



**UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO**

**POSGRADO EN CIENCIA E INGENIERÍA DE LA  
COMPUTACIÓN**

**EVOLUCIÓN DE COMPORTAMIENTOS REACTIVOS PARA  
NAVEGACIÓN DE ROBOTS MÓVILES REPRESENTADOS  
COMO MODELOS DE MÁRKOV**

**TESIS  
QUE PARA OPTAR POR EL GRADO DE:  
MAESTRO EN CIENCIAS (COMPUTACIÓN)**

**PRESENTA  
EDUARDO ALEJANDRO JIMENEZ RIOS**

**TUTOR:  
DR. JESÚS SAVAGE CARMONA  
FACULTAD DE INGENIERÍA,  
UNAM**

**CIUDAD UNIVERSITARIA, CD. MX.,  
ABRIL DE 2022**



Universidad Nacional  
Autónoma de México



**UNAM – Dirección General de Bibliotecas**  
**Tesis Digitales**  
**Restricciones de uso**

**DERECHOS RESERVADOS ©**  
**PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL**

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.



# Agradecimientos

- *Al CONACYT por los recursos otorgados para cursar este grado de maestría.*
- *Al proyecto DGAPA-PAPIIT IG101721*



# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Justificación . . . . .	1
1.2. Hipótesis . . . . .	2
1.3. Objetivo . . . . .	2
1.4. Organización de la tesis . . . . .	3
<b>2. Marco Teórico</b>	<b>5</b>
2.1. Robótica basada en el comportamiento . . . . .	5
2.1.1. Paradigmas de la robótica . . . . .	5
2.2. Máquinas de Estado Finito . . . . .	7
2.2.1. Definición . . . . .	7
2.2.2. Máquinas de Estado Finito Probabilístico . . . . .	8
2.2.3. Representación como FSM: Robot que evade obstáculos. . . . .	8
2.3. Modelo Oculto de Márkov Extendido . . . . .	11
2.3.1. Modelo Oculto de Márkov . . . . .	11
2.3.2. Modelo Oculto de Márkov con salidas . . . . .	13
2.3.3. Representación como EHMM: Robot que evade obstáculos . . . . .	14
2.4. Procesos de Decisión de Márkov . . . . .	16
2.4.1. Definición . . . . .	16
2.4.2. Política Óptima . . . . .	16
2.4.3. Representación como MDP: Robot que evade obstáculos . . . . .	17
<b>3. Robótica Evolutiva</b>	<b>19</b>
3.1. Definición y Antecedentes . . . . .	19
3.2. Algoritmos Genéticos . . . . .	20
3.3. Problema simulador a realidad . . . . .	22
3.3.1. Optimización basada en la realidad . . . . .	22
3.3.2. Optimización basada en el simulador . . . . .	23
3.3.3. Optimización basada en robot dentro del ciclo . . . . .	23
3.3.4. Solución al problema . . . . .	23
<b>4. Herramientas de desarrollo</b>	<b>25</b>
4.1. Software . . . . .	25
4.1.1. Simulador . . . . .	25

