UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
MECÁNCIA – MECATRÓNICA

DESARROLLO DE UN BANCO DE PRUEBAS PARA EL ESTUDIO DE ROBOTS MÓVILES MEDIANTE EL PROTOCOLO IEEE 802.11 EN TIEMPO REAL Y CON RETROALIMENTACIÓN VISUAL

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN INGENIERÍA

PRESENTA:
PEDRO JAVIER GÁLVEZ VALADEZ

TUTOR PRINCIPAL
DR VÍCTOR JAVIER GONZÁLEZ VILLELA,
FACULTAD DE INGENIERÍA

MÉXICO, D. F. ENERO 2015

JURADO ASIGNADO:

Presidente:	Dr. Ramírez Reivich Alejandro C.
Secretario:	Dr. Cuenca Jiménez Francisco
1 er. Vocal:	Dr. González Villela Víctor Javier
2 do. Vocal:	Dra. Corona Lira María Del Pilar
3 er. Vocal:	Dr. Diaz Hernández Octavio

Lugar o lugares donde se realizó la tesis:
Facultad de Ingeniería

TUTOR DE TESIS:

VICTOR JAVIER GONZALEZ VILLELA

_____

FIRMA

Agradecimientos

A la Universidad Nacional Autónoma de México por haber sido la cuna de mi formación profesional.

A los profesores de la Facultad de Ingeniería, en especial al Departamento de Mecatrónica, por sus enseñanzas, su dedicación y su tiempo.

Al Dr. Víctor Javier González Villela, por su motivación, orientación y confianza en la realización de esta Tesis.

Agradezco en lo que corresponde a DGAPA, por el apoyo brindado para la realización de este trabajo, a través del proyecto PAPIIT IN117614: "Robótica intuitiva, adaptable, reactiva, híbrida y móvil aplicada al servicio, el rescate y la medicina"

Así también, se extiende un agradecimiento al programa de becas de CONACYT por el apoyo económico que permitió dirigir la atención directamente a la realización de este proyecto.

Resumen

En este trabajo se presenta el desarrollo de un banco de pruebas para el estudio de robótica móvil colaborativa. La posición y orientación de cada robot se obtiene por medio de un sistema de visión a través del software Reactivision. La comunicación se basa en el protocolo de comunicación IEEE 802.11 (WiFi), creando una red local inalámbrica entre los robots y la computadora. Para la planeación de trayectorias y la evasión de obstáculos se utiliza el control por campos potenciales con la aproximación de González Villela V.

El procesamiento de datos se hace en tiempo real a través del software Simulink, comunicando cada subsistema por medio del protocolo UDP. En particular con la implementación de una comunicación basada en redes, y un procesamiento en tiempo real, se optimizará la velocidad del sistema en general.

Para validar el comportamiento del sistema se realizan experimentos en cada etapa del desarrollo.

Contenido

Índice de Tablas y Figuras

1. INTRODUCCIÓN

1.1. Generalidades

Un robot es un dispositivo mecánico versátil, equipado con sensores y actuadores para realizar movimientos, que es controlado por un sistema de cómputo. Algunos ejemplos de estos son un brazo manipulador, un vehículo con patas o ruedas, o una combinación de estos. Los movimientos que son ejecutados por el robot pueden ser pre-programados, tele-operados o autónomos. [1]

La robótica móvil es un área de la robótica que estudia aquellos robots que no tienen ningún eslabón anclado a un medio físico no móvil, como podría ser la tierra. Existen acuáticos, voladores y terrestres. Todos ellos por la manera en que están configurados. Y por la intrínseca relación que guardan con las disciplinas de la ingeniería mecánica, electrónica y el control, son considerados productos inherentemente mecatrónicos. [2]

Con el avance acelerado del desarrollo de la inteligencia artificial, sensores, control automático y equipos de sistemas de visión, los robots móviles han tenido un amplio campo de aplicación en el mundo real, como búsqueda, rescate, vigilancia, y exploración por mencionar los más destacados. Esto se debe principalmente a que los robots pueden mejorar la eficiencia de llevar a cabo tareas complejas que una persona tardaría en realizar, así como la capacidad de trabajar en lugares peligrosos o con sustancias nocivas para él hombre.

Uno de los objetivos finales de la robótica es el crear robots autónomos. Estos aceptarán una descripción de alto nivel de una tarea y la ejecutarán sin intervención humana. Esta descripción especificará que es lo que requiere el usuario que sea realizado en vez de especificar cómo realizarlo. [1]

En la investigación de la robótica móvil y en particular de los robots autónomos algunas de las cuestiones más importantes para su desarrollo es su arquitectura, es decir su comportamiento; la forma en la cual planea y ejecuta una tarea, como se comunica con otros robots y/o computadora(s) y el método por el cual conoce su ubicación. Por lo tanto los robots autónomos necesitan un sistema de inteligencia que les permita resolver problemas y desarrollar estrategias para cumplir un conjunto de tareas asignadas mientras operan en ambientes dinámicos. Meystel en [3] menciona que la autonomía está determinada por el nivel de inteligencia de una máquina: un alto nivel de inteligencia conlleva un alto nivel de autonomía.

Desafortunadamente, hay una serie de cuestiones que deben abordarse antes de que los robots móviles autónomos se conviertan en algo cotidiano de nuestras vidas, uno de los problemas más restrictivos es la velocidad de procesamiento y el sistema de comunicación,

Estos sistemas ofrecen al robot herramientas para realizar una tarea y moverse en su entorno. Dichos sistemas tienen un gran impacto en el rendimiento de los robots autónomos, tolerancia a fallos, robustez, utilidad y costo.

En este trabajo se presenta el desarrollo de un banco de pruebas para el estudio de robots móviles autónomos. El sistema de comunicación está basada en el protocolo IEEE 802.11 (WiFi) empleando el WifiShield de Arduino. El procesamiento se realiza configurando el software de Simulink Matlab en tiempo real con la teoría de los campos potenciales. La retroalimentación visual se basa en el uso de Reactivision.

1.2. Trabajos Previos

En los últimos años, los robots se han hecho cada vez más comunes en la vida diaria de los seres humanos, a lo largo de la historia los robots han ido evolucionando hasta sustituir actividades de los humanos que requieren alta precisión, largos periodos de trabajo, actividades repetitivas o que conllevan un alto riesgo. La investigación de la robótica ha crecido ampliamente y sus principales temas de investigación son: [4]

- Comunicaciones
- Arquitecturas.
- Control
- Planificación de tareas.
- Inspiraciones Biológicas.
- Coordinación
- Robots reconfigurables.
- Localización y exploración.
- Manipulación y transporte de objetos.

Esta tesis se ha centrado en la comunicación y arquitecturas. La comunicación se refiere al protocolo de comunicación entre robot y computadora. La arquitectura es un diseño conceptual y estructural de las operaciones que hará el sistema.
Se ha realizado una investigación sobre los sistemas de comunicación y arquitectura de los sistemas de robots móviles más novedosos en los últimos años. Como primer lugar en [5] se presenta al grupo de robots llamados Kilobots. Este grupo de robots utiliza una comunicación descentralizada por medio de un emisor y receptor infrarrojo en la base del robot, todos los robots mandan información enviando datos hacia la base del área de trabajo, de tal manera que la luz infrarroja rebota en el área para ser recibida por el otro robot, este tipo de comunicación a base de luz infrarroja, es bastante común en robots de tamaño pequeño. En [6] un grupo de robots de tamaño mediano, utiliza un protocolo de comunicación a base de bluetooth, creando una pequeña red loca entre los robots emparejados en el sistema. En [7] se propone una arquitectura por medio de protocolos de

comunicación similares al IEEE 802.11 para el control de robots colaborativos, esta arquitectura está basada en la creación de redes inalámbricas por medio de routers para optimizar el envío de datos. En [8] se propone un banco de pruebas para el estudio de robótica móvil para el grupo MRG (Mechatronics Research Group) a cargo del Dr. Víctor Javier González Villela, se utiliza un sistema de visión para obtener las coordenadas X,Y de los robots en el plano, así como su orientación, la comunicación se basa en una red por medio de Xbee, la cual utiliza un protocolo peer to peer con el fin de llevar la información a cada uno de los robots, este banco de pruebas ha sido base para la generación de proyectos en los cuales se han involucrado: optimización de técnicas de planeación de trayectorias y evasión de obstáculos, técnicas de visión y reconocimiento de patrones, así como de colaboración entre robots. A pesar de que el banco de pruebas ha sido útil, su mayor desventaja es el tiempo que tarda en realizar las acciones.

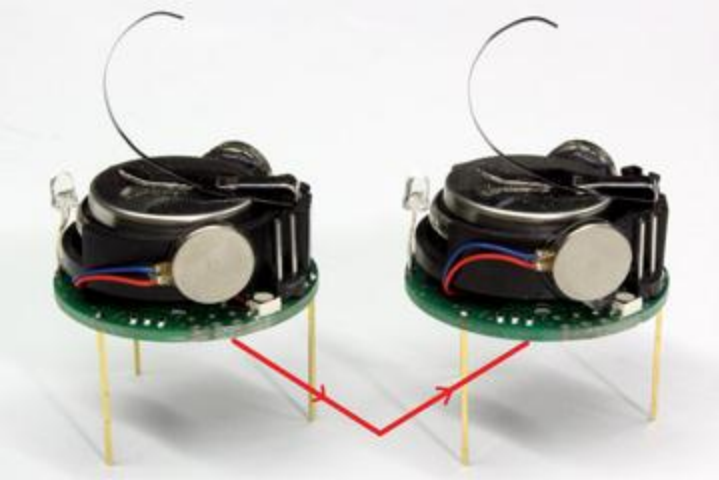


Fig. 1.1 Kilobots, Robots autónomos desarrollados en la universidad de Harvard Su sistema de comunicación es a base de infrarrojos

1.3. Aportación

La robótica móvil tiene más de cincuenta años de investigación y desarrollo, y en esos años se han explorado una gran cantidad de técnicas con el fin de optimizar su tamaño, eficiencia e interacción con los humanos, estas técnicas abarcan desde su manufactura, hasta los algoritmos de control, incluyendo sus modelos cinemáticos y dinámicos. Día con día evolucionan estos sistemas robóticos, sin embargo la aplicación de nueva tecnología en el uso de robótica móvil ha quedado limitada por la velocidad de procesamiento del software dedicado al control, tolerancia a fallos en las comunicaciones e incluso limitantes mecánicas.

Con base en [8] se desea desarrollar un sistema capaz de utilizar tecnología de comunicación novedosa, un sistema de procesamiento en tiempo real y un sistema de visión que funcione con rapidez. La propuesta de este trabajo es aprovechar las nuevas tecnologías

de comunicación implementadas en microcontroladores y el procesamiento de los últimos procesadores de computadoras para aumentar la velocidad del sistema en general.

1.4. Objetivos

El objetivo general de este trabajo es desarrollar un banco de pruebas para el estudio de robots móviles, mediante el uso de comunicación inalámbrica y procesamiento en tiempo real. Como objetivos particulares se tienen:

- Montar un sistema de experimentación, basado Simulink y con retroalimentación visual que permita obtener la posición, orientación y velocidad de uno o varios robots dentro de un área de trabajo en tiempo real.
- Utilizar la información del sistema de visión para retroalimentar el lazo de control de un campo potencial, con la finalidad que el robot llega a un objetivo.
- Usar un protocolo de comunicación inalámbrica que soporte las velocidades de muestreo del tiempo real y haga más fácil la comunicación en sistemas de robótica colaborativa.

1.5 Justificación

Los robots móviles son productos mecatrónicos que pueden satisfacer las necesidades de las personas o hacer su vida cotidiana más cómoda y segura, su uso en el transporte de materiales peligrosos o de construcción en el área industrial es un gran impacto tecnológico. El conocer que la ayuda de estos robots ya es una realidad en industrias como Amazon para la planeación y control dentro de sus almacenes, nos da la pauta para considerar que en el futuro la interacción con los robots será una realidad, sin embargo la tecnología empleada en muchos casos es costosa, voluminosa y lenta. [9] Por lo tanto los resultados de este trabajo son de alto impacto en las aplicaciones del grupo MRG así como en el desarrollo de la robótica móvil.

2. DESCRIPCIÓN DEL SISTEMA.

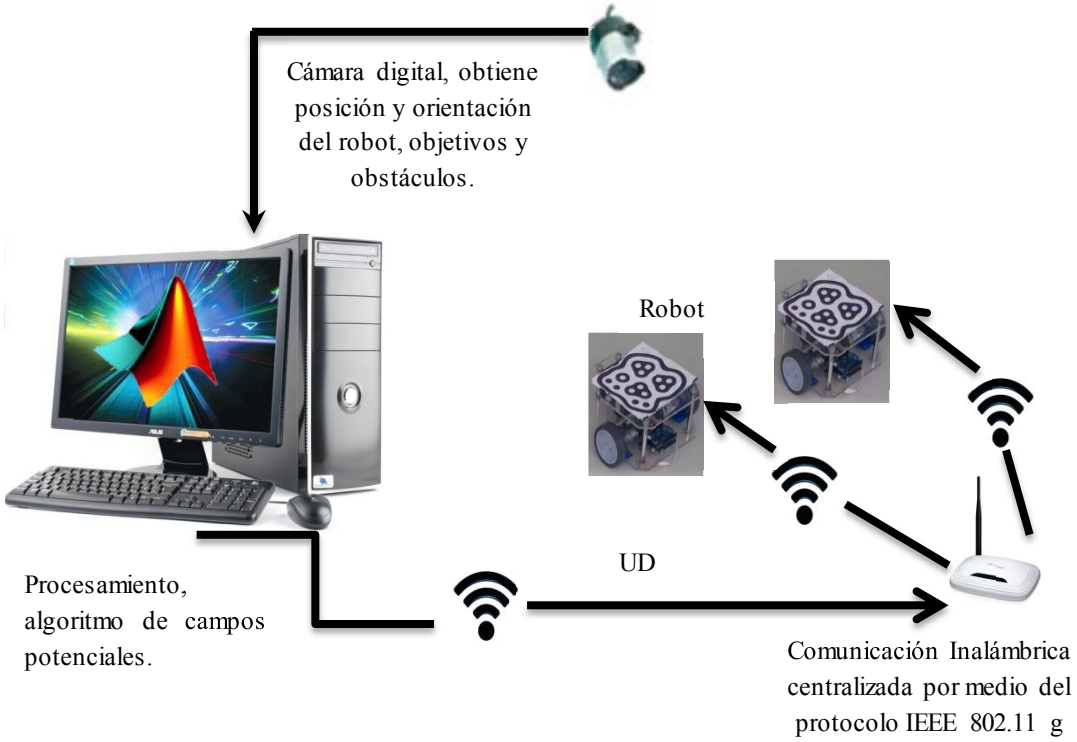


Fig 2. 1 Banco de pruebas propuesto para el estudio de robótica móvil.

El sistema general está compuesto por cuatro subsistemas interconectados entre sí, un sistema de visión el cual identifica la posición y orientación de cada marcador a través de una cámara web, una computadora la cual con la información del sistema de visión usa el algoritmo de campos potenciales, este algoritmo calcula las velocidades de cada llanta para que el robot llegue a su objetivo y evada obstáculos, estas velocidades se mandan a través de una red LAN inalámbrica, el robot recibe la información y por medio de un microcontrolador el cual controlará la velocidad de cada rueda para llegar a su objetivo.

2.1 Modelo OSI

Para lograr comprender la comunicación que se realiza entre cada uno de los subsistemas es necesario saber que es y cómo funcionan los protocolos de comunicación en una red, en específico el protocolo de comunicación UDP. [10]

El modelo de interconexión de sistemas abiertos (ISO/IEC 7498-1), también llamado OSI (Open System Interconnection) es el estándar de modelo de red creado por la Organización Internacional para la Estandarización (ISO) en el año de 1980. Es un marco de referencia para la definición de arquitecturas en la interconexión de los sistemas de comunicaciones.

Fig 2. 2 Capas del modelo OSI

Cada una de las capas del modelo OSI describen el proceso de transmisión de los datos dentro de una red. Las dos únicas capas del modelo con las que, interactúa el usuario son la primera capa, la capa Física, y la última capa, la capa de Aplicación.

2.1.1 Capa de Aplicación

La capa de aplicación proporciona la interfaz y servicios que soportan las aplicaciones que el usuario utiliza, también se encarga de ofrecer los servicios de red relacionados con estas aplicaciones, como mensajes, la transferencia de archivos y las consultas a bases de datos. Entre los servicios de intercambio de información que gestiona la capa de aplicación se encuentran la Web, los servicios de correo electrónico como SMTP (Simple Mail Transfer Protocol), así como aplicaciones especiales de bases de datos cliente/servidor.

2.1.2 Capa de Presentación

La capa de presentación puede considerarse el traductor del modelo OSI. Esta capa toma los paquetes de la capa de aplicación y los convierte a un formato genérico que pueden leer todas las computadoras. Por ejemplo, los datos escritos en caracteres ASCII se traducirán a un formato más básico y genérico.

La capa de presentación también se encarga de cifrar los datos así como de comprimirlos para reducir su tamaño. El paquete que crea la capa de presentación contiene los datos prácticamente con el formato con el que viajarán por las restantes capas del modelo OSI

2.1.3 Capa de Sesión

La capa de sesión es la encargada de establecer el enlace de comunicación o sesión entre las computadoras emisora y receptora. Esta capa también gestiona la sesión que se establece en ambos nodos.

Una vez establecida la sesión entre los nodos participantes, la capa de sesión pasa a encargarse de ubicar puntos de control en la secuencia de datos. De esta forma, se proporciona cierta tolerancia a fallos dentro de la sesión de comunicación. Si una sesión falla y se pierde la comunicación entre los nodos, cuando se restablezca la sesión solo

tendrán que volver a enviarse los datos situados detrás del último punto de control recibido. Así se evita el tener que enviar de nuevo todos los paquetes.

2.1.4 Capa de Transporte

La capa de transporte es la encargada de controlar el flujo de datos entre los nodos que establecen una comunicación; los datos no sólo deben entregarse sin errores, sino además en la secuencia que proceda. La capa de transporte se ocupa también de evaluar el tamaño de los paquetes con el fin de que éstos tengan el tamaño requerido por las capas inferiores. El tamaño de los paquetes lo dicta la arquitectura de red que se utilice.

La comunicación también se establece entre computadoras del mismo nivel; la aceptación por parte del nodo receptor se recibe cuando el nodo emisor ha enviado el número acordado de paquetes.

Esta comunicación en la capa de transporte resulta muy útil cuando la computadora emisora manda demasiados datos a la computadora receptora. En este caso, el nodo receptor tomará todos los datos que pueda aceptar de una sola vez y pasará a enviar una señal de "ocupado" si se envían más datos. Una vez que la computadora haya procesado los datos y esté lista para recibir más paquetes, enviará a la computadora emisora un mensaje que envíe los restantes.

2.1.5 Capa de Red

La capa de red encamina los paquetes además de ocuparse de entregarlos. La determinación de la ruta que deben seguir los datos se produce en esta capa, lo mismo que el intercambio efectivo de los mismos dentro de dicha ruta, esta capa es donde las direcciones lógicas como las direcciones IP pasan a convertirse en direcciones físicas, como direcciones MAC.

Los routers operan precisamente en esta capa y utilizan protocolos de encaminamiento para determinar la ruta que deben seguir los paquetes de datos.

2.1.6 Capa de enlace de datos

Cuando los paquetes de datos llegan a esta capa, estos pasan a ubicarse en tramas, que vienen definidas por la arquitectura de red que se utiliza (Ethernet, Token Ring, etc). La capa de enlace de datos se encarga de deslazar los datos por el enlace físico de comunicación hasta el nodo receptor, e identifica cada computadora incluida en la red de acuerdo con su dirección física.

La información de encabezamiento se añade a cada trama que contenga las direcciones de envío y recepción. La capa de enlace de datos también se asegura de que las tramas enviadas por el enlace físico se reciben sin error alguno.

2.1.7 Capa Física.

En la capa física las tramas procedentes de la capa de enlace de datos se convierten en una secuencia única de bits que puede transmitirse por el entorno físico de la red**.** La capa física también determina los aspectos físicos sobre la forma en que el cableado está enganchado, el grosor del cable y parámetros de corriente y voltaje por ejemplo.

