se basa en formular el modelo de observacion del sensor del robot,
3  # esta dado en base a la transformacion entre la posicion del robot
4  # y la informacion que se obtiene del sensor laser, la funcion
5  # regresa un rango (r) y un angulo alpha (angulo del sensor hacia
6  # la marca).
7  @staticmethod
8  def h(estado, marca, desplazamiento_sensor):
9
10     dx = marca[0] - (estado[0] + desplazamiento_sensor *\
11                      cos(estado[2]))
12     dy = marca[1] - (estado[1] + desplazamiento_sensor *\
13                      sin(estado[2]))
14     r = sqrt(dx * dx + dy * dy)
15     alpha = (atan2(dy, dx) - estado[2] + pi) %(2*pi) - pi
16     return np.array([r, alpha])
17
18 # Funcion para el calculo de la medicion "esperada" hacia cierta
19 # marca de mapa, esta funcion se basa en proponer una distancia
```

```
20 # y angulo esperado en funcion a la marca detectada y la pose
21 # de una particula en especifico, que despues se usara como
22 # parametro para comparar la distancia detectada "real" por el
23 # sensor, esta seccion tambien es conocida como la
24 # Distancia de Mahalanobis.
25 def h_expected_measurement_for_landmark(self, numero_marca,\
26                                         desplazamiento_sensor):
27
28     z_esperada = self.h(self.pose,\
29                         self.landmark_positions[numero_marca],\
30                         desplazamiento_sensor)
31     return z_esperada
```

Algoritmo 4.2.3.1.h. Etapa de corrección de "Fast SLAM", función modelo de observación.

El algoritmo (4.2.3.1.h) aplica el segundo método estático, en las líneas 7 a 16, este método corresponde a la aplicación del modelo de observación para ASTEC basado en las ecuaciones (4.2.2.2.2) y (4.2.2.2.3), en las líneas 10 y 13 se realiza el cálculo de la distancia entre la marca en (x_m, y_m) con respecto de la posición de la partícula (el cálculo de la medición se hará para cada partícula) (x, y), después estas distancias se elevan al cuadrado y se toma la raíz cuadrada de la suma en la línea 14, en la línea 15 se realiza el cálculo del ángulo alfa (α), este ángulo corresponde al ángulo entre la marca y el ángulo de orientación (ilustración 4.2.2.2.1) del robot θ en ese instante, la idea de aplicar la operación de $mod(2\pi)$ es mantener el ángulo entre 2π. La función $h()$ regresa un *array* (tipo numpy) con el cálculo de la distancia a la marca r y el ángulo de "apertura" α (alfa).

La función "h_expected_measurement_for_landmark" es la función encargada de llamar al método estático $h()$ para realizar el cálculo de la distancia "esperada" (y ángulo) de la posición de la marca de mapa con respecto a cierta partícula $\hat{Z}$, esta medición se llama "esperada" debido a que es una medición tentativa que depende de la posición generada para la partícula analizada en ese instante, donde posiblemente no corresponde a la distancia que detecte de manera real el sensor LIDAR, ya que no todas las partículas concuerdan con la distancia r y ángulo α de ese instante de tiempo.

En las líneas 35, 36 y 37 del algoritmo (4.2.3.1.g) se realiza el calculo de H y Q_l llamando a la función "self.H_Ql_jacobian_and_measurement_cov..." programada en el algoritmo (4.2.3.1.i).

```
1 # Se calcula la matriz jacobiana (H = 2 x 2) de la funcion h() en base
2 # a la funcion "dh_dlandmark", donde se usa la pose de la particula,
3 # la posicion de la marca y el desplazamiento del escaner, ademas,
4 # se calcula la matriz de covarianza (Ql) de la medicion de la marca
5 # de mapa, en el caso de FAST_SLAM, se considera que las matrices de
6 # las marcas de mapa son independientes.
7
8 def H_Ql_jacobian_and_measurement_covariance_for_landmark( self,
    numero_marca, Qt_covarianza_medicion, desplazamiento_sensor):
```

```
9
10      H = self.dh_dlandmark(self.pose,\
11                            self.landmark_positions[numero_marca],\
12                            desplazamiento_sensor)
13
14      Ql = np.dot(H, (np.dot(\
15          self.landmark_covariances[numero_marca], H.T))) +\
16          Qt_covarianza_medicion
17
18      return (H, Ql)
19
20  # Metodo Estatico #3
21  # Se encarga de generar las derivadas parciales de la matriz Jacobiana
22  # "H", estas derivadas parciales se calculcan a partir de la funcion
23  # "h()" con respecto de las marcas de mapa (Xm, Ym).
24  @staticmethod
25  def dh_dlandmark(estado, marca, desplazamiento_sensor):
26
27      theta = estado[2]
28      cost, sint = cos(theta), sin(theta)
29      distancia_laser_x = estado[0] + desplazamiento_sensor*cost
30      distnacia_laser_y = estado[1] + desplazamiento_sensor*sint
31      dx = marca[0] - (distancia_laser_x)
32      dy = marca[1] - (distancia_laser_y)
33      q = dx * dx + dy * dy
34      dist_laser_marca = sqrt(q)
35      dr_dmx = dx / dist_laser_marca
36      dr_dmy = dy / dist_laser_marca
37      dalpha_dmx = -dy / q
38      dalpha_dmy =  dx / q
39
40      return np.array([[dr_dmx, dr_dmy],
41                       [dalpha_dmx, dalpha_dmy]])
```

Algoritmo 4.2.3.1.i. Etapa de corrección de "Fast SLAM", calculo de matriz jacobiana H, matriz de covarianza Q_l y cálculo de derivadas parciales.

La primera parte del algoritmo (4.2.3.1.i) en la línea 10 se llama a la función "self.dh_dlandmark", que se encarga de calcular las derivadas parciales del modelo de observación $h()$ con respecto de la posición de la marca de mapa (x_m, y_m), después en la línea 14 (usando la matriz H calculada) se calcula la matriz de covarianza Q_l aplicando la ecuación (4.2.3.10).

(4.2.3.13)

$$H = \frac{\partial h(r,\alpha)}{\partial m(x_m, y_m)} = \begin{bmatrix} \frac{\partial r}{\partial x_m} & \frac{\partial r}{\partial y_m} \\ \frac{\partial \alpha}{\partial x_m} & \frac{\partial \alpha}{\partial y_m} \end{bmatrix}$$

Donde:

- H es la matriz Jacobiana, dimensión (2 x 2), que relaciona la "variación" del modelo de movimiento $h()$ con respecto a la ubicación de las marcas de mapa (x_m, y_m).

- $h(r, \alpha)$ es el modelo de movimiento, donde las variables que la función $h()$ opera, son r (distancia a la marca) y α (ángulo de apertura a la marca, tomando como referencia el ángulo de orientación del robot "θ").

- $m(x_m, y_m)$ es la ubicación de las marcas del mapa en plano (x, y).

El cálculo de las derivadas parciales de $h()$ con respecto de las marcas del mapa m, es una forma de linealizar al modelo de observación ($h()$) usando "series de Taylor" de primer orden (propiamente estas derivadas parciales), estas derivadas permiten conocer las variaciones que existen entre las variables "r" y "α" del modelo de observación, con respecto de las marcas del mapa (x_m, y_m). Un aspecto importante de esta matriz "H" es en el enfoque de EKF SLAM (SLAM usando filtros de Kalman extendidos) donde existe una matriz que relaciona las variaciones de la función $h()$ con respecto del *estado* (sin tomar en cuenta la orientación del robot θ) e internamente esta matriz esta compuesta por la matriz $((-1) * H)$ (es la matriz H multiplicada por (-1) en todos sus elementos) y la matriz H calculada en esta función, donde se puede establecer como la localización del robot es dada con respecto de las marcas (en la sub-matriz $((-1) * H)$) y como la localización de las marcas es dada con respecto al robot (en la submatriz H).

Entonces se tiene que, las derivadas parciales de la matriz Jacobiana "H" son:

(4.2.3.14)
$$\frac{\partial r}{\partial x_m} = \frac{x_m - x_l}{\sqrt{(x_m - x_l)^2 + (y_m - y_l)^2}}$$

(4.2.3.15)
$$\frac{\partial r}{\partial y_m} = \frac{y_m - y_l}{\sqrt{(x_m - x_l)^2 + (y_m - y_l)^2}}$$

(4.2.3.16)
$$\frac{\partial \alpha}{\partial x_m} = \frac{(-1) * (y_m - y_l)}{(x_m - x_l)^2 + (y_m - y_l)^2}$$

(4.2.3.17)
$$\frac{\partial \alpha}{\partial x_m} = \frac{x_m - x_l}{(x_m - x_l)^2 + (y_m - y_l)^2}$$

Donde:

- (x_m, y_m) corresponden a las ubicaciones de las "marcas" del mapa.

- (x_l, y_l) es la ubicación del láser con respecto al centro del robot y al desplazamiento del sensor LIDAR.

- r es la distancia de medición del LIDAR a cierta marca de mapa.

- α es el ángulo entre la marca de mapa y el ángulo de orientación del robot θ.

Una vez calculadas las derivadas parciales en el algoritmo (4.2.3.1.i), se pueden establecen los valores para la matriz H y la matriz de covarianza Q_l, este proceso se repite por cada marca de mapa. Después, con estos datos se pueden calcular las probabilidades de correspondencia con el algoritmo (4.2.3.1.g), una vez que se tienen todos los valores de correspondencia, regresando al algoritmo (4.2.3.1.f) en la línea 16 se pregunta si el elemento máximo ($max(prob\_correspondecia)$) de los valores calculados en el vector de "prob_correspondencia" es menor al valor de correspondencia mínimo ($prob\_min\_correspondencia$), si esta condición se cumple, entonces esa "marca detectada" no corresponde a una marca de mapa ya establecida, entonces se inicializa una nueva marca de mapa con "self.initialize_new_landmark", y se extiende el vector de los "contadores" de las marcas del mapa (en la linea 22) con "self.landmark_counters.extend". En caso contrario de que la condición de correspondencia mínima sea mayor que la probabilidad mínima de correspondencia (es decir, que se encuentra cercano a una marca de mapa) se actualiza la marca de mapa a la que corresponda esa asignación.

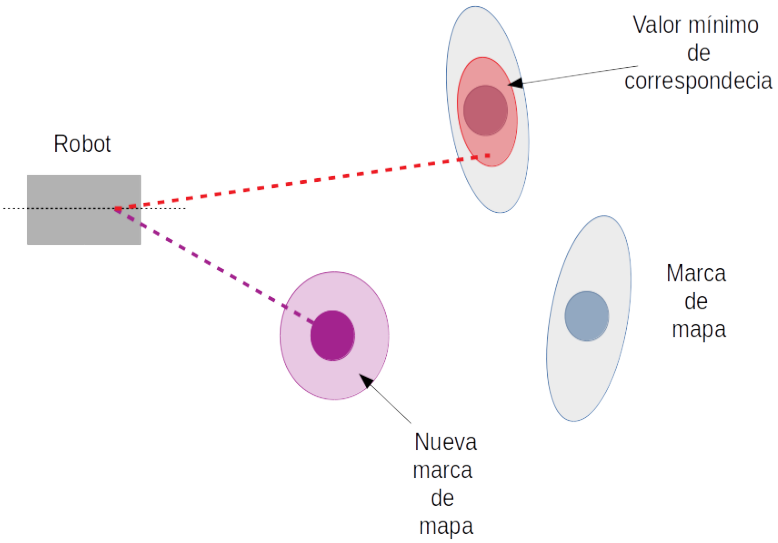


Ilustración 4.2.3.2. Inicialización de "nueva marca" o actualización de marca de mapa.

En la ilustración (4.2.3.2) se ejemplifica este fenómeno de inicialización de nueva marca (en color morado) cuando el elemento con el valor máximo del arreglo de "probabilidades de correspondencia" sea menor al valor mínimo de pertenencia establecido, en color rojo se tiene el caso cuando el valor del elemento máximo es mayor al valor mínimo de pertenecía establecido (se realiza la actualización de la marca).

```
1  # Funcion para inicar una nueva marca de mapa, esta funcion calcula
2  # la pose del escaner en funcion de la posicion del robot,
3  # posteriormente realiza una funcion inversa para calcular la
4  # posicion de la marca del mapa dada la posicion de escaner y la
5  # medicion en el sistema de coordenadas del escaner, luego
6  # calculamos las matrices jacobianas H de las marcas, que por la
7  # teoria de observo que corresponden a la matriz inversa de H
8  # (que es la matriz jacobiana de la funcion h() con respecto del
9  # estado "x, y, theta" y marca "mx, my"), despues se calcula la
10 # covarianza de la marca, por ultimo, la funcion regresa la nueva
11 # posicion de la marca y la nueva covarianza de la marca (NUEVA).
12
13 def initialize_new_landmark(self, medicion_sistema_lidar,\
14                              Qt_covarianza_medicion,\
15                              desplazamiento_sensor):
16
17     pose_sensor = (self.pose[0] + cos(self.pose[2]) *\
18                    desplazamiento_sensor,
19                     self.pose[1] + sin(self.pose[2]) *\
20                    desplazamiento_sensor,
21                     self.pose[2])
22
23     marca_mapa = LegoLogfile.scanner_to_world(pose_sensor,\
24                                               medicion_sistema_lidar)
25     H = self.dh_dlandmark(self.pose, marca_mapa, desplazamiento_sensor)
26     H_inv = np.linalg.inv(H)
27     covarianza_marca = np.dot(H_inv, np.dot(Qt_covarianza_medicion H_inv.T))
28
29     self.landmark_positions.append(marca_mapa)
30     self.landmark_covariances.append(covarianza_marca)
```

Algoritmo 4.2.3.1.j. Etapa de corrección de "Fast SLAM", inicialización de marca de mapa.

En el algoritmo (4.2.3.1.j) se tiene el programa para la "inicialización" de una nueva marca de mapa. En la línea 17 se realiza el cálculo de la pose del sensor, esta posición se obtiene sumando el "desfase" del sensor LIDAR con respecto del centro del robot, usando la posición del robot se calcula en la línea 23 la posición de la "marca_mapa" usando una función que proporciona *Claus Brenner* del curso "SLAM Lectures" [50] llamada "LegoLogfile.scanner_to_world()", este método aplica la función inversa del modelo de observación $m(x, y) = h^{-1}((x_s, y_s, \theta_s), z)$ que obtiene las coordenadas en (x, y) de la marca de mapa usando la posición del sensor (x_s, y_s, θ_s) y la medición esperada z. En la línea 25 se calcula la matriz jacobiana H $(\frac{\partial h}{\partial m_n})$, en la línea 26 se calcula el inverso de la matriz H denotada "H_inv" (H^{-1}) usando la función de numpy "np.linalg.inv()", una vez obtenido H^{-1} se aplica la ecuación (4.2.3.18).

(4.2.3.18)

$$\Sigma = H^{-1} Q_t (H^{-1})^T$$

Donde:

- H^{-1} es el inverso de la matriz H.
- Q_t es la matriz de varianza asociada al proceso de medición, es decir, la varianza en r y α debido a la medición z.

El calculo de H^{-1} se realiza para poder establecer la "incertidumbre asociada" Σ (matriz de covarianza de la marca) a una determinada marca de mapa, donde el error en el proceso de medición Q_t se propaga en función de H^{-1} (en la ecuación (4.2.3.18)). Finalmente, en el algoritmo (4.2.3.1.j) en las líneas 29 y 30 se agrega la posición de la "nueva marca" del mapa (línea 29) y la matriz de covarianza asociada a la marca (línea 30).

En caso de que la condición de "correspondencia mínima" del algoritmo (4.2.3.1.f) se cumpla (linea 32 - linea 41), entonces se toma el valor máximo y el índice (de dicho valor máximo) en el vector de "probabilidades de correspondencia" para llamar al método "self.update_landmark()" para "actualizar" la marca de mapa en función al valor de correspondencia máximo.

```
1  # Funcion de actualizacion de la marca, en base a la medicion generada,
2  # el numero de marca, la pose de la particula, la covarianza de
3  # medicion (fija) y el desplazamiento del escaner, calculamos las
4  # matrices H, Ql, la cov_vieja (covarianza anterior) y la
5  # mu_vieja (media anterior), calculamos la ganancia para el filtro de
6  # Kalman (K), se calcula la inovacion "delta_z", que es la resta entre
7  # la medicion y la medicion esperada a la marca, calculamos los nuevos
8  # parametros (nueva_mu, nueva_covarianza), y suplantamos las posicones
9  # y covarianzas de dichas marcas en el "indice" del array que
10 # corresponda a la marca corregida.
11
12 def update_landmark(self, numero_marca, medicion,\
13                      Qt_covarianza_medicion, desplazamiento_sensor):
14
15     H, Ql = self.H_Ql_jacobian_and_measurement_covariance_for_landmark(\
16         numero_marca, Qt_covarianza_medicion, desplazamiento_sensor)
17     inv_Ql = np.linalg.inv(Ql)
18     cov_vieja = self.landmark_covariances[numero_marca]
19     mu_vieja = self.landmark_positions[numero_marca]
20
21     ganancia_K = np.dot(cov_vieja, np.dot(H.T, inv_Ql))
22     delta_z = medicion - self.h_expected_measurement_for_landmark(\
23         numero_marca, desplazamiento_sensor)
24
25     nueva_mu =  mu_vieja + np.dot(ganancia_K, delta_z)
26     k_h = np.dot(ganancia_K, H)
27     nueva_covarianza = np.dot((np.eye(len(k_h)) - k_h), cov_vieja)
28
29     self.landmark_covariances[numero_marca] = nueva_covarianza
30     self.landmark_positions[numero_marca] = nueva_mu
```

Algoritmo 4.2.3.1.k. Etapa de corrección de "Fast SLAM", actualización de marca de mapa.

La actualización de la marca de mapa esta escrita en el algoritmo (4.2.3.1.k), esta actualización se realiza aplicando filtros extendidos de Kalman para cada marca de mapa usando las ecuaciones (2.2.46), (2.2.47) y (2.2.48) escritas en el marco teórico. Estas ecuaciones "corrigen" la posición $\bar{\mu}$ y la covarianza $\bar{\Sigma}$ a una nueva posición de la marca μ y una nueva covarianza (elipse de error) Σ, donde la posición y la covarianza pasadas están dadas por las coordenadas en (x, y) previas de cada marca "self.landmark_positions" y la covarianza Σ_m que se tenia en el vector de covarianzas "self.landmark_covariances".

Para la actualización de la posición e incertidumbre de la marca de mapa (algoritmo (4.2.3.1.k)) en la línea 15 se calcula la matriz H y la matriz Q_l (la matriz Q_l corresponde al elemento entre paréntesis del cálculo de la ganancia de Kalman K en la ecuación (2.2.46)), en la línea 17 se calcula el inverso de la matriz Q_l usando la función de numpy "np.linalg.inv()". En las líneas 18 y 19 se extrae la posición y covarianza previa de la marca de mapa en función del número de marca que se este actualizando, posteriormente, se calcula la "ganancia_K" (ganancia de Kalman) en la línea 21, seguida (en la línea 22) del cálculo de "delta_z" que es la diferencia entre la medición hecha por el sensor LIDAR y la medición esperada según la pose de la partícula analizada y la posición previa que se conocía de la marca de mapa. Finalmente, en las líneas 25 a 27 se realiza la actualización a la "nueva_mu" y la "nueva_covarianza" de la marca de mapa, μ y Σ son agregados en "reemplazo" de la posición anterior y covarianza anterior de la marca de mapa "numero_marca" en los vectores correspondientes.

La penúltima parte de la "actualización de partícula" *update_particle* del algoritmo (4.2.3.1.f), en específico de la línea 41, es la actualización de los contadores de marcas de mapa "self.landmark_counters", es una propiedad que ayuda a evitar la generación de "marcas erróneas" de mapa, esto se relaciona con las líneas 18 y 19 del algoritmo (4.2.3.1.d) donde al inicio del cómputo y actualización de los pesos por partícula, existe un decremento de las marcas visibles del mapa, es decir, marcas que se encuentran dentro de la zona de censado del LIDAR. Este proceso de decrementar e incrementar los contadores de las marcas del mapa, permite evitar problemas cuando existen inicializaciones de marcas nuevas de mapa que posiblemente se deban a errores de medición. La función que se encarga de realizar la depuración de las marcas de mapa se llama "remove_spurious_landmarks".

```
1 # Esta funcion remueve las marcas de mapa con contadores negativos,
2 # es decir que despues de decrementar sus contadores porque no
3 # fueron vistas (- 1), o cuando su contador fue aumentado (+ 1),
4 # nos arroja un valor negativo para las marcas erroneas del mapa,
5 # entonces dichas marcas son removidas, junto con su posicion, y
6 # covarianza correspondientes.
7
8 def remove_spurious_landmarks(self):
9
```

```
10      numero_marcas = len(self.landmark_counters)
11
12      posiciones = self.landmark_positions
13      covarianzas = self.landmark_covariances
14      contadores = self.landmark_counters
15
16      for i in range(len(self.landmark_counters)-1, -1, -1):
17
18          if self.landmark_counters[i] < 0:
19
20              del posiciones[i]
21              del covarianzas[i]
22              del contadores[i]
23
24      self.landmark_positions = posiciones
25      self.landmark_covariances = covarianzas
26      self.landmark_counters = contadores
```

Algoritmo 4.2.3.1.l. Etapa de corrección de "Fast SLAM", remover marcas erróneas de mapa.

En el algoritmo (4.2.3.1.l) se remueven las marcas erróneas de mapa, esta función realiza un ciclo sobre el número de marcas que se tienen detectadas por partícula (línea 16), donde si el contador de alguna o algunas marcas de mapa (línea 18) contiene un valor de su contador propio < 0 entonces ese elemento es eliminado de los parámetros (vectores) de la clase "particle", y finalmente, de la línea 24 a 25, se actualizan los valores de las marcas que no fueron removidas a los parámetros de la clase.

```
1  # Funcion para hacer el remuestreo, esta basado en el algoritmo de
2  # "Remuestreo de la Rueda", basicamente depura las particulas que
3  # tuvieron un menor peso global, y reproduce las particulas con
4  # mayor peso, el peso esta dado por la relacion entre las mejores
5  # mediciones hechas por cada particula.
6
7  def resample(self, pesos):
8
9      nuevas_particulas = []
10     peso_maximo = max(pesos)
11     indice = random.randint(0, len(self.particles) - 1)
12     offset = 0.0
13
14     for i in range(len(self.particles)):
15         offset += random.uniform(0, 2.0 * peso_maximo)
16         while offset > pesos[indice]:
17             offset -= pesos[indice]
18             indice = (indice + 1) %len(pesos)
19         nuevas_particulas.append(copy.deepcopy(self.particles[indice]))
20
21     return nuevas_particulas
```

Algoritmo 4.2.3.1.m. Etapa de corrección de "Fast SLAM", función de re-muestreo para las partículas.

Una vez generados los pesos acumulados para cada partícula (usando las precisión de sus marcas detectadas) se realiza un proceso de "re-muestreo" (resampling), que tiene el objetivo de replicar o reproducir las partículas con un mayor peso acumulado, este proceso es muy similar a los usados en algoritmos genéticos. La función mostrada en el algoritmo (4.2.3.1.m) esta basado en el método de "resampling wheel" (re-muestreo de la rueda) que supone la creación de un círculo (rueda) que contiene áreas asociadas para cada partícula según el peso que estas tengan, donde se realizan saltos aleatorios a partir de algún índice dentro del número de elementos (línea 11) y un offset aleatorio que se encuentre entre $(0, 2 * peso\_maximo)$ (líneas 10, 12, 15), cada salto es tomado a partir del índice que se encuentre y se aplica el valor del offset, si al aplicar el offset, el valor de llegada es menor al valor del offset no se "reproduce" esa partícula (líneas 16, 17, 18), solo en caso de que el valor supere el valor de offset que se tenga al momento se guarda esa partícula y sus valores asociados. El proceso de re-muestreo se repite a lo largo del número de partículas (linea 14), finalmente, se regresa la lista de nuevas partículas con un mayor peso asociado (línea 21).

Con el algoritmo (4.2.3.1.m) se termina la programación del algoritmo para localización y mapeo del diseño ASTEC, donde se propuso utilizar un enfoque de Fast SLAM con contadores de marca erróneos, a continuación se colocaran los programas para la captura de imágenes, el algoritmo para captura del espectro de calor y el programa tentativo para el envió de la información de ASTEC a un usuario humano para su interpretación y análisis.

4.3. Captura de Imágenes

Para el programa que se encarga de la captura de imágenes en el diseño de ASTEC se piensa como un proceso de adquisición de información de apoyo para un determinado usuario receptor para su interpretación. De esta forma, el robot ASTEC no se prevé realiza tareas de procesamiento de las imágenes debido a que la cantidad de recursos que estos procesos pueden demandar en muy alta (dependiendo de lo que se busque procesar o encontrar en las imágenes) y dado que el diseño del robot no cuenta con una gran cantidad de memoria y procesamiento.

El programa que captura las imágenes, utiliza dos bibliotecas para Python llamadas: "OpenCV" y "Numpy" (esta segunda también fue usada en otros programas del robot), la biblioteca de "OpenCV" es un compendio de algoritmos para procesamiento de imagen y vídeo, así como algunas funciones que contienen algoritmos de inteligencia artificial.