4.1.2.	ROS . . . . .	26
4.2.	Hardware . . . . .	26
4.2.1.	Robot . . . . .	27
<b>5.</b>	<b>Desarrollo</b>	<b>29</b>
5.1.	Modelado del robot móvil . . . . .	29
5.1.1.	Modelado de los sensores y actuadores . . . . .	29
5.1.2.	Representación como máquina de estado finito . . . . .	30
5.1.3.	Ejecución de la Máquina de Estado Finito que representa al robot . . . . .	31
5.1.4.	Representación como modelo oculto de Márkov extendido . . . . .	32
5.1.5.	Ejecución del modelo oculto de Márkov extendido que representa al robot . . . . .	32
5.1.6.	Representación como proceso de decisión de Márkov . . . . .	33
5.1.7.	Ejecución del proceso de decisión de Márkov que representa al robot . . . . .	33
5.2.	Optimización utilizando algoritmos genéticos . . . . .	34
5.2.1.	Evaluación de los individuos . . . . .	34
5.2.2.	Algoritmo Genético . . . . .	36
<b>6.</b>	<b>Pruebas y resultados</b>	<b>39</b>
6.1.	Pruebas simulador Tk . . . . .	39
6.1.1.	Parámetros pruebas . . . . .	39
6.1.2.	Ambiente simulado . . . . .	40
6.1.3.	Resultados . . . . .	41
6.2.	Pruebas simulador ROS . . . . .	43
6.3.	Parámetros de pruebas . . . . .	44
6.3.1.	Resultados . . . . .	45
6.4.	Pruebas robot real . . . . .	48
6.4.1.	Resultados . . . . .	49
<b>7.</b>	<b>Conclusiones y trabajo futuro</b>	<b>51</b>
7.1.	Conclusiones . . . . .	51
7.2.	Trabajo futuro . . . . .	52
<b>A.</b>	<b>Minibot Educativo - Diagrama Eléctrico</b>	<b>55</b>
A.1.	Diagrama eléctrico . . . . .	56
A.2.	Niveles . . . . .	56
A.2.1.	Componentes principales del nivel inferior . . . . .	57
A.2.2.	Componentes principales del nivel superior . . . . .	57
A.3.	Descripción . . . . .	57
A.3.1.	Sistema de Alimentación . . . . .	57
A.3.2.	Conexión de los motores . . . . .	57
A.3.3.	Conexiones de las fotorresistencias . . . . .	59
A.3.4.	Conexiones con los sensores infrarrojos Sharp . . . . .	59

<b>B. Interconexión mediante ROS</b>	<b>61</b>
B.1. Interacción entre componentes . . . . .	61
B.1.1. Comunicación robot - exterior . . . . .	61
B.1.2. Comunicación robot - PC . . . . .	62
B.2. Mensajes de ROS . . . . .	62
<b>C. Guía de evolución de comportamientos</b>	<b>63</b>
C.1. Evolución de comportamientos simulador Tk . . . . .	63
C.1.1. Instalación del simulador Tk . . . . .	63
C.1.2. Ejecución simulador Tk . . . . .	64
C.2. Evolución de comportamientos ROS . . . . .	67
C.2.1. Instalación del simulador ROS . . . . .	67
C.2.2. Transferir modelos derivados en simulador Tk . . . . .	67
C.2.3. Derivar comportamientos en el simulador ROS . . . . .	68
<b>D. Manual de conexión Minibot - PC</b>	<b>71</b>
D.1. Conexión vía inalámbrica con el robot . . . . .	71
<b>Bibliografía</b>	<b>76</b>



# Capítulo 1

## Introducción

### 1.1. Justificación

La navegación es una de las principales tareas de un robot móvil, si bien es posible tener conocimiento del ambiente a priori en forma de mapas o representaciones del medio ambiente que nos ayuden a calcular rutas y tomar decisiones para la navegación entre dos puntos, en ambientes dinámicos como son las casas particulares, lugares de trabajo, hospitales, entre otros, donde se espera opere un robot móvil de servicio, pueden existir obstáculos imprevistos y que difícilmente puedan ser representados a priori en la memoria interna del robot. En estas situaciones los comportamientos reactivos son una alternativa más flexible, que en conjunto con la representación interna del ambiente puede lograr la navegación al destino final al evadir estos obstáculos imprevistos. Los comportamientos reactivos son, de manera general, un mapeo entre las entradas sensoriales del robot y los actuadores, donde se busca una respuesta casi instantánea.

Existe una gran variedad de modelos reactivos: campos potenciales, máquinas de estado finito, redes neuronales, entre otros. Al desarrollar estos modelos es necesario considerar las características físicas del robot, por lo que a pesar de ser modelos generales requieren parámetros específicos de acuerdo al robot a utilizarse y la tarea a realizar. La obtención de estos parámetros a fuerza bruta resulta una tarea complicada y no se tiene garantía de que los parámetros obtenidos hasta un momento intermedio no se puedan mejorar, sin embargo el uso de técnicas meta heurísticas como son los algoritmos evolutivos ayudan a facilitar este proceso y obtener soluciones aceptables en un tiempo razonable.

La mayoría de los estudios de obtención de parámetros de modelos reactivos se basan en la neuroevolución [1] [2], es decir uso de algoritmos evolutivos para derivar modelos de redes neuronales principalmente recurrentes, en este trabajo se propone como alternativa el uso de algoritmos genéticos en la evolución

de máquinas de estado finito probabilísticas como lo son los modelos ocultos de Márkov y los procesos de decisión de Márkov. Estas máquinas de estado probabilístico dada su naturaleza estocástica son consideradas como alternativa para la exploración y navegación en entornos desconocidos de manera que las acciones tomadas por el robot se realicen de manera distinta dadas condiciones similares buscando de esta manera evitar mínimos locales.