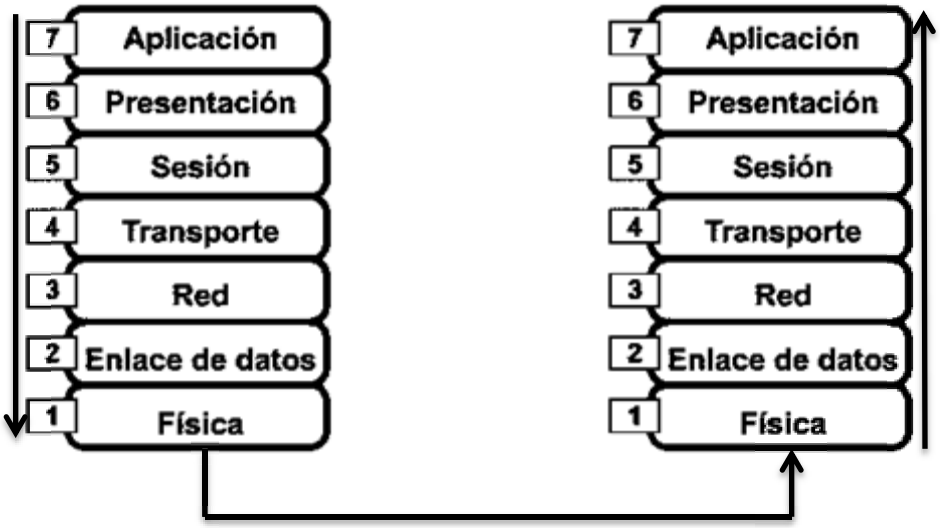


Fig. 2. 3 Forma de enlace de datos entre dos dispositivos mediante el modelo OSI

La comunicación entre dos dispositivos se realiza empezando por la capa de aplicación del primer dispositivo y va bajando hasta la capa física, las capas físicas de los dispositivos son el enlace de la comunicación, cuando el mensaje llega a la capa física del segundo dispositivo va subiendo por las capas hasta entregar el mensaje en la capa de aplicación.

2.1.8 Protocolo TCP y UDP

Los dos protocolos más comunes de la capa de Transporte del conjunto de protocolos TCP/IP son el Protocolo de Control de Transmisión (TCP) y el Protocolo de Datagramas de Usuario (UDP). Ambos protocolos gestionan la comunicación de múltiples aplicaciones. Las diferencias entre ellos son las funciones específicas que cada uno implementa.

UDP es un protocolo simple, sin conexión, descrito en la RFC768. Cuenta con la ventaja de proveer la entrega de datos sin utilizar muchos recursos. Las porciones de comunicación en UDP se llaman datagramas. Sus mensajes son poco fiables debido a la forma en la cual el protocolo manda estos mensajes, está orientado a la velocidad de transmisión, la aplicación debe de hacerse cargo de empaquetar y desempaquetar cada datagrama.

TCP es un protocolo orientado a la conexión, descrito en la RFC 793. TCP usa recursos adicionales para agregar funciones. Las funciones adicionales especificadas por TCP están en el mismo orden de entrega, son de entrega confiable y de control de flujo. El protocolo se encarga de empaquetar y desempaquetar cada segmento para su aplicación.

Cada segmento de TCP posee 20 bytes de carga en el encabezado, mientras que UDP solo posee 8 bytes de carga.

La velocidad para mandar mensajes a través del protocolo UDP deriva en la falta de un algoritmo que sea capaz de determinar si el paquete llegó bien a su destino, es importante saber que el protocolo sólo manda cadenas de texto, por lo que es necesario convertir las variables recibidas a enteros, flotantes o el tipo de variable de nuestro interés. Mientras que el protocolo IP permite mandar el tipo de variable que sea, ya que se dedica a empaquetar cada dato, colocar una cabecera y ordenarlo para que llegue sin errores al dispositivo de destino.

Las aplicaciones del protocolo UDP van orientadas a interfaces de audio y vídeo en tiempo real, es decir si se llega a perder un dato, no tiene gran importancia, porque el mensaje se seguirá entendiendo. Un pixel que se pierda en un vídeo de alta resolución no marcará gran diferencia ya que en el siguiente fotograma la imagen habrá cambiado.

Las direcciones IP (Internet Protocol) son un número único e irrepetible con el cual se identifica una computadora conectada a una red, en la capa de Red del modelo OSI las direcciones físicas denominadas MAC se convierten en una dirección IP. Una dirección IP es un conjunto de cuatro números que van del 0 al 255 separados por puntos; por ejemplo: 200.36.127.40. Por lo tanto una dirección IP es una dirección de 32 bits y se divide en dos partes; los números de la izquierda indican la red y se les denomina net ID o identificador de red y los números de la derecha indican los equipos dentro de esta red se les denomina host ID o identificador de host.

Existen muchas direcciones IP reservadas y especiales, sin embargo para esta tesis sólo es de interés conocer la dirección 127.0.0.1 la cual es conocida como local host o loopback, cualquier información mandada a esta dirección apunta al mismo dispositivo del que fue mandado, lo cual da la ventaja de comunicar aplicaciones dentro de la misma computadora sin necesidad de estar conectado a una red.

A la vez que los protocolos TCP y UDP trabajan con direcciones IP para identificar en la red el origen y el destino de un paquete, a nivel de transporte TCP y UDP emplean valores de 16 bits conocidos como puertos, que son identificadores de buffers en las máquinas de origen y destino respectivamente. Estos buffers son el interfaz entre la capa de aplicación y la de red, de forma que cada aplicación o proceso tiene asignado un buffer a través de la cual intercambia información con la capa de transporte. Posteriormente la capa de transporte envía esta información en bloques de tamaño adecuado a la capa de red. De esta manera una máquina puede tener varios procesos independientes que emitan y reciban paquetes a nivel de transporte.

Al conjunto formado por una dirección IP y el puerto del proceso origen, más la IP y puerto del proceso en el destino empleando un cierto protocolo de transporte (TCP o UDP) Se le

conoce como socket. Los sockets permiten la transmisión bidireccional de datos, como puede ser en modo full-duplex si el sistema operativo lo permite.

Con el uso de sockets se diferencian dos tipos de procesos: servidores y clientes. Los primeros son los que ofrecen algún tipo de servicio a los segundos. Siendo el cliente el que siempre solicita establecer una conexión creando un socket con un servidor para acceder a determinado servicio. Un servidor normalmente puede atender a varios clientes simultáneamente.

2.2 Sistema de Visión

El objetivo del sistema de visión es identificar la ubicación del robot y su entorno, es decir los obstáculos y los puntos a los cuales se pretende que llegue el robot.

Reactivision es un software abierto (open source) y multiplataforma para el reconocimiento de patrones llamados Fiducials o amibas y detección de dedos, creado por Martin Kaltenbrunner y Ross Bencia en la Universidad Pompeu Fabra como parte de la tecnología de la Reac Table. [11]

Su desarrollo está vinculado al uso de interfaces tangibles (TUI), ligado a objetos para el seguimiento de su posición.

La aplicación analiza en tiempo real una imagen de video obtenida desde una cámara web para identificar los patrones (Fiducials), o para rastrear las yemas de los dedos sobre alguna superficie táctil.

Un Fiducial es un patrón bi-tonal diseñado con un ID único que puede ser identificado por Reactivision a partir de un ordenamiento en forma de árbol con patrones alternados blancos y negros.

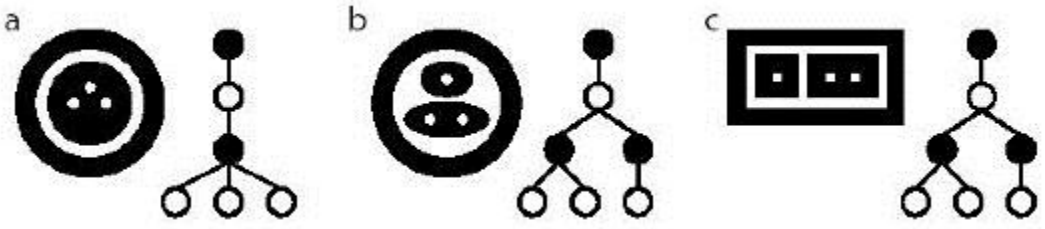


Fig. 2. 4 Reconocimiento de Patrones de Reactivision

En la figura 2.4 se tiene una estructura 1-1-1-3, en la cual una forma negra contiene una forma blanca, que a su vez contiene otra forma negra, que rodea tres formas blancas más.

En los otros dos casos Reactivision obtendrá el mismo ID ya que el árbol es el mismo, a pesar de que su imagen sea diferente. La forma y ubicación de los niveles de inclusión permite determinar las coordenadas X, Y así como el ángulo de rotación y la velocidad de estos parámetros.

2.2.2 Reactivision TUIO

TUIO es el protocolo de comunicación que utiliza Reactivision para enviar la información de los Fiducials que detecta, dicha información consiste en la posición, rotación y velocidad, esta información es mandada a través de UDP a alguna otra aplicación, la comunicación está codificada utilizando el protocolo OSC; Open Sound Control, el cual permite comunicar instrumentos de música o dispositivos multimedia en tiempo real sobre una red, es un remplazo del protocolo MIDI.

La aplicación puede mandar mensajes MIDI como otra alternativa, lo que permite asignar cualquier dimensión al objeto a través de un archivo XML, eliminar o añadir objetos con métodos sobre eventos, sin embargo el protocolo MIDI tiene un ancho de banda y resolución menor en comparación con Open Sound Control, por lo cual MIDI solo se toma como una alternativa dependiendo el caso.

Un perfil TUIO define dos mensajes centrales: “Set” y “Alive”; los mensajes Set se utilizan para comunicar información sobre el estado del token en el caso de Reactivision es el estado de cada objeto Fiducial, los mensajes Alive indican el conjunto actual de tokens presentes en el campo de visión utilizando una lista de identificadores.

Por defecto Reactivision envía paquetes a través del puerto 3333 por medio de UDP.

Un paquete contiene la información de cada Fiducial y debe ser desempaquetado dependiendo las variables de interés; la sintaxis del mensaje para una superficie en dos dimensiones es:

/tuio/2Dobj set s i x y a X Y A m r

Parámetro	Descripción	Tipo de Dato
s	ID de la sesión	int32
i	ID del Fiducial	int32
x, y, z	Posición	float32, 0… 1
a, b, c	Ángulo	float32, 0… 2PI
w, h, d	Dimensión	float32, 0… 1
f, v	Área, Volumen	float32, 0… 1
X, Y, Z	Vector de velocidad lineal	float32
A, B, C	Velocidad Angular	float32
m	Aceleración Lineal	float32
r	Aceleración Angular	float32
p		Encabezado del mensaje

Tabla 2. 1 Descripción de los parámetros enviados por Reactivision

Es importante mencionar que el ID de la sesión es un indicador cuando entra un Fiducial al campo de visión y se mantiene hasta que el Fiducial sale del campo, esto da la ventaja de tener varios Fiducial con el mismo ID sin que Reactivision se confunda en el seguimiento de ellos.

Reactivision ofrece librerías para desempaquetar el protocolo OSC en los siguientes lenguajes de programación:

- C++
- Java
- C#
- Processing
- Pure Data
- Max/Msp

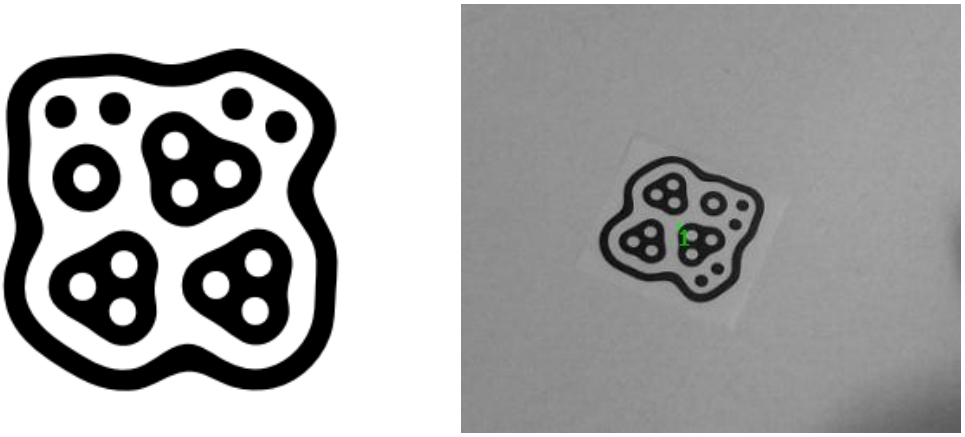


Fig. 2. 5 Fiducials Reactivision

Para iniciar Reactivision sólo es necesario ejecutar el programa y tener conectada una cámara web a la computadora, cuando Reactivision detecte un Fiducial mandará la información por el socket 127.0.0.1 a través del puerto UDP 3333.

2.2.3 Iluminación, Calibración y Distorsión.

Para el seguimiento, los objetos deben permanecer en un entorno adecuadamente iluminado, por lo que la cámara y por lo tanto la aplicación de visión pueda reconocerlos correctamente, dependiendo del tipo de cámara es recomendable evitar situaciones donde la luz solar influya de manera directa sobre los patrones de visión.

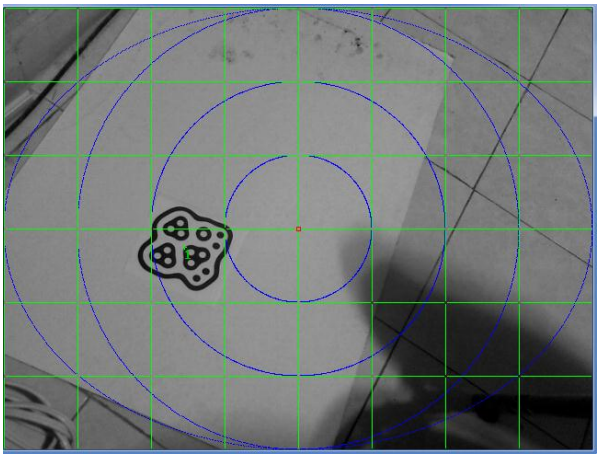


Fig. 2. 6 Modo de Calibración Reactivision

Muchas aplicaciones utilizan lentes angulares o lentes de ojo de pez con el fin de aumentar el área visible de la cámara a una distancia mínima. Lamentablemente estas lentes distorsionan la imagen; Reactivision incluye un modo de calibración el cual es capaz de corregir los errores de lentes en la mayoría de las cámaras a través de una tabla, en la cual el usuario puede ajustar los puntos de la malla.

2.3 Matlab Simulink

El nombre de Matlab proviene de la contracción de los términos Matrix Laboratory y fue concebido para el fácil acceso a las librerías que son de gran importancia en el campo de la computación y el cálculo matricial.

Matlab es un entorno de computación y desarrollo de aplicaciones totalmente integrado, orientado para el desarrollo de proyectos con elevados cálculos matemáticos y la visualización gráfica de estos. Matlab integra análisis numérico, cálculo matricial, procesado de señal, todo ello en un entorno fácil para el usuario.

Matlab se ha convertido en una herramienta básica para la resolución de complejos problemas matemáticos en diferentes áreas como la computación, el cálculo numérico, prototipo de algoritmos, teoría de control automático, estadística, etc.

El software consta de diferentes aplicaciones o toolboxes especializados orientados a ingeniería, ciencia y profesiones técnicas, en particular destacan: Sistemas de Control, Adquisición de Datos, Tiempo Real, Lógica Fuzzy, y Procesamiento de imágenes, Redes Neuronales, Optimización, Procesamiento de Señales, etc.

Simulink es la interfaz gráfica de simulación de Matlab. Permite el análisis y estudio de sistemas mediante la simulación de modelos matemáticos. La creación de estos modelos se basa en un lenguaje de alto nivel, mediante la interconexión gráfica de distintos bloques, estos bloques, se conectan y se parametrizan para su posterior simulación.

2.3.1 Sistema en Tiempo Real

Existen muchas definiciones sobre la naturaleza de un sistema de tiempo real, sin embargo, todas tienen en común la noción de tiempo de respuesta: el tiempo que precisa el sistema en generar la salida a partir de alguna entrada asociada.

Young en 1982 define un sistema de tiempo real como:

> *...cualquier actividad o sistema de proceso de información que tiene que responder a un estímulo de entrada generado externamente en un periodo finito y especificado.*

El proyecto PDCS (Predictably Dependable Computer Systems) proporciona la definición siguiente:

> *Un sistema de tiempo real es aquél al que se le solicita que reaccione a estímulos del entorno (incluyendo el paso de tiempo físico) en intervalos del tiempo dictados por el entorno.*

Por lo tanto se puede decir que para el correcto funcionamiento de un sistema en tiempo real, no solo se depende del resultado lógico, sino también depende del tiempo en que se produce un cambio o un resultado, y este tiempo está asociado a eventos externos, en otras palaras, se refiere al menor tiempo posible en el que un sistema puede dar un resultado en relación a una entrada. [12] [13]

2.3.2 Real Time Windows Target

Matlab presenta una aplicación para hacer simulaciones en tiempo real, la toolbox Real Time Windows Target. Esta herramienta permite realizar aplicaciones de control y simulaciones en tiempo real para plantas físicas. [14]

Real Time Windows Target es una herramienta que permite capturar y generar señales en tiempo real por medio de Simulink. Además, se pueden visualizar estas señales, cambiando y controlando parámetros, todo en tiempo real. Para hacerlo posible tiene que haber un elemento físico que interactúe entre Simulink y el elemento exterior que queremos controlar. En la mayoría de los casos este elemento es la placa de adquisición de datos DAQ, que es la que permite operar con señales de entrada y/o salidas analógicas y digitales.

El modo externo de Simulink y Real time Windows Target permite utilizar el modelo de Simulink con una interfaz gráfico para:

- Visualizar señales en tiempo real
- Cambio de parámetros mediante cajas de diálogo de los bloque en Simulink durante la ejecución en tiempo real.

2.3.3 Real-Time Windows Kernel

Un componente clave del Real Time Windows Target es un kernel en tiempo real que hace de interfaz con el sistema operativo Windows para asegurar que la aplicación en tiempo real se está ejecutando en el tiempo de muestreo seleccionado. El kernel asigna la prioridad más elevada de ejecución para la aplicación en tiempo real, y lo hace utilizando el reloj interno de la computadora como fuente principal de tiempo.

El kernel intercepta la interrupción del reloj de la computadora antes que el sistema operativo Windows la reciba, y bloquea cualquier llamada al sistema operativo, mientras el kernel utiliza la interrupción para iniciar la ejecución del modelo compilado.

Para garantizar un periodo de muestreo preciso el kernel reprograma el reloj de la computadora a una frecuencia mayor, Debido a que el reloj del PC es también la principal fuente de tiempo para el sistema operativo, el kernel envía una interrupción al sistema manteniendo la tasa de interrupción inicial. Durante la ejecución de la aplicación en tiempo real almacena los datos en buffers, y posteriormente el contenido de los buffers es recuperado por Simulink para imprimirlas en pantalla.

El kernel también sirve de interfaz y comunica con el hardware de entrada y salida utilizando los correspondientes drivers que comprueban que la instalación de la placa DAQ sea correcta. En el caso que esto no suceda, no se podrá ejecutar la aplicación.

Una aplicación en tiempo real tiene las siguientes características:

- Código compilado: Es el resultado de compilar el código fuente, en el caso de Simulink el código del modelo matemático es traducido y compilado en lenguaje C.
- Relación con el modelo de Simulink: El ejecutable contiene una relación de todos los componentes del modelo, es decir conexión entre bloques, dependencias de tiempo y variables.

- Relación con el kernel: el modelo tiene que ser cargado y ejecutado directamente por el kernel del Real Time Windows Target.
- Cheksum: el modelo y el ejecutable de Simulink contienen un valor de Cheksum. El kernel utiliza este valor para comparar el modelo y el ejecutable, si estos son coherentes permitirá realizar la ejecución.

2.3.4 Parámetros de simulación

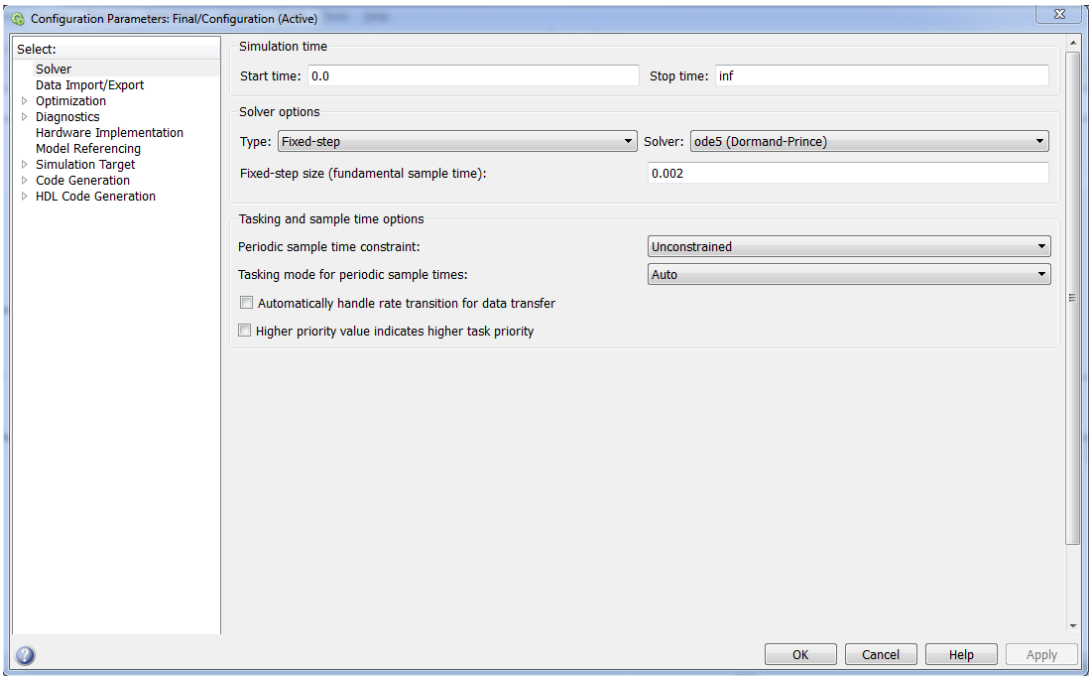


Fig. 2. 7 Parámetros de Configuración en Simulink

Esta sección dentro del programa de Simulink permite modificar las opciones básicas de la simulación. El tiempo de simulación es un parámetro en el cual se decide el tiempo que se requiere dure la simulación, teniendo la opción de decidir el tiempo de inicio y de fin, con las casillas start time y stop time, por defecto el tiempo de simulación será de 10 segundos.