```
1 # Programa para captura de imagenes de la camara WEB,
2 # se recibe un parametro de activacion que corresponde
3 # a la posible deteccion de alguna victima, este "activador"
4 # es un parametro que se obtiene de los datos de la camara
5 # termica, y si estos datos indican lecturas termicas en
```

```
6  # relacion a la temperatura de una victima, la camra seria
7  # activada y se tomaria una imagen en esa direccion, despues
8  # seria almacenada y enviada en los paquetes de datos de
9  # la informacion de exploracion.
10
11 import cv2
12
13 def captura_imagen_victima(numero_victima):
14
15     cap = cv2.VideoCapture(0)
16     ret, frame = cap.read()
17
18     nombre_archivo = '/home/brian/Desktop/Tesis_Info/ \
19                       Algoritmos_Tesis/imagen_victima_' + \
20                       str(numero_victima) + '.png'
21
22     cv2.imwrite(nombre_archivo, frame)
23     cap.release()
```

Algoritmo 4.3.1. Captura de imagen usando cámara WEB.

En el algoritmo 4.3.1 se realiza la captura de imagen usando la biblioteca "cv2" (OpenCV), donde la función "captura_imagen_victima" abre el canal del puerto de la cámara (línea 15) y posteriormente "lee" los datos de la cámara y los almacena en "ret" y "frame", "ret" es una variable indicativa de activación, cuando esta variable se encuentra en $TRUE$ indica que existe una matriz de datos almacenada en $frame$, en caso de que se encuentre en $FALSE$ indica que no se tiene un archivo de píxeles en $frame$. La parte más interesante de la captura de las imágenes es la generación del nombre de archivo destino, donde la función "captura_imagen_victima" recibe un parámetro que se entiende como un contador, este contador sirve para establecer la referencia al número de víctima detectado en ese instante. La función para captura de imagen es llamada por el algoritmo de "exploración y trazado de ruta" (QPA*).

4.4. Algoritmo para captura de espectro de calor

En lo que corresponde al algoritmo para la "captura de espectro de calor" se relaciona con el manejo de la cámara térmica. Para el diseño algoritmo se usó la hoja de datos técnicos de la cámara AMG8833 de Adafruit [61] y la matriz de sensores IR de Panasonic [66] (que conforman la cámara térmica AMG8833).

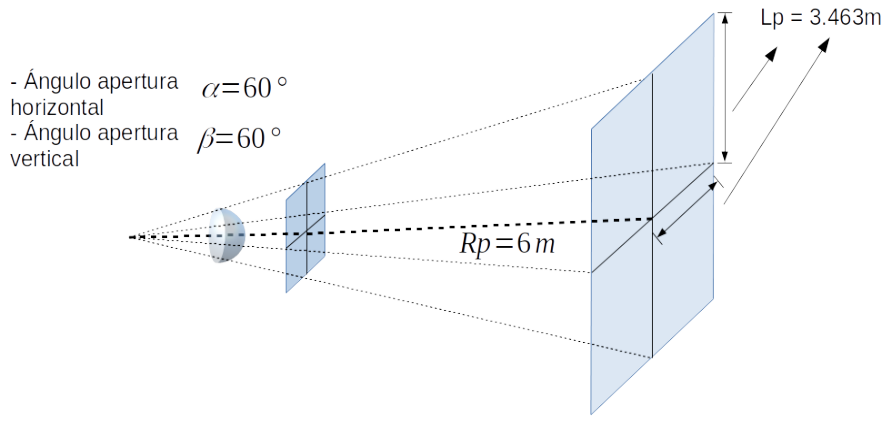


Ilustración 4.4.1 Proyección de cámara térmica para $6m$.

Usando las hojas de datos técnicos [61] [66] se realizó la ilustración (4.4.1) donde se ejemplifica el funcionamiento de como se simularía la cámara térmica tomando las condiciones reales de la misma, donde según el fabricante de la matriz de sensores IR (Panasonic) el ángulo de apertura del detector es de 60 grados verticales y 60 grados horizontales, además, Adafruit [61] menciona que el rango efectivo de detección de la cámara para temperaturas humanas es de 7 metros, dado que el generador de mapas pseudo-aleatorios se tiene configurado para generar mapas de 6 metros x 6 metros, la distancia posible de detección para la simulación de la cámara térmica se redujo a 6 metros, esto da como lineas de proyección de la cámara una altura "efectiva" de detección de 6,926m y de ancho 6,926m aproximadamente.

```
# Generador pseudo-aleatorio de camara termica, esta funcion
# simula las lecturas de la matriz de la camara terminca
# AMG8833, donde se simulan de 0 a 3 corridas con lecturas
# menores a las 25 C, y posteriormente se realizan "lecturas"
# hasta los 34 C con la finalidad de simular la deteccion de
# una posible victima.

import numpy as np
import random
import matplotlib.pyplot as plt

def camara_termica(contador_termico):

    if contador_termico < 4:

        matriz_termica = np.random.randint(25, size=(8, 8))
        tipo_meta = 's'