## 1.2. Hipótesis

Este trabajo se desarrolla con base en las siguientes hipótesis:

- Se puede modelar el comportamiento reactivo para navegación y evasión de obstáculos desconocidos como un modelo estocástico discreto sin memoria como son los modelos ocultos de Márkov y los procesos de decisión de Márkov.
- Mediante Algoritmos Genéticos se puede optimizar los parámetros de los modelos en un simulador, simplificando y mejorando el proceso de afinación de los parámetros en comparación a la afinación manual.
- El modelo obtenido en el simulador puede ser adaptado a un robot real, al realizar un proceso de re-entrenamiento en el entorno real.
- El modelo obtenido en el simulador será lo suficientemente robusto para no requerir un re-entrenamiento intensivo y no comprometer la integridad del robot al realizar las pruebas en el entorno real.

## 1.3. Objetivo

El objetivo principal de este trabajo es:

- Optimizar comportamientos reactivos estocásticos para la navegación y evasión de obstáculos de un robot móvil en un entorno desconocido.

Con base en este objetivo principal se tienen los siguientes objetivos específicos:

- Modelar el comportamiento del robot como una máquina de estado finito, un modelo oculto de Márkov extendido y como un proceso de decisión de Márkov.
- Encontrar los parámetros óptimos mediante un simulador para cada modelo y evaluar su desempeño en el simulador.
- Realizar un procedimiento de re-entrenamiento y afinación para aplicar estos modelos obtenidos mediante un simulador a un robot real y evaluar su desempeño en éste.

## 1.4. Organización de la tesis

En este primer capítulo se ha presentado la justificación y motivación de este trabajo, así como sus hipótesis y sus objetivos tanto particulares como generales.

En el capítulo 2 se dan los conceptos teóricos necesarios para entender los distintos modelos reactivos de navegación con los que se representarán los robots móviles y la manera en la que los robots móviles son representados. Los modelos a utilizar son: *máquinas de estado finito* (FSM), *modelos ocultos de Márkov extendidos* (EHMM) y *procesos de decisión de Márkov* (MDP).

En el capítulo 3 se plantean los conceptos referentes a la robótica evolutiva, partiendo de su fundamento básico, aplicaciones, así como la problemática de la *brecha de la realidad* que surge al trasladar modelos entrenados en ambientes simulados a ambientes reales.

En el capítulo 4 se describen las herramientas utilizadas tanto de software como de hardware. Respecto al software se define: el simulador utilizado, el cual fue desarrollado en el laboratorio de Bio-robótica de la Facultad de Ingeniería de la Universidad Nacional Autónoma de México, y el marco de trabajo ROS. Respecto al hardware se presenta el robot real utilizado describiendo su estructura y sus componentes básicos.

El capítulo 5 describe el proceso realizado para la obtención de los parámetros del robot, así como el modelado del robot para los tres distintos modelos reactivos utilizados y una breve explicación de la ejecución de estos. También se definen las implementaciones de la función de aptitud utilizada en los algoritmos genéticos y la metodología utilizada para la derivación de los modelos.

El capítulo 6 muestra los resultados obtenidos para los modelos tanto en el simulador como en el robot real y un análisis comparativo entre estos, resaltando las problemáticas y singularidades de cada comportamiento.

Por último en el capítulo 7 se dan las conclusiones de este trabajo de investigación de acuerdo a los resultados documentados en el capítulo 6, y se plantean áreas de mejora y preguntas abiertas como trabajo futuro.



# Capítulo 2

## Marco Teórico

En este capítulo se presentan los principales conceptos teóricos referentes a los distintos modelos reactivos utilizados en el presente proyecto. Se inicia con una breve introducción sobre los comportamientos reactivos y posteriormente se explican los modelos utilizados para representar al robot: máquinas de estado finito, modelos ocultos de Márkov extendidos y procesos de decisión de Márkov. Para cada modelo se da un ejemplo sencillo de la implementación de éste aplicado a la tarea de evasión de obstáculos.