Existen dos grupos de métodos numéricos para la integración de los modelos matemáticos dentro de Simulink, los de paso fijo y los de paso variable.

El grupo de los métodos de integración de paso fijo mantienen un tamaño de muestreo fijo para la resolución de las ecuaciones durante todas las simulaciones, mientras que el grupo de paso variable, varía el muestreo dependiendo de los resultados, si los resultados son constantes, aumentará la tasa de muestreo, por otro lado si existen variaciones el muestreo se irá reduciendo. De este modo, depende de la dinámica del modelo que se requiere simular, los resultados que se obtienen de las simulaciones con el método de integración.

El grupo de métodos de iteración de paso fijo permite tener un control del muestreo por lo que este grupo de métodos son los adecuados para trabajar en tiempo real, como desventaja,

el tiempo que tarde en obtener resultados de simulación es más lento comparado con los que utilicen métodos de paso variable.

El tiempo de muestreo, es el parámetro más importante para el método de integración que se elija, la opción Max Step Size indica el periodo de muestreo mínimo que debe tomar el método de integración, por el contrario Min Step Size, indica el periodo de muestreo mínimo que debe tomar el método en caso que la dinámica del sistema varíe.

2.3.5 Funciones S

Las funciones S son bloques de Simulink programables por el usuario y que sirven para describir modelos dinámicos complicados es decir parámetros variantes con el tiempo, sistemas no lineales, sistemas con diferentes periodos de muestreo, etc.

La comunicación con el exterior por medio de hardware también se realiza con funciones S. En este caso las funciones S están escritas en lenguaje C y deben contener los comandos para la comunicación con las tarjetas de adquisición de datos, deben ser compiladas con la utilidad mex de Matlab.

Para la simulación de sistemas dinámicos las funciones S ejecutan una estructura de rutinas básicas que Simulink ejecuta de manera secuencial:

- Inicialización, flag = 0
- Cálculo de derivadas en sistemas continuos, flag = 1
- Cálculo de la diferencia finita en sistemas discretos, flag = 2
- Cálculo de la salida del sistema, flag = 4
- Terminación, flag = 9

No es necesario implementar todas las funciones para desarrollar una función S.

La estructura simula muestra a muestra cada ecuación de estado del sistema.

Primero, simula la ecuación de estado:

$$\dot{x} = Ax + Bu \qquad (2.1)$$

Y después la ecuación de salida:

$$y = Cx + Du \qquad (2.2)$$

En el caso de sistemas discretos en el tiempo, la ecuación de estados es una ecuación en diferencias:

$$x[n+1] = Ax[n] + Bu[n] \qquad (2.3)$$

Donde n es el tiempo de muestreo del sistema.

2.4 Planeación de Trayectorias

Las acciones que pueden ser desempeñadas por los robots autónomos se clasifican en tres categorías, según la función que desempeñan dentro del sistema. Estas tres clasificaciones, denominadas primitivas de la robótica, son razonar, planear y actuar.

Razonar se refiere a las funciones que obtienen información de los sensores del robot, los cuales pueden ser de dos tipos dependiendo de la información que proporcionan: propioceptivos y exteroceptivos.

Los sensores propioceptivos sirven para medir el estado interno del robot. Uno de los sensores más utilizados de este tipo en los robots con ruedas, son los encoders, con los cuales se puede calcular la posición del robot a partir de la medición de la posición angular de sus llantas.

Los sensores exteroceptivos se utilizan para obtener información sobre el ambiente en el que se desempeña. Una de las aplicaciones más utilizadas de estos tipos de sensores es la detección de obstáculos a partir de sonares, sensores de presión o laser.

Planear se refiere a una función de planeación, cuando se generan sub-tareas o se toman decisiones, a partir de la información obtenida de los sensores o de alguna representación del conocimiento generado anteriormente, como pude ser un mapa o un comando por voz.

Actuar es el grupo de funciones donde se envían comandos a los actuadores para ejecutar el movimiento de alguna parte del robot, generalmente basados en una ley de control que permite que el actuador llegue a un valor deseado en un tiempo óptimo.

Primitivas	Entrada	Salida
SENSAR	Datos del sensor	Información sensada
PLANEAR	Información	Directivas
ACTUAR	Información sensada o directivas	Comandos de actuador

Fig. 2. 8 Funciones primitivas de los robots.

Utilizando estas funciones primitivas, se definen tres paradigmas para organizar la inteligencia artificial de los robots autónomos. Estos son modelos de la forma en que los robots ejecutan sus tareas. Dichos modelos se basan en la forma en que interactúan las funciones entre sí mediante sus entradas y salidas. Los paradigmas planteados por Murphy en [15] se describen a continuación.

2.4.1 Paradigma Deliberativo.

El primer paradigma utilizado fue el deliberativo, con el que se trató de emular la forma en que el humano piensa desde una perspectiva introspectiva.

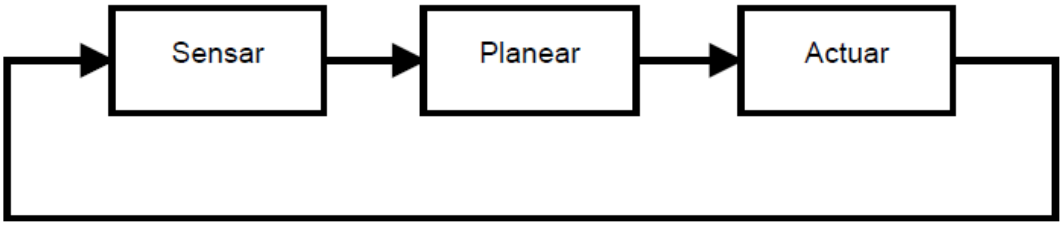


Fig. 2. 9 Paradigma Deliberativo (Flujo de instrucciones)

Para realizar una tarea utilizando este modelo, el primer paso es adquirir información sobre el ambiente y sobre el robot durante la etapa de sensado. Con esta información y datos adquiridos anteriormente, se genera una representación del mundo y del robot. A partir de esta representación, en la siguiente etapa, el robot planea todas las acciones y genera las instrucciones que necesita ejecutar para realizar uno o varios objetivos, con el fin de realizar la tarea encomendada. Por último, las instrucciones generadas se convierten en comandos que son ejecutados por los actuadores. Cuando se completan los objetivos planeados y se han ejecutado las instrucciones, este proceso termina y se repite nuevamente, hasta que el robot lleva a cabo la tarea especificada.

Con este, se pueden llevar a cabo tareas muy complejas dado que, el tener un modelo completo del mundo permite una planeación muy eficiente al momento de generar los objetivos y directivas para realizarlas.

Las desventajas de este tipo de paradigma son que, dependiendo de la complejidad del modelo del mundo, la etapa de planeación tiende a ser muy lenta y costosa computacionalmente, por lo que no permite una rápida ejecución de las funciones de actuado. Por tanto, si algún suceso inesperado surge después de la etapa de sensado, la planeación podría no ser óptima, dado que el modelo del mundo estará incompleto y no se generarán directivas para actuar ante dicho suceso.

2.4.2 Paradigma reactivo

Este modelo ésta basado, en gran medida, en observaciones del proceso que llevan a cabo los insectos para moverse. Fue propuesto inicialmente por Brooks en [16]. En este, la etapa de planeación es removida dando lugar a que las funciones de sensar y actuar se comuniquen directamente. Brooks afirma que, para un robot móvil autónomo, el único modelo necesario del mundo es la información de los sensores del ambiente en el que se está moviendo. La información de los sensores es enviada directamente a los actuadores, con lo que se activan una serie de directivas de bajo nivel simples, concurrentes y

asíncronas, para el control de estos. Cabe mencionar que las instrucciones de bajo nivel no conocen los comportamientos de las instrucciones de alto nivel, lo que da lugar a comportamientos emergentes que se pueden denominar inteligentes.

El paradigma reactivo presenta la ventaja de que, al no existir una fase de planeación, la velocidad del control y las capacidades de reacción del robot se ven incrementadas.

A pesar de estas ventajas, los robots con una arquitectura puramente reactiva no son capaces de resolver tareas demasiado complejas.

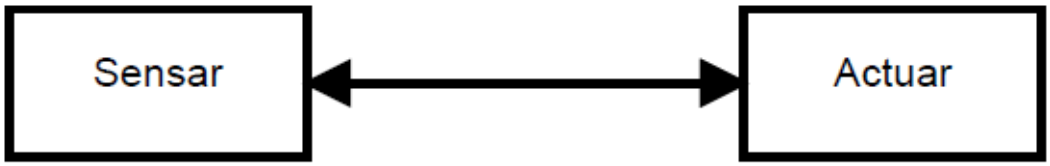


Fig. 2. 10 Paradigma Reactivo

2.4.3 Paradigma híbrido

El paradigma híbrido nació en la década de los 90 y actualmente sigue siendo objeto de estudio. Su desarrollo está basado en el paradigma reactivo, con la intención de ejecutar tareas más complejas que con este eran muy complicadas de manejar.

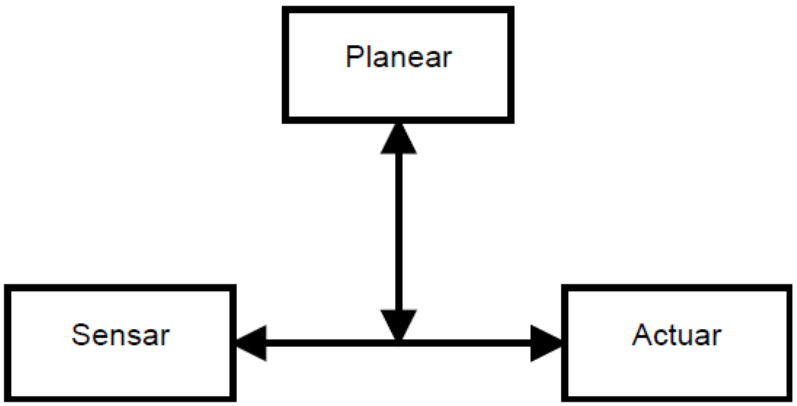


Fig. 2. 11Paradigma Híbrido

Al igual que en el paradigma reactivo, existe una comunicación directa entre la etapa de sensar y la de actuar, en donde la información detectada puede disparar comportamientos que serán ejecutados inmediatamente por los actuadores. Sin embargo, la información sensada también es procesada por funciones de planeación que genera un modelo global del mundo. Éstas realizan tareas de inteligencia artificial relacionadas con la planeación de objetivos globales, resolución de problemas y aprendizaje de máquina. Las funciones de planeación pueden generar directivas para los actuadores para cumplir con objetivos específicos.

Con este paradigma se pueden desarrollar robots autónomos de propósito general que reciban y ejecuten tareas encomendadas de los humanos, utilizando las ventajas de los dos paradigmas mencionados anteriormente. Se obtienen funciones cognoscitivas de nivel superior y un modelo global detallado del mundo, al mismo tiempo que una rápida ejecución de comportamientos básicos para adecuarse a las perturbaciones que se pueden presentar en ambientes no controlados.

Por lo tanto si se tiene conocimiento previo del espacio de trabajo de un robot, es sencillo determinar el tipo de paradigma a implementar, para ayudar a planear los movimientos del robot para que lleguen a cierto punto y evadan obstáculos.

El problema básico de la planeación se define de la siguiente manera

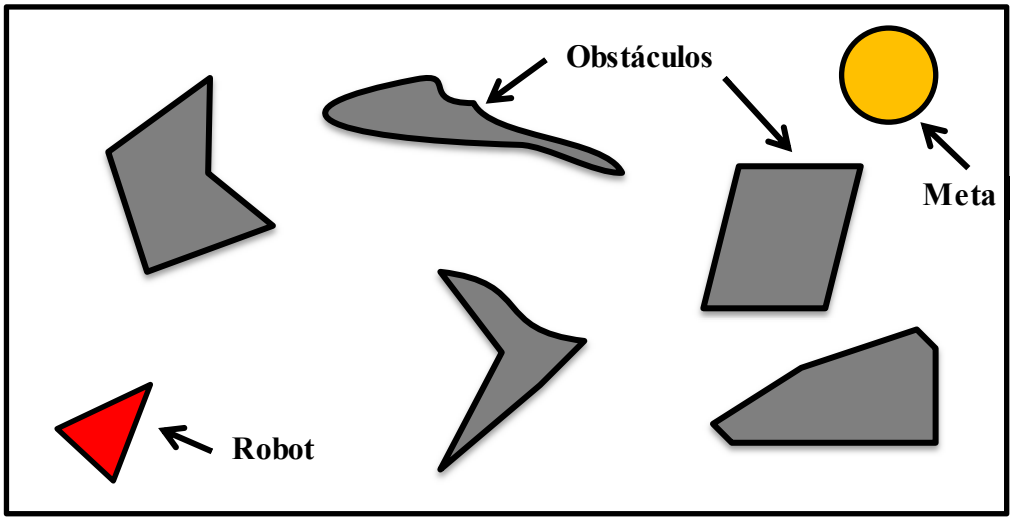


Fig. 2. 12 Representación del problema de planeación

- Sea A un objeto rígido (robot) el cual se mueve en un espacio euclidiano W llamado espacio de trabajo representado como R^2
- Sean $B_1 \ldots B_q$ objetos rígidos distribuidos en W a los cuales llamamos obstáculos.

- Tanto la forma geométrica como la posición del robot y de los obstáculos, distribuidos en W es conocida.

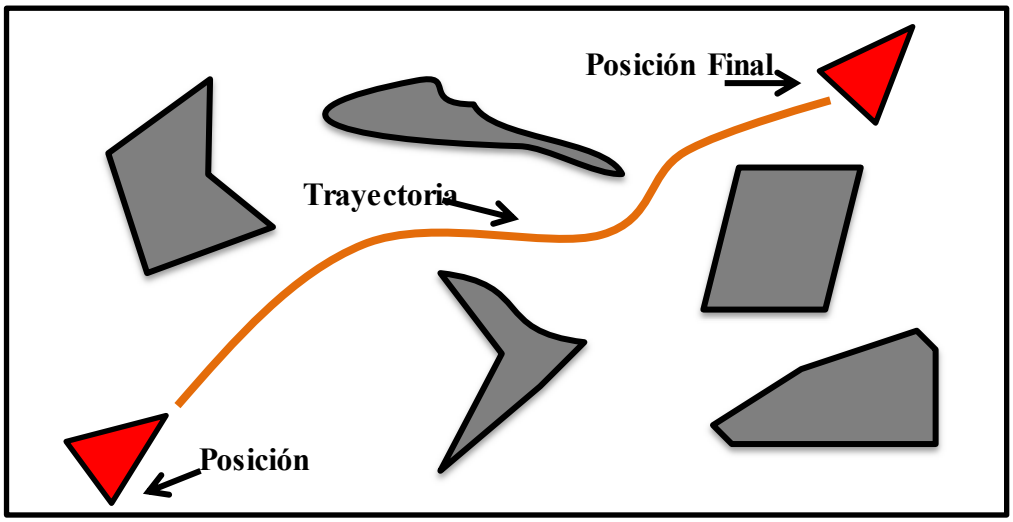


Fig. 2. 13 Trayectoria a seguir para llegar a la Posición Final

El problema consiste en dadas la posición y orientación inicial así como la orientación final del robot en el espacio W, se desea generar una trayectoria que especifique una secuencia continua de posiciones y orientaciones del robot, evitando el contacto con los obstáculos.

En la actualidad existe un gran número de métodos que resuelven parcialmente o con muchas restricciones el problema de planeación de movimientos. Puede ser clasificada en dinámica o estática dependiendo si los obstáculos se encuentran en movimiento o no. En un ambiente estático toda la información acerca de los obstáculos se conoce y el movimiento del robot se diseña a partir de ella, en un ambiente dinámico solo se tiene la información parcial del ambiente y el movimiento del robot se planifica con esta información, conforme el robot cambia de posición se planifica nuevamente la trayectoria.

Los métodos más utilizados en la actualidad son: mapas de carreteras, grafos, método probabilístico de carreteras, descomposición de celdas y campos potenciales por nombrar algunos.

2.4.1 Campos Potenciales

Los campos potenciales son un método para la planeación de trayectorias que consiste en discretizar el espacio de trabajo del robot en una muy fina malla que permita buscar una ruta libre de colisión. El método requiere de una poderosa heurística que guíe la búsqueda dado que quizás la malla que se genere sea muy grande.

En este método el robot es representado como un punto en el espacio, cuyo movimiento está influenciado por una fuerza artificial producida por la posición deseada y los obstáculos. La posición deseada genera una fuerza de atracción que lleva al robot hacia la

meta, mientras que los obstáculos producen un potencial de repulsión que aleja al robot de ellos.

Las suma de fuerzas total se trata como una fuerza artificial aplicada al robot, por lo que en cada configuración la dirección de esta fuerza se considera como la dirección más oportuna del movimiento.

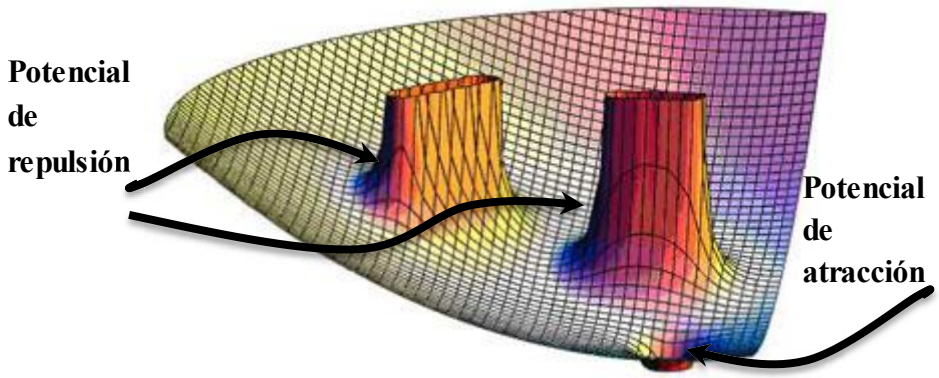


Fig. 2. 14 Representación de un campo potencial

El potencial de atracción disminuye conforme nos acercamos a la configuración final, mientras que el potencial de repulsión tiende a infinito conforme nos acercamos a él. La ruta encontrada entre las posiciones inicial y final se genera siguiendo el gradiente negativo del potencial total.

El método es eficiente sin embargo, tiene como desventaja de que la función potencial puede caer en un mínimo local, lo que significa que la fuerza de atracción y repulsión en algún punto sean iguales, teniendo como consecuencia que el robot no se mueva.

En [17] se plantea el modelo matemático del campo para un espacio determinado, la fuerza de atracción se define como:

$$Fatr\begin{cases} -\varepsilon_1(q-q_{des}) & \forall\ |q-q_{des}| \le d_1 \\ -\varepsilon_1 \frac{(q-q_{des})}{|q-q_{des}|} & \forall\ |q-q_{des}| \le d_1 \end{cases} \tag{2.4}$$

Y las fuerzas de repulsión están definidas por:

$$Frep\begin{cases} \mu\left(\frac{1}{|q-q_{obs}|}-\frac{1}{d_0}\right)*\frac{1}{|q-q_{obs}|^2}*\frac{(q-q_{obs})}{|q-q_{obs}|} & \forall\ |q-q_{obs}| \le d_0 \\ 0 & \forall\ |q-q_{obs}| \le d_0 \end{cases} \tag{2.5}$$

Dónde:

$q \quad \triangleq posición\ inicial\ del\ robot.$
$q_{obs} \quad \triangleq posición\ del\ obstáculo.$
$q_{des} \quad \triangleq posición\ del\ destino.$
$\varepsilon_1 \quad \triangleq coonstante\ de\ atracción.$
$\mu \quad \triangleq coonstante\ de\ repulsión.$
$d_0, d_1 \triangleq radio\ de\ influencia\ al\ robot\ y\ al\ obstáculo.$

Los valores obtenidos por medio de este algoritmo, brinda la trayectoria óptima para llegar al objetivo por la ruta más eficiente.

Este método es se clasifica como un paradigma híbrido, debido a que es capaz de evadir obstáculos, y a la vez tiene un razonamiento en el cual decide la trayectoria más corta para resolver la tarea propuesta.

2.5 Comunicación

La comunicación es un proceso importante para cualquier arquitectura sistema, en la mayoría de ocasiones se hace uso de la comunicación serial, para la comunicación entre microcontroladores y computadoras, en esta tesis se propone utilizar una red inalámbrica bajo el protocolo IEEE 802.11.

Una red informática es un conjunto de dispositivos interconectados entre sí a través de un medio, que intercambian información y comparten recursos, básicamente, la comunicación dentro de una red informática es un proceso en el que existen dos roles bien definidos para los dispositivos conectados. La estructura y el modo de funcionamiento de las redes actuales están definidos por el conjunto de estándares del modelo OSI.