    else:

```

```
21          matriz_termica = np.random.randint(35, size=(8, 8))
22          lectura = matriz_termica[np.where(matriz_termica >= 25)]
23
24          if len(lectura) > 10:
25              tipo_meta = 'p'
26          else:
27              tipo_meta = 's'
28
29      return tipo_meta
```

Algoritmo 4.4.1. Simulación de cámara térmica.

La biblioteca creada por *Dean Miller* de "Adafruit Industries", contiene una función llamada "amg.pixels()" que consiste en la creación de una matriz de "píxeles" de dimensión (8 x 8) que permite establecer las lecturas de calor de la cámara por posición en la matriz de sensores IR internos de la cámara térmica, esta matriz se envía al llamar la función "amg.pixels()".

En el algoritmo (4.4.1) se realiza la simulación de los datos obtenidos por la cámara térmica.

En la línea 12 se tiene la función de "camara_termica()" esta función recibe un parámetro llamado "contador_termico", este contador tiene la finalidad de alterar las tendencias de las lecturas pseudo-aleatorias generadas con la función de Numpy "np.random.randint()" en la líneas 16 y 21, cuando este contador sobrepasa la iteración 3 (puede establecerse otro número de iteraciones, pero para propósitos de datos de simulaciones se estableció que 4 iteraciones, es decir, de 0 a 3 son suficientes para experimentar ciertos comportamientos de los algoritmos) la "matriz_termica", que simula la matriz que se obtendría de "amg.pixels()", se le permite generar valores por encima de los 25 grados celcius, donde si existen más de diez elementos (elementos dentro de la matriz) que tengan lecturas iguales o por encima de los 25 grados entonces se relaciona un cambio en el "tipo_meta" a "p" que es protocolo de una "posible víctima", en caso contrario, cuando no se alcanzan más de diez elementos superiores o iguales a los 25 grados, el "tipo_meta" permanece como "s" que indica el protocolo de exploración del mapa.

Una vez explicado el último programa (o algoritmo) pensado para el diseño de ASTEC, se continuará con el capítulo de "Resultados y Conclusiones" donde se mostrarán los resultados y simulaciones que se obtuvieron con estos algoritmos, y con ello se obtendrán algunas conclusiones que podrán ser de ayuda para la implementación física del robot ASTEC en un trabajo futuro, así como su mejoramiento en esta etapa de diseño en próximos trabajos.

Capítulo 5

Resultados y Simulaciones

En este capítulo se expondrán los resultados, simulaciones y conclusiones logradas en el trabajo de tesis en función del diseño de ASTEC. Las simulaciones generadas y los resultados de las mismas serán explicados en el orden en como se prevé puedan aparecer en una zona de desastre "real", y tomando en cuenta las características mencionadas en capítulos anteriores que se buscaron implementar en este diseño.

5.1. Simulaciones de mapa de exploración pseudo-aleatorio en $2D$

Dado que este trabajo de tesis no se plantea llegue a la etapa de implementación, resulta fundamental diseñar un método en el cual se puedan generar "mapas de exploración". A continuación, se muestran algunos mapas que se generaron con el programa encargado de la generación de mapas pseudo-aleatorios.

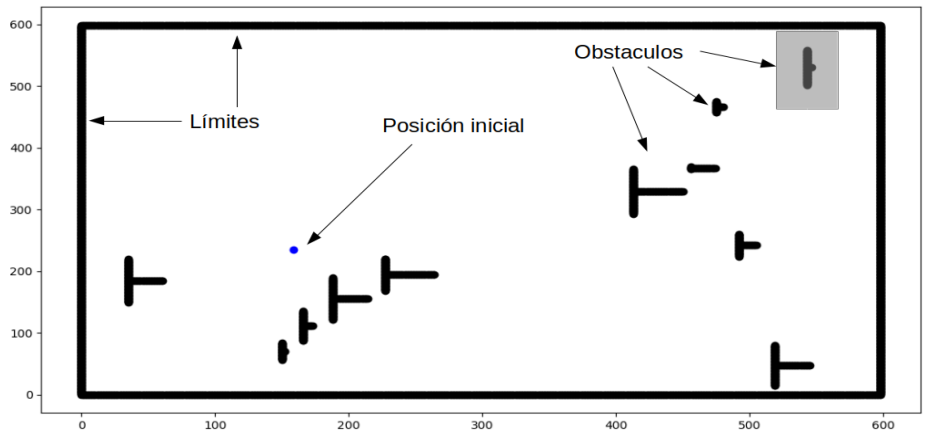


Figura 5.1.1. Mapa pseudo-aleatorio generado.

El la figura (5.1.1) se muestra un mapa pseudo-aleatorio generado. En estos mapas pseudo-aleatorios se fijan los límites posibles de "exploración" en 600 unidades por 600

unidades que representarían las fronteras de lo que podría detectar el sensor LIDAR, esta idea de plantear "limites" en las matrices de exploración es con la finalidad de generar zonas de búsqueda o exploración bien definidas, y permitirle a ASTEC lograr explorar zonas de un tamaño relativamente reducido, y posteriormente avanzar a otras zonas y repetir este proceso mientras se desee.

En general, estos mapas contienen tres elementos primordiales, que son: un inicio dentro del mapa, dicho inicio es generado de manera pseudo-aleatoria, obstáculos de mapa generados de manera pseudo-aleatoria y los limites del mapa de exploración.

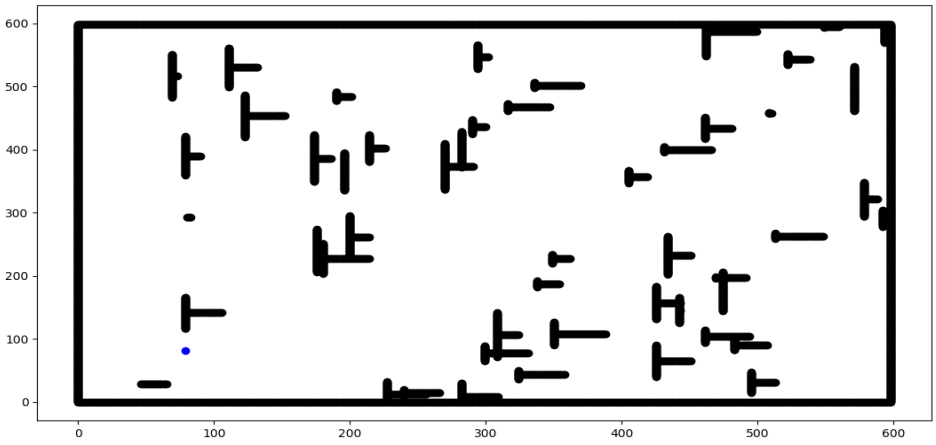


Figura 5.1.2. Mapa pseudo-aleatorio generado con mas obstáculos.

El programa para generar los mapas pseudo-aleatorios puede ser modificado para generar un mayor número de obstáculos en el mapa. Al aumentar el número de obstáculos pueden generarse mapas más complejos. La perspectiva que se tiene de los mapas generados puede interpretarse como una toma por encima del mapa, donde los objetos (obstáculos) que se generan están a nivel del sensor LIDAR de ASTEC.

En la figura (5.1.3) se muestran algunos mapas pseudo-aleatorios generados con diferentes números de obstáculos por mapa, estos mapas son generados para cada simulación, siendo que cada mapa es prácticamente "único", es decir, que este mapa no se repite para la siguiente simulación. Debe mencionarse que para cada simulación el algoritmo de exploración y trazado de ruta lleva a cabo tres iteraciones de busqueda dentro de cada mapa creado.

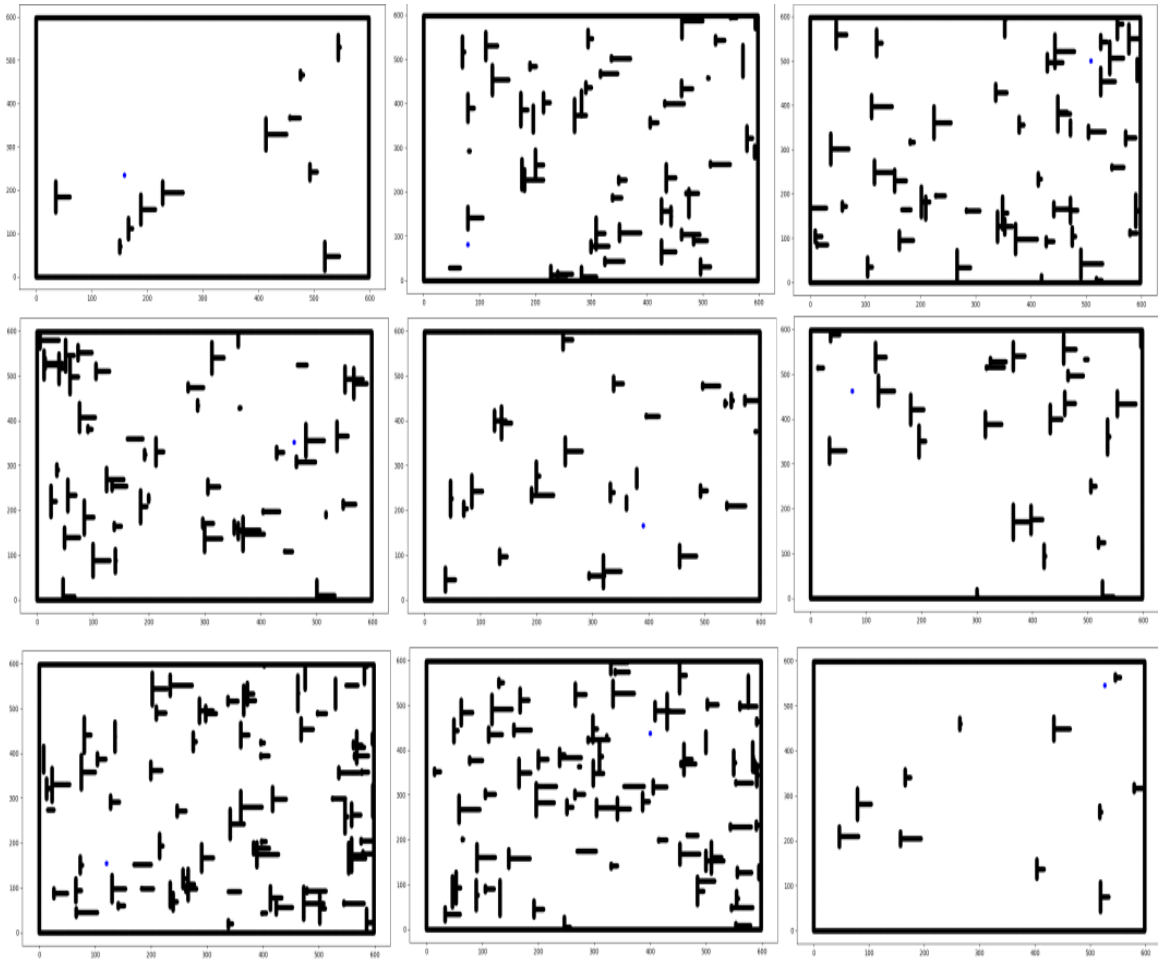


Figura 5.1.3. Mapas pseudo-aleatorios generados con diferente numero de obstáculos.

La siguiente etapa de los algoritmos creados para ASTEC, consiste en el establecimiento de los campos artificiales de potencial binarios alrededor de los obstáculos generados en los mapas de simulación.

5.2. Generación de campos de potencial binario artificial para obstáculos del mapa de exploración pseudo-aleatorio en $2D$

El siguiente elemento que corresponde al desarrollo del algoritmo de la etapa de exploración y trazado de ruta es la generación de los campos de potencial binario artificial. Los campos de potencial son asociados a los obstáculos y limites del mapa antes mostrados, estos campos tienen la finalidad de ayudar en la reducción de colisiones del robot ASTEC en una aplicación en campo, dado que la exploración y el trazado de ruta tienen un enfoque como punto masa para representar al cuerpo y trayectoria del robot. Entonces, para evitar que las rutas y el robot puedan colisionar por sus

dimensiones físicas “reales” la idea de incorporar campos de potencial “inhabilita” zonas de exploración y trazado de ruta para el diseño ASTEC.

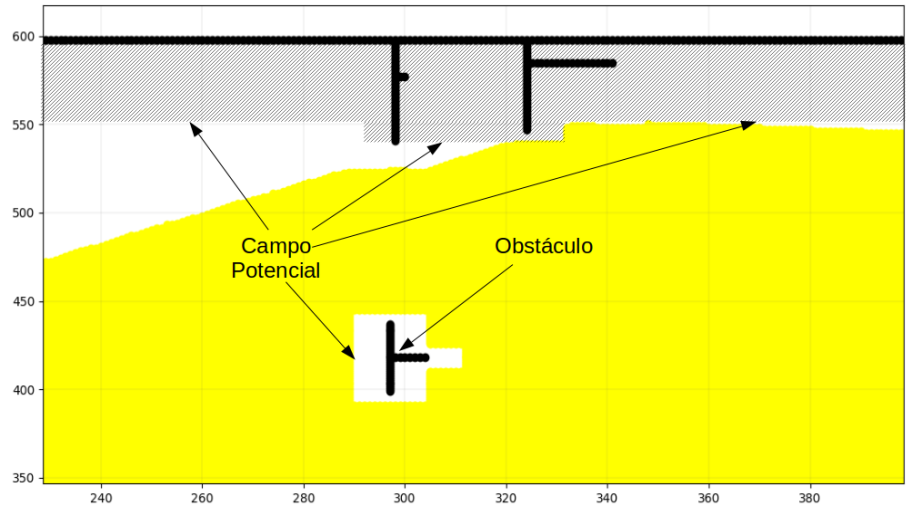


Figura 5.2.1. Campos de potencial binario artificial generados al perímetro de los obstáculos y limites del mapa.

En la figura (5.2.1) se observa la generación del campo de potencial binario artificial, el campo es generado después de la creación del mapa pseudo-aleatorio, donde cada obstáculo genera un campo al rededor de sí mismo. El algoritmo de trazado de ruta interpreta estos campos binarios como zonas de no acceso para exploración, donde estos nodos (los que corresponden a los campos de potencial) y los nodos de los obstáculos no se pueden contemplar como parte de los nodos “libres” dentro del mapa de exploración.

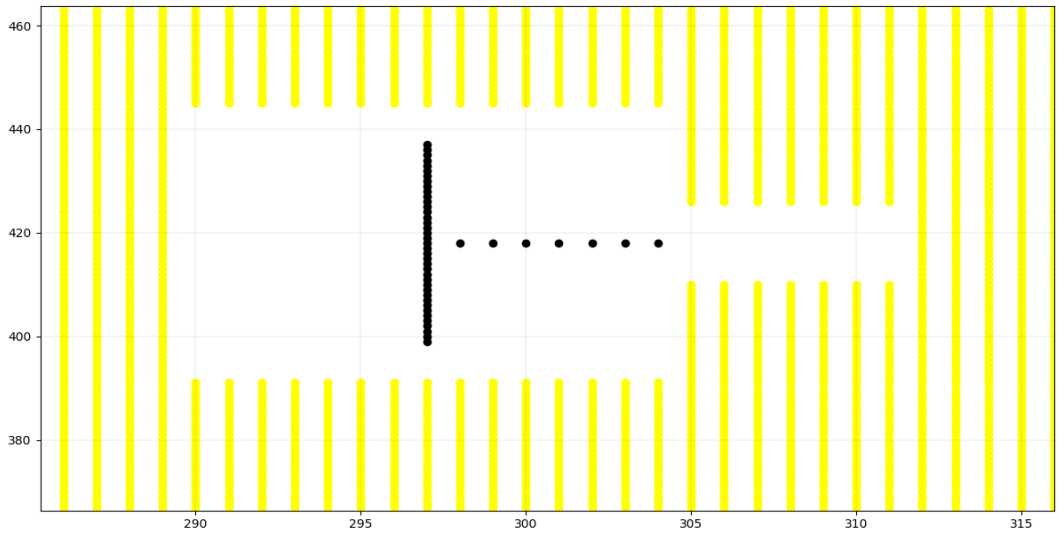


Figura 5.2.2. Campos de potencial binario artificial.

Al aumentar la resolución de la figura (5.2.1) se obtiene la figura (5.2.2). En color amarillo se tienen los nodos visitados por el algoritmo de trazado de ruta, estos nodos

son analizados como nodos tentativos para establecer la ruta mínima a la meta, y como puede apreciarse, los campos de potencial binario inhabilitan un área en función a las dimensiones que se tengan del robot. En las simulaciones, el área que corresponde a los campos binarios no contendrá color (sera blanca), para contemplar que estas zonas están deshabilitadas por el algoritmo que genera los campos de potencial.

5.3. Simulaciones de Cámara Térmica

La simulación de la cámara térmica se propone como el método "principal" para la determinar la posible localización de una víctima, esta cámara térmica simulada se basa en la cámara AMG8833 de Adafruit. Como se explico en capítulos anteriores, la cámara térmica simula la obtención del espectro de calor de una zona determinada a través de matrices de (8 x 8) píxeles, donde según datos de Adafruit [61] la distancia efectiva de detección es de 7 metros aproximadamente, donde para propósitos de compatibilidad con el generador de mapas pseudo-aleatorios que genera mapas de 6 metros x 6 metros, la distancia de detección térmica se simula a 6 metros.

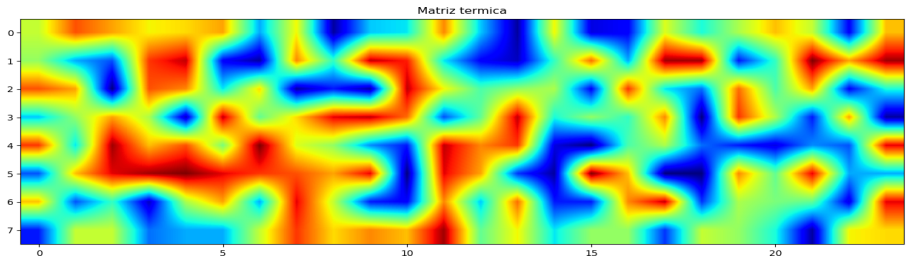


Figura 5.3.1. Simulación de cámara térmica para protocolo de detección de "posible víctima".

La figura (5.3.1) simula el barrido de la cámara térmica para 3 secciones del mapa, la cámara térmica propiamente no realiza ningún proceso de "barrido" sin embargo para poder tener una mayor cobertura del espectro de calor de la zona donde el robot ASTEC se propone (como se explico en el capítulo de "Materiales y Diseño") que a través de la conexión de cámara térmica con un actuador como un servo motor, la cámara térmica rote y tome al menos 3 secciones de captura de espectro de calor de la zona. Como se aprecia en la figura (5.3.1) las lecturas pseudo-aleatorias simuladas no presentan un patrón en especifico, dado en principal medida a la técnica usada para la generación de los datos y también con el propósito de establecer que la "posible víctima" puede encontrarse "parcialmente", es decir, que posiblemente no se detecte el cuerpo completo de la persona y tal vez solo sean extremidades.

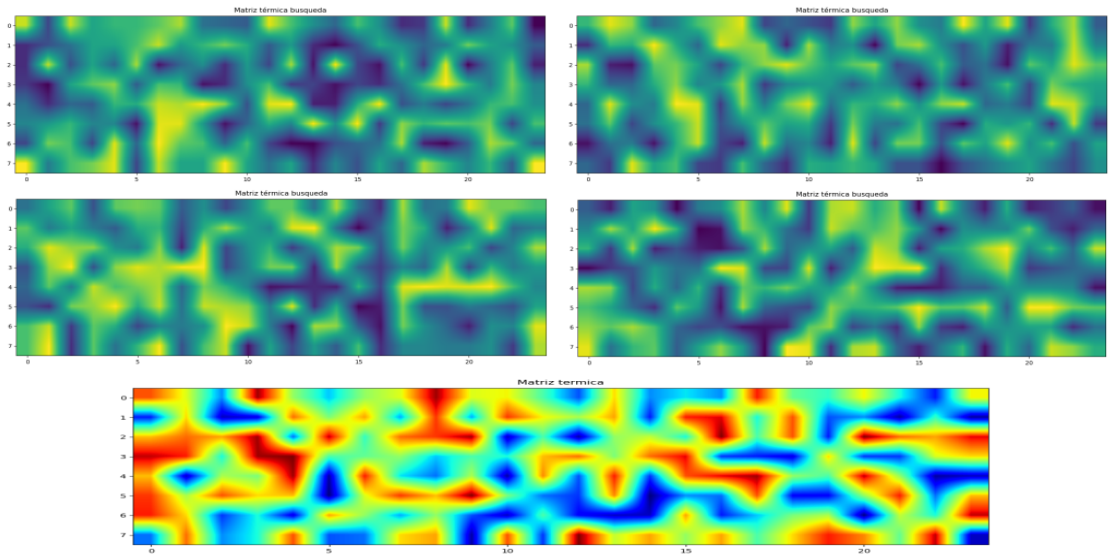


Figura 5.3.2. Simulación de cámara térmica con protocolo de búsqueda y detección de "posible víctima".

Ahora, en la figura (5.3.2) se muestra el protocolo de búsqueda (o exploración) y el protocolo de detección de "posible víctima", como se menciono en secciones pasadas las simulaciones se basan en cinco ejecuciones del los protocolos de exploración y trazado de ruta, captura de espectros de calor, captura de imágenes y localización y mapeo. En estas ejecuciones cuatro de ellas se declinan por protocolo de exploración y una corresponde a la posible localización de una víctima. En la figura (5.3.2) se muestra como las primeras cuatro iteraciones, la cámara simula no obtener lecturas de calor que puedan indicar la presencia de una posible víctima, entonces el protocolo de exploración se confirma y este programa envía un "tipo_meta" secundario "s", para la ultima iteración que corresponde a la gráfica inferior de la figura (5.3.2), se simula la posible detección del espectro de calor que indica la "presencia" de una posible víctima, entonces este programa regresa un "tipo_meta" principal "p".

5.4. Simulaciones del algoritmo de trazado de ruta, realizando comparativa entre A* clásico, A* modificado y QPA*

En esta sección, se realizaron las simulaciones que corresponden a la parte de trazado de ruta del "algoritmo de exploración y trazado de ruta", en esta parte se compararon tres enfoques para el algoritmo A*, estos enfoques fueron:

- A* clásico, que implementa búsqueda no ordenada para el vector de "frente".

- A* modificado, implementa búsqueda ordenada usando el algoritmo de Heap Queue de Python 3 para el vector de "frente".
- QPA* (algoritmo creado por autor de la tesis), que implementa la búsqueda ordenada de Heap Queue para el vector de "frente" y una función de "costo extra" para nodos visitados en iteraciones previas a t (que se considera como la iteración de exploración "actual").

Para las simulaciones comparativas de los tres enfoques listados se realizara en dos bloques, el primer bloque consta de la realización de las simulaciones visuales de los tres enfoques, con un total de tres iteraciones por simulación (dos trazados de ruta en protocolo de exploración y un trazado de ruta a posible víctima), y se realizaran un total de tres simulaciones por algoritmo, para cada simulación se colocará su mapa inicial (el mapa pseudo-aleatorio generado) y el mapa de exploración resultante después de las tres iteraciones, además, se agregaran datos importantes respecto a las simulaciones realizadas. En la segunda etapa, se realizaran cien simulaciones (con tres iteraciones por simulación, al igual que en la primera etapa) por algoritmo para la obtención de gráficas y datos más solidos del comportamiento de estos enfoques, en esta etapa no habrán elementos visuales para fines prácticos, y se colocarán las gráficas generadas con respecto a los elementos más relevantes y al final se realizarán las gráficas comparativas de los distintos enfoques.

Además, para las simulaciones gráficas de los distintos enfoques de A* comparados, se calculo el porcentaje de cobertura y el total de nodos visitados por segundo, con las siguientes formulas:

(5.4.1)

$$P_{cov} = \frac{Vis_{real} * 100}{M_{total} - OA}$$

Donde:

- P_{cov} es el porcentaje de cobertura del mapa de exploración generado.
- Vis_{real} es el total de nodos visitados menos los nodos repetidos entre iteraciones.
- M_{total} es el total de nodos de la matriz de exploración (600 x 600).
- OA es el número de nodos generados de obstáculos y campos de potencial binario artificial.

(5.4.2)

$$NvT = \frac{Vis_{total}}{t_e}$$

Donde:

- NvT es la relación del total de nodos visitados entre el tiempo de ejecución de la simulación.
- Vis_{total} es el total de nodos visitados incluyendo los nodos repetidos entre iteraciones.
- t_e corresponde al tiempo de ejecución de la simulación.

5.4.1. Simulaciones visuales para algoritmo A* clásico

Como se menciono anteriormente, el algoritmo A* clásico es un algoritmo de trazado de ruta, donde existe una posición inicial y una posición final (meta) donde el algoritmo se encarga de generar la ruta mínima entre estos dos puntos.

El algoritmo A* clásico realiza las labores de búsqueda del nodo o los nodos mínimos del vector "frente" (inspirado en el algoritmo Dijkstra) de manera "no ordenada", en este caso la complejidad computacional asociada para la búsqueda del min (mínimo) es de $O(F)$ [1], donde F es el número de elementos en el arreglo del vector de "frente". Realmente los tiempos de búsqueda tienen muchas variaciones dado que el elemento (en este caso, de valor mínimo) puede estar en cualquier parte del arreglo.

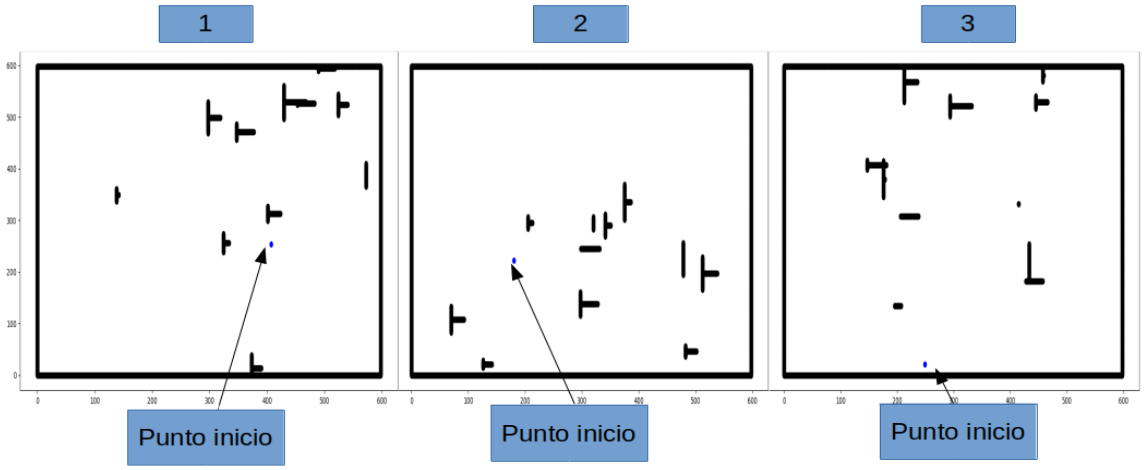


Figura 5.4.1.1. Simulación de protocolo de exploración y detección de víctima con algoritmo A* clásico.

En la figura (5.4.1.1) se muestran los tres mapas generados de manera pseudo-aleatoria que corresponden a las tres simulaciones realizadas para el algoritmo A* clásico. El mapa

[1] En cómputo, la notación $O()$ es usada para clasificar la "complejidad" de un algoritmo basada en los tiempos de ejecución o la memoria requerida en relación con el crecimiento de los datos operados por el algoritmo.

consiste en una matriz de exploración total de 600 unidades por 600 unidades, en color azul se muestran los puntos de inicio para cada mapa, y en color negro se muestran los obstáculos (u objetos) creados por el generador de mapas pseudo-aleatorio. Para la primer simulación del A* clásico, se generaron un total de 30,304 nodos de "obstáculos" y "campo artificial" (que se abreviara como "OyCA"), para la segunda simulación se generaron 31,391 nodos de OyCA y para la tercer simulación se crearon 32,936 nodos de OyCA.

Ahora, en la figura (5.4.1.2) se muestran los mapas "resueltos", es decir, los mapas resultantes después de las tres iteraciones en el protocolo de exploración y detección de posible víctima, en color amarillo se tienen los nodos "visitados" en el protocolo de exploración del algoritmo, en color violeta se tienen los nodos generados en la activación del protocolo de "detección de posible víctima", en la detección de una posible víctima (en la ultima iteración de la simulación) se genera un punto de inicio final en color azul, y en color rojo se establece la localización de la posible víctima.

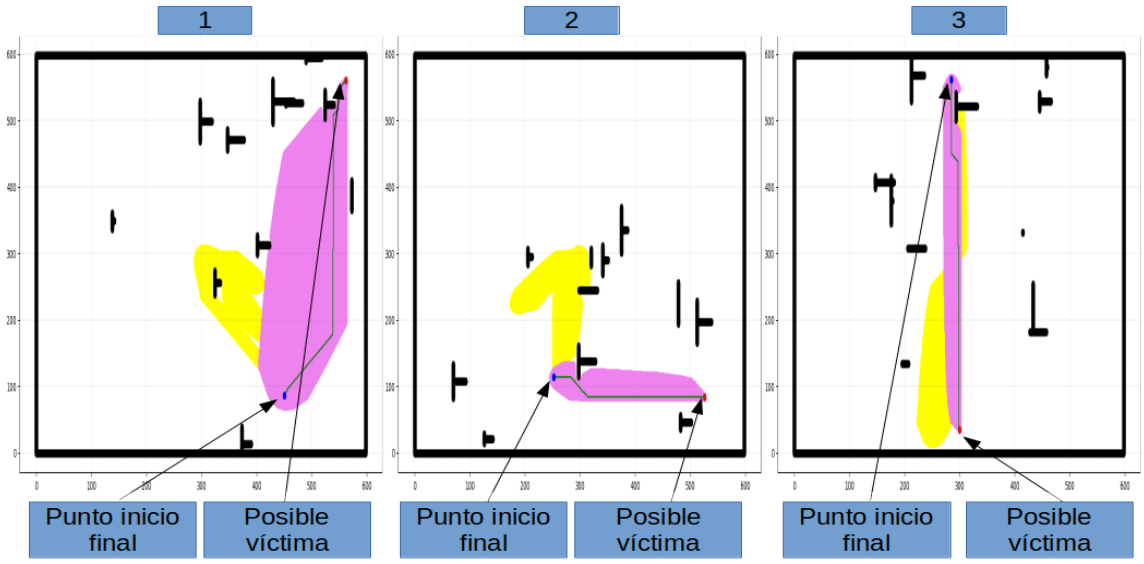


Figura 5.4.1.2. Mapas resueltos para algoritmo A* clásico.

Para el cálculo del "tiempo de ejecución" de los tres enfoques (clásico, modificado y QPA*) se utilizo la función *time.time*() de la biblioteca de funciones estándar de Python 3, donde se colocó un registro de tiempo al inicio del llamado a la función de alguno de los enfoques (llamado al algoritmo de trazado de ruta basado en A* clásico, A* modificado o QPA*) y posteriormente un registro de tiempo al final del llamado a la función, donde solo el tiempo de ejecución del enfoque de trazado de ruta que se este simulando es contabilizado. Después, para establecer la cantidad de nodos visitados, los nodos que correspondieron a OyCA y nodos "re-visitados" se uso la función *numpy.count_nonzero*() de la biblioteca Numpy de Python 3, que sirve para contabilizar todos los nodos que tienen

un valor distinto de cero y regresa el total de elementos de las matrices que tuvieron valores por arriba de cero.

Para las simulaciones del A* clásico, usando la fig (5.4.1.2), la primer simulación tuvo un tiempo total de ejecución fue de 461.65(seg) con un total de 65,478 nodos visitados, de los cuales 2,382 fueron nodos "re-visitados" (nodos repetidos entre iteraciones en la matriz de nodos visitados), en la segunda simulación el tiempo de ejecución fue de 88.32 (seg) con un total de 20,739 nodos visitados y 1,301 nodos re-visitados.