### 2.1. Robótica basada en el comportamiento

#### 2.1.1. Paradigmas de la robótica

Un paradigma es un conjunto de técnicas y supuestos que caracterizan el enfoque con el que se trabaja una problemática.

En la robótica se tienen tres paradigmas para organizar la inteligencia en los robots: jerárquico, reactivo e híbrido [3]. Estos paradigmas se distinguen por la relación entre las tres primitivas de la robótica:

- Percibir: son las funciones en las que el robot recibe información a través de sus sensores.
- Planear: son las funciones que toman la información obtenida a través de sensores o de la representación interna del mundo y la utiliza para producir una o más tareas a realizar por el robot.
- Actuar: las funciones que producen una salida a los actuadores del robot.

El paradigma jerárquico es el más antiguo, en éste se sigue una secuencia estricta de estados conocida como el ciclo percibir-planear-actuar, donde el robot percibe el ambiente con sus sensores, planea la siguiente acción, actúa y después vuelve a percibir el ambiente para reiniciar el ciclo, esto se efectúa hasta lograr

el objetivo (ver Figura 2.1).

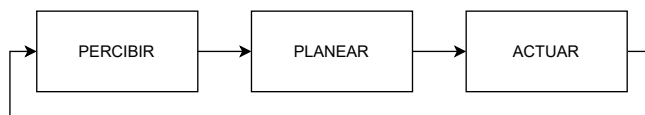


Figura 2.1: Esquema del paradigma jerárquico

El paradigma jerárquico presenta dos problemáticas principales: el supuesto del mundo cerrado y el problema del recuadro [3]. El supuesto del mundo cerrado asume que el robot posee toda la información necesaria acerca del ambiente en la representación interna que tiene de éste. En cuanto al problema del recuadro presenta la incapacidad para representar toda la información necesaria de una manera computacionalmente viable. Ambas problemáticas limitan el campo de aplicación del paradigma jerárquico pues en ambientes continuamente cambiantes o con un grado de incertidumbre resulta demasiado compleja y tediosa la representación del ambiente.

Como respuesta a las problemáticas del paradigma jerárquico y dados los avances en psicología cognitiva y etología, surge el paradigma reactivo donde se elimina la etapa de planificación y se busca tener un conjunto de instancias percepción-acción o estímulo-respuesta (ver Figura 2.2).

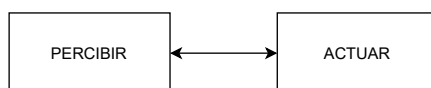


Figura 2.2: Esquema del paradigma reactivo

A pesar de que el paradigma reactivo resuelve la problemática de la lenta velocidad de respuesta, encuentra limitantes para lograr comportamientos complejos que requieren capacidades cognitivas.

Nuevamente dadas las limitaciones de ambos paradigmas se busca un balance entre las ventajas y desventajas de cada uno y de esta manera surge el paradigma híbrido, donde el robot primeramente descompone de la mejor manera posible la tarea en subtareas, las cuales se resuelven mediante instancias estímulo-respuesta (ver Figura 2.3), con lo que se busca agilizar las acciones del robot al no tener que planear cada acción (Paradigma jerárquico), pero se tiene cierta planificación central que facilita la coordinación de las instancias percepción-acción (Paradigma reactivo).

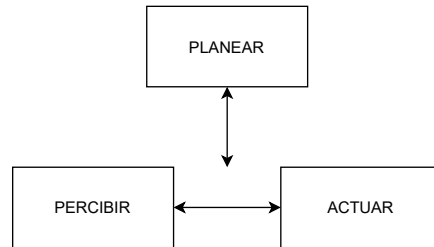


Figura 2.3: Esquema del paradigma híbrido

## 2.2. Máquinas de Estado Finito

Las máquinas de estado finito son uno de los distintos tipos de autómatas, jerárquicamente se encuentran en un nivel bajo, sin embargo son suficientemente buenas para distintas aplicaciones y áreas donde pueden ayudar significativamente a implementar tareas repetitivas [4].