Los dispositivos conectados en una red, puede clasificarse en dos tipos: los que gestionan el acceso y las comunicaciones en una red, como lo son el módem, router, switch, Access point, bridge, etc. y los que se conectan para utilizarla, como una computadora, teléfono celular, impresora, televisión en este caso el dispositivo es un robot.

Considerando el tamaño y a función de la red, se pueden clasificar de la siguiente manera:

- PAN (Personal Area Network) o red de área personal; está conformada por dispositivos utilizados por una sola persona. Tiene un rango de alcance de unos pocos metros. WPAN (Wireless Personal Area Network) o red inalámbrica de área personal: es una red PAN que utiliza tecnologías inalámbricas como medio.
- LAN (Local Area Network) o red de área local; es una red cuyo alcance se limita a un área relativamente pequeña, como una habitación, un edificio, etc.
- WLAN (Wireless Local Area Network) o red de área local inalámbrica; es una red LAN que emplea medios inalámbricos de comunicación.

- CAN (Campus Area Network) o red de área de campus; es una red de dispositivos de alta velocidad que conecta redes de área local a través de un área geográfica limitada, como un campus universitario, una base militar, etc.
- MAN (Metropolitan Area Network) o red de área metropolitana: es una red de alta velocidad que da cobertura en un área geográfica más extendida, por ejemplo una ciudad.
- WAN (Wide Area Network) o red de área amplia; se extiende sobre un área geográfica extensa empleando medios de comunicación como satélites, cables interoceánicos, fibra óptica, etc.

La topología de una red representa la disposición de los enlaces que conectan los nodos de una red. Las redes pueden tomar muchas formas diferentes dependiendo de cómo están interconectados los nodos, hay dos formas de describir la topología de una red: física y lógica, la topología física se refiere a la configuración de cables, antenas, computadoras y otros dispositivos de red, mientras la topología lógica hace referencia a un nivel más abstracto, considerando por ejemplo el método de flujo de la información transmitida entre nodos.

A continuación se hace referencia de algunas topologías de red básicas.

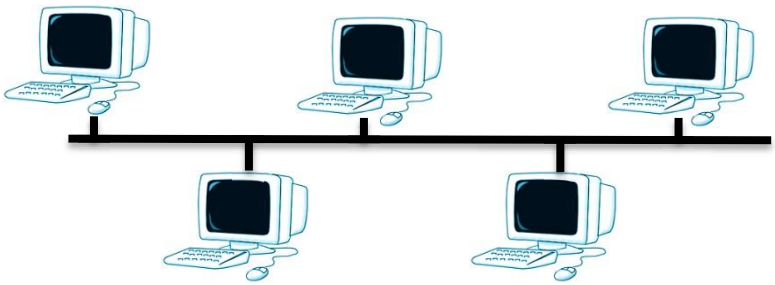


Fig. 2. 15 Red Tipo Bus

- Bus o Barra: Todos los nodos están conectados a un cable común o compartido. Las redes Ethernet normalmente usan esta topología

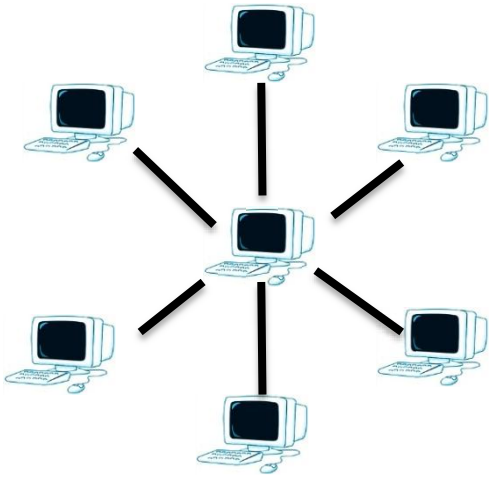


Fig. 2. 16 Red Tipo Estrella

- Estrella: Cada nodo se conecta directamente a un concentrador central. En una topología de estrella todos los datos pasan a través del concentrador antes de alcanzar su destino. Esta es una topología común tanto en redes Ethernet como inalámbricas.

Fig. 2. 17 Red Línea

- Línea o multi concentrador: Un conjunto de nodos conectados en una línea. Cada nodo se conecta a sus dos nodos vecinos excepto el nodo final.

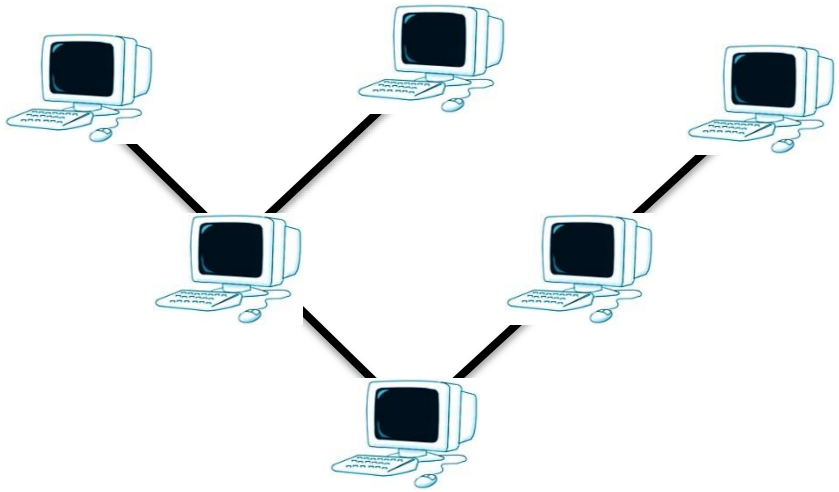


Fig. 2. 18 Red Tipo Árbol

- Árbol: Una combinación de las topologías de bus y estrella, Un conjunto de nodos configurados como estrella se conectan a aun dorsal.

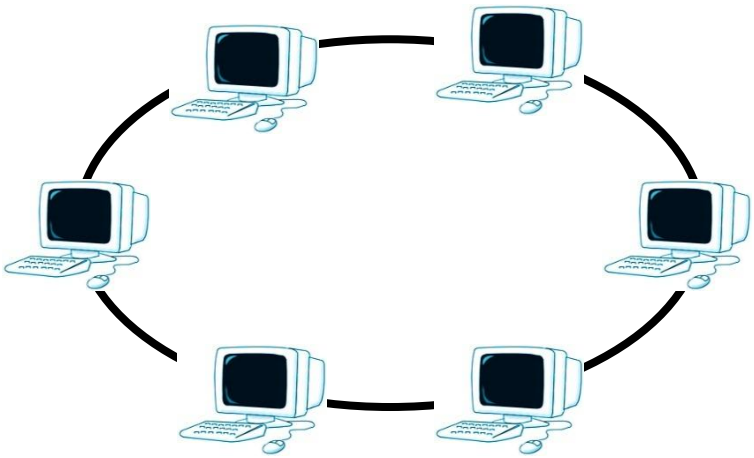


Fig. 2. 19 Red Tipo Anillo

- Anillo: Todos los nodos se conectan entre sí formando un lazo cerrado, de manera que cada nodo se conecta directamente a otros dos dispositivos.

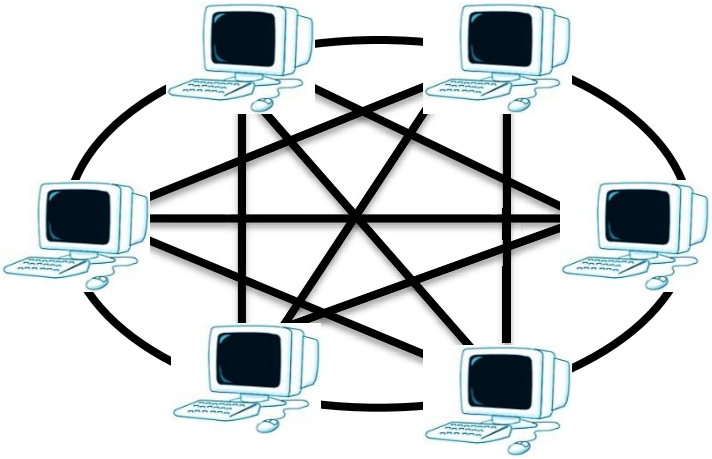


Fig. 2. 20 Red Tipo Malla Completa

- Malla completa: Existe enlace directo entre todos los pares de nodos de la red. Una malla completa con n nodos requiere de n(n-1)/2 enlaces directos. Debido a esta característica, es una tecnología costosa pero muy confiable. Su principal uso es de aplicaciones militares.

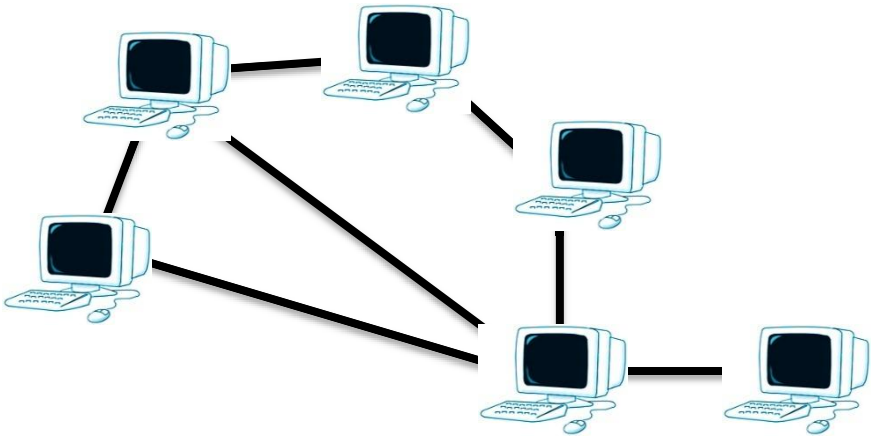


Fig. 2. 21 Red Tipo Malla Parcial

- Malla parcial: Algunos nodos están organizados en una malla completa, mientras otros se conectan solamente a uno o dos nodos de la red. Esta topología es menos costosa que la malla completa pero por supuesto, no es tan confiable ya que el número de enlaces es reducido.

2.5.1 Protocolo IEEE 802.11

La especificación IEEE 802.11 (ISO/IEC 8802-11) es un estándar internacional que define las características de una red de área local inalámbrica. WiFi que significa Fidelidad Inalámbrica es el nombre de la certificación otorgada por la WiFi Alliance, grupo que garantiza la compatibilidad de dispositivos que utilizan el estándar 802.11.

Una red WiFi es en realidad una rede que cumple con el estándar 802.11. WiFi permite crear redes inalámbricas de alta velocidad siempre y cuando el equipo que se vaya a conectar no esté muy alejado del punto de acceso. En la práctica, WiFi admite computadoras, equipos móviles, televisiones, etc. Se considera una conexión de alta velocidad a la red que supere los 11Mbps dentro de un radio de varias docenas de metros en ambientes cerrados o de cientos de metros al aire libre.

El estándar 802.11 establece los niveles inferiores del modelo OSI para las conexiones inalámbricas, es decir la capa física y la capa de enlace de datos.

La capa física define la modulación de las ondas de radio y las características de señalización para la transmisión de datos, mientras que la capa de enlace de datos define la interfaz entre el bus del equipo y la capa física.

El estándar 802.11 en realidad es el primer estándar y permite un ancho de banda de 1 a 2 Mbps. El estándar original se ha modificado para optimizar el ancho de banda para especificar componentes de mejor manera con el fin de garantizar mayor seguridad o compatibilidad.

Nombre del estándar	Nombre	Descripción
802.11a	WiFi 5	Admite un ancho de banda superior, el rendimiento total máximo es de 54 Mbps. Provee ocho canales de radio en la banda de frecuencia de 5GHz
802.11b	WiFi	Es el más utilizado actualmente. Ofrece un rendimiento total máximo de 11Mbps y tiene un alcance hasta de 300 metros en espacio abierto. Usa un rango de frecuencia de 2.4GHz con tres canales de radio.
802.11c		Es un estándar combinado, el cual es una versión modificada del estándar 802.1d que permite conectarlo

		con el protocolo 802.11
802.11d	Internacional	Es un complemento pensado para permitir información a diferentes rangos de frecuencia según el país de origen del dispositivo.
802.11e		Está destinado a mejorar la calidad del servicio en la capa de enlace de datos, define los requisitos de diferentes paquetes en cuanto al ancho de banda y al retardo de transmisión.
802.11f	Itinerancia	Es una recomendación para proveedores de puntos de acceso para mejorar la compatibilidad.
802.11g		Tiene un rendimiento total de 54Mbps en el rango de frecuencia de 2.4GHz. Es compatible con el estándar 802.11b
802.11h		Tiene por objetivo unir el estándar 802.11 con el estándar europeo HiperLAN 2
802.11i		Está destinado a mejorar la seguridad en la transferencia de datos implementando el cifrado.
802.11lr		Se elaboró para el uso de señales infrarrojas (obsoleto)
802.11j		Regulación japonesa para el uso de frecuencias y rendimiento energético de Japón.

Tabla 2. 2 Protocolos IEEE 802.11

3. IMPLEMENTACIÓN DEL SISTEMA

3.1 SISTEMA DE VISIÓN

Para las pruebas publicadas en esta tesis se ha hecho uso de una cámara Microsoft LifeCam Studio, la cual cuenta con las siguientes características.

- Sensor HD 1080p, lo que da una nitidez y calidad de imagen superior a la mayoría de cámaras web.
- Tecnología TrueColor, la cual controla de forma automática la exposición de colores y luz.
- La principal ventaja de la cámara es que su lente no tiene distorsión en los bordes por lo cual el proceso de calibración y distorsión se omite en esta tesis.

3.1.2 Reactivision-Simulink

Fig. 3. 1 Processing como middlework entre Reactivision y Matlab

Se requiere que Simulink sea el software que reciba los datos de Reactivision, sin embargo no existen librerías para dicho software, por esta razón ha sido necesaria la creación de un Middle Work, es decir un software que pueda recibir la información de Reactivision y traducirla en paquetes que Matlab pueda interpretar.

Processing es un lenguaje de programación basado en Java orientado especialmente a proyectos multimedia, es de código abierto y multiplataforma. Se ha elegido este Software para obtener los datos que Reactivision procese debido a su simplicidad para trabajar con el protocolo OSC así como la herencia al importar las librerías y funciones de Java ya existentes.

La librería que ofrece Reactivision para Processing nos permiten iniciar sesiones así como obtener el atributo de cada Fiducial que Reactivision detecta, es decir su posición en el plano, su orientación, velocidad y aceleraciones.

Se hace uso del protocolo UDP para comunicar Processing con Simulink. El algoritmo utilizado para dicha comunicación es el siguiente:

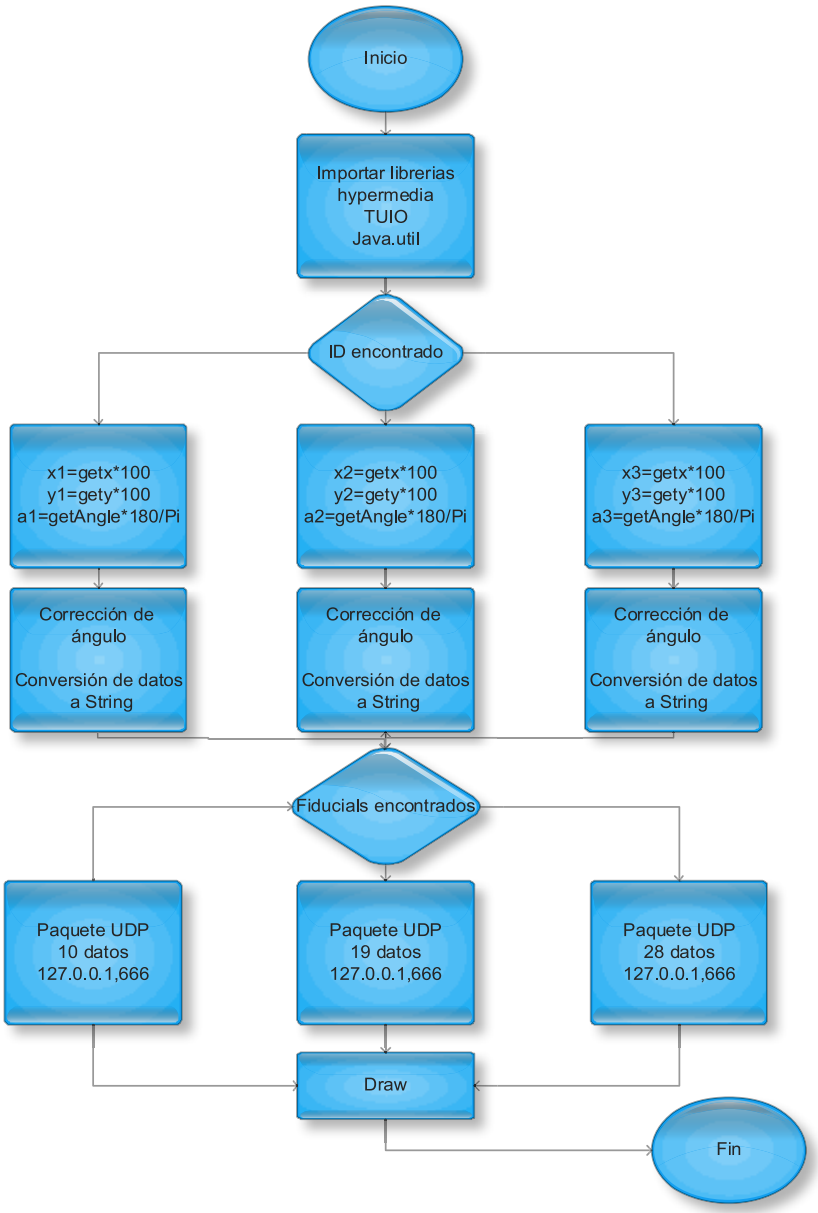


Fig. 3. 2 Diagrama de flujo Processing

En el apéndice sección 6.1, se muestra el código completo del programa.

El software realizado en Processing sólo funciona para tres Fiducials; los que corresponden al ID, uno, dos y tres. Y dependiendo el número de objetos que encuentre Reactivision, será el tamaño del paquete enviado a Simulink.

El programa comienza importando las librerías de Reactivision llamada TUIO, también es necesario importar la librería hypermedia, la cual permite la comunicación por medio del protocolo UDP y la creación de sockets, y la librería java.util, la cual nos permite utilizar las funciones básicas de Java en Processing.

Posteriormente se crea un objeto UDP, y por medio de los métodos importados en la librería TUIO, se desempaquetan los datos mandados por Reactivision.

Dependiendo el ID del Fiducial que encuentre, crea las variables asociadas a la posición X, Y y a su orientación, es decir el ángulo, debido a que la posición y el ángulo son obtenidas como tipo flotante, es necesario convertirlas a entero, y después a una cadena, se aprovecha este paso para hacer una corrección en el ángulo de Reactivision y convertirlo de radianes a grados.

Simulink tiene un proceso primitivo para desempaquetar mensajes UDP, por lo cual ha sido necesario implementar un control de errores en la comunicación de estos dos programas.

Dependiendo el número de ID que encuentre Reactivision, será el tamaño del paquete mandado a Simulink.

En el caso en el que sólo se encuentre un ID, Processing mandará la siguiente cadena:

"M+x1+y1+a1"

Donde "M" es un identificador o cabecera con el fin de que Simulink conozca donde comienza cada paquete.

x1, y1 y a1, son números enteros que varían entre 0 y 360, debido a que el paquete que se manda a través de la red, debe ser una cadena de caracteres y como consecuencia el tamaño del paquete puede variar, por ejemplo

x1=10

y1=100

a1=29

La cadena corresponderá a:

"M1010029"

Para evitar estos errores se ha creado un sencillo algoritmo con el cual el paquete siempre tendrá el mismo número de datos, teniendo como resultado:

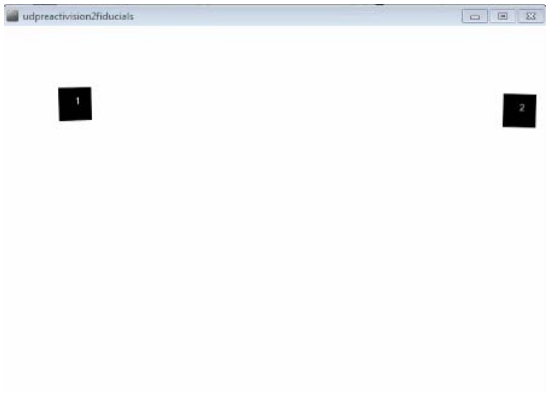
x1=010

y1=100

a1=029

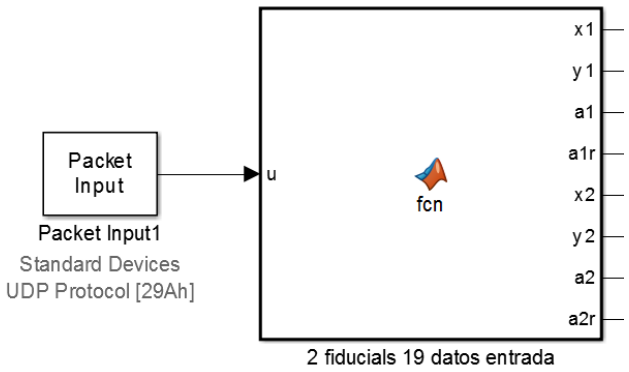
"M010100029"

Todos los paquetes mandados para un solo Fiducial, tendrán una longitud de 10 datos, para dos Fiducials, se mandarán 19 datos y 28 para 3 Fiducials.