Usando las ecuaciones (5.4.1) y (5.4.2) se calculó la tabla (5.4.1.1).

Número de simulación	Porcentaje de cobertura	Nodos vs tiempo
1	19.137629	141.834723
2	6.149861	234.816576
3	7.560526	150.205853

Tabla 5.4.1.1. Análisis de simulaciones gráficas de algoritmo A* clásico.

5.4.2. Simulaciones visuales para algoritmo A* modificado

Para el algoritmo A* modificado se usaron los mismos parámetros de simulación que en el A* clásico, con un total de tres simulaciones gráficas y tres iteraciones por simulación.

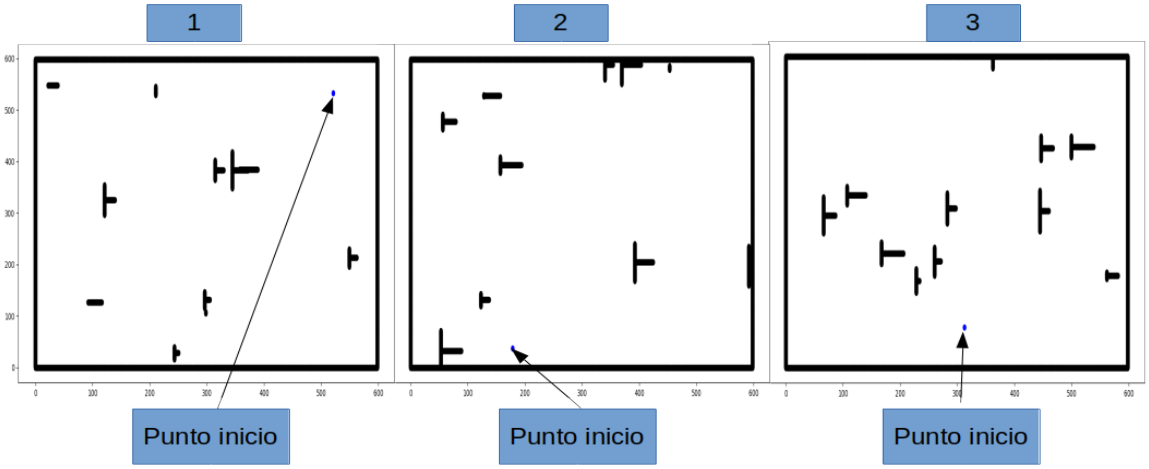


Figura 5.4.2.1. Simulación de protocolo de exploración y detección de víctima con algoritmo A* modificado.

El enfoque del algoritmo de A* modificado consiste en la implementación de un método de "búsqueda ordenada", el método que se implemento para este algoritmo usa la biblioteca

"heapq" de Python 3. La búsqueda ordenada usando el "ordenamiento de pila" (heap queue algorithm) se usa en la búsqueda e inserción de los nodos "mínimos" del algoritmo A* clásico.

En la figura (5.4.2.1) se obtuvieron para la primera simulación un total de 28,641 nodos de OyCA, para el segundo mapa se crearon un total de 29,913 nodos de OyCA y para la tercer simulación se generaron 33,206 nodos de OyCA.

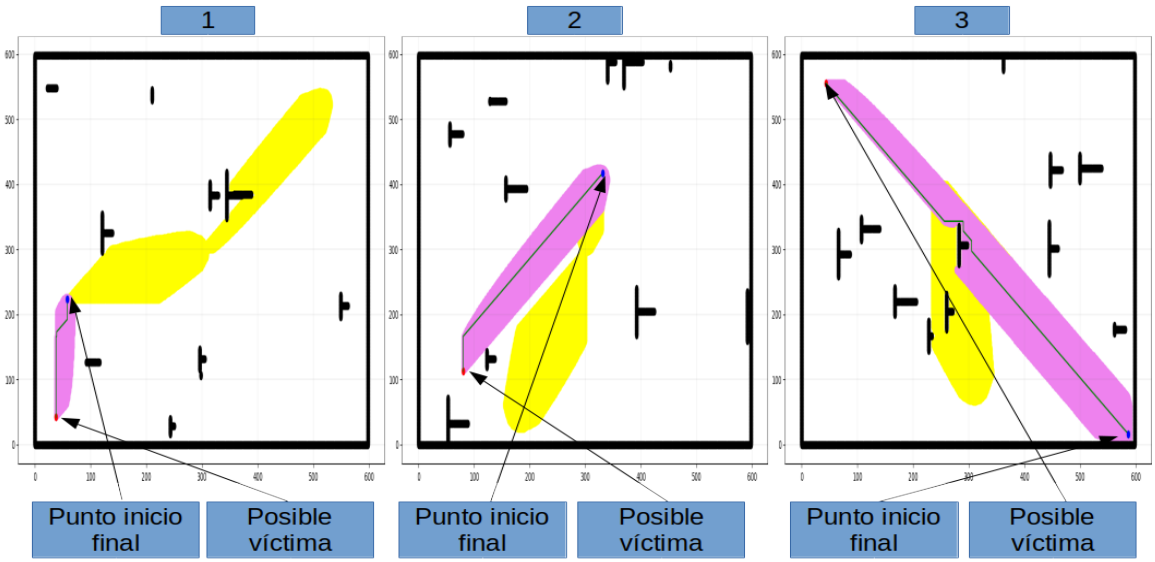


Figura 5.4.2.2. Mapas resueltos para algoritmo A* modificado.

La figura (5.4.2.2) muestra las simulaciones "resueltas" para el algoritmo A* modificado después de la exploración del mapa y detección de una posible víctima (en tres iteraciones), en color amarillo se encuentran los nodos visitados, en color violeta los nodos visitados en el protocolo de detección de víctima. Para la primer simulación se generó una matriz de nodos visitados de 35,496 nodos con 61 nodos repetidos entre iteraciones, y con un tiempo de ejecución de 35.16 (seg), para la segunda simulación gráfica se generaron 45,693 nodos visitados con 2,010 nodos repetidos entre iteraciones, y con un tiempo de ejecución de 48.35 (seg), para la tercera simulación se generaron 84,192 nodos visitados con 24,066 nodos repetidos en un tiempo de ejecución de 39.86 (seg).

Número de simulación	Porcentaje de cobertura	Nodos vs tiempo
1	10.693839	1009.556313
2	13.233783	945.046535
3	18.398746	1641.169590

Tabla 5.4.2.1. Análisis de simulaciones gráficas de algoritmo A* modificado.

5.4.3. Simulaciones visuales para algoritmo QPA*

Para las simulaciones gráficas del algoritmo QPA* (propuesto por el autor de la tesis), se usaron los mismos parámetros que para las simulaciones gráficas de los enfoques anteriores.

El enfoque del algoritmo QPA* utiliza como su "kernel" al algoritmo A* modificado con el ordenamiento de pila (Heap Queue algorithm) y agrega una función de costo extra en relación a los nodos visitados en iteraciones pasadas (ver ecuación (4.17) del capítulo 4), esta modificación se plantea para realizar una "tendencia" del algoritmo a no "re-visitar" nodos (visitados) de las iteraciones pasadas.

En la figura (5.4.3.1) se muestran los mapas generados para las simulaciones del algoritmo QPA*, en el primer mapa se crearon 30,671 nodos de OyCA, para el segundo mapa se crearon 31,016 nodos de OyCA, y para el tercer mapa se generaron un total de 32,390 nodos de OyCA.

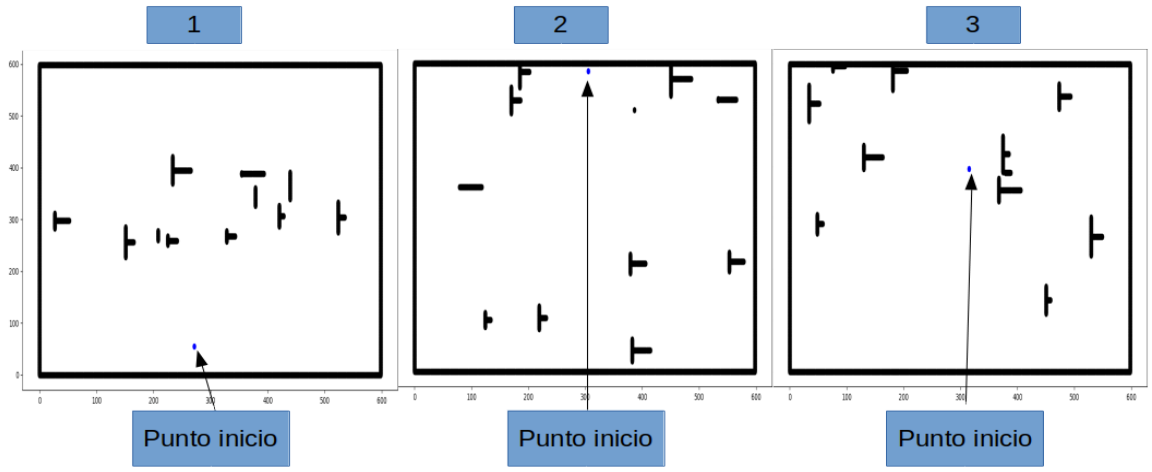


Figura 5.4.3.1. Simulación de protocolo de exploración y detección de víctima con algoritmo QPA*.

En la figura (5.4.3.2) están colocados los mapas "resueltos" generados por el algoritmo QPA* posterior a la aplicación del protocolo de exploración y detección de posible víctima. En el primer mapa resuelto se generaron 59,638 nodos visitados con 2,883 nodos repetidos entre iteraciones y un tiempo de ejecución de 44.22 (seg), para el segundo mapa 103,047 nodos se visitaron con 26,019 nodos repetidos entre iteraciones en un tiempo de ejecución de 63.97 (seg), y en la tercera simulación se generaron un total de 157,308 nodos en la matriz de visitados con 46,337 nodos repetidos en un tiempo de ejecución de 87.39 (seg).

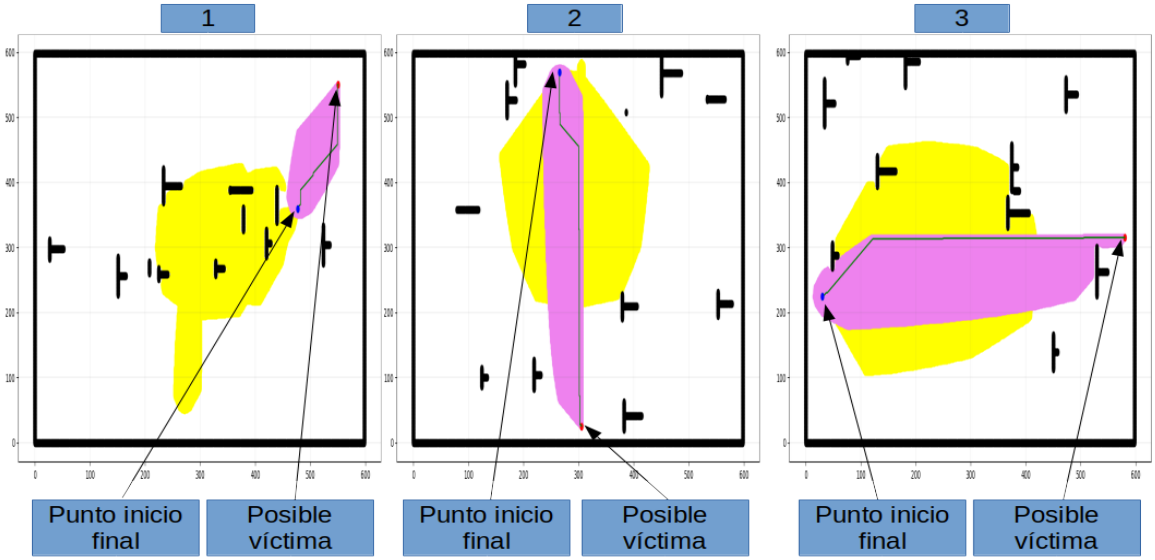


Figura 5.4.3.2. Mapas resueltos para algoritmo QPA*.

Con la información generada de las simulaciones gráficas del algoritmo QPA*, y utilizando las ecuaciones para cálculo de cobertura y nodos visitados vs tiempo, se lleno la tabla (5.4.3.1).

Número de simulación	Porcentaje de cobertura	Nodos vs tiempo
1	17.233526	1348.665762
2	23.413904	1610.864467
3	33.872897	1800.068657

Tabla 5.4.3.1. Análisis de simulaciones gráficas de algoritmo QPA*.

Dado, que el número de simulaciones visuales realizadas no fue considerable, en la siguiente sub-sección se realizaron cien simulaciones por enfoque del algoritmo A*, para verificar los datos obtenidos en la etapa visual, y con esto fundamentar de mejor manera la efectividad del algoritmo QPA* diseñado para ASTEC.

5.4.4. Simulaciones comparativas a cien corridas por enfoque

En esta subsección se realizaron las simulaciones "no visuales" de enfoques del algoritmo A* comparados, la estrategia de generación de estas simulaciones fue similar a la implementada para el caso de las simulaciones visuales, donde se realizaron un total de cien simulaciones por enfoque, es decir, cien simulaciones del algoritmo A* clásico, cien simulaciones del algoritmo A* modificado (con ordenamiento de pila) y cien simulaciones para el algoritmo QPA*.

Además, cada simulación genera un mapa pseudo-aleatorio diferente, con diferente distribución de obstáculos y campos artificiales. Con la información generada de las cien simulaciones, se crearon gráficas para observar su comportamiento de forma individual, y en la parte final, tablas comparativas de los tres enfoques.

Simulaciones para algoritmo A* clásico

En el caso de la búsqueda "no ordenada" que se usa en la búsqueda de los nodos mínimos en el algoritmo A* clásico, se establece que la complejidad computacional para la inserción de elementos es $O(1)$ [67] dado que la inserción se da en cualquier posición de vector (o matriz) que contenga la lista (suele insertarse al final de la lista), para el caso de la búsqueda de algún elemento la complejidad computacional se establece como $O(F)$ [67] (que es el caso del algoritmo A*) F es el vector de nodos de "frente" (los nodos que están siendo explorados).

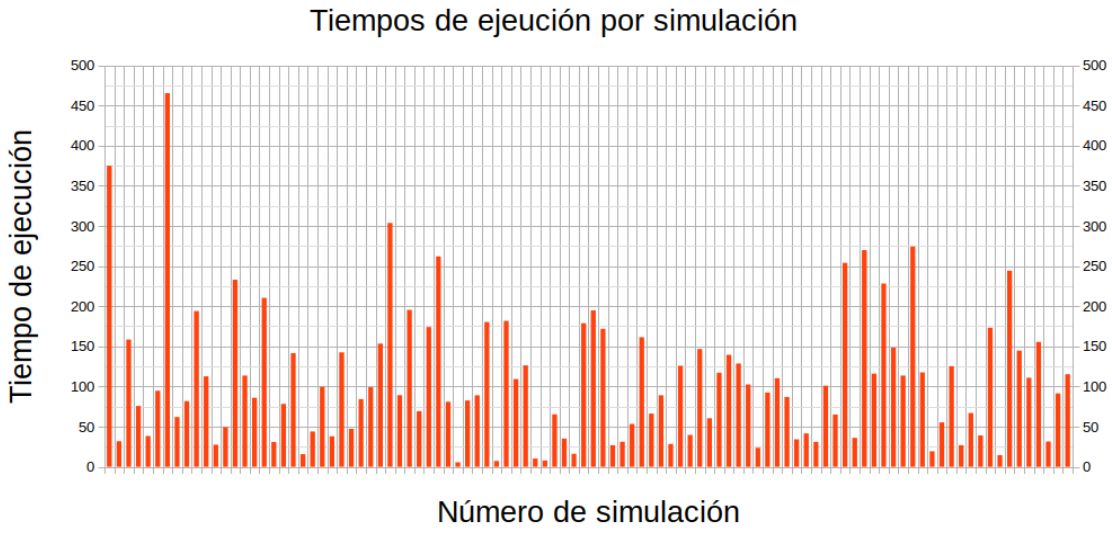


Figura 5.4.4.1. Gráfica del tiempo de ejecución de las cien simulaciones para algoritmo A* clásico.

Observando la figura (5.4.4.1) se establece que el tiempo de ejecución para el caso del algoritmo A* clásico puede ser muy "volátil", esto se debe principalmente al tiempo de búsqueda "no ordenada", ya que en el caso de que los elementos mínimos se encuentren dentro de los primeros elementos del vector (o matriz) puede reducir considerablemente el tiempo de búsqueda, pero en caso de que los elementos se encuentren en posiciones alejadas de la zona de inicio de búsqueda el tiempo de ejecución de este enfoque se puede extender hasta casi los 500 (seg).

En la figura (5.4.4.2) mostrada a continuación, se graficaron los nodos visitados totales (incluyendo los nodos repetidos entre iteraciones), en esta gráfica también se

observan comportamientos muy abruptos en la cantidad de nodos visitados, que se pueden encontrar en un máximo observado de 150,000 nodo visitados (aproximadamente) pero con un mínimo de solo 2,000 nodos visitados (aproximadamente).

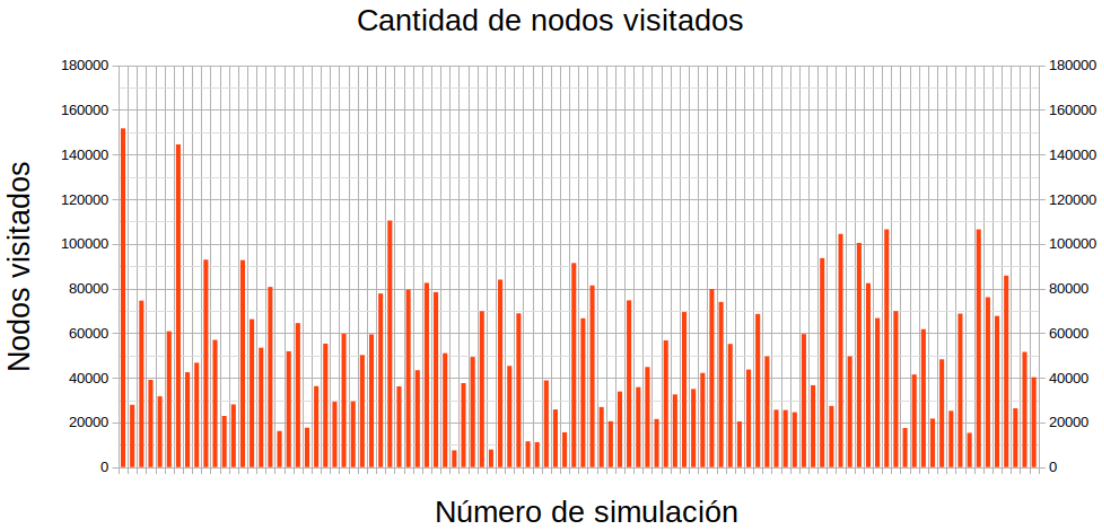


Figura 5.4.4.2. Gráfica del nodos visitados en las cien simulaciones para algoritmo A* clásico.

Al observar las figuras (5.4.4.1) y (5.4.4.2) se observa un comportamiento de relación entre el número de nodos visitados y el tiempo de ejecución, a mayor número de nodos visitados generados el tiempo de ejecución también se eleva de forma considerable.

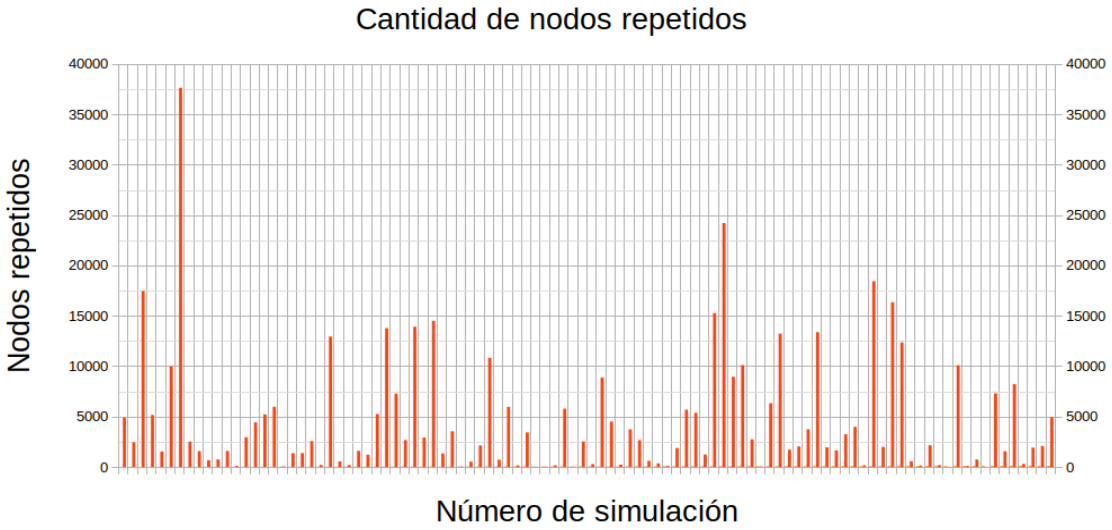


Figura 5.4.4.3. Gráfica del nodos repetidos entre iteraciones en las cien simulaciones para algoritmo A* clásico.

La figura (5.4.4.3) contiene los datos referentes a la cantidad de nodos repetidos entre iteraciones durante las simulaciones creadas, en esta gráfica el comportamiento se puede observar más uniforme que en caso del tiempo de ejecución o los nodos visitados, sin embargo, también se observan algunos incrementos de hasta 37,000 nodos repetidos (aproximadamente).

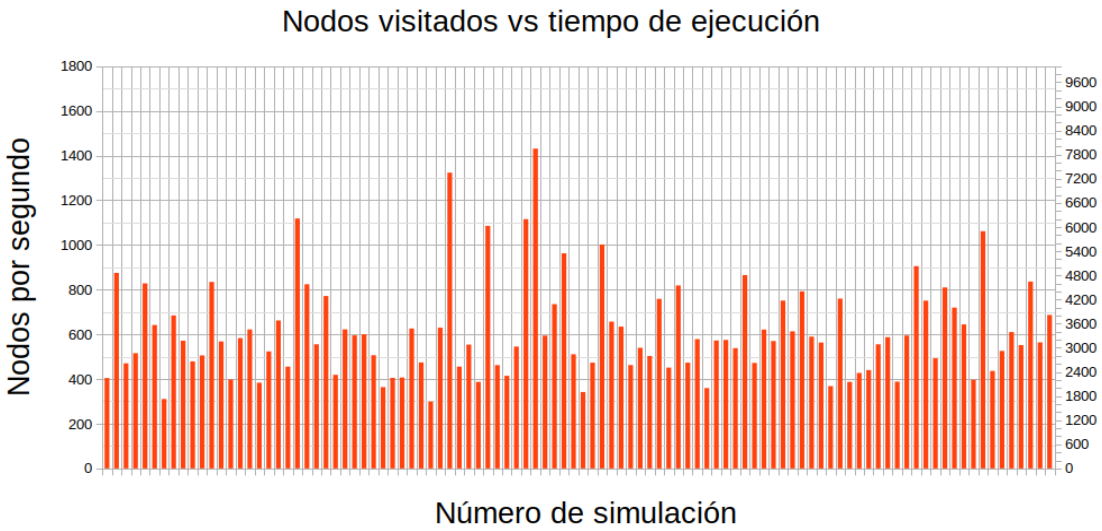


Figura 5.4.4.4. Gráfica del nodos visitados totales entre el tiempo de ejecución, en las cien simulaciones para algoritmo A* clásico.

La gráfica que relaciona la cantidad de nodos visitados por segundo se puede observar en la figura (5.4.4.4), la relación de (n/s) tiene un comportamiento relativamente "estable" dado que existe una relación entre la cantidad de nodos y el tiempo de ejecución.

Usando los datos de las simulaciones del enfoque A* clásico, se realizo la tabla (5.4.4.1).

Promedios de simulaciones de algoritmo A* clásico	
Tiempo ejecución	108.40 (s)
Nodos visitados	53,664.23 (n)
Nodos repetidos	4,620.84 (n)
Nodos visitados vs tiempo	611.80 (n/s)

Tabla (5.4.4.1). Generación de promedios de datos de simulaciones de A* clásico.

Simulaciones para algoritmo A* modificado

Para el enfoque del algoritmo A* modificado, se utiliza un método de ordenamiento de pila a través del algoritmo "Heap Queue" de Python 3, este enfoque se desarrollo con la

idea de mejorar la complejidad computacional del algoritmo, para el caso de inserción en búsqueda "ordenada" se tiene una complejidad de $O(log(F))$ [67] (F representa al vector de "frente") que en comparación con la complejidad computacional del enfoque clásico que es de $O(1)$, la búsqueda ordenada genera un incremento en la complejidad computacional, esto se debe principalmente a que los métodos de ordenamiento (de la función *heapq*() de Python 3) se realizan de forma "dinámica", ya que cada nodo "mínimo" es agregado en la posición en la que cumple con las condiciones del árbol binario de la función *heapq*() (véase ecuaciones (4.1.5) y (4.1.6) del capitulo 4). En el caso del tiempo de búsqueda (usado para encontrar los nodos mínimos en el algoritmo) se tiene una reducción de la complejidad computacional a $O(log(F))$ [67], en las simulaciones del A* clásico los tiempos pueden variar dependiendo de la localización de los elementos buscados en el vector de "frente" donde se pueden encontrar casos que la búsqueda sea relativamente rápida pero a medida que el vector crece (para el caso clásico) el tiempo de búsqueda necesario crece también.

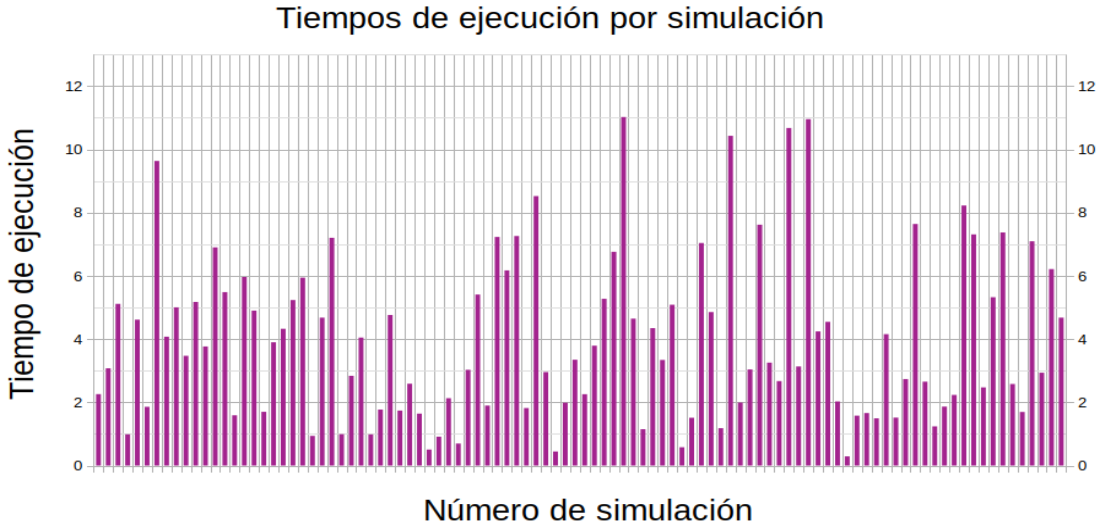


Figura 5.4.4.5. Gráfica del tiempo de ejecución de las cien simulaciones para algoritmo A* modificado (con ordenamiento de pila).

La gráfica del tiempo de ejecución para el enfoque modificado se tiene en la figura (5.4.4.5), en la gráfica se puede apreciar una reducción "log()" en comparación con el enfoque del algoritmo A* clásico que usa búsqueda "no ordenada". El tiempo de ejecución tiene un comportamiento que oscila entre los 10 (seg) y 0.5 (seg) (aproximadamente). Uno de los aspectos más interesantes que presenta la gráfica de la figura (5.4.4.5) es el hecho de que el incremento en la complejidad computacional (que se relaciona con el tiempo de inserción) de inserción a $O(log(F))$ se "compensa" con el tiempo de búsqueda ($O(log(F))$) del algoritmo, y se tiene una reducción en los tiempos de ejecución bastante considerable.

En lo que respecta a los nodos visitados, como puede observarse en la figura (5.4.4.6), el enfoque de A* modificado presenta incrementos y decrementos bastante pronunciados

en el número de nodos visitados, el número máximo de nodos visitados que se observo fue de 160,000 nodos aproximadamente y el mínimo fue de 5,000 nodos visitados aproximadamente.

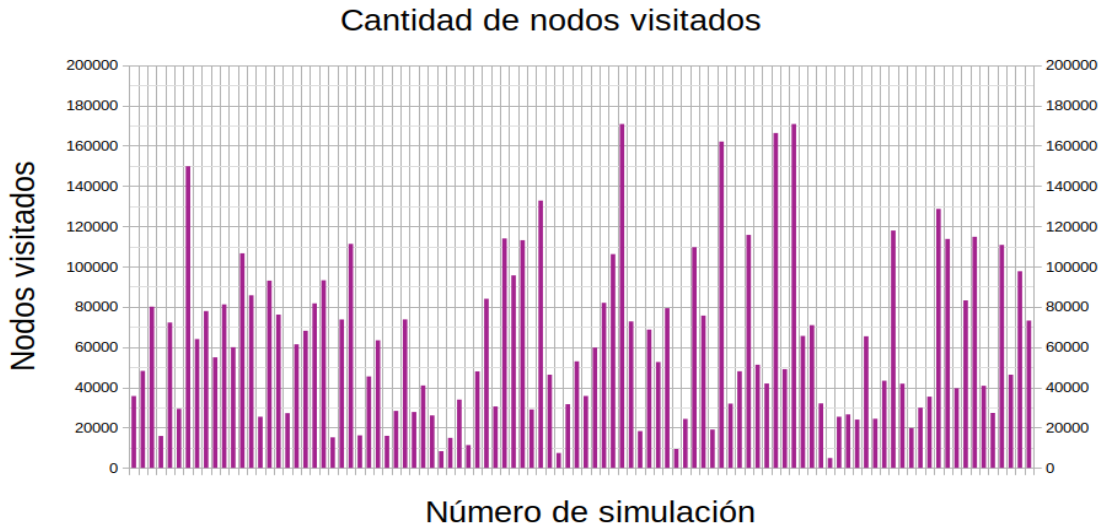


Figura 5.4.4.6. Gráfica del nodos visitados en las cien simulaciones para algoritmo A* modificado.

En la figura (5.4.4.7) se tiene la gráfica de los nodos repetidos entre iteraciones para el enfoque A* modificado, la gráfica muestra comportamientos similares a los observados en el caso del algoritmo A* clásico, las marcas más altas registradas se presentaron a los 60,000 nodos aproximadamente, y las marcas mas bajas oscilaban entre los 100 a 400 nodos.

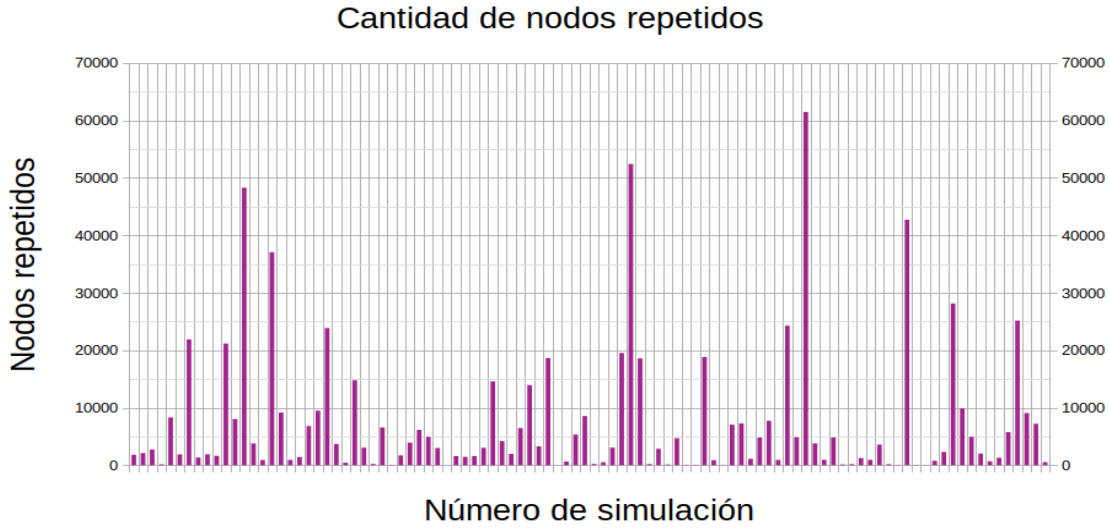


Figura 5.4.4.7. Gráfica del nodos repetidos entre iteraciones en las cien simulaciones para algoritmo A* modificado.

La relación entre el número de nodos visitados (totales) entre el tiempo de ejecución se muestra en la figura (5.4.4.8), en la gráfica se puede observar un aumento bastante considerable en el número de nodos visitados por segundo (n/s), este incremento se debe a la disminución del tiempo de ejecución por simulación, las diferencias entre los tiempos de ejecución permiten al algoritmo A* modificado con *heapq*() ser mucho más eficiente en la cantidad de nodos que se analizan por segundo. La mayor cantidad de (n/s) que se registro fue de 16,750 n/s aproximadamente y la menor cantidad fue de 15,250 n/s aproximadamente.

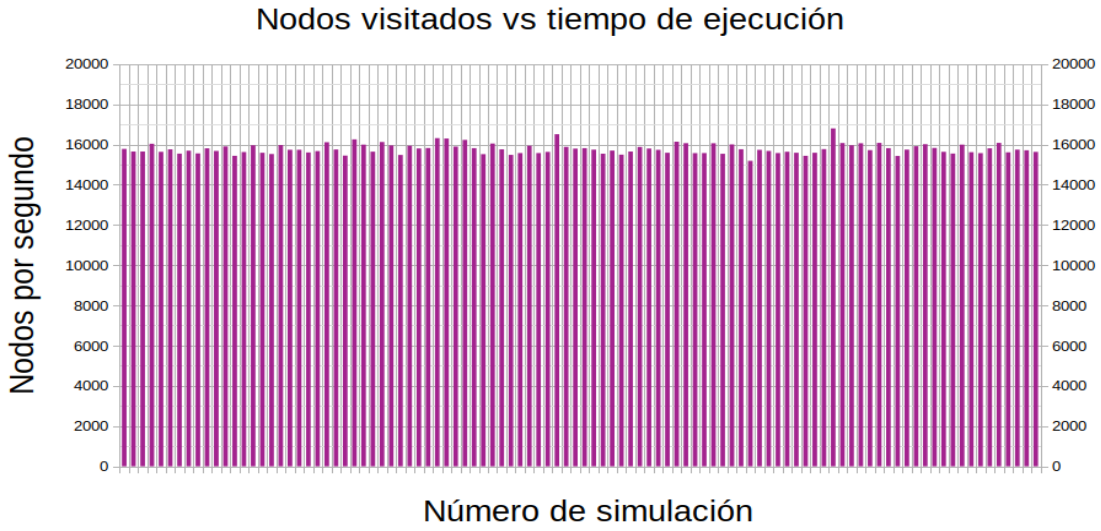


Figura 5.4.4.8. Gráfica del nodos visitados totales entre el tiempo de ejecución, en las cien simulaciones para algoritmo A* modificado.

Utilizando los datos generados de las simulaciones y gráficas presentadas, se creo la tabla (5.4.4.2) donde se colocaron los promedios de las simulaciones para A* modificado.

Promedios de simulaciones de algoritmo A* clásico	
Tiempo ejecución	3.95 (s)
Nodos visitados	61,994.39 (n)
Nodos repetidos	7,473.78 (n)
Nodos visitados vs tiempo	15,782.25 (n/s)

Tabla 5.4.4.2. Generación de promedios de datos de simulaciones de A* modificado.

Simulaciones para algoritmo QPA*

El algoritmo QPA* (en conjunto) se encarga de generar el protocolo de exploración y trazado de ruta para ASTEC, este algoritmo fue diseñado con el propósito de presentar una solución para la carencia del factor de "exploración" de los algoritmos de trazado de ruta, pero una vez siendo analizado el protocolo de exploración de QPA* podía ser incorporado a distintos algoritmos de trazado de ruta que estuviesen basados en teoría de grafos. Es por ello que surge la propuesta de modificar el algoritmo A* para combinarlo de una manera más eficiente con la parte de exploración, QPA* utiliza el modelo base del algoritmo A* modificado con ordenamiento de pila (Heap Queue), donde a este enfoque se le agrega una función de costo en relación a los "nodos repetidos" entre iteraciones para el protocolo de exploración, para las simulaciones se realizaran cambios en la función de costo asociada. La idea central de QPA* es permitir un método de exploración "real" mientras se incorporan las ventajas de un algoritmo de trazado de ruta (como el algoritmo A* modificado en este caso) para realizar los caminos mínimos entre inicio y meta.

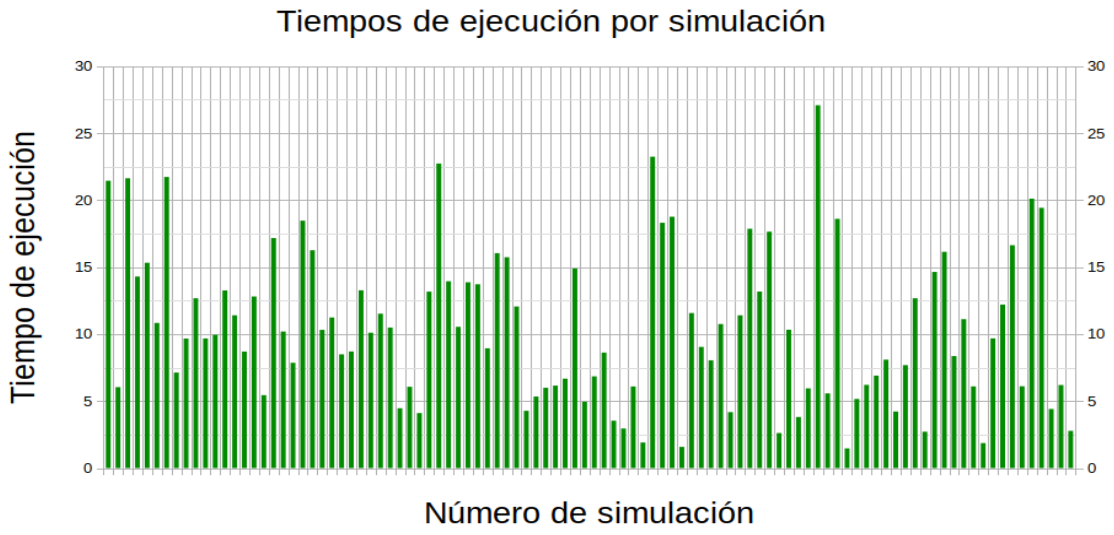


Figura 5.4.4.9. Gráfica del tiempo de ejecución de las cien simulaciones para algoritmo QPA* usando costo de movimiento de conexión como función de costo extra. [2]

La gráfica de tiempo de ejecución para el algoritmo QPA* se tiene en la figura (5.4.4.9), el tiempo muestra un comportamiento similar a los enfoques de A* clásico y A* modificado, donde existen mínimos de tiempo de alrededor de 1 (seg) y máximos de 25 (seg). En general, se observa una reducción considerable de tiempo con respecto al A* clásico, pero con un pequeño aumento de 6 (seg) en promedio con respecto al A* modificado.

[2]Usar el costo de movimiento como costo de "penalidad" se refiere a que el algoritmo "ve" a los nodos pre-visitados similar a repetir un movimiento.

Puede observarse , en la figura (5.4.4.10), que es la gráfica que presenta un máximo de alrededor de 350,000 nodos visitados, este número representa más de la mitad de la matriz de exploración propuesta para las simulaciones, que es de 600,000 nodos; el mínimo registrado entre las simulaciones (en relación a los nodos visitados) es de 18,000 nodos visitados aproximadamente.

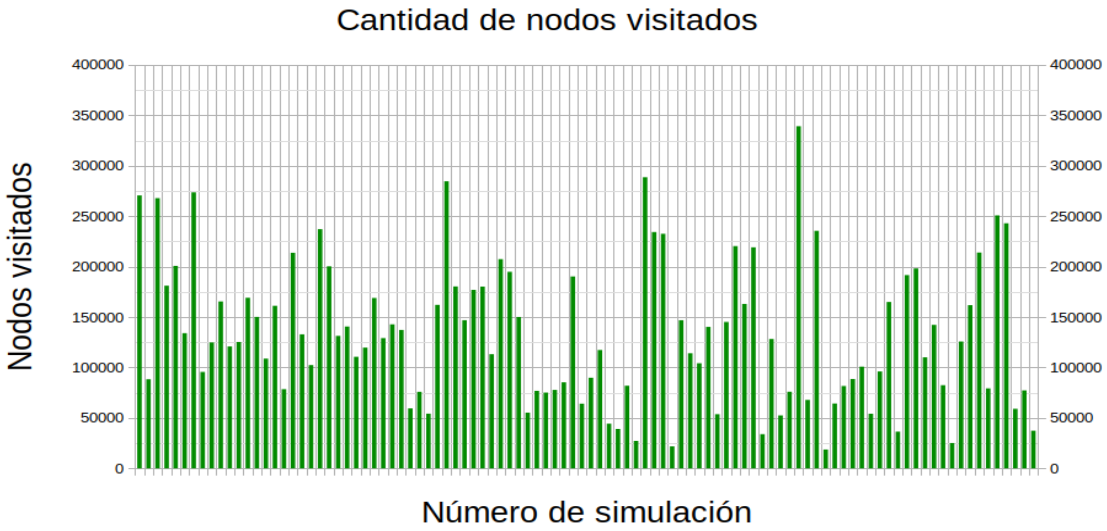


Figura 5.4.4.10. Gráfica del nodos visitados en las cien simulaciones para algoritmo QPA* usando costo de movimiento de conexión como función de costo extra.

En general, la figura (5.4.4.10), muestra una tendencia de aumento en el número de nodos visitados de las simulaciones del algoritmo QPA*, aunque existen saltos irregulares, mantiene una mayor relación de nodos visitados a los otros enfoques.

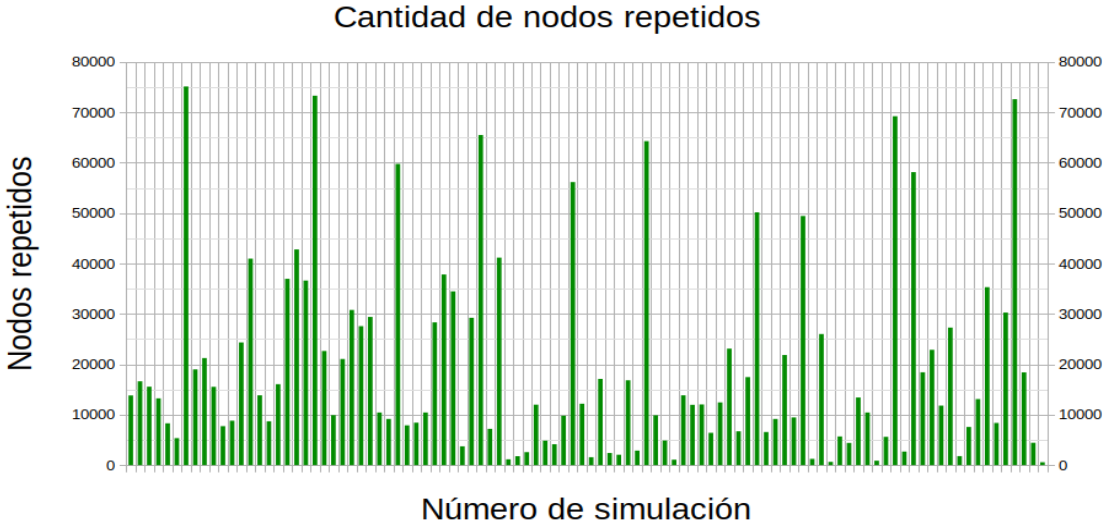


Figura 5.4.4.11. Gráfica del nodos repetidos entre iteraciones en las cien simulaciones para algoritmo QPA* usando costo de movimiento de conexión como función de costo extra.

En la figura (5.4.4.11) se muestran los nodos repetidos para las simulaciones de QPA*, en la gráfica se muestra un aumento en la cantidad de nodos repetidos entre iteraciones, con máximos cercanos a los 80,000 nodos y mínimos de 200 nodos (aproximadamente).