### 2.2.1. Definición

Una máquina de estado finito (FSM) o autómata finito funciona intuitivamente de la manera siguiente:

- Se tiene un conjunto finito de estados  $Q$  que son descripciones instantáneas del sistema, los cuales contienen la información necesaria para determinar la evolución del sistema a partir de ese momento.
- Existen transiciones  $\delta$  entre estados, las cuales son cambios de un estado a otro del sistema, estos cambios surgen de manera espontánea o dadas ciertas condiciones externas.
- Dada una cadena de símbolos de entrada  $\Sigma^*$  e iniciando en un estado inicial  $s$ , se van realizando las transiciones dado cada símbolo de entrada y el estado del sistema, una vez terminada la cadena se acepta la cadena si se llega a un estado final  $F$  y se rechaza en caso contrario.

Formalmente una máquina de estados se define [5] como una estructura:

$$M = (Q, \Sigma, \delta, s, F)$$

donde:

- $Q$  es un conjunto finito, los elementos de  $Q$  se llaman estados.
- $\Sigma$  es un conjunto finito, el alfabeto de entrada.
- $\delta : Q \times \Sigma \rightarrow Q$  es la función de transición. Intuitivamente  $\delta$  es una función que indica a qué estado moverse en respuesta a una entrada: si  $M$  está en un estado  $q$  y ve una entrada  $a$ , se mueve al estado  $\delta(q, a)$ .

- $s \in Q$  es el estado inicial.
- $F$  es un subconjunto de  $Q$ , los elementos de  $F$  se llaman estados finales,  $F$  puede ser un conjunto vacío.

Una extensión de la estructura descrita previamente son las máquinas de estado finito con salidas, las cuales adicionalmente tienen un alfabeto de salida  $S$  y una función de salida  $\lambda : Q \times \Sigma \rightarrow S$ . El funcionamiento de la FSM en este caso consiste en dado un conjunto de símbolos de entrada generar un conjunto de símbolos de salida por lo que el conjunto de estados finales  $F$  se considera vacío, quedando la nueva estructura formada por seis elementos de la forma:

$$M = (Q, \Sigma, S, \delta, \lambda, s)$$

A partir de este punto al utilizar una máquina de estado finito nos referiremos a este segundo caso que incluye salidas.

### 2.2.2. Máquinas de Estado Finito Probabilístico

En algunas ocasiones se tiene información incompleta sobre el sistema por lo que no es posible determinar el estado actual y/o las función de transición y de salida de éste de manera exacta, por lo que representarla como una máquina de estado finito tradicional resultaría una tarea compleja, sin embargo existe una extensión de las máquinas de estado finito llamadas máquinas de estado finito probabilístico (PFSM) que nos permiten modelar estos sistemas.

Un PFSM  $T$  se define [6] como:

$$T = (Q, \Sigma, S, \delta, I, F, P)$$

Donde:

- $Q, \Sigma$  y  $S$  continúan siendo el conjunto de estados, el alfabeto de entrada y de salida respectivamente.
- $\delta \subseteq Q \times \Sigma \times S \times Q$  es el conjunto de transiciones.
- $I : Q \rightarrow \mathbb{R}^+$  y  $F : Q \rightarrow \mathbb{R}^+$  son la probabilidad de estado inicial y estado final respectivamente.
- $P : \delta \rightarrow \mathbb{R}^+$  son las probabilidades de transición.

Una PFSM también puede ser representada como un EHMM, el cual se trata en la sección 2.3.1.

### 2.2.3. Representación como FSM: Robot que evade obstáculos.

Las FSM pueden ser representadas por la estructura formal descrita en la sección 2.2.1, sin embargo esta representación es poco práctica, por lo que también se utilizan otras representaciones como el diagrama de transiciones y la



tabla de estado-transiciones.

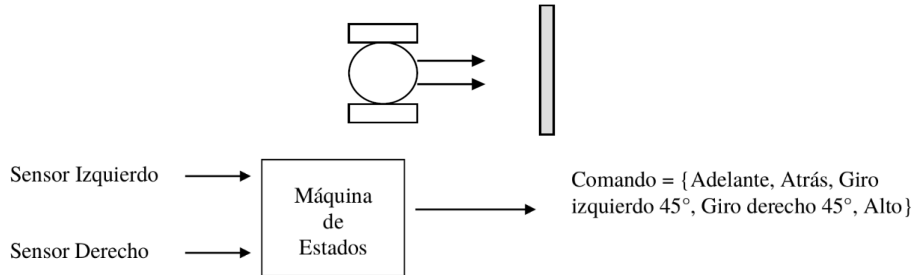


Figura 2.4: Robot móvil

Para ejemplificar esto consideremos la máquina de estados que representa el comportamiento de un robot móvil que navega libremente y evade obstáculos, el cual tiene las siguientes características físicas: dos motores, uno en el lado izquierdo y uno en el derecho, y dos sensores colocados en la parte delantera para detectar obstáculos (ver Figura 2.4), el robot se comporta de acuerdo a las siguientes condiciones (ver Figura 2.5):