Los datos apuntan a la dirección del local host: 127.0.0.1 a través del puerto 666.

El programa además aprovecha las variables de la posición y orientación para dibujar en una ventana cada Fiducial correspondiendo a lo que encuentre Reactivision con el fin de verificar el correcto funcionamiento del programa.



Para desempaquetar con Simulink los datos que manda Processing, es necesario abrir un socket UDP por medio de las librerías que incluye el software, y para interpretar la información recibida se han creado tres bloques por medio de funciones de Matlab, dependiendo el número de Fiducials con los que se esté trabajando.

Una función de Matlab se puede interpretar como un método el cual necesita un número determinado de entradas y salidas, se programa en el lenguaje de programación de Matlab y a diferencia de las funciones .m, se guarda en el modelo de Simulink.

$$e(n) = [0_1, 0_2, 0_3, 0_4, 0_5, \ldots, 0_n]$$

El algoritmo crea un vector de ceros de n elementos, donde n varía de 9, 18 o 27 elementos dependiendo del número de Fiducials de los cuales se desea adquirir su posición y orientación.

Después Simulink recibirá a través del protocolo UDP una cadena de caracteres de n+1 elementos, por ejemplo suponiendo el caso intermedio donde n = 18, la cadena que manda Processing es:

x1=010

y1=100

a1=029

x2=030

y1=120

a1=056

"M010100029030120056"

Como Simulink no cuenta con un control de flujo para la comunicación, se ha implementado un algoritmo de búsqueda.

Si la información llega desfasada a Matlab:

"9030120056M01010002"

El algoritmo busca la posición donde se encuentra el valor 77 que en ASCII corresponde a la letra "M", una vez encontrada la posición, divide la cadena en dos partes.

"9030120056M01010002"

La cadena del lado derecho se va acomodando en orden en el vector de ceros inicializado anteriormente, para posteriormente colocar la cadena de lado derecho.

"010100029030120056"

Cada elemento de la cadena sigue siendo un carácter, no el valor numérico que representa, por esta razón el algoritmo hace la conversión de ASCII a entero de la siguiente manera.

$$\boldsymbol{X1 = (e(1) - 48) * 100 + (e(2) - 48) * 10) + e(3) - 48}$$

$$\boldsymbol{Y1 = (e(4) - 48) * 100 + (e(5) - 48) * 10) + e(6) - 48}$$

$$\boldsymbol{a1 = (e(7) - 48) * 100 + (e(8) - 48) * 10) + e(9) - 48}$$

$$\boldsymbol{X2 = (e(10) - 48) * 100 + (e(11) - 48) * 10) + e(12) - 48}$$

$$\boldsymbol{Y2 = (e(13) - 48) * 100 + (e(14) - 48) * 10) + e(15) - 48}$$

$$\boldsymbol{a2 = (e(16) - 48) * 100 + (e(17) - 48) * 10) + e(18) - 48}$$

El código completo se encuentra en la sección 6.2 del apéndice.

Con este algoritmo se implementa una tolerancia a fallos en la comunicación del protocolo UDP de Processing y Simulink

3.2 Procesamiento

3.2.1 Campos Potenciales

Conociendo los datos de la posición inicial y final del robot así como la de cada uno de los obstáculos. Es posible implementar el algoritmo de campos potenciales. En [18] propone que el problema de seguir la trayectoria puede ser resuelto con las siguientes reglas:

$$V = \begin{cases} V_{max} & si\ d_g > k_r \\ \\ \frac{V_{max}}{k_r} * d_g & si\ d_g \leq k_r \end{cases} \qquad (3.1)$$

Donde d_g es la distancia a la meta y $k_r$es la constante definida para campos atractivos, por otro lado la velocidad angular se plantea como función del ángulo de error, el cual es el resultado de la resta entre el ángulo de la fuerza resultante y el ángulo actual del robot.

$$\theta_e = \theta_{campo\ potencial} - \theta_{robot} \qquad (3.2)$$

La velocidad angular queda determinada por:

$$\omega = \omega_{max} Sen(\theta_e) \qquad (3.3)$$

Con estas ecuaciones se espera que el robot se dirija a la posición final a su máxima velocidad lineal cuando se encuentre lejos de la meta y reduzca su velocidad al aproximarse a dicho punto, mientras que la velocidad angular varía en función del ángulo de error determinado por el método de los campos potenciales artificiales.

3.2.2 Modelos matemáticos del sistema

Para validar la teoría expuesta en esta tesis se ha utilizado un robot del tipo (2,0), el cual está totalmente descrito en [18].

Para poder modelar el robot diferencial es necesario hacer las siguientes suposiciones:

- El robot móvil es rígido y no deformable.
- Las llantas no se deslizan.
- El robot se mueve sobre una superficie horizontal.

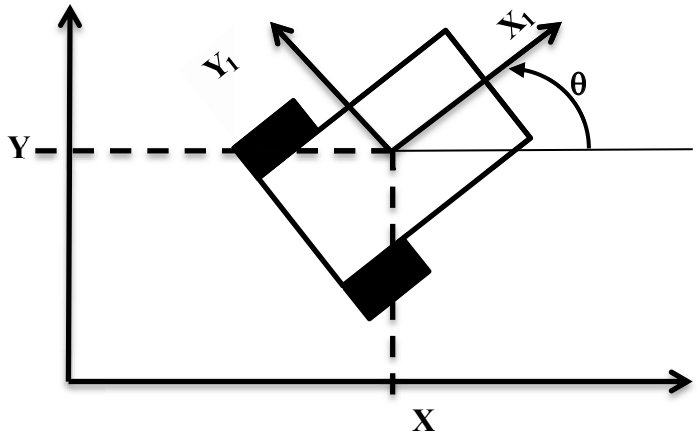


Fig. 3. 3 Postura de Robot Diferencial

Se hace uso de dos sistemas de referencia, siendo el primero un sistema inercial, el cual es la base sobre el cuál se mueve el robot, y está denominado como (X, Y), y el segundo está referido a la posición del robot, denominado como (X_1, Y_1).

La postura del robot se representa por:

$$\xi = [x, y, \theta]' \qquad (3.4)$$

Donde X y Y son la posición en el sistema de referencia inercial y θ es la orientación sobre el mismo sistema. La matriz ortonormal de rotación para cambiar del sistema inercial, al sistema de rotación se define como:

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (3.5)$$

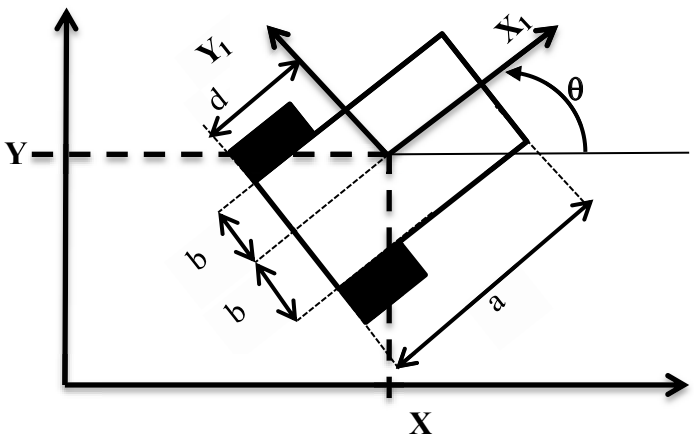


Fig. 3. 4 Cinemática Robot Móvil

Esta matriz nos permite también trasladar las velocidades de un sistema a otro, el modelo del robot móvil diferencial representado en variables de estado es:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}\cos\theta & \mathrm{d} * \sin\theta\\ -\sin\theta & -d * \cos\theta\\ 0 & 1\end{bmatrix}\begin{bmatrix}v\\ w\end{bmatrix} \qquad (3.6)$$

De acuerdo con el trabajo de González Villela, el número de coordenadas de configuración $q = [q_1, \dots, q_{n_T}]'$ que representa un robot móvil esta dado por la ecuación:

$$n_T = 3 + N_T + N_C + N_{OC} \qquad (3.7)$$

En donde:

N_T Es el número total de ruedas.

N_C Es el número de ruedas centradas.

N_{OC} Es el número de ruedas descentradas.

Estas coordenadas generalizadas se encuentran distribuidas entre las coordenadas de postura, los ángulos de las ruedas centradas, los ángulos de las ruedas descentradas y de rotación de las ruedas. De acuerdo a esto, las coordenadas generalizadas del robot son:

$$q = [x \;\; y \;\; \theta \;\; \phi_r \;\; \phi_l] \qquad (3.8)$$

Donde X, Y son la posición y θ la orientación en el sistema de referencia del robot. ϕ_r, ϕ_l son los ángulos de rotación de las llantas izquierda y derecha respectivamente alrededor de su eje.

Haciendo el desarrollo matemático necesario se puede obtener la representación en variables de estado del modelo cinemático, obteniendo así:

$$\dot{q} = \begin{bmatrix}\cos\theta & d * \sin\theta\\ \sin\theta & -d * \cos\theta\\ 0 & 1\\ \frac{1}{r} & \frac{b}{r}\\ \frac{1}{r} & -\frac{b}{r}\end{bmatrix}\begin{bmatrix}v\\ w\end{bmatrix} \qquad (3.9)$$

En donde el vector de velocidades generalizadas $\dot{q}$ esta definido para el caso de un robot diferencial tipo (2,0) es:

$$\dot{q} = [\dot{x} \;\; \dot{y} \;\; \dot{\theta} \;\; \dot{\phi}_r \;\; \dot{\phi}_l]' \qquad (3.10)$$

El desarrollo matemático completo se puede encontrar en [18]

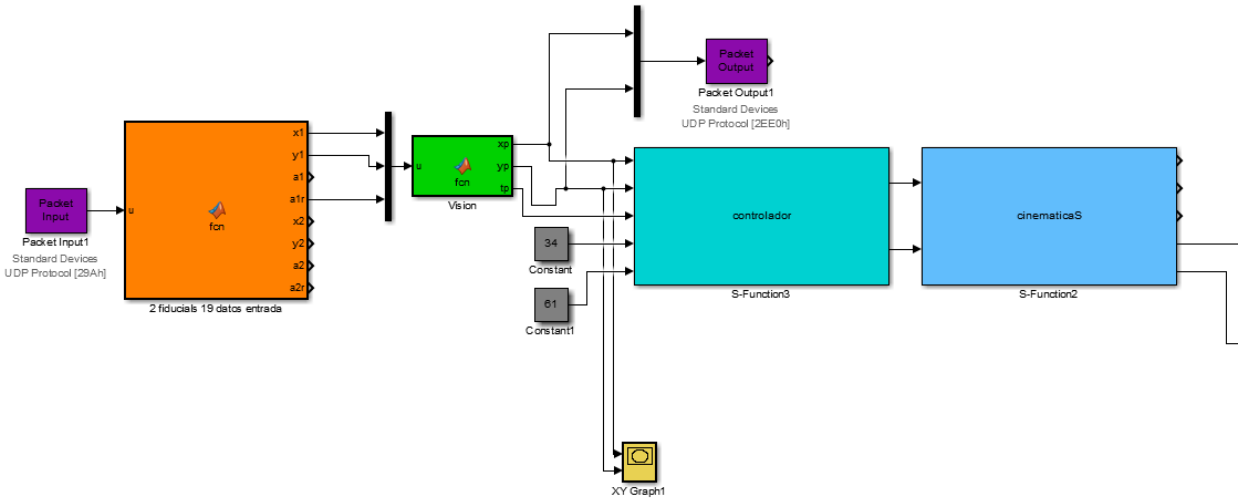


Fig. 3. 5 Funcion Controlador y Cinemática en Simulink

Para implementar estos dos algoritmos en Simulink se han creado dos funciones S, la función "controlador" que contiene el algoritmo de los campos potenciales, y la función "cinemática" la cual contiene la cinemática del robot diferencial.

La función controlador necesita la posición y orientación del robot obtenida por el sistema de visión, así como la posición de la meta, la salida de la función son las velocidades lineales en X,Y del robot, por ello es necesaria la cinemática del robot, la función cinemática transforma las velocidades lineales en velocidades angulares para cada llanta.

Estas dos funciones se pueden encontrar completas en el apéndice 6.3 y 6.4 respectivamente.

3.2.3 Real Time Windows Target

```
>> rtwintgt -setup
```

Fig. 3. 6 Instrucción para instalar Real Time Windows Target

Para que Simulink trabaje en tiempo real, primero es necesario instalar el módulo RTWT (Real Time Windows Target), en la ventana de comandos se teclea la instrucción de la figura 3.6.

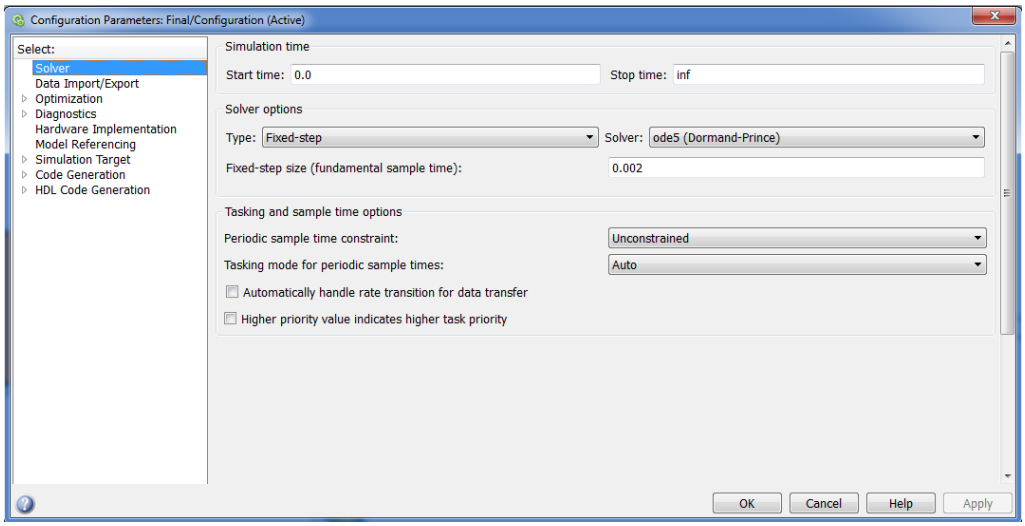


Fig. 3. 7 Parámetros de Configuración para Tiempo Real

Es necesario tener un compilador de C para que Matlab pueda compilar el código del modelo matemático. En los parámetros de configuración de Simulink, deben elegirse métodos de solución de paso fijo. Es importante mencionar que el tiempo de parada, se configura como "inf" para establecer que no parará de ejecutar el código hasta que el operador lo indique mediante la interfaz de Simulink.

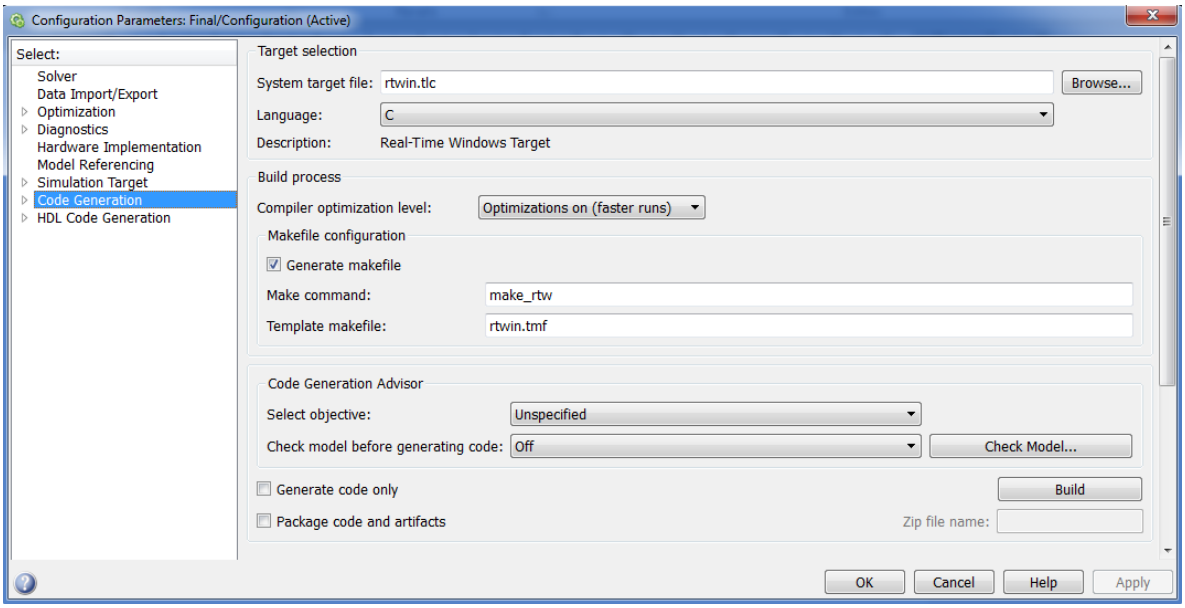


Fig. 3. 8 Generación de Código en C

En la mayoría de modelos Simulink trabaja interpretando la información del modelo para dar una simulación rápida, es decir, si se requiere simular el modelo dinámica de un sistema Masa – resorte –amortiguador, basta con programar el modelo y ejecutarlo, en unos pocos segundos de procesamiento, nosotros sabremos cómo se comportará el modelo en una escala de tiempo mucho mayor a la que se está corriendo el proceso, la naturaleza del Software es de simulación, sin embargo para poder hacer la comunicación con el mundo exterior, es necesario cambiar esta forma de funcionamiento en Matlab, en los parámetros de configuración, en la pestaña de generación de código, podemos elegir el tipo de compilador a utilizar para el código, para trabajar en tiempo real es necesario elegir la opción rtwin.tlc en lenguaje C.

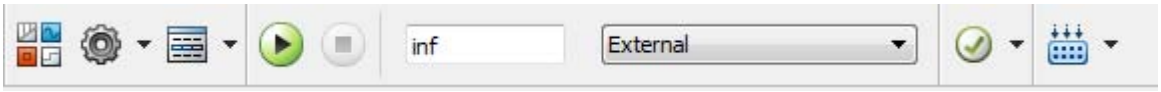


Fig. 3. 9 Tipo de Simulación

Por último al momento de ejecutar el programa es necesario configurar el tipo de simulación que se requiere utilizar, para Tiempo Real se debe fijar en "External".

En la mayoría de las ocasiones, es necesario contar con una tarjeta de adquisición de datos para comunicar Simulink con el sistema físico. El protocolo de comunicación que se realiza varía dependiendo del tipo de tarjeta, puede ser por medio de drivers USB, comunicación serial, puertos paralelos, o mediante protocolos de red.

Aprovechando que existe la comunicación por medio del protocolo UDP en Simulink, y que no son necesarios drivers para que soporte la comunicación, se ha implementado esta comunicación para comunicarse con el robot.

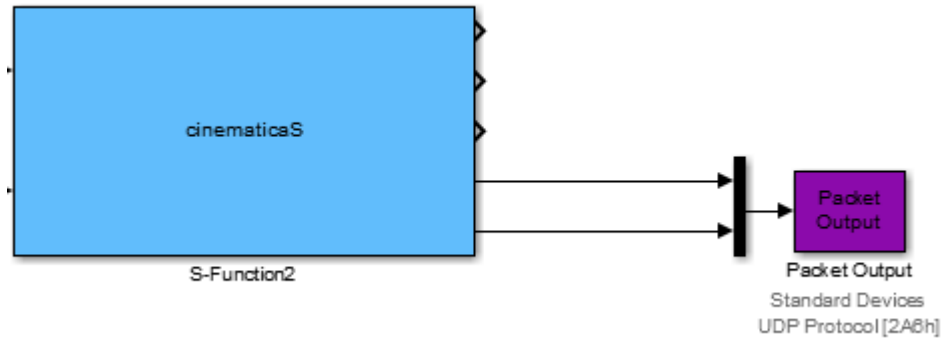


Fig. 3. 10 Las velocidades de cada llanta son enviadas por el protocolo UDP

De la función cinemática, se han obtenido las velocidades de cada llanta, esta información necesita ser empaquetada para mandarse a través del protocolo UDP.

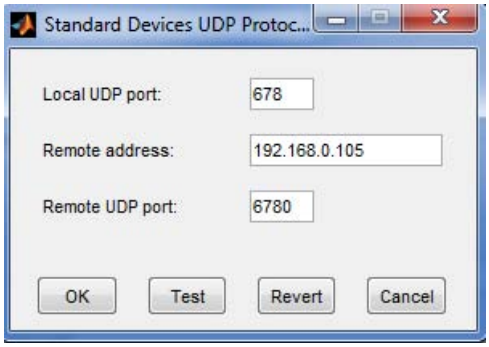


Fig. 3. 11 Parámetros para la creación del Socket

Primero es necesario configurar el socket para lograr la comunicación UDP, seleccionar la dirección IP del robot y el puerto al cual queremos conectar.