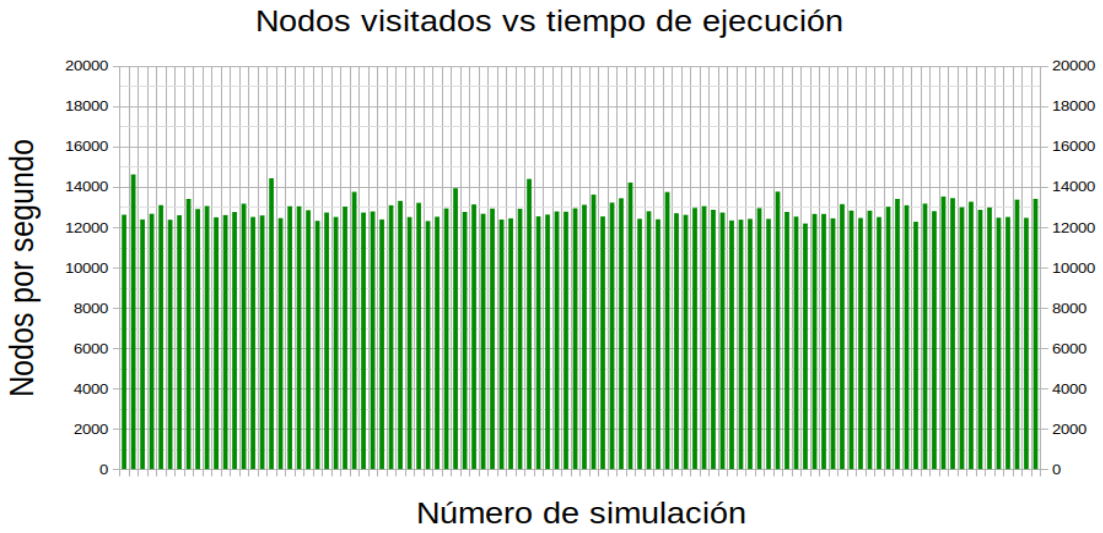


Figura 5.4.4.12. Gráfica del nodos visitados totales entre el tiempo de ejecución, en las cien simulaciones para algoritmo QPA* usando costo de movimiento de conexión como función de costo extra.

Los datos obtenidos para nodos visitados vs tiempo se gráfico la figura (5.4.4.12), la gráfica muestra un comportamiento mas "regular", donde los valores de (n/s) oscilan entre los 14,700 (n/s) y los 12,250 (n/s).

Simulaciones para algoritmo QPA* con aumento de costo de conexión (x100)

Ahora, aumentando el costo de conexión entre nodos "$x100$" (el costo de conexión hacia nodos repetidos se multiplica por 100) como función de costo extra para nodos visitados en el enfoque del algoritmo QPA*.

La gráfica mostrada en la figura (5.4.4.13) se observan los tiempos de ejecución con el cambio de función de costo en QPA*, en la gráfica se observa un aumento en los tiempos de las simulaciones del enfoque QPA*, con máximos al rededor de los 30 segundos y mínimos en los 5 seg, con un promedio de tiempo de ejecución de 18.2791 seg.

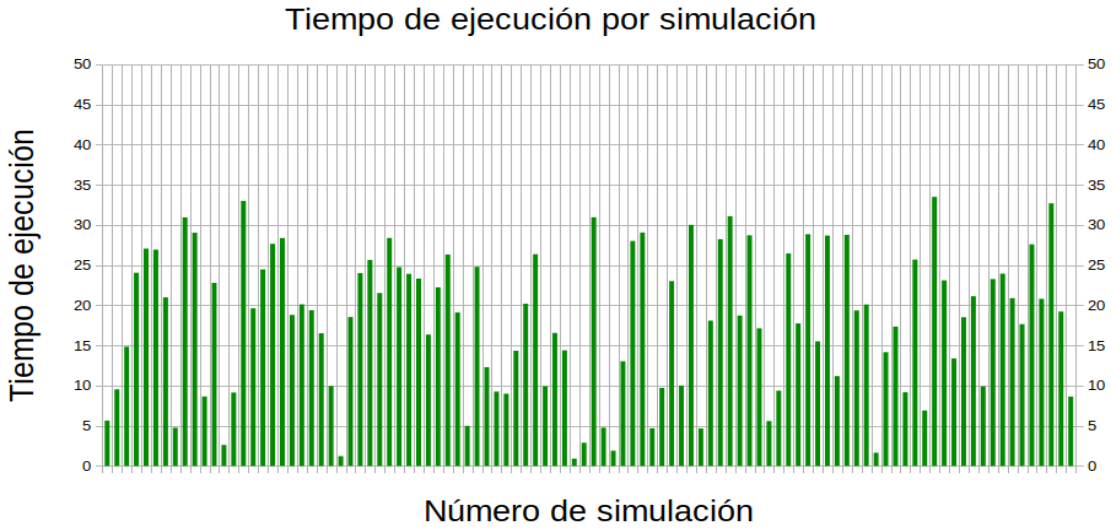


Figura 5.4.4.13. Gráfica del tiempo de ejecución de las cien simulaciones para algoritmo QPA* cambiando función de costo de nodos visitados.

En la figura (5.4.4.14) se muestran los datos de simulación referentes a la cantidad de nodos visitados con el cambio de función de costo.

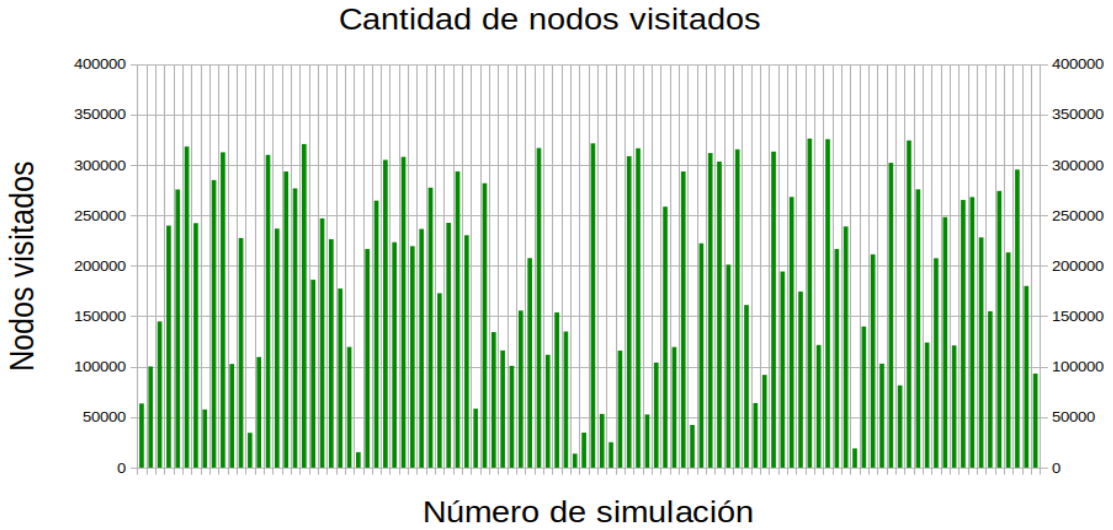


Figura 5.4.4.14. Gráfica del nodos visitados en las cien simulaciones para algoritmo QPA* cambiando función de costo de nodos visitados.

Como se observa en la gráfica (figura (5.4.4.14)) la cantidad de nodos visitados aumenta de manera considerable , donde los máximos oscilan entre los 250,000 nodos y los mínimos entre los 50,000 nodos. El promedio observado de nodos visitados es de 197280.77 nodos.

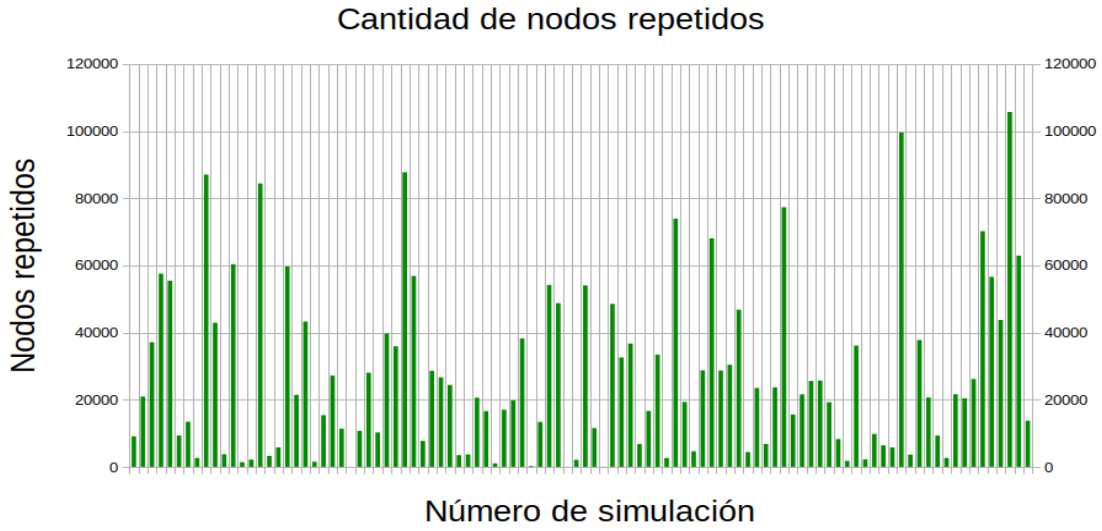


Figura 5.4.4.15. Gráfica del nodos repetidos entre iteraciones en las cien simulaciones para algoritmo QPA* cambiando función de costo de nodos visitados.

En la figura (5.4.4.15) existe un aumento en la cantidad de nodos repetidos entre iteraciones de la simulación, siendo el enfoque que obtuvo el mayor promedio de nodos repetidos con 27236.32 nodos repetidos, con máximos cercanos a los 100,000 nodos y mínimos cercanos a los 1,000 nodos.

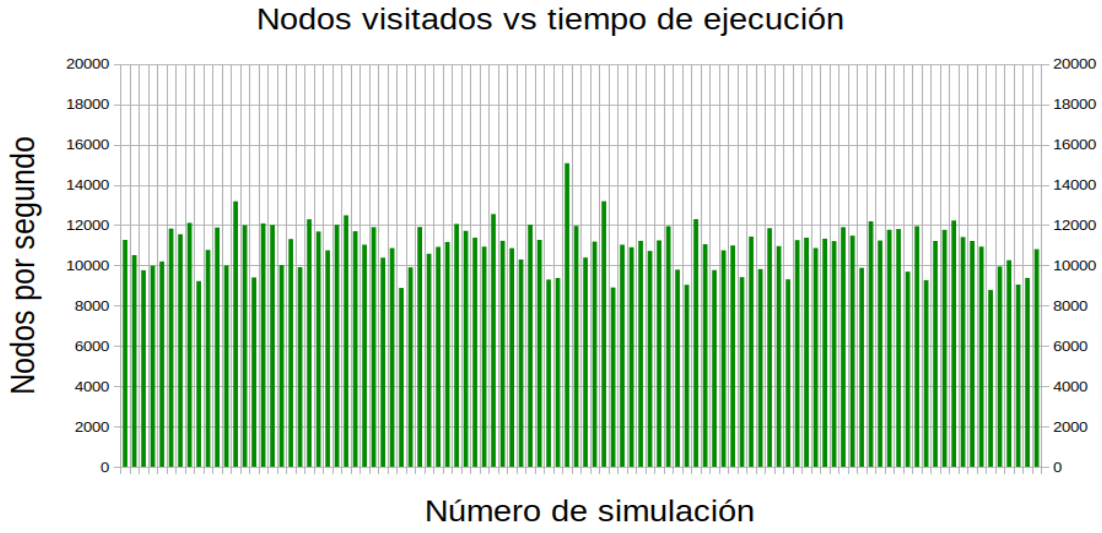


Figura 5.4.4.16. Gráfica del nodos visitados totales entre el tiempo de ejecución, en las cien simulaciones para algoritmo QPA* cambiando función de costo de nodos visitados.

La ultima gráfica presentada en la figura (5.4.4.16) se muestra un decremento en la "eficiencia" de la cantidad de nodos visitados por segundos, el promedio de (n/s) cambiando la función de costo en QPA* es de 10985.24 (n/s).

Promedios de simulaciones para algoritmos QPA*			
Algoritmo QPA* costo de conexion (x1)		Algoritmo QPA* costo de conexion (x100)	
Tiempo ejecución	10.49 (s)	Tiempo ejecución	18.27 (s)
Nodos visitados	134,071.55 (n)	Nodos visitados	197,280.77 (n)
Nodos repetidos	19,576.09 (n)	Nodos repetidos	27,236.32 (n)
Nodos visitados vs tiempo	12,892.99 (n/s)	Nodos visitados vs tiempo	10,985.24 (n/s)

Tabla 5.4.4.3. Generación de promedios de datos de simulaciones de algoritmos QPA*.

En la siguiente subsección se muestran las gráficas comparativas (desplegando los múltiples resultados obtenidos) con el fin de proporcionar un mejor análisis y demostración de los resultados de los tres enfoques (algoritmo A* clásico, algoritmo A* modificado con ordenamiento de pila y algoritmo QPA* con costo de conexión $x1$ como función de costo para nodos visitados).

Gráficas comparativas entre enfoques de algoritmo A*

En esta sección se presentan las gráficas comparativas para el tiempo de ejecución, cantidad de nodos visitados, nodos repetidos entre iteraciones y nodos visitados vs tiempo de ejecución, para los enfoques de algoritmo A* clásico, A* modificado y QPA* (con costo de conexión $x1$).

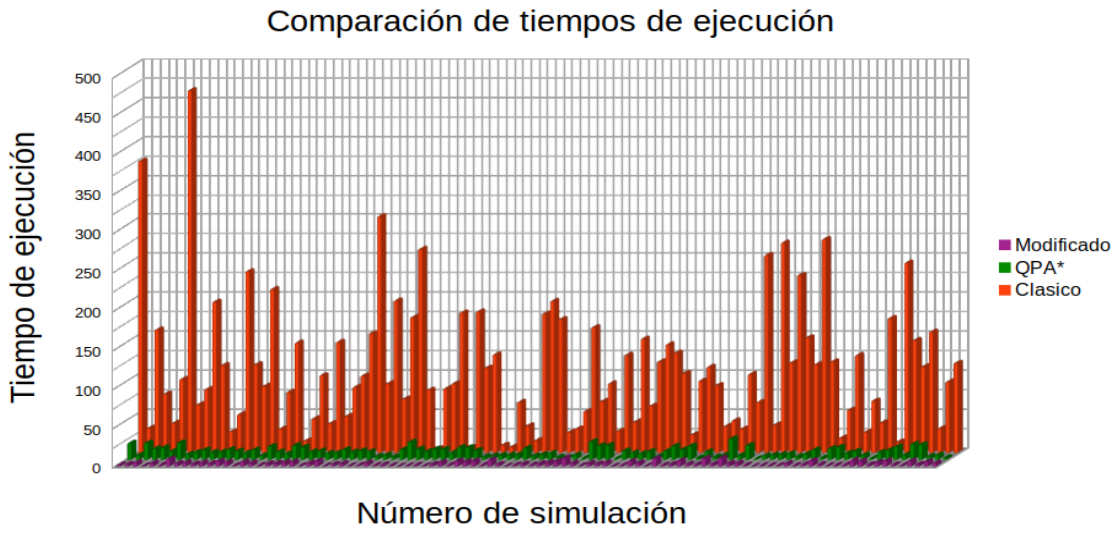


Figura 5.4.4.17. Gráfica comparativa de tiempos de ejecución para los tres enfoques.

En la figura (5.4.4.17) se pueden apreciar las diferencias que se presentaron en las simulaciones entre el tiempo de ejecución de los distintos enfoques, la principal diferencia

se puede observar en el algoritmo A* clásico, este comportamiento se esperaba debido al uso de búsqueda "no ordenada", donde pueden existir tiempos muy altos o tiempos muy bajos dependiendo de la localización de los elementos en el vector, sin embargo, existe una tendencia a tiempos por encima de los 100 (seg) aproximadamente.

En lo que corresponde al algoritmo A* modificado y al algoritmo QPA*, los tiempos de ejecución se mantienen "relativamente" similares entre los dos, con una tendencia de 3 a 6 (seg) en aumento de tiempo para el algoritmo QPA* con respecto al A* modificado, pero como puede observarse el principio de la búsqueda "ordenada" en ambos enfoques (A* modificado y QPA*) presenta una mejora considerable en el tiempo de ejecución con respecto al algoritmo A* clásico.

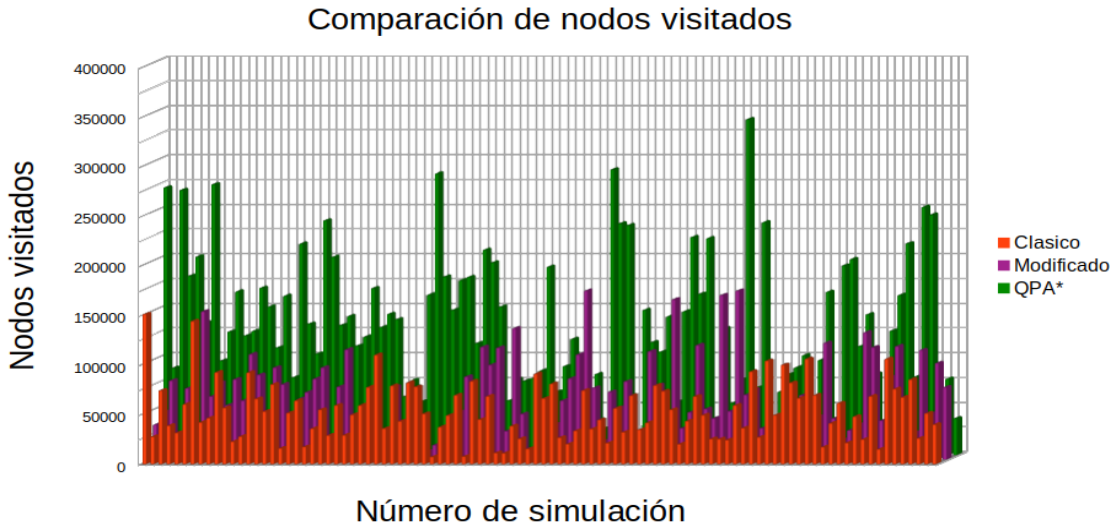


Figura 5.4.4.18. Gráfica comparativa de nodos visitados totales para los tres enfoques.

Los datos comparativos de los nodos visitados se tienen en la figura (5.4.4.18), en general, el comportamiento de los enfoques A* clásico y A* modificado son relativamente "análogos", donde existen simulaciones donde el algoritmo A* clásico presenta una mayor cantidad de nodos visitados que el algoritmo A* modificado, y viceversa.

También, en la figura (5.4.4.18), el algoritmo QPA* fue el enfoque que tuvo la mayor cantidad de nodos visitados (en promedio) y en los casos específicos presento el número más alto de nodos visitados totales, este comportamiento comprueba una de las premisas creadas para implementar este enfoque, dado que la ecuación de evaluación del costo de los nodos agrega un parámetro de nodos visitados entre iteraciones, el algoritmo tiende a visitar una mayor cantidad de nodos para encontrar la ruta con costo mínimo, esto contrasta con la estrategia de evaluación por nodo para A* clásico y A* modificado que solo obtienen el costo de conexión entre nodos (que depende del tipo de movimiento que sea explorado) y la distancia Euclidiana entre ese nodo y la meta. La función de extra

costo puede ser modificada en función de la tarea que se busque, en el caso del algoritmo QPA* que se diseño para ASTEC, el costo extra no es tan "elevado", este costo se puede interpretar como un "doble" movimiento, entonces el costo por movimiento de exploración aumenta al doble, sin afectar el costo de distancia del nodo a la meta. Si el costo extra de nodos visitados aumenta, la tendencia del algoritmo a no "re-visitar" esas zonas aumenta, esto se asemeja al comportamiento del algoritmo de campos de potencial, en este algoritmo existen zonas con un mayor costo si son visitadas y zonas que son menos costosas si son visitadas, de manera similar la función p_{vis}^{t-n} (de la ecuación (4.1.7)) presenta estas zonas visitadas como zonas mas costosas, y se puede modificar la función para generar zonas más "rigurosas", es decir, zonas que son tan costosas que el algoritmo QPA* puede prácticamente descartar de los nodos posibles para la ruta mínima.

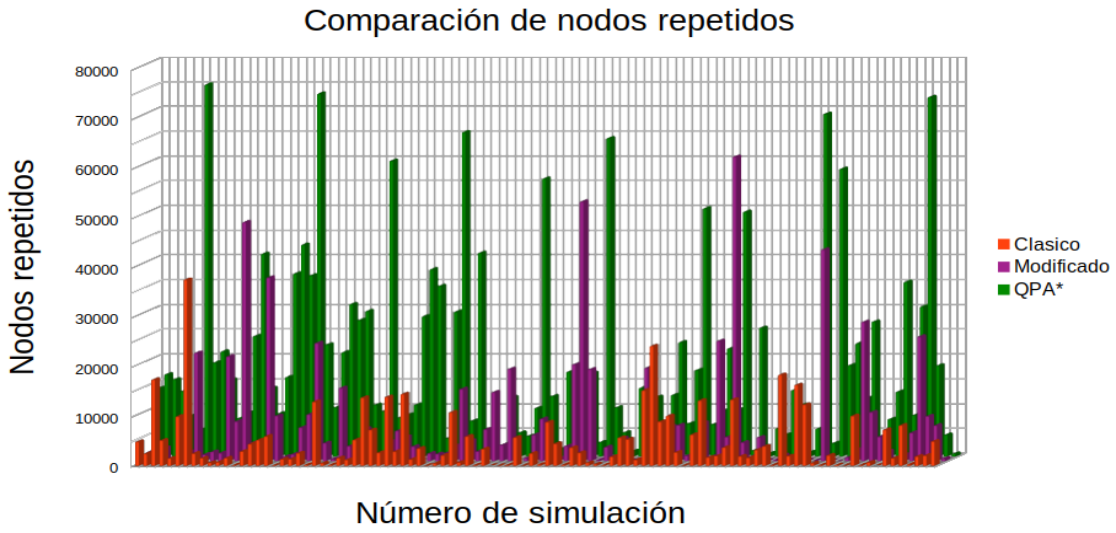


Figura 5.4.4.19. Gráfica comparativa de nodos repetidos entre iteraciones para los tres enfoques.

La información de los nodos repetidos de los tres enfoques (figura (5.4.4.19)) mostró datos bastante interesantes en la comparativa, esto se debe principalmente en el algoritmo QPA* que se esperaba un descenso en la cantidad de nodos repetidos entre iteraciones debido al costo extra hacia nodos visitados, es decir, que se pretendía un aumento en nodos visitados (como en que se mostró en la gráfica anterior) y un descenso en la cantidad de nodos repetidos, esta tendencia se puedo observar en las simulaciones gráficas, cuando la trayectoria cumplía con dos condiciones: que la meta se encontrara en un cuadrante opuesto al del nodo de inicio, y que en esta trayectoria existieran áreas delimitadas previamente exploradas; sin embargo, al realizar las cien simulaciones por enfoque los datos presentaron otra tendencia, siendo el algoritmo QPA* es el enfoque con más nodos repetidos, seguido por el enfoque del algoritmo A* modificado y con el número menor de nodos repetidos es el algoritmo A* clásico. Al analizar los datos con mayor detenimiento al información de la

gráfica, a pesar de que QPA* presenta una mayor cantidad de nodos repetidos, la cantidad de nodos "reales" (nodos visitados totales menos los nodos repetidos) son mayores que el enfoque A* clásico y A* modificado, este resultado se desarrolla debido a que cuando el algoritmo cae en una zona de costo extra (debido a los nodos visitados) cualquiera de los caminos que tome en esa zona tendrá un costo "doble" en función al movimiento explorado, y debido a que el costo extra es relativamente "bajo", el algoritmo aumenta su zona de exploración y aumenta la cantidad de los nodos repetidos, sin embargo, el aumento en nodos visitados es mucho mayor al aumento generado en nodos repetidos.

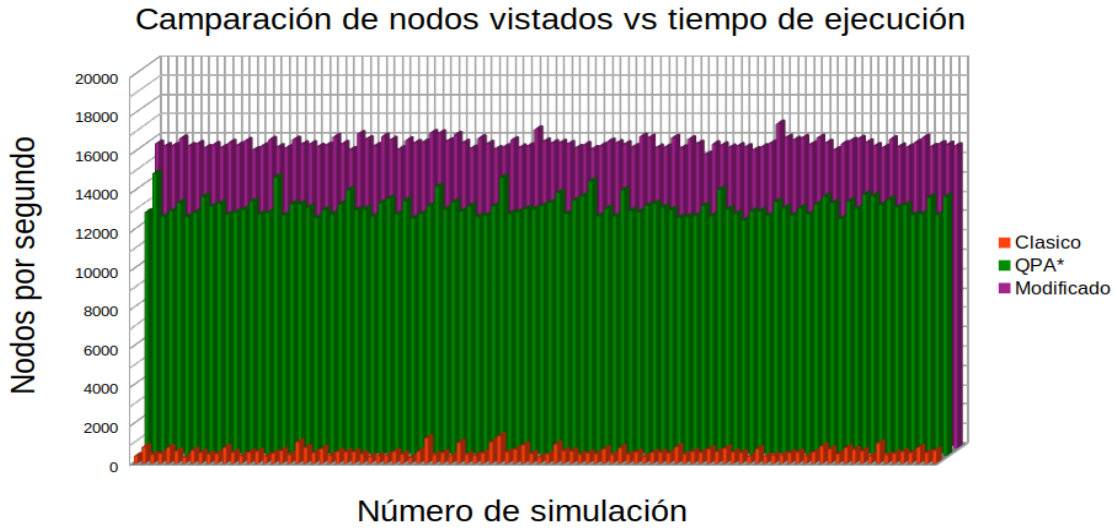


Figura 5.4.4.20. Gráfica comparativa de nodos visitados totales entre el tiempo de ejecución para los tres enfoques.

Por ultimo, en la figura (5.4.4.20) se tiene la gráfica comparativa para nodos visitados vs tiempo de ejecución (n/s), para el caso del algoritmo A* clásico (en color rojo) es el algoritmo que presenta la menor relación de nodos visitados por segundo, este resultado era previsto por la proporción de tiempo de ejecución y los nodos visitados del algoritmo, debido a que la búsqueda "no ordenada" presenta un mayor tiempo de búsqueda que los otros enfoques, el A* clásico visita menos nodos por segundo que el algoritmo A* modificado o el algoritmo QPA*.

El enfoque que presenta una mayor relación de (n/s) es el A* modificado (color morado) y el algoritmo QPA* presenta un ligero decremento en relación al A* modificado, en la relación del número de nodos visitados por segundo, esto se presume que pueda deberse a la incorporación de una "condición" dentro del kernel de exploración del algoritmo, esta condición "pregunta" si en esa posición i y j que esta siendo explorada, existe algún valor dentro de la matriz de "nodos visitados anteriores", y a pesar de que se busca una ubicación especifica en (i, j), al repetirse esta pregunta en cada nodo explorado el tiempo de exploración por nodo aumenta, lo que al final se traduce en un decremento de

la eficiencia de nodos visitados por segundo del algoritmo QPA*. Se puede decir que a pesar de que existe un decremento en la cantidad de (n/s) la eficiencia obtenidas para QPA* supera por mucho al enfoque A* clásico, y aunque no es tan eficiente como el A* modificado, el algoritmo QPA* presenta una mayor cantidad de nodos visitados.

En la teoría que se ha investigado referente al algoritmo de trazado de ruta A*, no existe una gran diferencia si el "número de nodos visitados" aumenta o decrementa, debido a que estas áreas de nodos solo son contemplados como rutas mínimas posibles hacia la meta, sin embargo, para el problema de localización y mapeo (y la exploración de la zona de despliegue), la cantidad de nodos visitados si puede tener una repercusión considerable en la información obtenida del mapa, esto se debe a que las rutas mínimas solo pueden crearse por zonas de nodos visitadas, entonces, si la zona de nodos visitados es reducida, la posibilidad de trazar rutas por "nuevas" zonas también se reduce y si existe una mayor área de nodos visitados existe una mayor posibilidad de trazar rutas por zonas "nuevas", zonas donde no se tenga información precisa del entorno, que al final resultan en datos importantes para las tareas de localización y mapeo del diseño de ASTEC, y en general, de la cantidad de información generada del mapa de exploración.

5.5. Simulaciones del algoritmo de localización y mapeo basado en Fast SLAM

En esta sección se mostrarán los datos obtenidos de las simulaciones del algoritmo de localización y mapeo que incorpora el enfoque de Fast SLAM. La figura (5.5.1) muestra un escenario ejemplo de las lecturas de *Claus Brenner* en "SLAM Lectures" [50], este escenario ejemplo cuenta con seis obstáculos distribuidos en el mapa, los obstáculos son representados como cilindros. Los archivos que se ocupan para estas simulaciones consisten en dos archivos de texto (.txt), uno corresponde a los comandos de control (los movimientos de desplazamiento del robot en el mapa), en total existen 260 movimientos en la lista de comandos, donde por cada comando se realiza un barrido del sensor LIDAR con 660 mediciones por cada movimiento (en los 260 movimientos totales).

Las simulaciones de esta sección mostrarán los distintos elementos que conforman a Fast SLAM, después se mostrara la simulación del enfoque Fast SLAM con diferentes números de partículas y algunas comparativas con otro tipo de enfoques que se pueden ocupar para resolver el problema de SLAM (localización y mapeo simultaneo).

El mapa ejemplo se muestra en la figura (5.5.1), en color gris se muestran las "marcas de mapa", que son representadas como círculos (esto se da por la representación en dos dimensiones del mapa, estos círculos corresponden a cilindros en tres dimensiones que están colocados en el mapa), y en color negro se tiene la "trayectoria real" del robot, esta trayectoria es dada como una serie de comandos de control, y lo que se busca con el

algoritmo de localización y mapeo es obtener una representación de estas marcas de mapa y trayectoria del robot, que sean lo mas cercanas a la “trayectoria real” y a la “ubicación real” de las referencias del mapa.

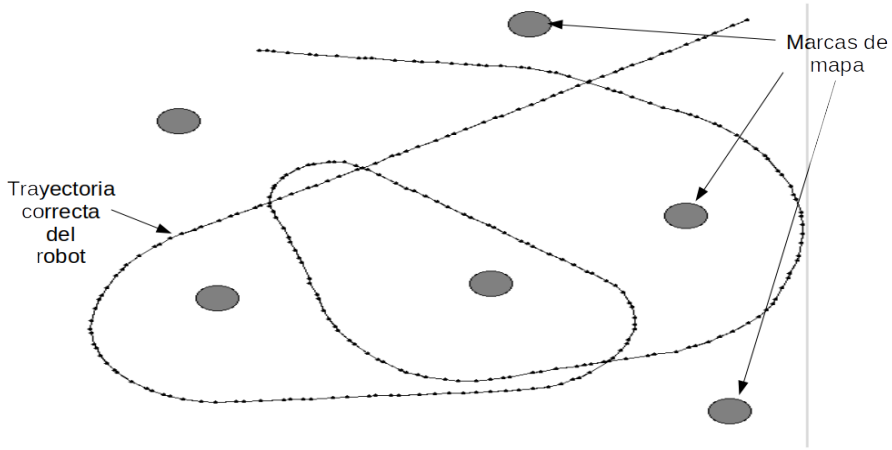


Figura 5.5.1. Mapa de simulación para Fast SLAM, con la localización de los cilindros y la trayectoria correcta del robot.

Filtro Extendido de Kalman para enfoque Fast SLAM

Como se comento en el marco teórico, el enfoque de “Fast SLAM” es básicamente, la combinación de los filtro extendidos de Kalman (para la localización y actualización de las marcas) y el filtro de partículas Rao-Blackwellised (para la localización y actualización de la posición del robot). Como primera instancia tenemos la aplicación de los filtros extendidos de Kalman (observar figura (5.5.2), estos filtros aplican el modelo de observación planteado para el diseño ASTEC, este modelo de observación sirve para detectar y actualizar las marcas de mapa.

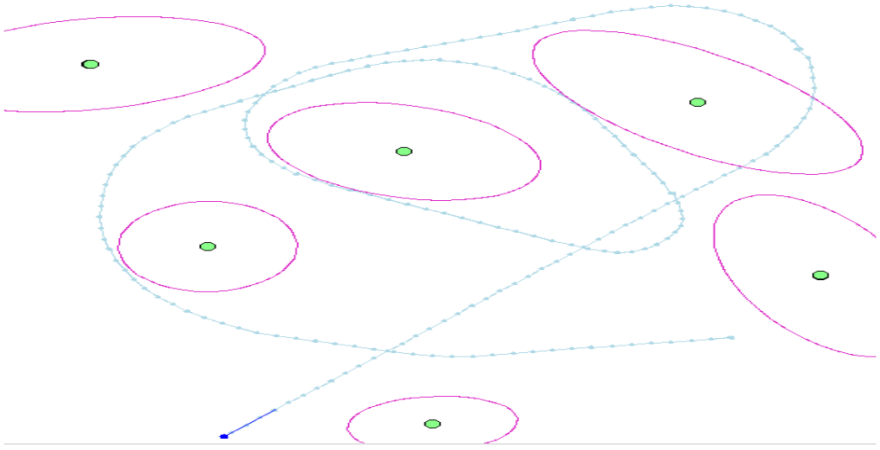


Figura 5.5.2. Mapa de simulación para Fast SLAM con otro punto de inicio, y con los filtro extendidos de Kalman asociados a las marcas.

En la figura (5.5.2) se puede observar la asignación de los filtro de Kalman a las marcas de mapa, en el enfoque de "Fast SLAM" cada filtro de Kalman es independiente del resto, es decir, que cada filtro se establece y se actualiza de forma independiente. La estrategia de generar filtro separados para cada marca se propone para poder lidiar con la "maldición de dimensionalidad" inherente del enfoque "EKF SLAM" (SLAM usando solo filtros extendidos de Kalman), y en general de los filtros de Kalman. Cada marca se expresa en color verde, todas las marcas (es decir, cada filtro) posee una incertidumbre asociada (al inicializar la marca), esta incertidumbre va en decremento o en aumento dependiendo de las mediciones que se realicen sobre ella, estas funciones en la mayoría de los casos ayudan a decrementar la incertidumbre asociada a la marca (mostrada como la elipse violeta al rededor de las marcas de mapa en la figura (5.5.2)), los puntos verdes presentes en la figura (5.5.2) se pueden interpretar como la estimación de la posición "real" de la marca del mapa que esta siendo detectada y procesada por los sensores y la unidad de control y procesamiento del robot, esta ubicación es la media de las distribuciones marginales para (x, y), la incertidumbre en el ángulo de orientación θ se refleja como una "rotación" de la elipse violeta (la incertidumbre) de la marca en cuestión. En color azul (en la parte inferior de la figura (5.5.2)) se muestra la representación del robot en la trayectoria, con una linea en color azul indicando la orientación del robot.

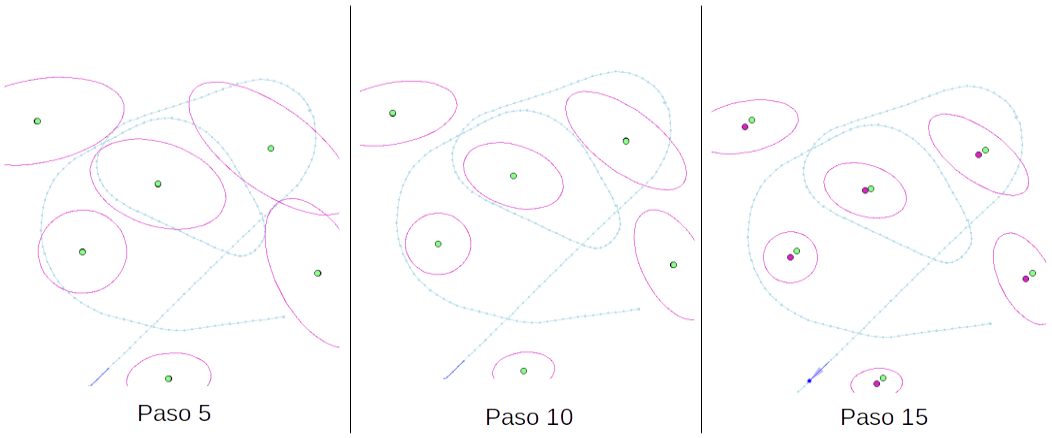


Figura 5.5.3. Mapa de simulación para Fast SLAM con otro punto de inicio, decremento de incertidumbre de las marcas de mapa.

En la figura (5.5.3) se muestra la progresión de la "actualización" y "corrección" de las incertidumbres de las marcas de mapa a través de los filtro de Kalman, en la imagen se muestran etapas de 5, 10 y 15 pasos (cada paso representa un instante o un comando de control y medición generado por el robot en el mapa) del paso 0 al paso 12 el robot se simula "estático", sin embargo, en cada paso realiza un barrido de mediciones con el