- Si los sensores no detectan un obstáculo, el robot sigue avanzando.
- Si el sensor derecho lo detecta y el izquierdo no, el robot se hace para atrás y después gira hacia la izquierda  $45^\circ$  y sigue avanzando.
- Si el sensor izquierdo lo detecta y el derecho no, el robot se hace para atrás y gira hacia la derecha  $45^\circ$  para seguir avanzando.
- Si los dos sensores detectan el obstáculo, el robot se hace para atrás y gira hacia la izquierda  $90^\circ$ . Después avanza una cierta distancia y gira hacia la derecha  $90^\circ$  y sigue avanzando.

La máquina de estado finito correspondiente tendría los siguientes elementos:

- Conjunto de estados:  $Q = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$
- Alfabeto de entrada:  $\Sigma = \{00, 01, 10, 11\}$
- Alfabeto de salida:  $S = \{0, 1, 2, 3, 4\}$
- Estado inicial:  $s = s_0$

Donde los elementos de  $\Sigma$  corresponden a las distintas combinaciones de los sensores donde el primer dígito corresponde al sensor izquierdo, el segundo al derecho, un 0 indica que no se detectó un obstáculo en dicho sensor y un 1 que sí se detectó, los elementos de  $S$  son las acciones posibles del robot siendo 0:detenerse, 1:avanzar, 2:retroceder, 3:giro izquierda, 4:giro derecha.  $\delta$  y  $\lambda$  se pueden

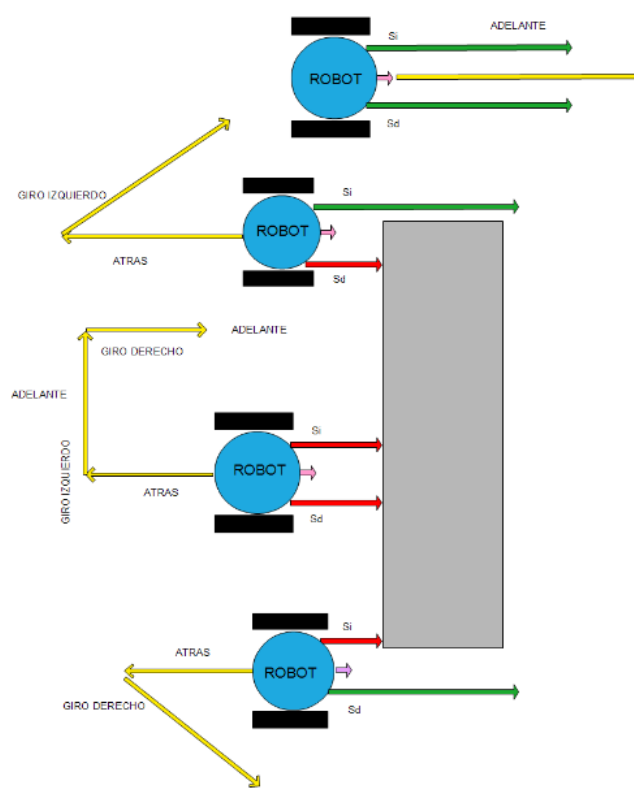


Figura 2.5: Comportamiento de evasión de obstáculos

observar más fácilmente en el diagrama de transiciones de la máquina de estado finito que se muestra en la Figura 2.6 o en la tabla de estado-transiciones que se muestra en el Cuadro 2.1.

Tanto en el diagrama como en la tabla el símbolo \* se utiliza para representar condiciones libres, esto quiere decir que sin importar cual sea el símbolo de entrada se realizará la transición entre estado.

En el diagrama de transiciones cada estado se representa por una circunferencia, las transiciones se representan por aristas dirigidas del estado actual al estado siguiente, en cada transición se coloca una etiqueta condición/acción donde la condición viene dada por el símbolo de entrada recibido y la acción indica el símbolo de salida, el estado inicial se indica con una flecha entrante.