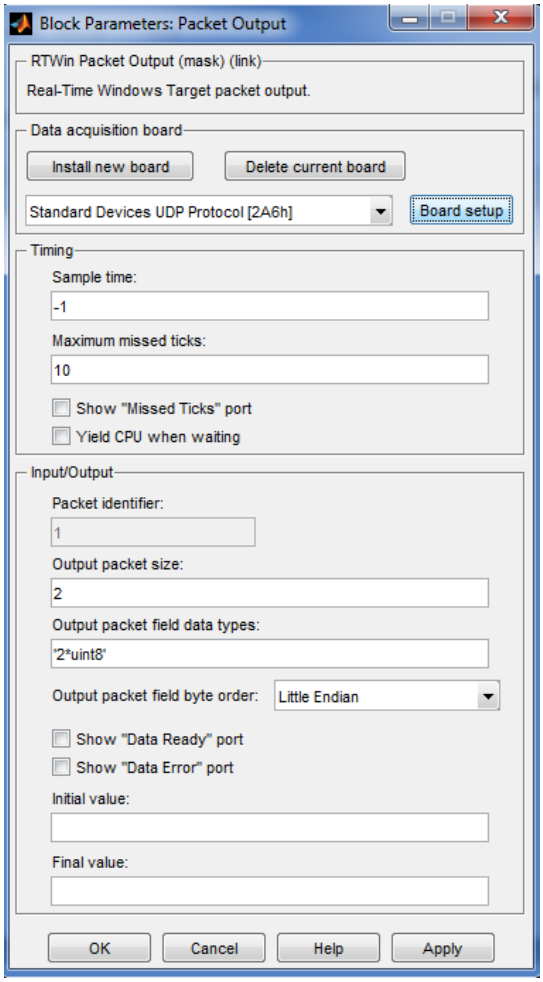


Fig. 3. 12 Empaquetamiento de Datos a través de Simulink

El empaquetado lo hace Matlab, dependiendo el tipo de dato y número de variables que se envíen por el socket.

3.2.4 Robot Diferencial

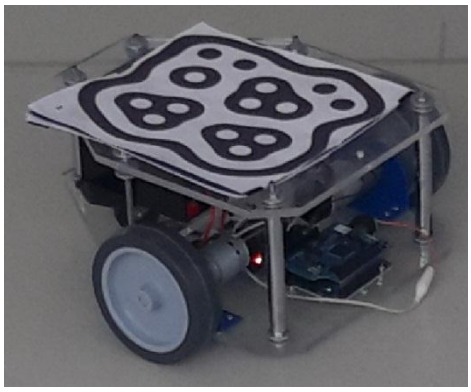


Fig. 3. 13 Robot Diferencial del grupo MRG

El robot diferencial está integrado por una placa Arduino, una placa WiFi Shield, y una tarjeta MD25.

El WiFi Shield es una tarjeta que permite la comunicación WiFi entre el microcontrolador Arduino y la red inalámbrica, se basa en el estándar IEEE 802.11b/g y soporta las comunicaciones bajo los protocolos UDP y TCP/IP, así como la creación de servidores web.

La tarjeta MD25, es un circuito electrónico que a través del protocolo I2C se conecta con el microcontrolador Arduino, el fin de la tarjeta MD25 es servir como etapa de potencia para los motores del robot, y controlar sus velocidades a través de encoders.

El WiFi Shield sirve como comunicación entre el robot y la computadora, sin embargo para lograrla sería necesaria una red AD-HOC la cual no es conveniente cuando se requieran conectar más robots. Por lo tanto es necesaria la implementación de un Router.

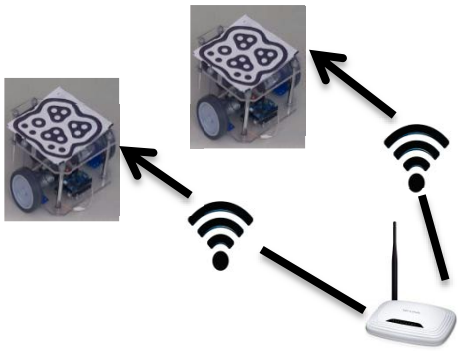


Fig. 3. 14 Red Tipo Estrella para la comunicación WiFi

El Router es un dispositivo físico que opera en la capa tres del modelo OSI, su función en esta tesis es centralizar la comunicación y ayudar a dirigir los paquetes dependiendo su destino, así como aprovechar el servicio DHCP, que por sus siglas en inglés corresponden a Dynamic Host Configuration Protocol, o Protocolo de Configuración Dinámica de Servidor, el cual permite que uno o más dispositivos obtengan su configuración de red de manera automática, es decir cualquier dispositivo o en este caso cualquier robot que se conecte a la red creada por este Router, obtendrá la dirección IP, mascara de subred y puerta de enlace de manera automática.

El algoritmo del microcontrolador del robot funciona de la siguiente manera:

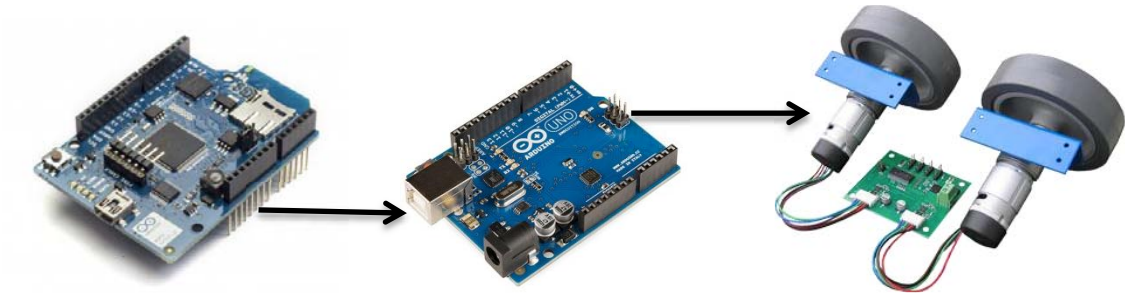


Fig. 3. 15 el Shield recibe los datos, los manda a Arduino y este a su vez a la tarjeta MD25

El Shield WiFi está programado para conectarse a la red creada por el router, para esto es necesario que el microcontrolador conozca, el BSSID (Basic Service Set Identifier), o nombre de la red, el tipo de cifrado que en este caso es WPA2, y la contraseña de la red.

El microcontrolador ha sido programado con la función de servidor, por lo cual presta sus servicios al cliente que en este caso es Simulink, este servidor crea un socket UDP con un puerto local, una vez creado este socket, recibe información por medio de la técnica del poleo, en la cual pregunta una y otra vez si existen datos en el buffer del protocolo, cuando haya datos, los transforma al tipo de dato int, estos datos recibidos son las velocidades de cada llanta, por medio del protocolo I2C manda estas velocidades a la tarjeta MD25, la cual tiene un microcontrolador programado para controlar las velocidades del robot por medio de los encoders que se sitúan en cada llanta.

De esta forma el robot cumplirá con las velocidades calculadas por el algoritmo de campos potenciales artificiales y la cinemática del robot. Para más robots, el router se encargará de enumerarlos y dar una dirección IP a cada uno, por lo tanto el algoritmo se repite para cada uno de estos, formando así una red estrella, si alguno de los robots se llegase a desconectar, los demás robots podrán seguir funcionando.

La cantidad de datos generados cada segundo con este sistema es de 500 datos por variable, el hecho de que Simulink trabaje en tiempo real no permite guardar el historial de la variable de manera completa, por lo tanto, aprovechando que se ha implementado la

comunicación UDP en el modelo, se hace uso del mismo protocolo para comunicarlo con algún otro programa. Por su simplicidad se ha elegido Python para programar este algoritmo, el código completo se puede ver en el apéndice 6.6. Básicamente, el algoritmo es capaz de detectar el número de variables que están entrando a través del puerto, debe especificarse el tipo de dato para que pueda desempaquetar cada datagrama, una vez que la cadena ha sido interpretada por el programa, crea un archivo de Excel con extensión .CSV con el fin de importar los datos a Matlab con mayor facilidad y guarda la información enviada por Simulink.

4. PRUEBAS Y RESULTADOS

Para validar los experimentos realizados con este banco de pruebas, es necesario compararlo con la tecnología utilizada anteriormente en el grupo MRG y así poder analizar los resultados.

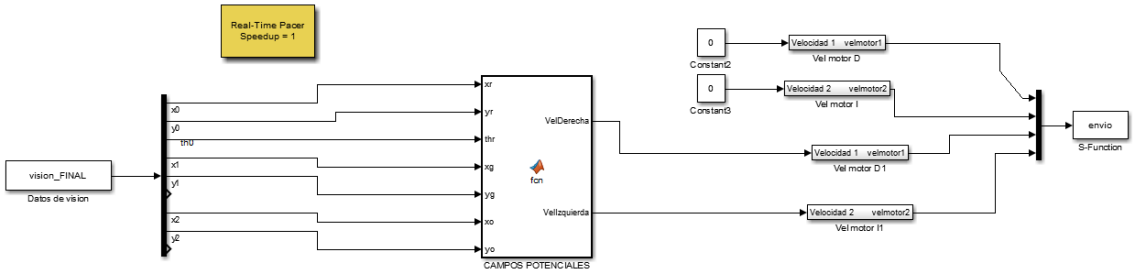


Fig. 4. 1 Sistema propuesto en [8]

En [8] se hace una explicación detallada del funcionamiento de dicho banco de pruebas, en esta tesis solo se hará referencia al procesamiento y comunicación.

Como primer punto, se utiliza una función S, para comunicar de manera directa Reactivision con Simulink, la página de Reactivision otorga las instrucciones para esta comunicación, sin embargo Simulink y en lo particular Matlab, es un programa basado en el lenguaje C, importar una librería nativa de Java reduce de manera drástica la velocidad de adquisición del sistema de visión. El procesamiento se ejecuta en el modo simulación, lo que resta velocidad al momento de las comunicaciones, se utiliza un Real-Time Pacer, este bloque crea un retardo en la simulación para que se sincronice con el tiempo externo, tiene un rango de error de entre 10 a 30 milisegundos y es necesaria una computadora con más de un procesador. Por último la comunicación se realiza utilizando módulos Xbee los cuales hacen una conversión de comunicación serial a comunicación de radiofrecuencia, la gran desventaja es el tipo de arquitectura para mandar la información, el primer robot recibe la cadena de datos completa, la divide y manda al segundo robot parte de la cadena, este a su vez la divide y pasa el resto al tercer robot y así de manera consecutiva, si un robot llegase a fallar, la red completa falla. Como punto importante cabe mencionar que Xbee usa un dispositivo que emula la comunicación serial dentro de la computadora, por lo tanto la velocidad entre la computadora y el primer robot, está limitada al máximo de velocidad de la comunicación serial (115200 bps máximo). Este conjunto de combinaciones limitaban al robot a una velocidad lineal de .2 m/s.

4.1 Prueba 1 Robot diferencial a un punto fijo.

Como primera prueba se ha decidido utilizar un robot diferencial con el sistema descrito anteriormente. El objetivo es iniciar el robot en una posición y orientación conocida, y llevarlo a un punto aleatorio por medio de los campos artificiales, en este caso el punto se definió en x=34, y=61.

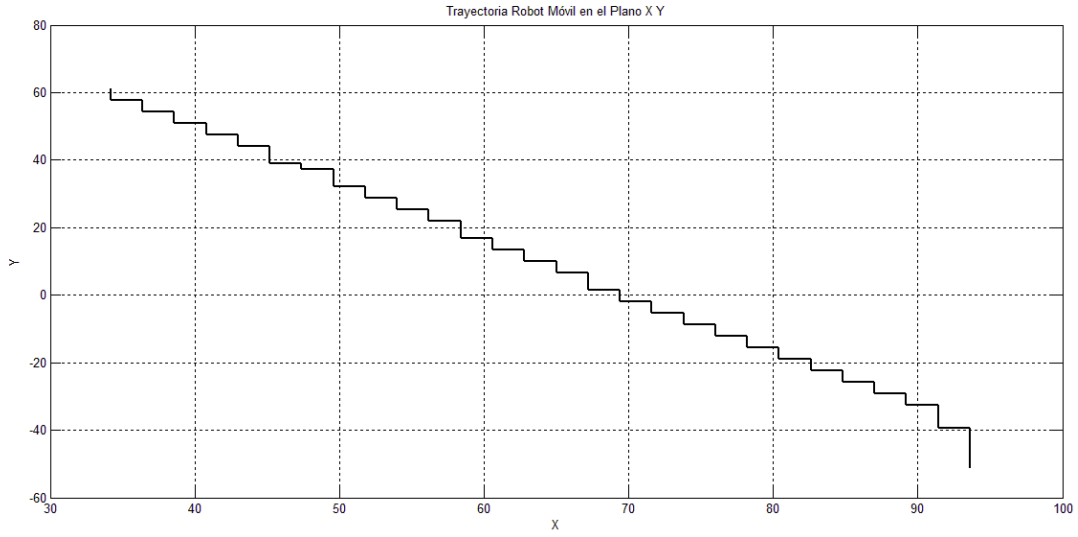


Fig. 4. 2 Respuesta del sistema propuesto en [8]

La figura muestra la trayectoria generada por el robot, el tiempo total que el robot tardó en ejecutar esta prueba fue de 37.918 segundos, con una velocidad lineal máxima de .2 m/s, la cual es la velocidad máxima descrita en el sistema según [19].

Los resultados que se exponen a continuación han sido desarrollados con el banco de pruebas propuesto en la tesis. Como segunda prueba, se ha aumentado la velocidad lineal a 50m/s.

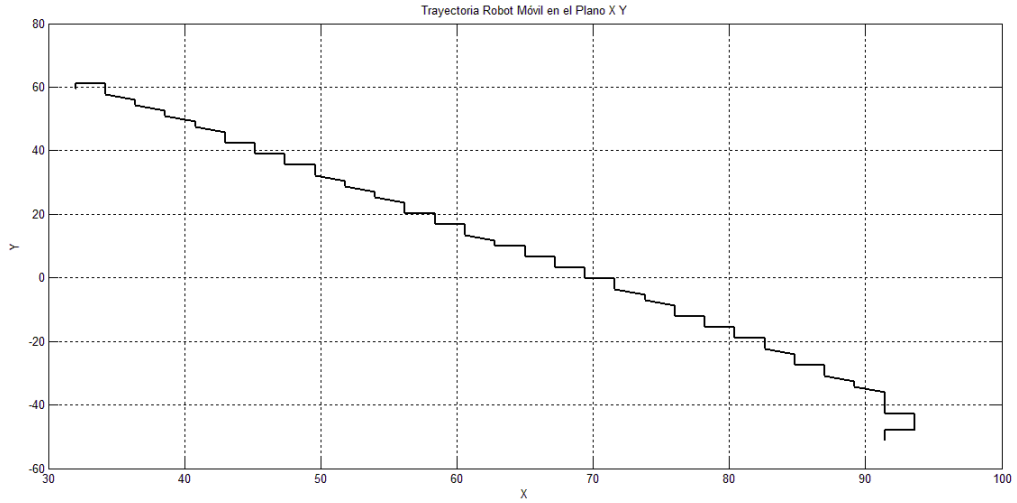


Fig. 4. 3 Respuesta del Sistema a 50cm/s

El tiempo que tardó en ejecutar esta trayectoria fue de 13.506 segundos. En la gráfica también es posible observar que el patrón de pequeños escalones mostrados en la figura, van cambiando hacía una línea recta.

Se realizaron diferentes pruebas a diferentes velocidades, sabiendo que la tasa de muestreo es de 0.002 segundos y con el número total de datos de cada prueba, se puede estimar el tiempo que el robot tarda en realizar la trayectoria.

	Vmax [m/s]	Tiempo [s]	Número de Datos
Antes	.2	37.918	
Ahora	.5	13.506	6753
	1.0	5.152	2576
	1.5	3.688	1844
	2.0	2.926	1463
	3.0	2.196	1098
	5.0	2.178	1089

Tabla 4. 1 Datos obtenidos a diferentes velocidades.

En la tabla se registra la velocidad máxima programada en el algoritmo de campos potenciales y el tiempo que tardó el robot en realizar la trayectoria.

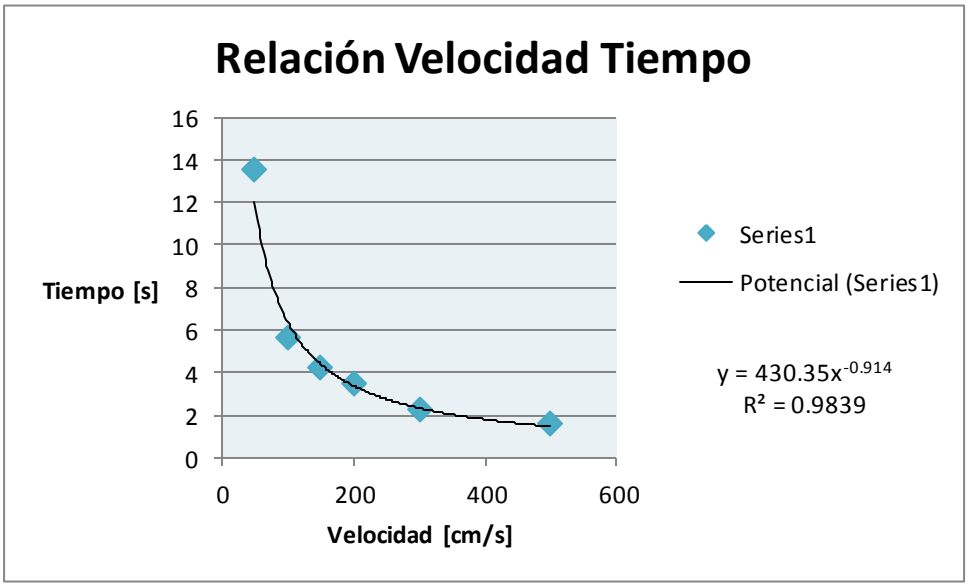


Fig. 4. 4 Velocidades vs Tiempo

Estos datos se trazan en una gráfica, para obtener su línea de tendencia, con lo que se puede observar que aunque la velocidad siga aumentando, el tiempo será aproximadamente el mismo, por lo tanto la velocidad máxima del robot se fija en 5 m/s

4.2 Prueba 2 Robot diferencial siguiendo una trayectoria.

Para validar el procesamiento de los campos potenciales, se ha propuesto que el robot diferencial, siga la trayectoria de un círculo, para esto se han utilizado dos funciones Seno desfasadas noventa grados.

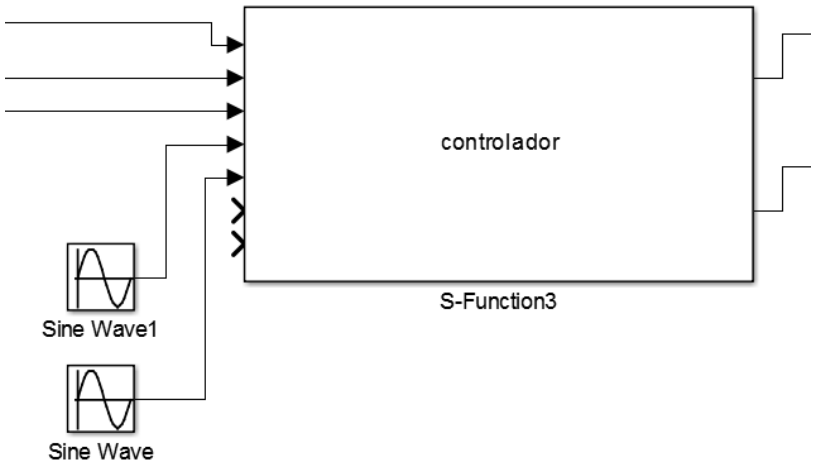


Fig. 4. 5 Dos funciones Seno desfasadas, formarán la trayectoria Circular

Estas funciones irán avanzando respecto al tiempo y dictarán los puntos X Y a los cuales el robot deberá llegar. Los datos dados por el sistema de visión, los cuales son la posición y orientación del robot son guardados junto a los datos de las funciones seno.