sensor LIDAR. En la etapa que corresponde a los 5 primero pasos se observa que las incertidumbres asociadas a las marcas de mapa (elipses color rosa) son muy grandes en relación a las medidas físicas de las marcas e inclusive en relación al mapa, después para el paso 10 (aún el robot permanece inmóvil) se observa que las incertidumbres asociadas a las marcas de mapa va decrementando, pero, aún las marcas más alejadas con respecto al robot (que se encuentra en la parte inferior izquierda de cada etapa) permanecen con una incertidumbre muy elevada, en la última etapa mostrada (que corresponde los 15 pasos) se observa que las posiciones e incertidumbres de las marcas comienzan a variar debido a que para esta etapa el robot comienza a seguir la trayectoria planteada, los puntos color violeta representan la posición "conocida" o previa de las marcas en función a los datos procesados en iteraciones pasadas, los puntos verdes representan la posición que se esta detectando en ese instante, aun que la posición de las marcas ya no se considere estática (debido a que el robot en un principio no se movía) la incertidumbre sigue decrementando a medida que aumentan los barridos del sensor LIDAR.

Una característica importante que se aprecia en estas simulaciones, es el hecho de que los parámetros de calibración del filtro de Kalman asocian una menor incertidumbre al proceso de medición, es decir, que se considera más exacto, que el error al presentar una transición de estado (realizar un movimiento), esto se comprueba con el decremento de la elipse de error de las marcas de mapa a medida que el número de "actualizaciones" (mediciones) aumenta a pesar de que el robot realiza movimientos a lo largo del mapa.

Filtro de Partículas Rao-Blackwellised para enfoque Fast SLAM

Ahora, como siguiente elemento del enfoque de Fast SLAM, es la sección del filtro de partículas llamado "Filtro de Partículas Rao-Blackwellised". En general, se puede decir que el etapa del filtro de partículas dentro de Fast SLAM se encarga principalmente de la localización del robot durante la trayectoria, sin embargo, como se expuso en capítulos anteriores.

Cada partícula (en la figura (5.5.4)) del filtro corresponde a un posible estado hipotético de ASTEC, donde cada partícula posee una ubicación y sus propias referencias de mapa (que son actualizadas y mapeadas con los filtros de Kalman), y en conjunto todas estas partículas forman el filtro de partículas. En la mayoría de las aplicaciones, y también para propósitos de este trabajo, no es conveniente expresar todos los estados de cada partícula generada en las simulaciones y es de hecho una técnica bastante común obtener los promedios de ubicación y orientación de estas partículas, y con ello seleccionar la partícula más cercana al promedio y expresar tanto su ubicación y trayectoria así como las referencias de mapa que esta partícula "observa".

En la figura (5.5.4), se observa la distribución de las partículas en una cierta posición del mapa, tomando como referencia la linea roja como la trayectoria "deseada", se ve que las partículas se encuentran distribuidas en una cierta área que puede ser ajustada según la tarea que se busque resolver.

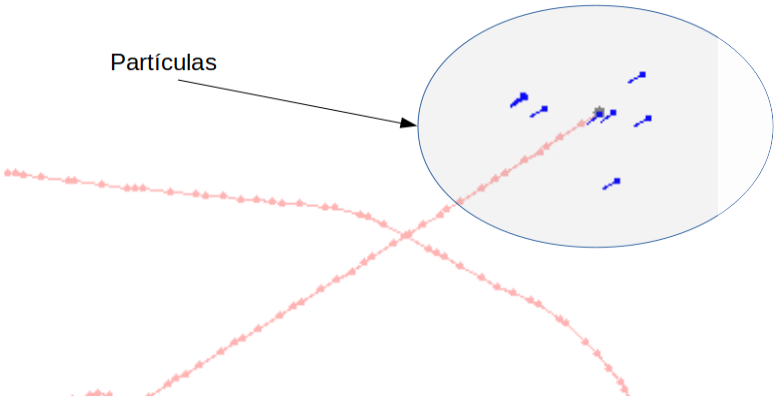


Figura 5.5.4. Distribución de filtro de partículas, sin trayectoria y sin marcas de mapa.

Ahora, en la figura (5.5.5) se realiza una simulación del mapa ejemplo, donde se generaron 500 partículas repartidas acorde a una distribución gaussiana de tres variables (x, y, θ) con desviación estándar de 100 milímetros aproximadamente (para x, y) y de 0,0017 grados para θ.

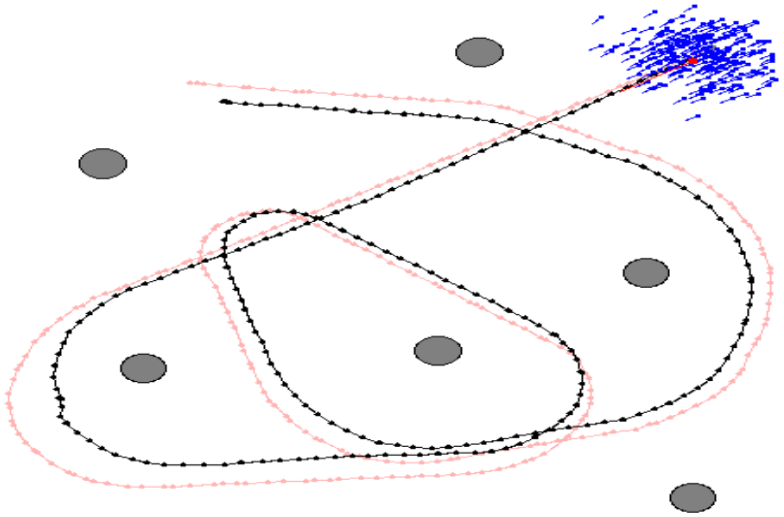


Figura 5.5.5. Filtro con cien partículas, calculo de densidad y trayectoria generada.

En la parte superior derecha de la figura (5.5.5) se observa el cúmulo de partículas, y en el centro del cúmulo de partículas se encuentra la "partícula promedio" en color rojo,

esta partícula refleja los promedios de todas las partículas en (x, y, θ) (entendiéndose θ como la orientación de las partículas); la trayectoria generada a partir del cálculo de la densidad de las partículas se muestra en color negro, la trayectoria deseada (o trayectoria ejemplo) se muestra en color rosa y las referencias del mapa en color gris.

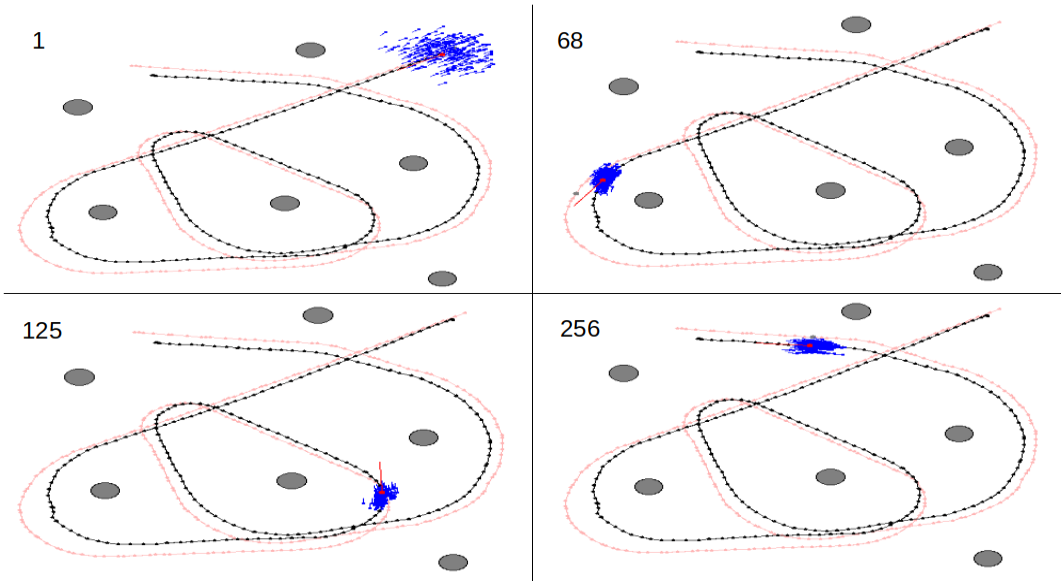


Figura 5.5.6. Filtro con cien partículas, calculo de densidad en distintas etapas de la trayectoria generada.

En la figura (5.5.6) se muestra una serie de imágenes de distintos "pasos" o "etapas" de la trayectoria ejemplo para ASTEC, cada imagen cuenta con un numero en la parte superior izquierda que representa el numero del paso correspondiente a la imagen. Empezando por la imagen superior izquierda, que representa al primer paso de la trayectoria, se observa el cumulo de partículas distribuido acorde a las especificaciones de la simulación, en general, los primeros pasos representan una dispersión mayor de las partículas, esto se debe a que la generación de las partículas se establece (al inicio) sin contemplar las referencias "visibles" para cada partícula, pasando a la imagen del paso 68 observamos una dispersión reducida en comparación con los primero pasos, sin embargo, como puede verse en la trayectoria en color negro (que representa la trayectoria generada por las partículas) existe una curva mas pronunciada que la trayectoria "deseada" (la trayectoria real, puesta en color rosa), y esto genera un error que se conserva en cierta medida hasta el final de la trayectoria. Para los pasos 125 y 256, el comportamiento de las partículas se mantiene, pero nunca se llega mejorar la proximidad de la trayectoria de las partículas a la trayectoria deseada.

5.5.1. Simulación de etapa de predicción para enfoque Fast SLAM

Teniendo en cuenta los elementos principales que forman el enfoque de Fast SLAM pensado para ASTEC, se realizaron simulaciones para entender las etapas de "predicción" y "corrección" internas de Fast SLAM, y algunas simulaciones para problemas como: el problema del "robot secuestrado" (Kidnapped Robot Problem) y observar como el número de partículas puede afectar a los procesos de localización y mapeo simultáneos de este enfoque.

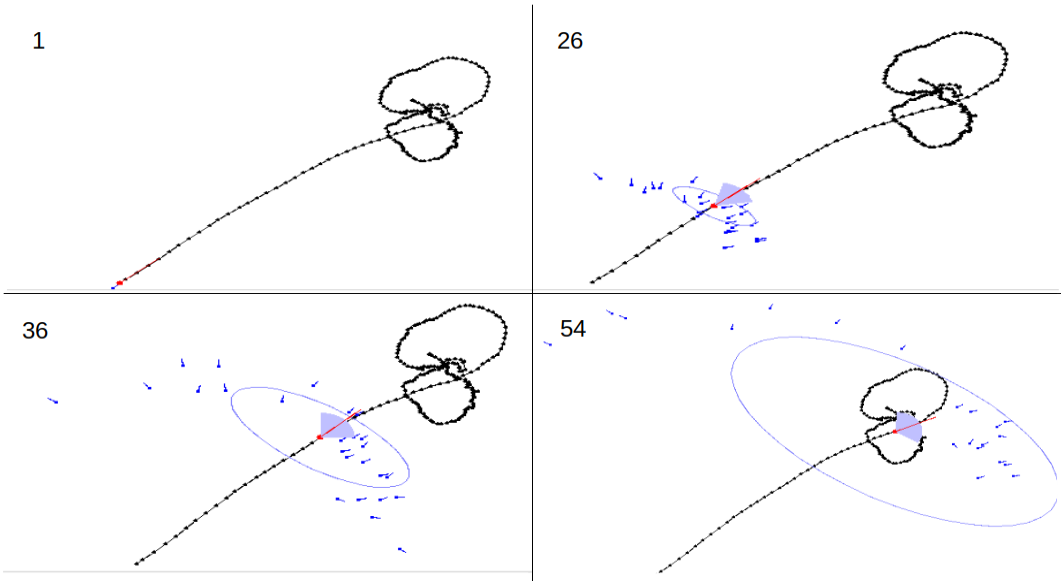


Figura 5.5.1.1. Etapa de predicción para enfoque Fast SLAM con 25 partículas.

La primer etapa interna del enfoque Fast SLAM consiste en la predicción, esta etapa establece la aplicación del modelo de transición de estado o modelo de movimiento planteado para ASTEC explicado en el capítulo 4. La etapa de predicción se encarga de generar y aplicar un movimiento de manera pseudo-aleatoria para cada partícula, este movimiento esta relacionado con la posición de la partícula y la incertidumbre asociada a los movimientos del robot, estas incertidumbres se establecen como parámetros, para el caso de ASTEC, la incertidumbre para el movimiento en linea recta (cuando los valores de las ruedas r_izq y r_der son iguales) es de 0,35 milímetros, es decir, que existe un error asociado al movimiento del robot cuando cambia de un lugar a otro en linea recta. El error asociado al movimiento de curva de ASTEC (cuando los valores de las ruedas r_izq y r_der son diferentes) se estableció en 0.6, este error en específico afecta a la orientación del robot, que al final se traduce en una desviación de la trayectoria aunado con el error de traslado en linea recta.

En la figura (5.5.1.1) se muestra una serie de capturas de cuatro eventos dentro de la simulación donde solo se aplica la etapa de predicción de "Fast SLAM", cada evento

cuenta con su número de paso asociado, además, el inicio "tentativo" para las partículas se establece en la parte inferior, a diferencia de las simulaciones anteriores. Para el paso 1 se tiene que las partículas parten de un punto en especifico en ($x = 500,\ y = 0,\ \theta =$ 0,78539 $rads$), el primer factor observado es que la trayectoria del robot generada por las partículas y los filtros de Kalman del enfoque (en color negro) es completamente errónea, para el paso 26 se comienza a observar el proceso llamado "diversidad", este proceso se manifiesta de manera inherente en el enfoque de Fast SLAM, sin embargo, dado que no existe etapa de corrección, la diversidad de las partículas comienza a notarse de manera abrupta, dando como resultado la ubicación errónea de ASTEC, una mayor incertidumbre en posición y orientación, y como resultado una trayectoria errónea, a partir del paso 26 y los subsecuentes (mostrados en la figura) existe una elipse en la "partícula promedio" (que se ve en color rojo) esta elipse representa la incertidumbre sobre la posición del robot en (x, y) y tiene una relación de incremento a medida que las dispersión de las partículas aumenta, además, la partícula promedio contiene un "cono" en dirección de la linea roja que se coloca como la media de este cono que se asocia como el error en θ de ASTEC, el error de orientación también aumenta a medida que las direcciones de las partículas existentes diverge.

5.5.2. Simulación de etapa de corrección para enfoque Fast SLAM

El siguiente elemento que se aplica para el enfoque Fast SLAM en ASTEC es la etapa de corrección. La etapa de corrección se aplica posterior a la etapa de predicción, la corrección que se genera en Fast SLAM se lleva acabo a través de la información de los sensores del robot, estos sensores pueden ser cámaras, sensores de distancia ultrasónicos o láser, etc. En la etapa de predicción se aplica de manera pseudo-aleatoria un movimiento para cada partícula usando el modelo de movimiento (ecuaciones (4.2.2.1.3) - (4.2.2.1.5) y (4.2.2.1.12) - (4.2.2.1.13)) planteado para ASTEC, para la etapa de corrección se realiza la aplicación del modelo de observación (ecuaciones (4.2.2.2.1) - (4.2.2.2.3)) que utiliza un sensor láser omnidireccional para su modelado, esta etapa de corrección se aplica para cada una de las partículas.

Cada partícula en el mapa cuenta con su sistema de referencias (marcas de mapa) detectado, cada marca de mapa cuenta con su filtro extendido de Kalman asociado y es actualizado (corregido) individualmente cada vez que las mediciones relacionan a la ubicación de esa referencia en especifico. Para evitar que existan ambigüedades en la relación de la localización de ASTEC, dado que cada partícula "ve" sus propias marcas de mapa, cada partícula tiene un peso asociado o una "puntuación" asociada, este peso que tiene cada partícula a su vez se conforma de pesos individuales por cada marca de mapa detectada, que nos dicen que tan cercana se encuentra de los datos reales del sensor de

distancia, es decir, que entre más cerca se encuentren las detecciones de marcas de mapa por cada partícula, eso le da una especie de puntuación que se acumula por marca, y el resultado total de la acumulación es dado a cada partícula como una especie de peso (véase el algoritmo (4.2.3.1.f)), entonces las partículas son sometidas al proceso de "resampling wheel" (véase el algoritmo (4.2.3.1.m)) para "reproducir" o "recrear" las partículas con un mayor peso acumulado asociado, esto ocasiona que la localización y el mapeo de las marcas de mapa sea mejor mientras estos procesos se den de manera adecuada, sin embargo, al reproducir solo partículas que concuerdan más con los datos del sensor LIDAR, se pierde el factor de diversidad de las partículas, que en algunos casos puede ser necesario para sobreponerse a errores globales de localización debido a algún error de detección, modelado, etc.

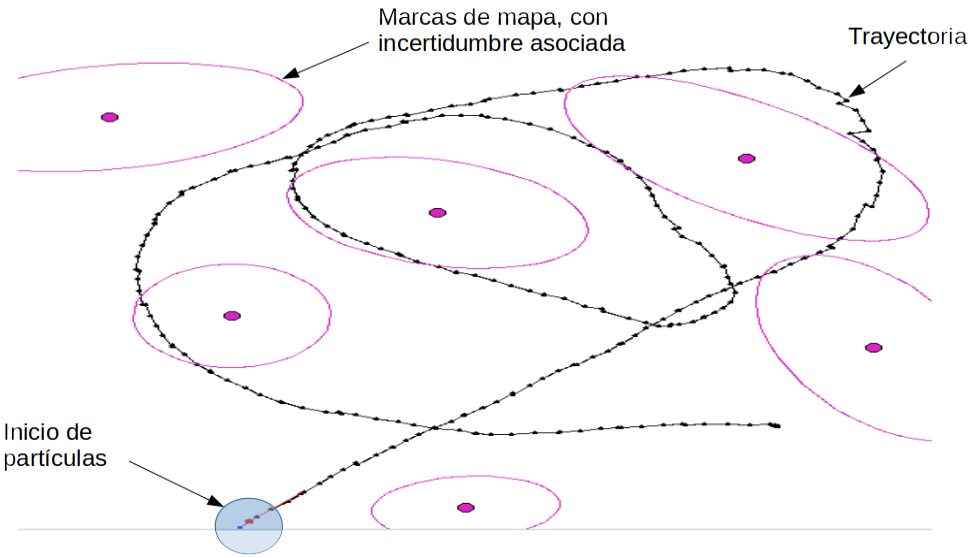


Figura 5.5.2.1. Etapa de corrección para enfoque Fast SLAM con 25 partículas.

En la figura (5.5.2.1) se representa el primer instante del escenario prueba, en la parte inferior izquierda se encuentra el inicio de las 25 partículas. De este cumulo de partículas se calculo la partícula que se encontraba más cercana al promedio del conjunto, y usando esta partícula se mapean en color violeta las marcas detectadas y las elipses que representan la incertidumbre de cada marca para el instante cero del recorrido y en color negro se establece la trayectoria "filtrada" de la partícula promedio a lo largo de la trayectoria, debe aclarase que esta partícula "promedio" en la mayoría de los casos no es una sola partícula, dado que puede ser que la partícula más cercana al promedio en el instante 25 no sea la partícula "ideal" para la trayectoria en el instante 50 o posteriores, entonces, esta partícula se observo que puede ir cambiando a través de las etapas de predicción, corrección y re-muestreo.

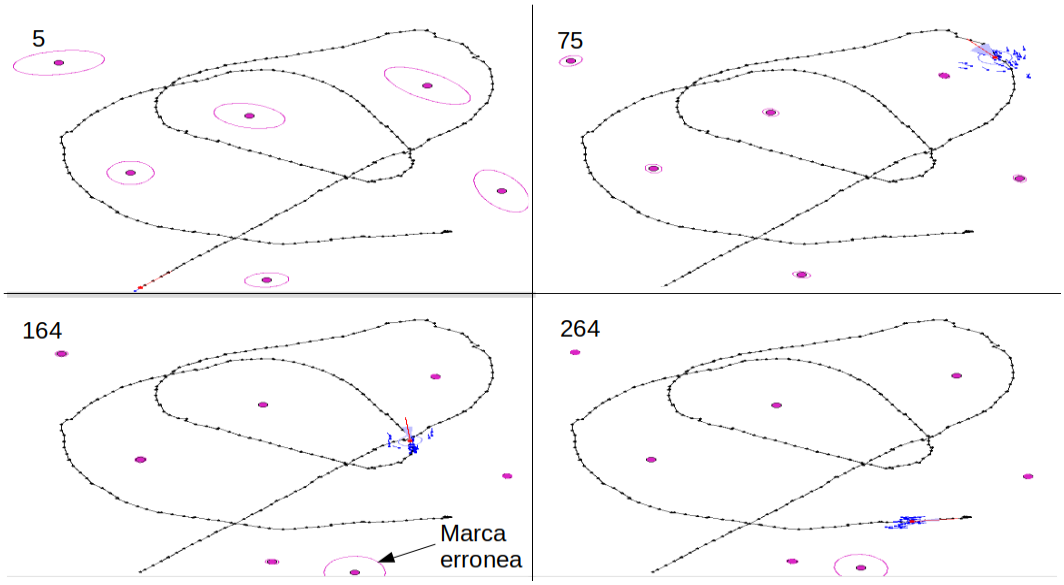


Figura 5.5.2.2. Etapa de corrección para enfoque Fast SLAM con 50 partículas en distintas etapas.

Ahora, en la figura (5.5.2.2) se muestran las imágenes obtenidas en la simulación incorporando la etapa de corrección del enfoque Fast SLAM. El escenario generado en la figura (5.5.2.2) utiliza un total de 50 partículas, con seis marcas de mapa y un total de 280 pasos de trayectoria. En la parte superior izquierda se muestra la distribución de las marcas de mapa para el instante 5 de la trayectoria, en comparación con la figura (5.5.2.1) que muestra el instante 0, la incertidumbre de las marcas de mapa se ve considerablemente reducida, esto se da debido a que los primeros 12 comandos de la trayectoria no establecen un movimiento, es decir, que ninguna partícula cambia su posición hipotética y solo se considera la etapa de corrección de "Fast SLAM" lo que permite actualizar los filtros de Kalman de cada marca y con ello reducir el error en su localización. En la parte superior derecha de la figura (5.5.2.2.) en el instante 75 se observa un aumento en el área de dispersión de las partículas debido a que existe un mayor error asociado a los movimientos en curva que los de línea recta, lo que genera que las partículas puedan aumentar su diversidad debido al aumento de rango en el espectro de movimientos posibles en la curva, después, en la parte inferior izquierda (en el instante 164) aparece una "marca errónea" de mapa, esto se debe a que las mediciones detectadas por la partícula más cercana al promedio determinan que la medición no corresponde a alguna de las marcas existentes, lo que crea una nueva marca, y en este caso, una marca errónea de mapa (dado que el mapa solo se genera con seis referencias de mapa totales) y finalmente, en la parte inferior derecha (en el instante 264) se observan instantes antes del final de la trayectoria simulada, el esparcimiento de las partículas se ve reducido, sin embargo, la marca errónea detectada en el momento 164 se conserva a partir del instante que apareció hasta el final de la simulación.

Simulación con conteo de marcas de mapa usando Fast SLAM

Al repetir el proceso de simulación utilizando las etapas de predicción y corrección de Fast SLAM, se observó que en ocasiones se presentan detecciones erróneas de marcas en el mapa, lo que resulta en algunos casos en problemas no solo de mapeo si no también de localización del robot (como puede observarse en la figura (5.5.2.2)), por ello es necesario implementar la fase de "depuración de marcas erróneas" [50] como se explico en capítulos anteriores.

La depuración de las marcas consiste en agregar contadores a cada marca de mapa detectada que aumentan o decrementan si son vistas o no, en relación a su ángulo de visión.

En la figura (5.5.2.3) se muestra el enfoque completo de Fast SLAM (con etapa de predicción y corrección) incorporando la depuración de marcas erróneas de mapa, en esta simulación se capturaron los mismos instantes del recorrido que en la simulación pasada, usando el mismo número de partículas y el mismo punto de inicio, para poder establecer diferencias en el comportamiento cuando se incorporan los contadores de las marcas de mapa.

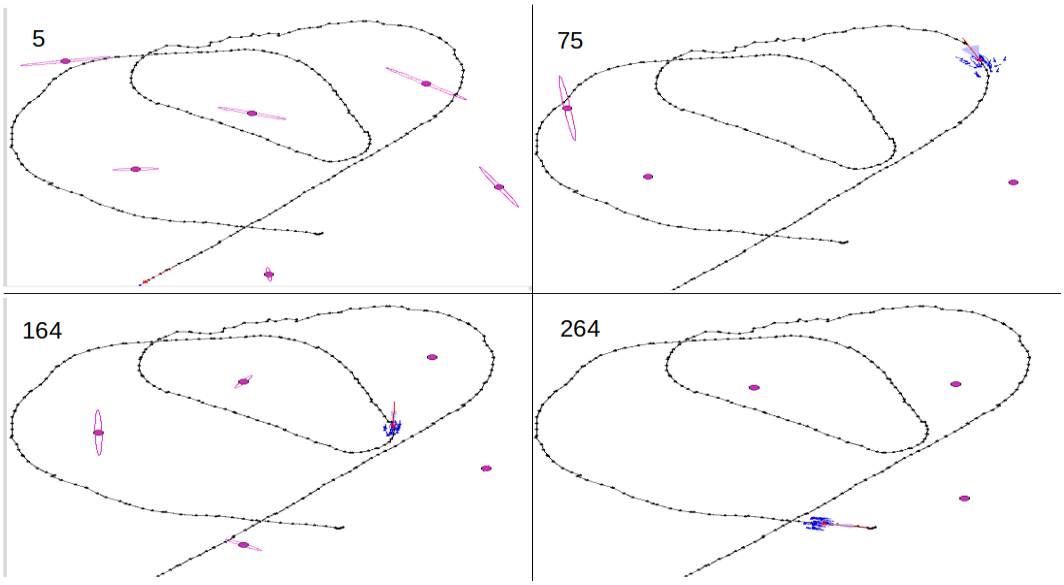


Figura 5.5.2.3. Etapa de corrección con depuración de marcas para enfoque Fast SLAM con 50 partículas en distintas etapas.

En la parte superior izquierda se muestra el instante 5 de la simulación (en la figura (5.5.2.3)), donde se observó que existe una reducción en la incertidumbre asociada a las marcas de mapa, en la parte superior derecha (en el instante 75) se puede observar que solo son detectadas tres marcas de mapa de las seis marcas supuestas en la simulación, esto se debe a que los contadores asociados a las marcas de mapa se continúan decrementando cuando no son vistos (aunque se trate de marcas "correctas" de mapa), y después de

un cierto número de pasos estas marcas de mapa desaparecen y solo las marcas más recientes permanecen para generar la localización y el mapeo. En la parte inferior de la figura (5.5.2.3), en el costado izquierdo se tiene el instante 164, algunas marcas de mapa se vuelven a detectar siendo un total de cinco marcas detectadas, sin embargo, la marca detectada a la izquierda (del instante 164) es una marca errónea de mapa, en la parte inferior derecha (en el instante 264) se observan los últimos instantes de la trayectoria simulada, donde existen solo tres marcas detectadas de mapa (del total de seis marcas puestas en el mapa) pero con la diferencia del instante 164, que las marcas detectadas son marcas correctas de mapa e incluso la incertidumbre asociada a las marcas es casi imperceptible, es decir, que las marcas se muestran casi como elementos "puntuales" en el mapa, y esto se relaciona con una buena exactitud en el mapeo, lo que se relaciona con una mejor localización de ASTEC en la trayectoria.