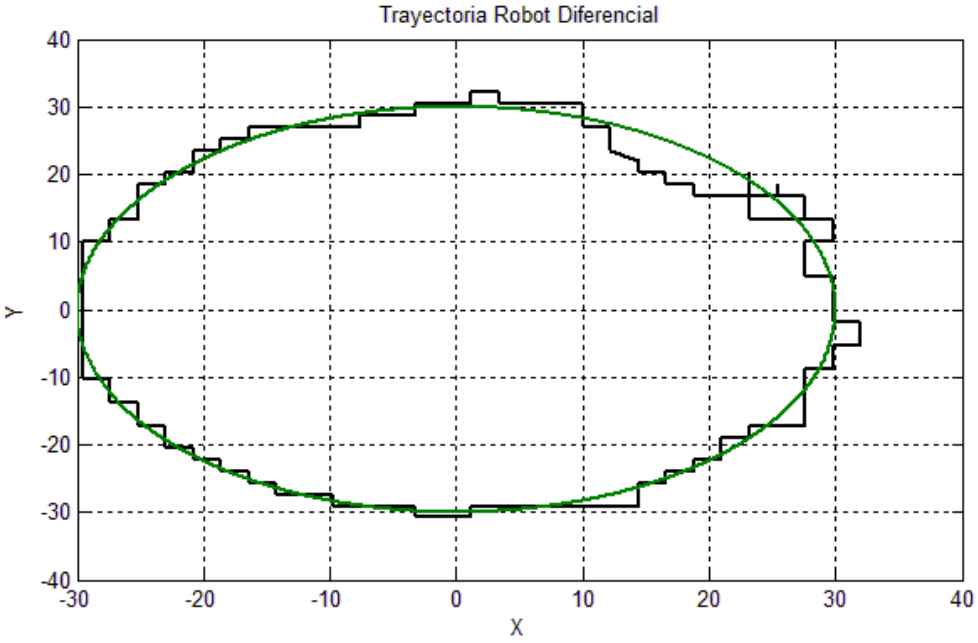


Fig. 4. 6 Trayectoria Robot (Negro) y Deseada (verde)

Acorde a la gráfica, el robot cumplió con la trayectoria deseada, se pueden apreciar ciertos errores, estos son debidos a que el robot diferencial es un robot no holonómico, es decir de los tres grados de libertad disponibles en el plano, son solo controlables dos, dadas estas restricciones el robot debe trazar trayectorias, similares para llegar al punto dictado por el campo potencial.

Para analizar el error en la trayectoria dada por las funciones Seno y la trayectoria seguida por el robot, se ha decidido hacer un análisis por separada en cada uno de los ejes X Y.

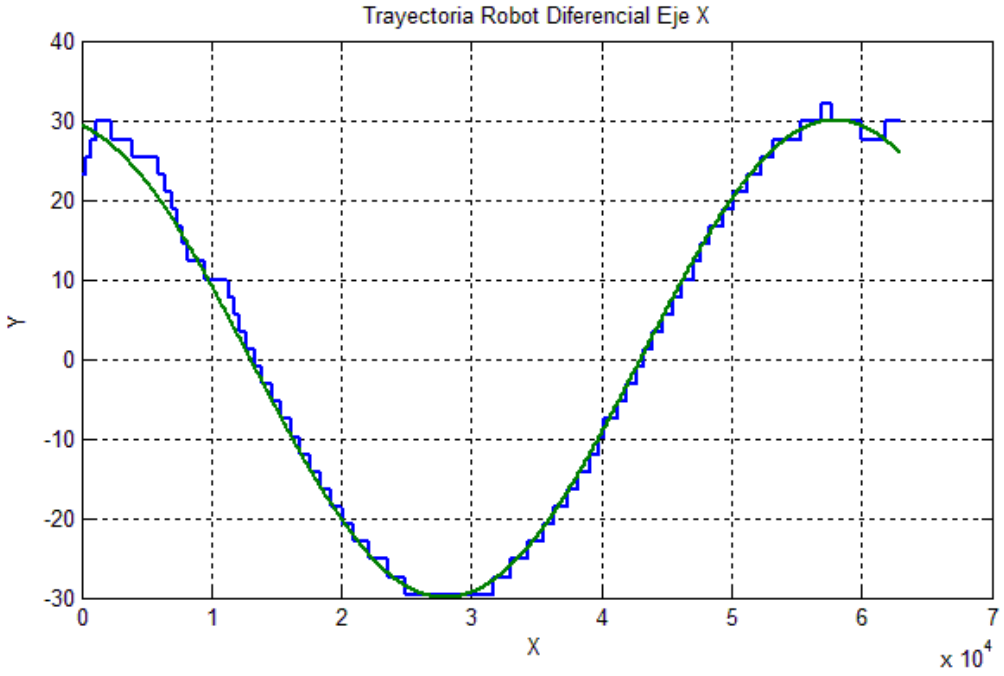


Fig. 4. 7 Trayectoria Robot Diferencial Eje X

En la gráfica se puede observar de color verde, la trayectoria a seguir por el robot, mientras que el trazo de color azul, indica el movimiento realizado por el robot. El mayor error presentado en el eje x se encuentra al principio de la trayectoria con un valor de 6.13 cm

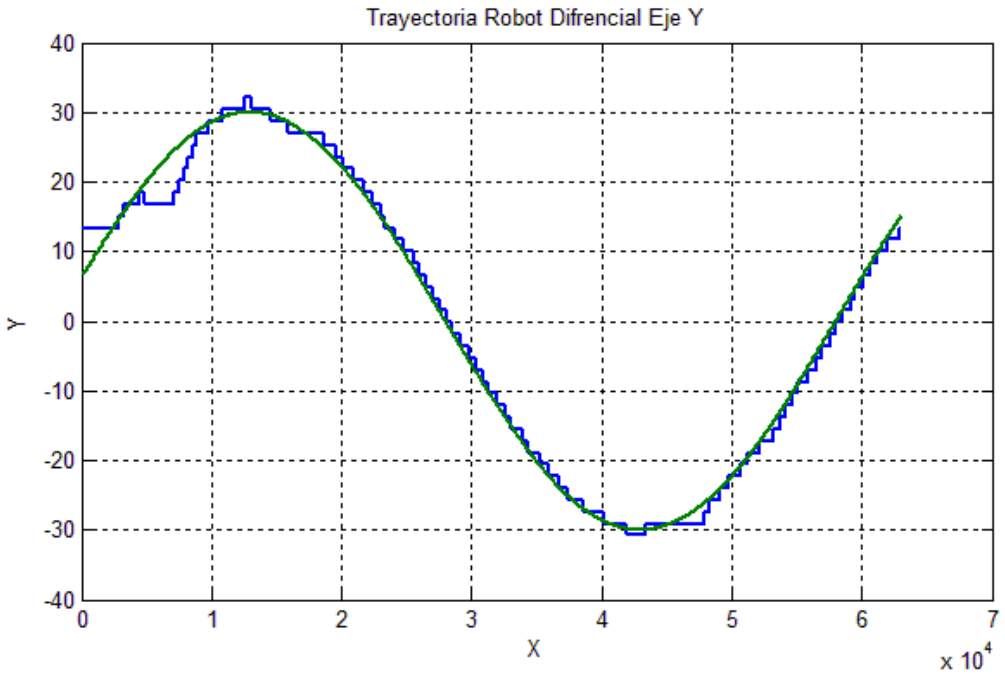


Fig. 4. 8 Trayectoria Robot Diferencial Eje Y

De manera similar el comportamiento en el eje Y, de color verde se presenta la trayectoria deseada, y de color azul la trayectoria real que realiza el robot, el error más grande ocurre al comienzo de la trayectoria con una magnitud máxima de 7.5 cm.

4.3 Prueba 3 Robot omnidireccional siguiendo una trayectoria.

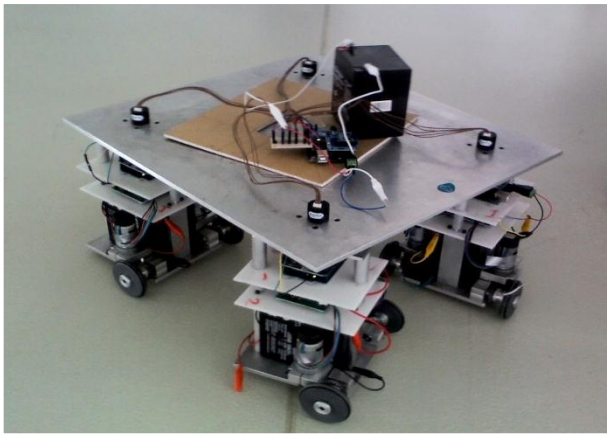


Fig. 4. 9 Robot Omnidireccional Redundante

Para validar la comunicación multi-robot y la arquitectura estrella se ha utilizado un robot omnidireccional conformado por un grupo de cuatro robots diferenciales. En esta tesis sólo se utiliza la comunicación, la cinemática de los robots no es el objetivo del presente trabajo, por lo tanto se omite.

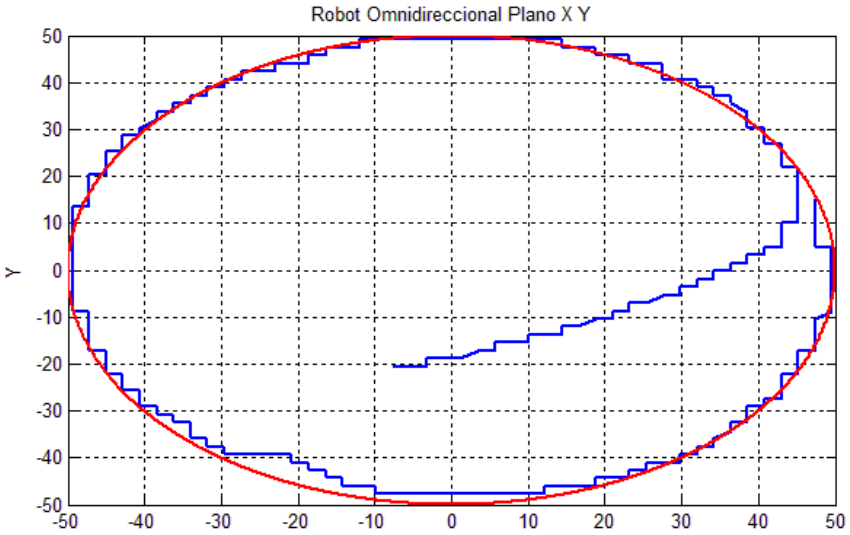


Fig. 4. 10 Azul Trayectoria Robot, Rojo Trayectoria Deseada

Se utilizan las mismas funciones de seno para que el robot siga la trayectoria del círculo.

En la figura se puede observar de color rojo, la trayectoria propuesta por las funciones seno, mientras que de color azul, la trayectoria seguida por el robot.

Para analizar el error en cada punto, se hace un análisis de la posición del robot en cada uno de los ejes.

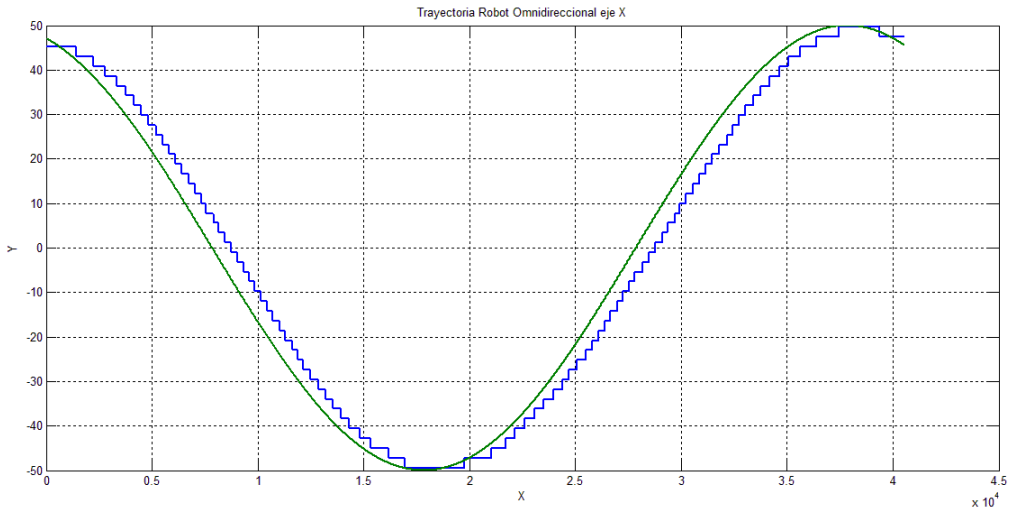


Fig. 4. 11 Trayectoria Eje X

De color verde se representa la trayectoria dictada por el campo potencial, mientras de color azul se muestra la trayectoria real, del robot, la magnitud del mayor error en la trayectoria es de 7.2 cm.

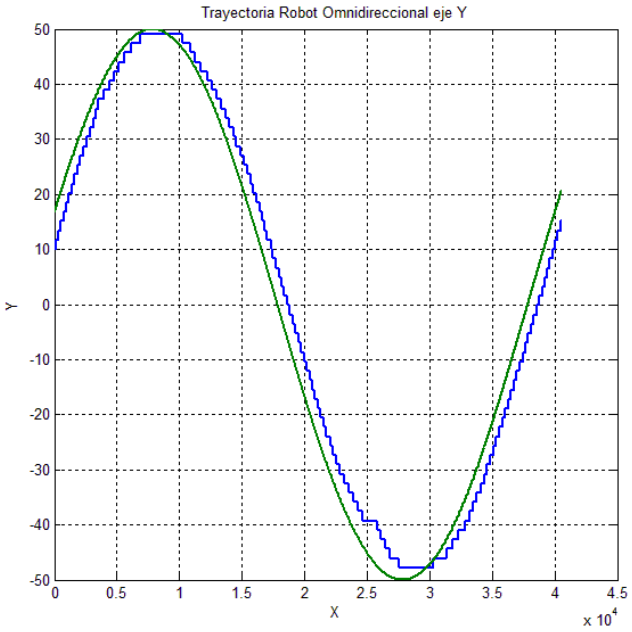


Fig. 4. 12 Trayectoria Eje Y

Los datos azules representan la trayectoria real del robot omnidireccional, los datos verdes representan la trayectoria deseada a través del algoritmo de campos potenciales, el error máximo en este caso es de 6.4 cm.

5. CONCLUSIONES Y TRABAJO A FUTURO

El presente trabajo muestra el desarrollo de un banco de pruebas orientado a la robótica móvil, para obtener la ubicación y orientación, de cualquier robot en el área de visión, así como el manejo de tiempo real en el procesamiento a través de Simulink, y por último la implementación de una red WiFi para la comunicación multi-robot dentro de una red local, en este caso se empleó un robot móvil diferencial para validar el sistema completo, y un grupo de robots móviles, para validar la comunicación multi-robot.

Es evidente, que al finalizar las pruebas, la diferencia de velocidades entre el sistema utilizado antes y el de ahora es drástica, el sistema actual es al menos 15 veces más rápido. El hecho de aumentar la velocidad, da como consecuencias el poder considerar controles dinámicos para el seguimiento de trayectorias. Tener un sistema de procesamiento en tiempo real ayudará a minimizar la tasa de muestreo de cada experimento, lo que optimizará los algoritmos de control.

El algoritmo de visión no fue modificado, sin embargo la creación de un middlework que ayudara la comunicación de Reactivision con Simulink tuvo una mejora importante en la velocidad de procesamiento, esto es debido a que Simulink que es el software que realiza el procesamiento en tiempo real y por lo tanto el que más recursos consume dentro de la computadora, sólo recibe la información de visión, en vez de importar librerías, crear objetos y obtener sus atributos de posición y orientación en cada muestra, como desventaja, el sistema general dependerá de las capacidades de la computadora, es decir tipo del procesador, velocidad del procesador, arquitectura y memoria RAM.

El procesamiento en tiempo real crea un código ejecutable en lenguaje C, para su interpretación por el compilador, esto ayuda a que Matlab modifique el espacio de memoria para los datos y procesos a lo largo de todo el algoritmo, haciendo el procesamiento más rápido, debemos tomar como una pequeña desventaja, que no se puede hacer uso de todos los bloques de Simulink para trabajar en tiempo real, sin embargo, estos pueden ser programados en código de funciones S.

Matlab permite la comunicación en tiempo real con tarjetas de adquisición de datos, se ha utilizado el protocolo de comunicación UDP, sin la necesidad de adquirir una DAQ, comunicándonos con el Router, el cual sirve para centralizar la comunicación entre los robots y la computadora. Esto da una gran ventaja, dado que se puede hacer uso de todo el ancho de banda de la comunicación WiFi y no estar limitados a una comunicación serial, como el caso del Xbee.

Gracias a este sistema es posible ejecutar, una gran cantidad de algoritmos en Funciones S, siempre y cuando cumplan con las condiciones de tiempo real y Simulink, en el mínimo

tiempo posible. Solucionando así una gran mayoría de problemas en cuanto al tiempo de procesamiento y comunicación que existían en la literatura.

Con base en este trabajo se propone como trabajo a futuro utilizar un sistema de visión que logre detectar obstáculos sin el uso de patrones, a través de una cámara con de alta velocidad, la cual ayudará a que el procesamiento la computadora sólo se dedique al algoritmo de planificación de tareas y trayectorias.

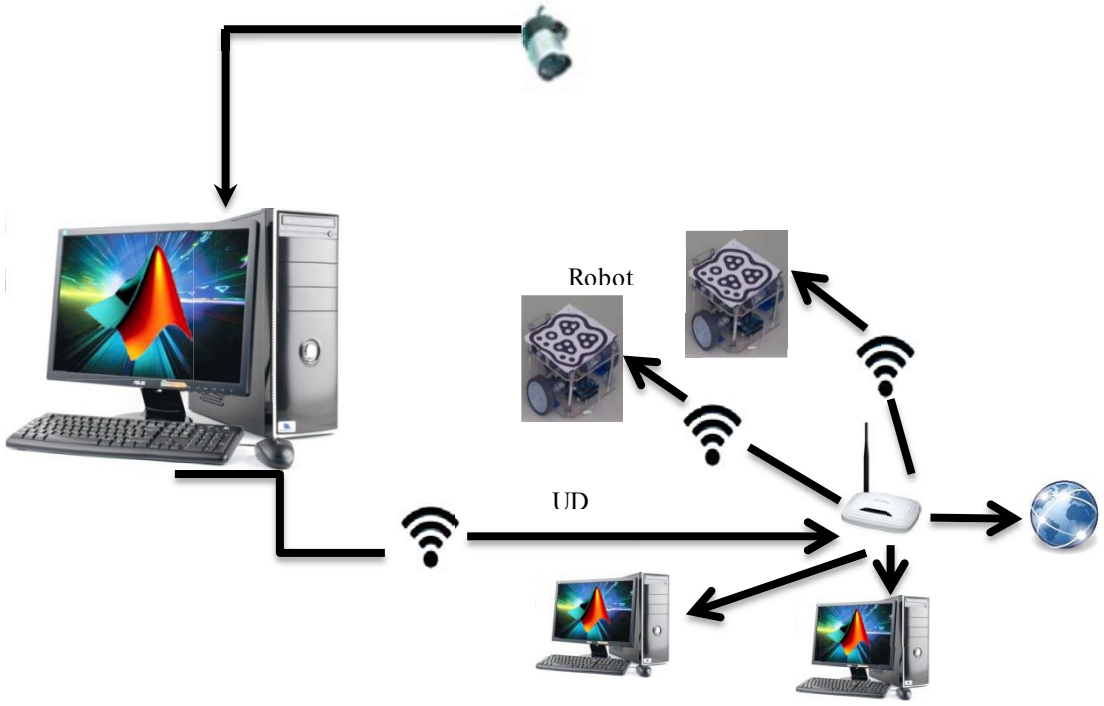


Fig. 5. 1 Sistema propuesto, conexión a internet.

El hecho de implementar una red WiFi y un Access Point, permitirá que con pocas modificaciones al sistema, contar con servidores web en cada uno de los robots, permitiendo así, la tele operación y la escritura y lectura de variables globales dentro de la red. También la creación de redes que incluyan más dispositivos, como otras computadoras, otros tipos de robots, celulares, tablets y la creación de redes y algoritmos más robustos.

6. APENDICES

A Código Processing comunicación entre Reactivision y Simulink

```
import hypermedia.net.*;
import TUIO.*;
import java.util.*;
TuioProcessing tuioClient;

float cursor_size = 15;
float object_size = 60;
float table_size = 760;
float scale_factor = 1;
PFont font;

UDP udp;

void setup()
{
 size(640,480);
 noStroke();
 fill(0);

 loop();
 frameRate(30);


 hint(ENABLE_NATIVE_FONTS);
 font = createFont("Arial", 18);
 scale_factor = height/table_size;


 tuioClient  = new TuioProcessing(this);
 smooth();
 udp=new UDP(this,27);
 udp.log(true);
}


void draw()
{
 background(255);
 textFont(font,18*scale_factor);
 float obj_size = object_size*scale_factor;
 float cur_size = cursor_size*scale_factor;
        String mx= new String();
```

```
  String my= new String();
  String ma= new String();
  String mx2= new String();
  String my2= new String();
  String ma2= new String();
  String mx3= new String();
  String my3= new String();
  String ma3= new String();

Vector tuioObjectList = tuioClient.getTuioObjects();
for (int i=0;i<tuioObjectList.size();i++) {
  TuioObject tobj = (TuioObject)tuioObjectList.elementAt(i);

  stroke(0);
  fill(0);
  pushMatrix();
  translate(tobj.getScreenX(width),tobj.getScreenY(height));
  rotate(tobj.getAngle());
  rect(-obj_size/2,-obj_size/2,obj_size,obj_size);
  popMatrix();
  fill(255);
  text(""+tobj.getSymbolID(), tobj.getScreenX(width), tobj.getScreenY(height));


  switch(tobj.getSymbolID()){



   case 1:
       float xf=tobj.getX()*100;
       float yf=tobj.getY()*100;
       float af=tobj.getAngle()*180/(3.1416);
       if(540-af>=360){af=180-af;}
       else{af=540-af;}
       int x=int(xf);
       int y=int(yf);
       int a=int(af);
       String sx = str(x);
       String sy = str(y);
       String sa = str(a);
       if(x<10){mx="00"+sx;}
       else if(x>=10 && x<100){mx="0"+sx;}
       if(y<10){my="00"+sy;}
       else if(y>=10 && y<100){my="0"+sy;}
       if(a<10){ma="00"+sa;}
       else if(a>=10 && a<100){ma="0"+sa;}
       else {ma=sa;}
```

```
        break;
    case 2:
        float xf2=tobj.getX()*100;
        float yf2=tobj.getY()*100;
        float af2=tobj.getAngle()*180/(3.1416);
        if(540-af2>=360){af2=180-af2;}
        else{af2=540-af2;}
        int x2=int(xf2);
        int y2=int(yf2);
        int a2=int(af2);
        String sx2 = str(x2);
        String sy2 = str(y2);
        String sa2 = str(a2);
        if(x2<10){mx2="00"+sx2;}
        else if(x2>=10 && x2<100){mx2="0"+sx2;}
        if(y2<10){my2="00"+sy2;}
        else if(y2>=10 && y2<100){my2="0"+sy2;}
        if(a2<10){ma2="00"+sa2;}
        else if(a2>=10 && a2<100){ma2="0"+sa2;}
        else {ma2=sa2;}
        break;
    case 3:
        float xf3=tobj.getX()*100;
        float yf3=tobj.getY()*100;
        float af3=tobj.getAngle()*180/(3.1416);
        if(540-af3>=360){af3=180-af3;}
        else{af3=540-af3;}
        int x3=int(xf3);
        int y3=int(yf3);
        int a3=int(af3);
        String sx3 = str(x3);
        String sy3 = str(y3);
        String sa3 = str(a3);
        if(x3<10){mx3="00"+sx3;}
        else if(x3>=10 && x3<100){mx3="0"+sx3;}
        if(y3<10){my3="00"+sy3;}
        else if(y3>=10 && y3<100){my3="0"+sy3;}
        if(a3<10){ma3="00"+sa3;}
        else if(a3>=10 && a3<100){ma3="0"+sa3;}
        else {ma3=sa3;}
        break;


  }
}
switch(tuioObjectList.size()){
  case 1:
```

```
println("  x "+mx+"  y "+my+"  a "+ma);
udp.send("M"+mx+my+ma,"127.0.0.1",666);
break;
 case 2:
println("  x "+mx+"  y "+my+"  a "+ma+"  x2 "+mx2+"  y2 "+my2+"  a2 "+ma2);
udp.send("M"+mx+my+ma+mx2+my2+ma2,"127.0.0.1",666);
break;
 case 3:
println("  x "+mx+"  y "+my+"  a "+ma+"  x2 "+mx2+"  y2 "+my2+"  a2 "+ma2+"  x3
"+mx3+"  y3 "+my3+"  a3 ");
udp.send("M"+mx+my+ma+mx2+my2+ma2+mx3+my3+ma3,"127.0.0.1",666);
break;
default:
break;

}

Vector tuioCursorList  = tuioClient.getTuioCursors();
for (int  i=0;i<tuioCursorList.size();i++)  {
  TuioCursor  tcur = (TuioCursor)tuioCursorList.elementAt(i);
  Vector pointList  = tcur.getPath();

  if (pointList.size()>0)  {
   stroke(0,0,255);
   TuioPoint  start_point  = (TuioPoint)pointList.firstElement();;
   for (int  j=0;j<pointList.size();j++)  {
     TuioPoint  end_point  = (TuioPoint)pointList.elementAt(j);

line(start_point.getScreenX(width),start_point.getScreenY(height),end_point.getScreenX(w
idth),end_point.getScreenY(height));
     start_point  = end_point;
   }

   stroke(192,192,192);
   fill(192,192,192);
   ellipse(  tcur.getScreenX(width),  tcur.getScreenY(height),cur_size,cur_size);
   fill(0);
   text(""+  tcur.getCursorID(),   tcur.getScreenX(width)-5,   tcur.getScreenY(height)+5);
  }
}
}

void addTuioObject(TuioObject  tobj) {

}
```

```
void removeTuioObject(TuioObject tobj) {
 }

void updateTuioObject (TuioObject tobj) {

}


void addTuioCursor(TuioCursor tcur) {
  }


void updateTuioCursor (TuioCursor tcur) {

}


void removeTuioCursor(TuioCursor tcur) {

}

void refresh(TuioTime bundleTime) {
  redraw();
}
```

B Función .m para desempaquetar los datagramas UDP de Processing

```
function [x1,y1,a1,a1r,x2,y2,a2,a2r]= fcn(u)

aux=0;
e=zeros(19);
for i=1:1:19
   y=u(i);
   if y==77
      aux=i;
   else

   end

end
r=19-aux;
for j=1:1:r
   e(j)=u(aux+j);
end
for k=1:1:aux
   e(r+k)=u(k);
end
x1=(e(1)-48)*100+(e(2)-48)*10+(e(3)-48);
y1=(e(4)-48)*100+(e(5)-48)*10+(e(6)-48);
a1=(e(7)-48)*100+(e(8)-48)*10+(e(9)-48);
a1r=a1*3.1416/180;

x2=(e(10)-48)*100+(e(11)-48)*10+(e(12)-48);
y2=(e(13)-48)*100+(e(14)-48)*10+(e(15)-48);
a2=(e(16)-48)*100+(e(17)-48)*10+(e(18)-48);
a2r=a2*3.1416/180;
```

C Función S controlador de campos potenciales.

```
/*
 * File : pruebaVel.c
 * Abstract:
 *      An example C-file S-function for multiplying an input by 2,
 *      y = 2*u
 *
 * Real-Time Workshop note:
 *   This file can be used as is (noninlined) with the Real-Time Workshop
 *   C rapid prototyping targets, or it can be inlined using the Target
 *   Language Compiler technology and used with any target. See
 *     matlabroot/toolbox/simulink/blocks/tlc_c/timestwo.tlc
 *   the TLC code to inline the S-function.
 *
 * Copyright 1990-2013 The MathWorks, Inc.
 */


#define S_FUNCTION_NAME controlador
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"
#include "math.h"
/*================*
 * Build checking *
 *================*/


/* Function: mdlInitializeSizes
=============================================
 * Abstract:
 *   Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{
   ssSetNumSFcnParams(S, 0);
   if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
      return; /* Parameter mismatch will be reported by Simulink */
   }

   if (!ssSetNumInputPorts(S, 5)) return;
   ssSetInputPortWidth(S, 0, DYNAMICALLY_SIZED);
   ssSetInputPortWidth(S, 1, DYNAMICALLY_SIZED);
   ssSetInputPortWidth(S, 2, DYNAMICALLY_SIZED);
   ssSetInputPortWidth(S, 3, DYNAMICALLY_SIZED);
   ssSetInputPortWidth(S, 4, DYNAMICALLY_SIZED);
```

```
    ssSetInputPortDirectFeedThrough(S, 0, 1);

    if (!ssSetNumOutputPorts(S,2)) return;
    ssSetOutputPortWidth(S, 0, DYNAMICALLY_SIZED);
    ssSetOutputPortWidth(S, 1, DYNAMICALLY_SIZED);


    ssSetNumSampleTimes(S, 1);

    /* specify the sim state compliance to be same as a built-in block */
    ssSetSimStateCompliance(S, USE_DEFAULT_SIM_STATE);

    ssSetOptions(S,
                 SS_OPTION_WORKS_WITH_CODE_REUSE |
                 SS_OPTION_EXCEPTION_FREE_CODE |
                 SS_OPTION_USE_TLC_WITH_ACCELERATOR);
}


/* Function: mdlInitializeSampleTimes
=============================================
 * Abstract:
 *   Specifiy that we inherit our sample time from the driving block.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
    ssSetModelReferenceSampleTimeDefaultInheritance(S);
}

/* Function: mdlOutputs
=======================================================
 * Abstract:
 *    y = 2*u
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    float           i,dg,drep,dres,tg,trep,te;
    float           rho01,rhoc1,rhor1,n1;
    float           x01,y01;
    float           vxpp,vypp,wpp;
    float           vxmax,vymax,wmax;
    float           xp,yp,tP;
    float           a,b,c,d,r;
```

```
float          xd,yd,kr,k_OWR;

//variables  de entrada
//Posición  y orientación  del robot móvi  diferencial
InputRealPtrsType   uPtrs0  = ssGetInputPortRealSignalPtrs(S,0);
InputRealPtrsType   uPtrs1  = ssGetInputPortRealSignalPtrs(S,1);
InputRealPtrsType   uPtrs2  = ssGetInputPortRealSignalPtrs(S,2);
//Posición  deseada para el robot móvil  diferencial
InputRealPtrsType   uPtrs3  = ssGetInputPortRealSignalPtrs(S,3);
InputRealPtrsType   uPtrs4  = ssGetInputPortRealSignalPtrs(S,4);


//Variables  de salida
real_T          *vxp    = ssGetOutputPortRealSignal(S,0);
real_T          *vyp    = ssGetOutputPortRealSignal(S,1);




//Velocidad  máxima  permisible
vxmax  =200;
vymax  =200;
//Posición  y orientación  del robot móvil  diferencial
xp   = *uPtrs0[0];
yp   = *uPtrs1[0];
tP   = *uPtrs2[0];
//Posición  deseada
xd   = *uPtrs3[0];
yd   = *uPtrs4[0];
//Obstáculos
x01= 5000;
y01= 5000;
rho01=20;
rhor1=4;
n1=100;
//Variables  de control
kr    = 1;
k_OWR = 1;


//Cálculo  de las  velocidades  de entrada
dg    = sqrt(pow((xd-xp),2)  + pow((yd-yp),2));
rhoc1 = sqrt(pow((x01-xp),2)  + pow((y01-yp),2));

if (yd==yp  &&  xd==xp){
   tg = tP;
}
```

```
    else {
        tg = atan2((yd-yp),(xd-xp));
    }

    if (rhoc1 > rho01+rhor1){
        drep=0;
        trep=2*tg;
    }
    else {
        drep= (n1/2)*pow((  (1/(rhoc1-rhor1))  - (1/rho01)  ),2)  ;
        trep= atan2((y01-yp),(x01-xp));
    }

    dres=drep+dg;
    te = (trep-tg)  - tP;


    if (dres <= k_OWR){
        vxpp = 0;
        vypp = 0;
    }
    else if (dres <= k_OWR + kr) {
        vxpp = vxmax*dres*cos(te)/(kr  + k_OWR);
        vypp = vymax*dres*sin(te)/(kr  + k_OWR);
    }
    else {
        vxpp = vxmax*cos(te);
        vypp = vymax*sin(te);
    }



   *vxp =vxpp;
   *vyp =vypp;


}


/* Function:  mdlTerminate
=============================================================
 * Abstract:
 *    No termination  needed,  but we '       are required  to have  this  routine.
 */
static  void mdlTerminate(SimStruct  *S)
{
}
```

```
#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

D Función S Cinemática del Robot Diferencial.

```
/*
 * File : pruebaVel.c
 * Abstract:
 *      An example C-file S-function for multiplying an input by 2,
 *      y = 2*u
 *
 * Real-Time Workshop note:
 *   This file can be used as is (noninlined) with the Real-Time Workshop
 *   C rapid prototyping targets, or it can be inlined using the Target
 *   Language Compiler technology and used with any target. See
 *     matlabroot/toolbox/simulink/blocks/tlc_c/timestwo.tlc
 *   the TLC code to inline the S-function.
 *
 * Copyright 1990-2013 The MathWorks, Inc.
 */


#define S_FUNCTION_NAME cinematicaS
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"
#include "math.h"
/*================*
 * Build checking *
 *================*/


/* Function: mdlInitializeSizes
=============================================
 * Abstract:
 *   Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{
   ssSetNumSFcnParams(S, 0);
   if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
      return; /* Parameter mismatch will be reported by Simulink */
   }

   if (!ssSetNumInputPorts(S, 2)) return;
   ssSetInputPortWidth(S, 0, DYNAMICALLY_SIZED);
   ssSetInputPortWidth(S, 1, DYNAMICALLY_SIZED);


   ssSetInputPortDirectFeedThrough(S, 0, 1);
```

```
    if (!ssSetNumOutputPorts(S,5))  return;
    ssSetOutputPortWidth(S,  0, DYNAMICALLY_SIZED);
    ssSetOutputPortWidth(S,  1, DYNAMICALLY_SIZED);
    ssSetOutputPortWidth(S,  2, DYNAMICALLY_SIZED);
    ssSetOutputPortWidth(S,  3, DYNAMICALLY_SIZED);
    ssSetOutputPortWidth(S,  4, DYNAMICALLY_SIZED);


    ssSetNumSampleTimes(S,  1);

    /* specify  the sim state compliance  to be same as a built-in  block */
    ssSetSimStateCompliance(S,  USE_DEFAULT_SIM_STATE);

    ssSetOptions(S,
                 SS_OPTION_WORKS_WITH_CODE_REUSE |
                 SS_OPTION_EXCEPTION_FREE_CODE |
                 SS_OPTION_USE_TLC_WITH_ACCELERATOR);
}


/* Function:  mdlInitializeSampleTimes
=========================================
 * Abstract:
 *    Specifiy  that we inherit  our sample  time  from the driving  block.
 */
static  void mdlInitializeSampleTimes(SimStruct  *S)
{
    ssSetSampleTime(S,  0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S,  0, 0.0);
    ssSetModelReferenceSampleTimeDefaultInheritance(S);
}

/* Function:  mdlOutputs
=======================================================
 * Abstract:
 *    y = 2*u
 */
static  void mdlOutputs(SimStruct  *S, int_T  tid)
{
    float           i,dg,tg,te;
    float           vxpp,vypp,wpp;
    float           vxmax,vymax;
    float           xp,yp,tP;
    float           a,b,c,d,r;
    float           xd,yd,kr,k_OWR;
```

```
    //variables  de entrada
    //Velocidades  deseadas para el robot móvi  diferencial
    InputRealPtrsType   uPtrs0  = ssGetInputPortRealSignalPtrs(S,0);
    InputRealPtrsType   uPtrs1  = ssGetInputPortRealSignalPtrs(S,1);

    //Variables  de salida
    real_T            *vx     = ssGetOutputPortRealSignal(S,0);
    real_T            *vy     = ssGetOutputPortRealSignal(S,1);
    real_T            *wp     = ssGetOutputPortRealSignal(S,2);
    real_T            *phi1   = ssGetOutputPortRealSignal(S,3);
    real_T            *phi2   = ssGetOutputPortRealSignal(S,4);

    //Dimensiones  de la plataforma
    d    = 17;
    b    = 27.4/2;
    r    = 12.5/2; //125mm  diameter


     //Posición  y orientación  del robot móvil  diferencial
    vxpp   = *uPtrs0[0];
    vypp   = *uPtrs1[0];



   *vx = cos(tP)*vxpp  + sin(tP)*vypp;
   *vy = -vxpp*sin(tP)  + vypp*cos(tP);
   *wp  = vypp*(1/d);
   *phi1  = vxpp/r + vypp*(-b)/(d*r)  ;
   *phi2 = vxpp/r + vypp*(b)/(d*r)  ;


}


/* Function:  mdlTerminate
=============================================================
 * Abstract:
 *    No termination  needed,  but we are required  to have this  routine.
 */
static  void mdlTerminate(SimStruct  *S)
{
}



#ifdef  MATLAB_MEX_FILE     /* Is this file  being compiled  as a MEX-file?  */
#include  "simulink.c"       /* MEX-file  interface  mechanism  */
```

```
#else
#include  "cg_sfun.h"        /* Code generation  registration  function  */
#endif
```

E Programa Arduino

```
#include <SPI.h>
#include <Wire.h>
#include <WiFi.h>
#include <WiFiUdp.h>
#define i2cAddress 0x58
#define speed1 0x00                                  // speed to first motor
#define speed2 0x01
/*
// the IP address for the shield:
IPAddress ip(192, 168, 0, 110);
*/
char ssid[] = "RobotMovil";  //  your network SSID (name)
char pass[] = "holamundo";    // your network password (use for WPA, or use as key for
WEP)

int status = WL_IDLE_STATUS;
byte a,b;
unsigned int localPort =6780;      // local port to listen on
long cont=0;
char packetBuffer[255]; //buffer to hold incoming packet


WiFiUDP Udp;
void setup() {
 Wire.begin();
 //Initialize serial and wait for port to open:


 if (WiFi.status() == WL_NO_SHIELD) {
  //Serial.println("WiFi shield not present");
  // don't continue:
  while(true);
 }
//  WiFi.config(ip);  // ip forzada
 // attempt to connect to Wifi network:
 while ( status != WL_CONNECTED) {
  //Serial.print("Attempting to connect to SSID: ");
  //Serial.println(ssid);
  // Connect to WPA/WPA2 network. Change this line if using open or WEP network:
  status = WiFi.begin(ssid, pass);
}
```

```
  // start the server:


  // you're connected now, so print out the status:
  printWifiStatus();

  Udp.begin(localPort);

 }


void loop() {
  //Serial.print("Entre  al loop ");
  // if there's data available,  read a packet
  int packetSize = Udp.parsePacket();
  if(packetSize)
  {
    int len = Udp.read(packetBuffer,255);
    if (len >0)
    packetBuffer[len]=0;
    a=packetBuffer[0];
    b=packetBuffer[1];
         // send a reply,  to the IP address and port that sent us the packet


    digitalWrite(11,HIGH);
      Wire.beginTransmission(i2cAddress);                  // Drive  motor  1 at speed value
stored  in  x
      Wire.write(speed1);
      Wire.write(a);
      Wire.endTransmission();


      Wire.beginTransmission(i2cAddress);                  // Drive  motor  2 at speed value
stored  in  x2
      Wire.write(speed2);
      Wire.write(b);
      Wire.endTransmission();
      digitalWrite(11,LOW);




}
}

void printWifiStatus()  {
```

```
  // print the SSID of the network you're attached to:
 // Serial.print("SSID: ");
  //Serial.println(WiFi.SSID());

  // print your WiFi shield's IP address:
  IPAddress ip = WiFi.localIP();
 // Serial.print("IP Address: ");

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  //Serial.print("signal strength (RSSI):");
  //Serial.print(rssi);
  //Serial.println(" dBm");
}
```

F Programa en Python para exportar los datos de Simulink

```
import SocketServer
import struct
archi = open('datos.csv','w')
archi.close()
class MatlabUDPHandler(SocketServer.BaseRequestHandler):

    def handle(self):
        data = self.request[0]
        socket = self.request[1]
        print "%s wrote:" % self.client_address[0]
        numOfValues = len(data) / 8
        unp=struct.unpack('>' + 'd' * numOfValues, data)
            ar = open('datos.csv','a')
            for x in range(0, numOfValues):
                    print unp[x]
                    ar.write(str(unp[x]))
                    ar.write('\t')
            ar.write('\n')
            ar.close()


HOST, PORT = "127.0.0.1", 1001
server = SocketServer.UDPServer((HOST, PORT), MatlabUDPHandler)
server.serve_forever()
```

BIBLIOGRAFÍA

[1] Latombe, Jean-Claude *Robot Motion Planning*. Springer Science & Business Media, 1991.

[2] Gopalakrishnana, B., S. Tirunellayi, and R. Todkar, *Design and development of an aoutonomous mobile smart vehicle: a mechatronics application.* Mechatronics, 2004.

[3] A Meystel *Basic Theory of Cognitive Controle. Autonomous Mobile Robots*, 1991.

[4] T. Arai, E. Pagello, and L. E. Parker. Guest editorial advances in multirobot systems. *Robotics and Automation, IEEE Transactions,* 2002

[5] Rubenstein, M *Kilobot: A low cost scalable robot system for collective behaviors,* IEEE Transactions, 2012

[6] S. Kornienko *Generation of desired emergent behavior in swarm of micro-robots,* 2004

[7] Ming Li *Robot Swarm Communication Networks: Architectures, Protocols, and Applications,* 2008

[8] J. A. Peña *Localization and control of a mobile robot for the pursuit of trajectories through visual feedback*. Thesis, Department of Mechatronics, UNAM, 2009

[9] http://www.kivasystems.com/

[10] Andrew S. Tnenbaum, *Redes de computadoras,* Pearson Educación, 2003

[11] http://reactivision.sourceforge.net/

[12] Alan Burns, *Sistemas de Tiempo Real y Lenguajes de Programación,* Addison Wesley, 2003

[13] David Vallejo Fernández, *Programación Concurrente y Tiempo Real,* Escuela Superior de Informática, 2014.

[14] http://www.mathworks.com/products/rtwt/

[15] Murphy R.R, *Introduction to AI Robotics,* MIT Press, 2000

[16] Brooks R, *A robot that walks. Emergent behaviors from a carefully evolved network*. MIT AI LAB Memo, 1989.

[17] Latombe, J. C., *Robot Motion Planning*, Boston,1991

[18] González Villela V. J. *Research on a semiautonomous mobile robot for loosely structures environments focused on transporting mail trolleys.* PhD Thesis, Loughborough University, 2006

[19] Ruiz Eduardo, *Robots Móviles Colaborativos para el Transporte Autónomo de Objetos Vía Visión,* Tesis, Departamento de Mecatrónica, UNAM, 2013