Pruebas en simulaciones de enfoque Fast SLAM

A continuación, se realizaron distintas simulaciones para estudiar el efecto del numero de partículas en el enfoque de Fast SLAM.

La primer prueba (mostrada en la figura (5.5.2.4)) consintió en incorporar la trayectoria de referencia, así como las marcas de mapa, para poder observar la relación entre la trayectoria generada por Fast SLAM y la trayectoria real del robot, y también comparar la capacidad de precisión de mapeo usando como referencia las marcas de mapa reales.

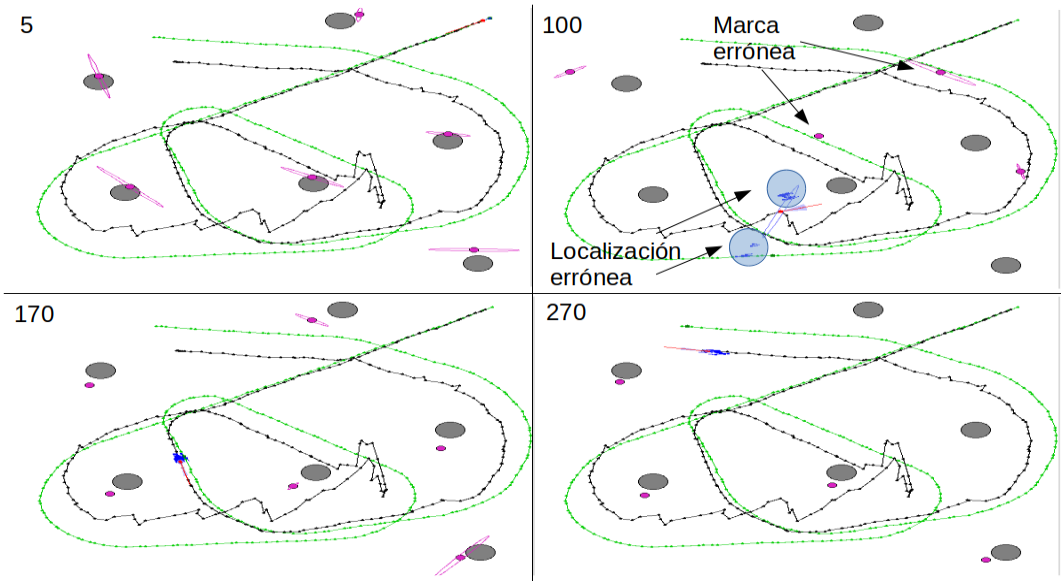


Figura 5.5.2.4. Fast SLAM con 50 partículas, trayectoria de referencia y marcas de mapa.

En la figura (5.5.2.4) se muestra la comparación de la trayectoria generada por el Fast SLAM de ASTEC (en color negro), la trayectoria "correcta" en color verde, las referencias de mapa en color gris y las marcas de mapa detectadas por Fast SLAM en color violeta.

Como se puede observar desde el instante 5 (de la figura (5.5.2.4) la trayectoria generada por el algoritmo de localización y mapeo es aproximada a la trayectoria real (en verde) usada para las simulaciones, para el instante 100 de la simulación, se observo como existe una zona de error bastante considerable en la trayectoria donde se generan dos cúmulos de partículas (círculos azules) y esto genera que exista un error considerable durante bastantes pasos de la trayectoria, además, se observa (también en el instante 100) que el mapeo de al menos tres de las marcas es equivocado, lo que explica los problemas de localización de ASTEC en esos pasos. Para los instantes 170 y 270 (de la figura (5.5.2.4)) se tuvo un comportamiento mas cercano a la trayectoria deseada, pero aun con algo de error, que se muestra al final de la trayectoria con un desfase en el punto final o meta del recorrido.

Después de la prueba con 50 partículas, se realizo una simulación con 100 partículas que se puede ver en la figura (5.5.2.5).

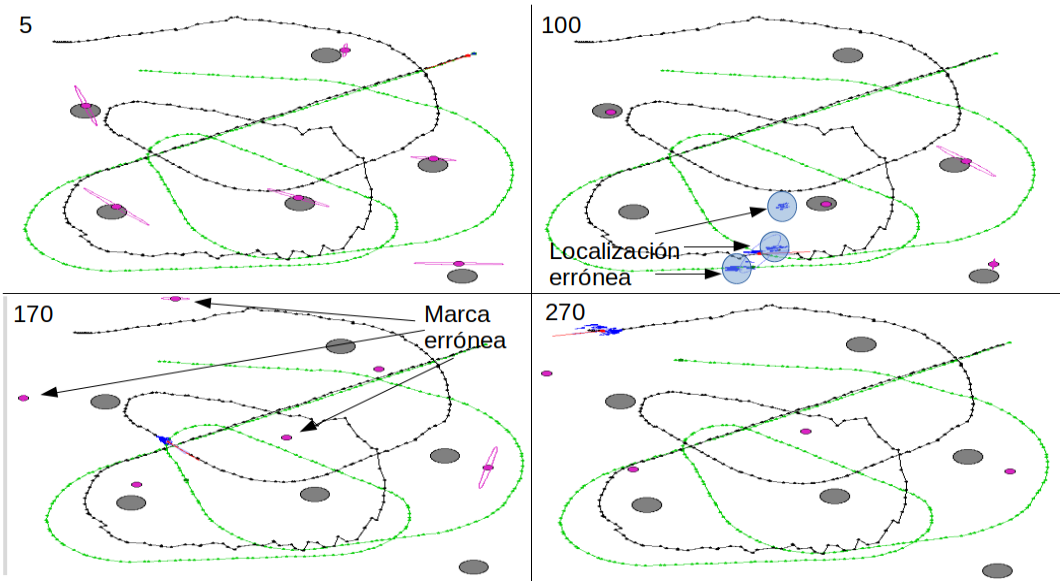


Figura 5.5.2.5. Fast SLAM con 100 partículas, trayectoria de referencia y marcas de mapa.

La simulación con 100 partículas (figura (5.5.2.5)) mostró un error mayor en la trayectoria global generada por Fast SLAM, donde en el instante 100 se crean aproximadamente tres conjuntos de partículas, que al final son responsables del desvió de la trayectoria en las vueltas centrales, además, en el instante 100 no se muestra de manera clara un desfase o un mapeo erróneo de las marcas, dado que las tres marcas que son detectadas coinciden con referencias “reales” del mapa; después, en el paso 170 se muestran las detecciones erróneas de al menos tres marcas de mapa, este error se puede asociar debido al esparcimiento generado en pasos anteriores de las partículas. En los instantes finales de la trayectoria (instante 270) resulto que las localizaciones finales de las partículas tenían menos dispersión pero tenían un gran desfase con respecto a la trayectoria deseada (en verde).

Ya que la simulación con 100 partículas no demostró una mejoría notable en la trayectoria final del robot se opto por realizar una simulación con 500 partículas (véase figura (5.5.2.6)), en esta prueba se aumento considerablemente el número de partículas, lo que en teoría debería ayudar a la localización "global" de ASTEC debería verse mejorada.

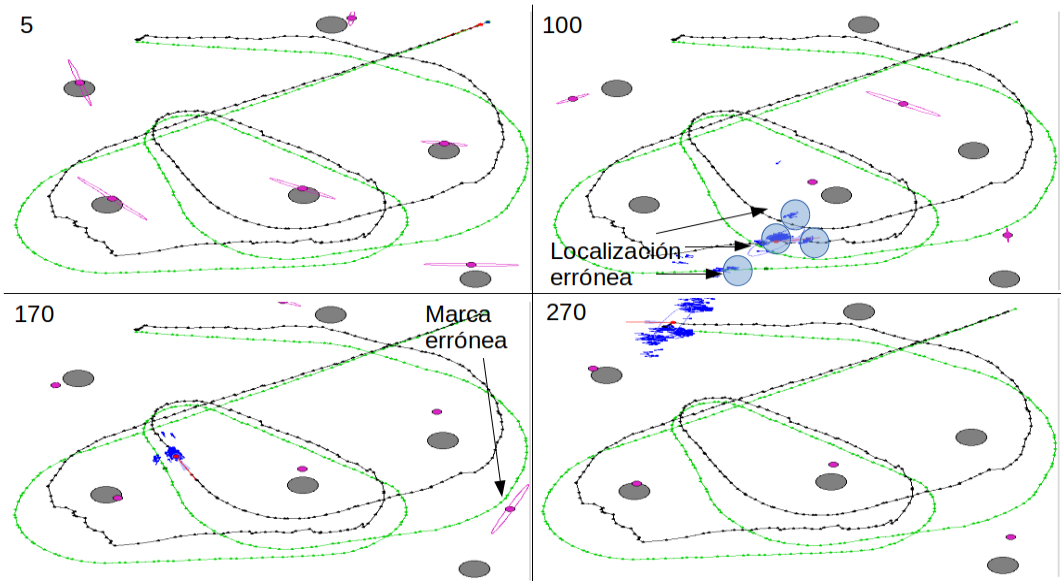


Figura 5.5.2.6. Fast SLAM con 500 partículas, trayectoria de referencia y marcas de mapa.

En la figura (5.5.2.6) se muestran los resultados observados para la simulación del escenario ejemplo con 500 partículas. El primer efecto que se muestra en las imágenes, es la mejoría en la ruta generada por Fast SLAM, que se encuentra de manera general, más cercana a la ruta "ideal" del robot, en el instante 100 se muestra que se generaron más grupos de partículas (aproximadamente cuatro), lo cual es un resultando bastante interesante y no previsto al inicio de la simulación. En el instante 170 (de la figura (5.5.2.6)) solo se observo una detección errónea de marca, en comparación con la simulación de 100 partículas (en el instante 170) que se detectaron al menos tres marcas erróneas de mapa, al final de la trayectoria (paso 270 en adelante) existen dos grupos principales de partículas, que en su promedio resultan en un punto "meta" bastante cercano al de la trayectoria ejemplo.

Capítulo 6

Conclusiones

Con base a las investigaciones realizadas, las simulaciones generadas y los resultados obtenidos, se obtuvieron algunas conclusiones expresadas a continuación.

En lo que respecta al objetivo general de la tesis, se puede decir que "ASTEC" (el sistema robótico inteligente), cuenta con los elementos necesarios para poder realizar las tareas de búsqueda y rescate (como producto de la localización) de víctimas en las simulaciones hechas, si bien, existe la posibilidad de realizar modificaciones para una aplicación de escenarios reales, los elementos de hardware y software contemplados en el diseño de ASTEC, cumplen con los elementos "mínimos" de un robot de rescate (véase el capitulo 1, "Características de los robots de rescate").

El hardware propuesto para el robot ASTEC, permitió incorporar un diseño prototipo que cuenta con los requerimientos investigados en el marco teórico, además, esto ayudó a instituir parámetros y características consideras en el desarrollo de los algoritmos para realizar las tareas de búsqueda y rescate. La elección del "sistema de locomoción" (unidad de locomoción) de rodamiento por oruga permitió incorporar un modelo de movimiento similar al modelo de locomoción de dos ruedas, que acorde a las simulaciones, permite establecer una aproximación de la transición de estados del robot ASTEC. Para la "unidad de control y procesamiento" la elección de una microcomputadora o SBC (single board computer) resulto fundamental para poder reducir el tiempo de desarrollo de los programas dada la portabilidad del código en distintas plataformas (SBCs, Laptops, PCs, etc.), el usar una microcomputadora como la Raspberry Pi 3 permite tener una cantidad adecuada de procesamiento y memoria *in situ*, a diferencia del uso de plataformas que solo incorporan microcontroladores. También se concluye que el uso del interprete (o lenguaje de programación) Python 3 facilitó la implementación de varios algoritmos haciendo uso de varias bibliotecas de programas disponibles para este lenguaje.

Para la "unidad de sensores", la incorporación del sensor láser omnidireccional (LIDAR) como elemento de localización y mapeo en el enfoque de Fast SLAM es un

método factible para incorporar en una aplicación física, donde el procesamiento de los datos en la "unidad de control" es relativamente bajo en costo computacional, y poder mantener una buena precisión en el mapeo del entorno. También, el uso de una cámara térmica y una cámara WEB, como elementos auxiliares, ayudo a crear protocolos para poder determinar la posible existencia de una víctima en el lugar (y en conjunto con el sensor LIDAR cumplen con el primer objetivo especifico de la tesis) considerando que el adquirir una imagen (cámara WEB), un espectro de calor (cámara térmica) y la información de distancias a los objetos del mapa (sensor LIDAR) son distintos métodos para percibir el entorno, debido a que cada uno aporta una parte de información que puede ser distinta a la que perciben cada uno de ellos en forma independiente.

Ya que los escenarios propuestos en la tesis debían abordarse a través de simulaciones, la generación de los mapas pseudo-aleatorios ayudo en distintas etapas, en especifico, en el desarrollo del "algoritmo QPA*". El "algoritmo QPA*" cumple con el segundo objetivo especifico de la tesis, que consiste en un algoritmo capaz de resolver las tareas de exploración y trazado de ruta; los distintos enfoques que utiliza el algoritmo QPA* resuelve distintos problemas inherentes en los sistemas deterministas de trazado de ruta, dado que el uso de los campos de potencial binario artificial se encargan de evitar las colisiones de la ruta con los objetos de mapa (dado que el robot ASTEC es considerado como punto masa por el algoritmo de trazado de ruta). El aspecto más relevante del enfoque de QPA* es la capacidad de establecer un protocolo de exploración "real", esto permite que el robot pueda realizar un recorrido (o exploración) del mapa sin la necesidad de detectar una víctima a priori, y con ello aumentar las zonas de búsqueda de víctimas.

Las simulaciones y comparativas de los distintos enfoques del algoritmo A* explorados (algoritmo A* clásico, algoritmo A* modificado con ordenamiento de pila, y algoritmos QPA* con costo de conexión $x1$ y costo de conexión $x100$ como función de costo hacia nodos visitados) cumplen con el tercer objetivo especifico de la tesis. Las estrategias del protocolo de exploración y los campos de potencial binario (del algoritmo QPA*), se observó que el enfoque "clásico" (del algoritmo A*) es el menos eficiente en cuestiones de tiempo de ejecución y la relación de nodos explorados contra tiempo de ejecución, pero mantiene un comportamiento similar en área de cobertura con el enfoque "modificado" de A* usando ordenamiento de pila (A* modificado), que mejora considerablemente (aproximadamente 10 veces más rápido) el tiempo de ejecución y la relación de nodos explorados contra tiempo de ejecución. Finalmente, el enfoque de A* modificado (con ordenamiento de pila) y la incorporación de un costo extra hacia nodos visitados (usando el costo de conexión $x1$) en iteraciones pasadas, fue el enfoque usado para el diseño de algoritmo QPA* final, tiene una mejoría en el tiempo de ejecución y la relación de nodos visitados contra tiempo de ejecución con respecto al enfoque clásico (aproximadamente 9 veces más rápido) pero

si solo se contempla el "trazado de ruta" es preferible utilizar el algoritmo A* modificado, sin embargo, el enfoque del algoritmo QPA* (ya sea con costo de conexión $x1$ o con costo de conexión $x100$) demostró tener una mayor área de cobertura comparado con el A* modificado y el A* clásico, esto al final se puede interpretar con una mayor área de exploración posible del robot, es decir, que el algoritmo QPA* es más eficiente para las cuestiones de exploración. Además, como se mostró en las simulaciones de los algoritmos QPA*, la función de costo extra para nodos visitados en QPA* puede modificarse para hacer que las zonas de exploración (nuevas) rodeen, de forma más radical (evita las zonas visitadas de manera constante), las zonas previamente visitadas, aumentando el área de cobertura (cantidad de nodos visitados) para la posible ruta, pero, para el caso cuando la función de costo es basada en el costo de conexión entre nodos $x100$ existe un decremento en la eficiencia de nodos por segundo (nodos visitados por segundo). Entonces, se puede concluir que el algoritmo QPA* permite tener una mayor versatilidad de ajustes, y con ello, aumentar las funciones que éste puede realizar.

El enfoque de "Fast SLAM" (SLAM - Simultaneous Localization and Mapping) usado para ASTEC, realiza las tareas planteadas en el cuarto objetivo específico de la tesis. El enfoque de Fast SLAM fue capaz de resolver los problemas de localización (del robot) y mapeo (marcas de mapa) de manera simultanea en las distintas simulaciones generadas. El uso de éste enfoque como un filtro, para resolver la localización y el mapeo, resulta bastante práctico, ya que en cada transición de estado (movimiento) se toma como un elemento independiente al igual que cada etapa de corrección (medición del sensor láser). A partir de los resultados de las simulaciones para Fast SLAM, se concluye que la calidad de la trayectoria generada (y de los procesos, en general, de localización y mapeo) aumentan con el número de partículas, es decir, que a mayor número de partículas existe una mejor certidumbre de la localización de ASTEC, también se observo que es importarte contar con un mínimo de dos marcas o referencias de mapa para poder establecer una buena etapa de corrección, ya que una sola marca no es suficiente, es por ello que la relación entre un mayor número de marcas de mapa y un mayor número de partículas mejora el proceso de SLAM, pero aumenta el tiempo de procesamiento y los recursos necesarios para poder realizar estas tareas desde el robot.

Finalmente, existen algunos aspectos que se reservan como trabajos a futuro, que se establecen para mejorar el diseño prototipo de ASTEC. Como parte esencial del trabajo a futuro se plantea construir el sistema físico utilizando el diseño propuesto en la tesis, observar el comportamiento del sistema y determinar si los componentes (de hardware y software) son suficientes para un despliegue "real", también se pretenden realizar pruebas en distintos escenarios y distintas tareas, para poder estudiar el comportamiento del robot y si es necesario modificar algunos enfoques o programas utilizados en las simulaciones de la tesis.

Bibliografía

[1] Roberto Carlo Ponticelli Lima, "Sistema de exploración de terrenos con robots móviles: aplicación en tareas de detección y localización de minas antipersonas", Tesis doctoral, Universidad Complutense de Madrid, Facultad de Ciencias Físicas, Departamento de Arquitectura de Computadoras y Automática, ISBN: 978-84-694-2460-5, Madrid-España, 2011.

[2] H. Adachi, N. Koyachi, T. Arai, A. Shimizu, Y. Nogami, "Mechanism and Control of a Leg-Wheel Hybrid Mobile Robot", Proceedings of the 1999 IEEE/RSJ Intenational Conference on Intelligent Robots and Systems, Kyongju - South Korea, October 1999.

[3] Iván Luciano Almeida Hernández, Jimmy Andrés Ochoa Urgiles, "Diseño y construcción de un robot explorador de terreno", Tesis para Ingeniería, Universidad Politécnica Salesiana sede Guayaquil, Facultad de Ingenierías, Ingeniería en Electrónica con mención en Sistemas Industriales, Guayaquil-Ecuador, Abril 2013.

[4] Sascha A. Stoeter, Nikolaos Papanikolopoulos, "Autonomous Stair-Climbing with Miniature Jumping Robots", IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, Vol. 35, No. 2, April 2005.

[5] Sheila Russo, Kanako Harada, Tommaso Ranzani, Luigi Manfredi, Casare Stefanini, Arianna Menciassi, Paolo Dario, "Design of a Robotic Module for Autonomous Exploration and Multimode Locomotion", IEEE/ASME Transactions on Mechatronics, Vol. 18, No. 6, December 2013.

[6] Jesús Salido Tercero, "Cibernética Aplicada – Robots educativos", Primera Edición, Editorial Alfaomega, México, diciembre 2009.

[7] Wenzeng Guo, Yu mu, Xueshan Gao, "Step-climbing Ability Research of a Small Scout Wheel-track Robot Plataform", Proceedings of the IEEE Conference of Robotics and Biomimetrics, Zhuhai, China, December 6-9, 2015.

[8] Lance Molyneaux, Dale A. Carnegie, Chris Chitty, "HADES: An Underground Mine Disaster Scouting Robot", 2015 IEEE International Symposium of Safety, Security, and Rescue Robotics (SSRR), West Lafayette, IN-USA, 18-20 October 2015.

[9] Robin R. Murphy, "Disaster Robotics", The MIT Press, ISBN: 978-0-262-02735-9, Cambridge, Massachusetts, London-England, 2014.

[10] Federación Internacional de Sociedades de la Cruz Roja y de la Media Luna Roja, "Informe Mundial sobre desastres 2015", 2015.

[11] Ahmet Denker, Mehmet Can Iseri, "Design and Implementation of a Semi-Autonomous Mobile Search and Rescue Robot: SALVOR", Dept. of Electrical-Electronics Engineering, Istanbul, Turkey, 2017.

[12] Katsuaki Tanaka, Di Zhang, Sho Inoue, Ritaro Kasai, Hiroya Yokoyama, Koki Shindo, Ko Matsuhiro, Shigeaki Marumoto, Hiroyuki Ishii, Atsuo Takanishi, "A design of a small mobile robot with a hybrid locomotion mechanism of wheels and multi-rotors", Department of Science and Engineering and Department of Modern Mechanical Engineering, Humanoid Robotics Institute (HRI), Waseda University, Tokyo-Japan, August, 2017.

[13] Robin R. Murphy, Jeffery Kravitz, Samuel L. Stover, Rahmat Shoureshi, "Mobile Robots in Mine Rescue and Recovery", Novel Application of Robotics, IEEE Robotics and Automation Magazine, June, 2009.

[14] Seyed Hossein Jenabi, Alireza Mohammad Shahri, Mohammad Reza Beyad, "Advanced Mechatronics Course based on Wheeled Mobile Robots and Real-Time Embedded Control Experimental Setups", 4th International Conference on Control, Instrumentation, and Automation (ICCIA), Qazvin Islamic Azad University, Qazvin-Iran, 27-28 January 2016.

[15] Nikolai P. Kobyzev, Sergei A. Golubev, Yuri G. Artamonov, Artur I. Karimov, Sergei V. Goryainov, "Embedded Control System for Tracked Robot", Department of Computer Aided Design, Saint Petersburg Electrotechnical University "LETI", Saint Petersburg-Russia, 2017.

[16] Han Jun, Chang Rui-li, "Development and Research on Embedded Control System for Intelligen Mobile Robot", Mechanical Engineering School, University of Science and Technology Inner Mongolia, Baotou-China, 2010.

[17] A. Denker, A. U. Dilek, B. Sarioglu, J. Savas, Y.D. Gökdel, "RoboSantral: An autonomous mobile guide robot", Department of Electrical and Electronics Engineering, Istanbul Bilgil University, Istanbul-Turkey, 2015.

[18] Ryan Krauss, "Combining Raspberry Pi and Arduino to Form a Low-Cost, Real-Time Autonomous Vehicle Platform", American Control Conference (ACC), Boston Marriott Copley Place, Boston-Massachusetts, USA, July 6-8, 2016.

[19] Wei-Fu Kao, Chun-Fei Hsu, Tsu-Tian Lee, "Intelligent Control for a Dynamically Stable Two-Wheel Mobile Manipulator", Department of Electrical Engineering, Tamkang University-Taipei City-Taiwan, IFSA-SCIS-2017, Otsu, Shiga-Japan, June 27-30, 2017.

[20] Stanislav Vechet, Jiri Krejsa, Michal Ruzicka, Petr Masek, "Building a Scaled Model of Autonomous Convoy Powered by Arm mbed Microcontrollers", Faculty of Mechanical Engineering, Brno. University of Technology, Brno-Czech Republic.

[21] T. Estier, Y. Crausaz, B. Merminod, R. Lauria, M. Piguet, and R. Siegwart, "An innovative space rover with extended climbing abilities", in Proc. Space and Robotics, Alburquerque, New México, Feb-Mar, 2000.

[22] H. Kwon, H. Shim, D. Kim, S. Park, and J. Lee, "A Development of a transformable caterpillar equipped mobile robot", in IEEE Int. Conf. On Control, Automation and Systems, 2007, pp.1062-1065.

[23] Weidong Wang, Wei Dong, Yanyu Su, Dongmei Wu, Zhijiang Du, "Development of Search-and-rescue Robots for Underground Coal Mine Applications", Published in: Journal of Field Robotics, Volume 31 Issue 3, Chichester-UK, May 2014, Pages 386-407.

[24] Keiji Nagatani, Seiga Kiribayashi, Yoshito Okada, Satoshi Tadokoro, Takeshi Nishimura, Tomoaki Yoshida, Eiji Koyanagi, Yasushi Hada, "Redesign of rescue mobile robot Quince –Toward emergency response to the nuclear accident at Fukushima Daiichi Nuclear Power Station on March 2011", Proceedings of the 2011 IEEE International Symposium on Safety, Security and Rescue Robotics, Kyoto – Japan, November 2011.

[25] Luca Cavanini, Gionata Cimini, Francesco Ferracuti, Alessandro Freddi, Gianluca Ippoliti, Andrea Monteriu, and Federica Verdini, "A QR-code Localization System for Mobile Robots: Application to Smart Wheelchairs", 2017 European Conference on Mobile Robots (ECMR), Paris - France, 6-8 Sept. 2017.

[26] Alexander Ceron Correa, "Sistemas Roboticos Teleoperados", Ciencia e Ingenieria Neogranadina, ISSN: 0124-8170, Universidad Militar Nueva Granada, Bogotá - Colombia, Pag. 62 - 69, 2005.

[27] Kun Zuang, Chunsheng Yang, Yubin Yang, Jie Liu and Nan Jiang, "Intelligent Stereo Camera Mobile Platform for Indoor Service Robot Research", Proceedings of the 2016 IEEE 20th International Conference on Computer Supported Cooperative Work in Design, Nanchang - China, 4-6 May 2016.

[28] Shuxiang Guo, Xin Li, Jian Guo, "Study on a Multi-Robot Cooperative Wireless Communication Control System for the Spherical Amphibious Robot", Proceedings of 2016 IEEE International Conference on Mechatronics and Automation, Harbin - China, August 7 - 10.

[29] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza, "Introduction to Autonomous Mobile Robots", MIT Press, Second Edition, Cambridge-Massachusetts, 2011.

[30] J. del Prado, Sunghyun Choi, "Experimental study on coexistence of 802.11b with alien devices",IEEE 54th Vehicular Technology Conference. VTC Fall 2001. Proceedings, New York - USA, Volume:2,Pages:977-981, 2001

[31] Scott Keller, "Determining the Best Wireless Frequency for Your Remote Monitoring and Control System", SignalFire Wireless Telemetry, www.signal-fire.com.

[32] Alberto Marroquin, Adalberto Gomez, Alejandro Paz, "Design and implementation of Explorer Mobile Robot controlled remotely using IoT Technology", Published in 2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON), Pucon - Chile, 18 - 20 Oct 2017.

[33] U Bharath Sai, K Sivanagamani, B Satish, M Ranga Rao, "Voice controlled Humanoid Robot with artificial vision", Published in 2017 International Conference on Trends in Electronics and Informatics (ICEI), Tirunelveli, India, 11 - 12 May 2017.

[34] Russell Stuart, Norvig Peter, "Artificial Intelligence: A Modern Approach", Prentice Hall - Third Edition, USA - New Jersey, 2010.

[35] M. Tim Jones, "Artificial Intelligence: A Systems Approach", Infinity Science Press, Massachusetts - USA, 2008.

[36] Cyrill Stachniss, Wolfram Burgard, "Exploring Unknown Environments with Mobile Robots using Coverage Maps", University of Freiburg, Department of Computer Science, Published in: IJCAI'03 Proceedings of the 18th international joint conference on Artificial intelligence, Acapulco-México, August 09-15, 2003.

[37] Alexander Ivanov, Mark Campbell, "Uncertainty Constrained Robotic Exploration: An Integrated Exploration Planner", Published in: IEEE Transactions on Control Systems Technology, Volume: 27, Issue: 1, Jan. 2019.

[38] Sebastian Thrun, Wolfram Burgard, Dieter Fox, "Probabilisti Robotics", The MIT Press, Massachusetts - USA, 2006.

[39] Tim Roughgarden, Alexa Sharp and Tom Wexler, "Guide to Greedy Algorithms", Handout 12 - CS161, July 29 - 2013. https://web.stanford.edu/class/archive/cs/cs161/cs161.1138/handouts/120\%20Guide\%20to\%20Greedy\%20Algorithms.pdf

[40] Alpaslan Yufka and Osman Parlaktuna, "Performance Comparison of Bug Algorithms for Mobile Robots", 5th International Advanced Technologies Symposium (IATS'09), Karabuk - Turkey, May 13 - 15, 2009.

[41] Lumelsky, V.J., Stepanov, "Dynamic path planning for a mobile automaton with limited information on the environment", IEEE Trans. Automat. Contr. 31, 1058 – 1063, 1986.

[42] Kamon, I., Rivlin, E., "Sensory-based motion planning with global proofs", IEEE Trans. Robot. Autom.13, 814 – 822, 1997.

[43] Evgeni Magid and Ehud Rivlin, "CAUTIOUSBUG: A Competitive Algorithm for Sensory-Based Robot Navigation", Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, Sendal - Japan, September 28 – October 2, 2004.

[44] Mr. S. Julius Fusic, Mr. P. Ramkumar, Dr. K. Hariharan, "Path planning of robot using modified dijkstra Algorithm", 2018 National Power Engineering Conference (NPEC), Madurai - India, 01 October 2018.

[45] Frantisek Duchon, Andrej Babinec, Martin Kajan, Peter Beno, Martin Florek, Tomás Fico, Ladislav Jurisica, "Path planning with modified A star algorithm for a mobile robot", Procedia Engineering 96, Elsevier, 2014.

[46] Duchon, F., Hunady, D., Dekan, M., Babinec, A., "Optimal navigation for a mobile robot in known environment", Applied Mechanicas and Materials 282, pp 33 - 38, 2013.

[47] Felipe Haro and Miguel Torres, "A Comparison of Path Planning Algorithms for Omni-Directional Robots in Dynamic Enviroments", LARS, 2006.

[48] Yadollah Rasekhipour, Amir Khajepour, Shih-Ken Chen, and Bakhtiar Litkouhi, "A Potencial Fiel-Based Model Predictive Path-Planning Controller for Autonomous Road Vehicles", IEEE Transactions on Intelligent Transportation Systems, Vol. 18, No. 5, May - 2017.

[49] Howie Choset, Kevin Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia Kavraki, and Sebastian Thrun, "Principles of Robot Motion: Theory, Algorithms, and Implementation", A Bradford Book, The MIT Press, Massachusetts - USA, 2005.

[50] Claus Brenner, "SLAM Lectures", 2013, https://www.youtube.com/watch?v=B2qzYCeT9oQ&list=PLpUPoM7Rgzi_7YWn14Va2FODh7LzADBSm

[51] Abraham Bachrach, Anton de Winter, Ruijie He, Garrett Hemann, Samuel Prentice, Nicholas Roy, "RANGE - Robust Autonomous Navigation in GPS-denied Environments", 2010 IEEE International Conference on Robotics and Automation, Anchorage Convention District, Alaska - USA, May 3 - 8, 2010.

[52] Greg Welch, Gary Bishop, "An Introduction to the Kalman Filter", University of North Carolina at Chapel Hill, Department of Computer Science, 2001.

[53] Xiaoqin Wang, Xiaolong Wang, "THE COMPARISON OF PARTICLE FILTER AND EXTENDED KALMAN FILTER IN PREDICTING BUILDING ENVELOPE HEAT TRANSFER COEFFICIENT", 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, Hangzhou - China, 30 Oct - 1 Nov, 2012

[54] A.M. Mathai, "Jacobians of Matrix Transformations and Functions of Matrix Argument", Ed. World Scientific, Singapore, 1997

[55] Dimitri Vvendensky, "Partial Differential Equations: with Mathematica", Ed. Addison-Wesley, Great Britain - Oxford, 1992.

[56] Maral Partovibakhsh and Guangjun Liu, "An Adaptive Unscented Kalman Filtering Approach for Online Estimation of Model Parameters and State-of-Charge of Lithium-Ion Batteries for Autonomous Mobile Robots", Published in: IEEE Transactions on Control Systems Technology, VOL. 23, NO. 1, JANUARY 2015.

[57] Melissa Morris and Sabri Tosunoglu, "Survey of Rechargeable Batteries for Robotic Applications", 2012 Florida Conference on Recent Advances in Robotics, Boca Raton - Florida, May 10-11, 2012

[58] Alliedelec, "Raspberry Pi 3 Model B", https://www.alliedelec.com/m/d/4252b1ecd92888dbb9d8a39b536e7bf2.pdf.

[59] Anker, "PowerCore 13000", https://www.anker.com/products/A1215011.

[60] Chengdu Ebyte Electronic Technology Co. Ltd., "E01-ML01DP5 Datasheet v1.0", https://fccid.io/2ALPH-E01/User-Manual/User-manual-3873277.iframe.

[61] Dean Miller, "Adafruit AMG8833 8x8 Thermal Camera Sensor", https://cdn-learn.adafruit.com/downloads/pdf/adafruit-amg8833-8x8-thermal-camera-sensor.pdf?timestamp=1556108404.

[62] SLAMTEC, "RPLIDAR A1 Low Cost 360 Degree Laser Range Scanner: Introduction and Datasheet", http://bucket.download.slamtec.com/e9e096e9d9f30205d665260abe2cfb0c2dd62efa/LD108_SLAMTEC_rplidar_datasheet_A1M8_v1.0_en.pdf.

[63] "heapq - Heap queue algorithm", Python Software Foundation[US], https://docs.python.org/3/library/heapq.html, fecha de consulta: 25 / Abril / 2019.

[64] Nick Parlante, "Binary Trees", Stanford CS Education Library, 2001, http://cslibrary.stanford.edu/110/BinaryTrees.pdf.

[65] Soheil Gharatappeh, Mohammad Ghorbanian, Mehdi Keshmiri, and Hamid D. Taghirad, "Modified Fast-SLAM For 2D Mapping And 3D Localization", Proceedings of the 3rd RSI International Conference on Robotics and Mechatronics, Tehran - Iran, October 7 - 9, 2015.

[66] Panasonic, "Infrared Array Sensor: Grid-Eye", datasheet 2015, https://eu.industrial.panasonic.com/sites/default/pidseu/files/downloads/files/grid-eye_flyer_english_web.pdf

[67] Amit Patel, "Implementation notes: From Amit's Thoughts on Pathfinding", Stanfort, http://theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes.html, fecha de consulta: 28 / Mayo / 2019.

[68] Python Software Foundation [US], https://www.python.org/

[69] Numpy, https://numpy.org/

[70] Matplotlib, https://matplotlib.org/

[71] Python Software Foundation, "Math Library" https://docs.python.org/3/library/math.html

Apéndice A

Material de apoyo y complementario

A.1. Teorema de Bayes

El Teorema de Bayes o Regla de Bayes, genera la relación de la probabilidad condicional $p(a|b)$ con su "inverso" $p(b|a)$, con la condición de que la probabilidad $p(b)$ se mayor que 0 ($p(b) > 0$)).

Para el caso discreto:

(a.1.1)
$$p(a|b) = \frac{p(b|a)p(a)}{p(b)} = \frac{p(b|a)p(a)}{\sum_{a'} p(b|a')p(a')}$$

Para el caso continuo:

(a.1.2)
$$p(a|b) = \frac{p(b|a)p(a)}{p(b)} = \frac{p(b|a)p(a)}{\int p(b|a')p(a')da'}$$

A.2. Matriz Jacobiana

Una matriz Jacobiana se puede entender como una matriz de transformación, en la cual se pretende mapear elementos de funciones (como elementos) de una matriz a otra [54], este mapa o transformación para el caso de modelos cinemáticos se puede establecer con una matriz Jacobiana.

La matriz Jacobiana J permite $f : \mathbb{R}^n \to \mathbb{R}^m$, donde la "entrada" de la matriz es un vector $x \in \mathbb{R}^n$ y la aplicación de la matriz Jacobiana J genera $f(x) \in \mathbb{R}^m$.

Para el caso de la transformación de coordenadas rectangulares para $3D$ denotadas por (x, y, z) a un nuevo sistema de coordenadas (j_1, j_2, j_3) se puede aplicar una matriz Jacobiana [55] como:

(a.2.1)

$$J = \begin{vmatrix} x_{j_1} & y_{j_1} & z_{j_1} \\ x_{j_2} & y_{j_2} & z_{j_2} \\ x_{j_3} & y_{j_3} & z_{j_3} \end{vmatrix}$$

Donde:

- x_{j_k} es igual a $\frac{\partial x}{\partial j_k}$.
- y_{j_k} es igual a $\frac{\partial y}{\partial j_k}$.
- z_{j_k} es igual a $\frac{\partial z}{\partial j_k}$.

En general, la matriz Jacobiana J contiene los elementos de las derivadas parciales de primer orden $\frac{\partial}{\partial j_k}$, donde j_k es nuestro espacio "objetivo" de mapeo.

A.3. Python 3 - Bibliotecas implementadas para diseño de ASTEC

La pagina de Python `https://www.python.org/about/` [68] contiene una breve definición general de Python "Python is a programming language that lets you work more quickly and integrate your systems more effectively". Python es un lenguaje ampliamente usado en la actualidad, este lenguaje cuenta con distintas características siendo la mas importante el enfoque "Open Source", ya que Python es un lenguaje de "código abierto" a permitido crear una gran comunidad alrededor de el, siendo la comunidad de desarrollo en aplicaciones de inteligencia artificial una de las mas grandes, además, Python se establece como un lenguaje de programación bastante amigable e intuitivo. Una de las características mas importantes de este lenguaje, es su flexibilidad de programación, ya que este Python puede ser programado en cuatro distintos enfoques (o paradigmas) de programación, estos son: programación imperativa, programación funcional, programación orientada a objetos y programación procedural, siendo esta flexibilidad una de las características principales para que el autor de la tesis prefiriese el uso de este lenguaje para el desarrollo de los algoritmos implementados en ASTEC.

Algunas de las bibliotecas de funciones para Python mas usadas en la implementación de los algoritmos de la tesis son:

- "Numpy" : Es un paquete de funciones para computo científico en Python, que puede manejar vectores (arreglos) objeto $N - dimensionales$, funciones de álgebra lineal, transformadas de Fourier, y generación de números aleatorios. [69] Además, "Numpy" cuenta con muchos algoritmos de optimización de memoria y distribución de tareas, lo que hace que sea una de las bibliotecas mas usadas de Python.

- "Matplotlib" : Es un biblioteca de funciones para gráficos 2D, que permite una gran calidad de las figurares generadas y tiene guardado de archivos en imagen para elementos gráficos generados. "Matplotlib" para Python, permite generar gráficas, histogramas, gráficas de espectro, gráficos de barras, etc. [70] Esta biblioteca fue usada para generar muchos de los resultados de las simulaciones de los algoritmos del presente trabajo, siendo una herramienta gráfica fundamental para poder entender y expresar algunos resultados obtenidos.

- "Math Library": La biblioteca "Math" de Python, es un paquete de funciones contenido en la versión de instalación estándar de Python. Esta modulo de funciones contiene operaciones y representaciones matemáticas básicas, trigonométricas, operaciones sobre variables u objetos (trucado, redondeado, etc.), operaciones con punto flotante, etcétera. [71] Esta biblioteca fue ampliamente usada para realizar operaciones matemáticas básicas en muchos algoritmos de la tesis, como la obtención de la raíz cuadrada, potencias, redondeado y trucado de números, etc.

A.4. Árbol Binario

Un árbol binario se puede definir de forma "recursiva" como un elemento (o función) que puede ser vació (cuando su "pointer" es "null"), o formado por un nodo "simple" que contiene apuntadores (pointers) que llaman o apuntan de forma recursiva a otros nodos (puntos) que son a su vez árboles binarios [64]. Sin embargo, los árboles binarios no solo pueden ser estructuras de funciones recursivas, sino que también pueden formarse a través de nodos que contengan apuntadores izquierdos y derechos (por cada nodo del árbol) y algún tipo de dato, en todos los arboles binarios existe un nodo "raíz" que es el primer nodo o el nodo maestro del árbol, y cada nodo subalterno a este genera o no, "sub-arboles" de la misma forma que el nodo maestro, estos nodos usualmente se les conoce como nodos "padre" (o madre) y los sub-nodos, se les conoce como nodos "hijo".

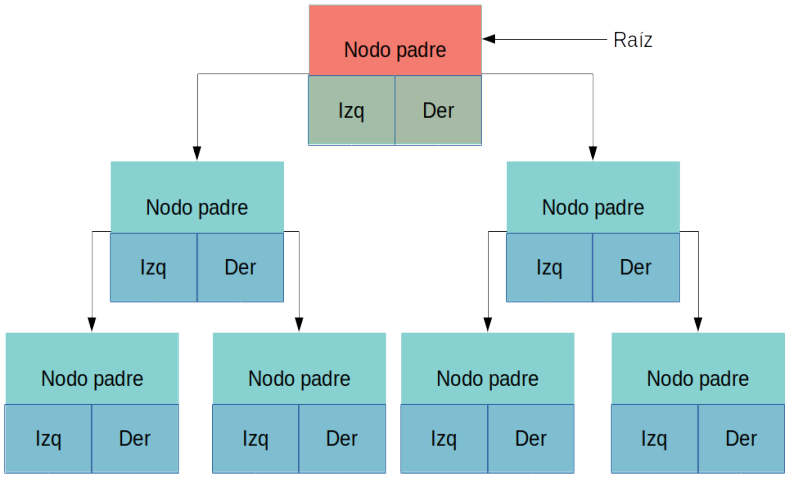


Diagrama A.3.1 Árbol binario.

En el caso del "Heap Queue Algorithm" implementado en la búsqueda del nodo mínimo del algoritmo A* modificado de QPA*, se desprende un concepto llamado "árbol de búsqueda binaria" (binary search tree), que es una estrategia de "búsqueda ordenada" donde los nodos en el árbol binario, se encuentran acomodados en un cierto orden [64], donde para el caso del algoritmo "heapq()" de Python 3, acomoda los nodos del árbol de tal manera que los padres siempre sean menores o iguales a los hijos (basados en algún parámetro establecido).