En las tablas de estado-transiciones las dos primeras columnas presentan todas las posibles combinaciones estado-símbolo de entrada, la tercera y cuarta columnas representan el estado siguiente y el símbolo de salida respectivamen-

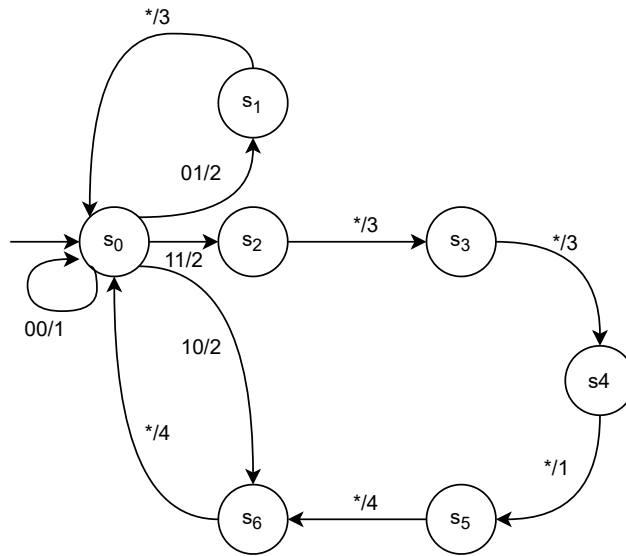


Figura 2.6: Diagrama de transiciones

te, dados el estado y el símbolo de entrada de las columnas anteriores el estado inicial corresponde al primer estado de la tabla.

Estado	Entrada	Estado siguiente	Salida
0	00	0	1
0	01	1	2
0	10	6	2
0	11	2	2
1	*	0	3
2	*	3	3
3	*	4	3
4	*	5	1
5	*	6	4
6	*	0	4

Cuadro 2.1: Tabla de Estado-Transiciones

## 2.3. Modelo Oculto de Márkov Extendido

### 2.3.1. Modelo Oculto de Márkov

Un modelo oculto de Márkov (HMM) es un proceso estocástico doble con un proceso subyacente que es no observable (oculto), pero puede ser observa-

do solamente a través de otro proceso que produce una secuencia de símbolos observable [7]. Un HMM cumple con la propiedad de Márkov, es decir que la probabilidad de que ocurra un evento depende solamente del evento anterior, como se describe en la ecuación siguiente:

$$P(X_{n+1} = x | X_1 = x_1, X_2 = x_2, \dots, X_n = x_n) = P(X_{n+1} = x | X_n = x_n)$$

Formalmente un HMM se define como una estructura:  $\lambda = (Q, V, A, B, \Pi)$  donde:

- $Q = \{s_1, s_2, \dots, s_N\}$  es un conjunto finito, los elementos de  $Q$  se conocen como estados.
- $V = \{v_1, v_2, \dots, v_M\}$  es un conjunto finito, los elementos de  $V$  se conocen como símbolos de entrada.
- $A = \{a_{ij}\}$  es el conjunto de probabilidades de transición de estado a estado, donde cada elemento  $a_{ij} = P(q_t = s_j | q_{t-1} = s_i)$  es la probabilidad de estar en el estado  $j$  en el tiempo  $t$  dado que en el tiempo  $t-1$  se estaba en el estado  $i$ .
- $B = \{b_j(v_k)\}$  es el conjunto de probabilidades de las observaciones donde cada elemento  $b_j(v_k) = P(o_t = v_k | s_t = j)$  es la probabilidad de observar el símbolo  $v_k$  dado que se esta en el estado  $s_j$ .
- $\Pi = \{\pi_i\}$  es el conjunto de probabilidades iniciales, donde  $\pi_i$  es la probabilidad de que el estado  $s_i$  sea el estado inicial.

El conjunto  $O = \{O_1, O_2, \dots, O_T\}$  es la secuencia de observaciones donde cada elemento de  $O$  es uno de los símbolos de  $V$ . Dado que el número de estados y símbolos viene implícito en los conjuntos  $A$  y  $B$ , se utiliza la notación compacta  $\lambda = (A, B, \Pi)$ .

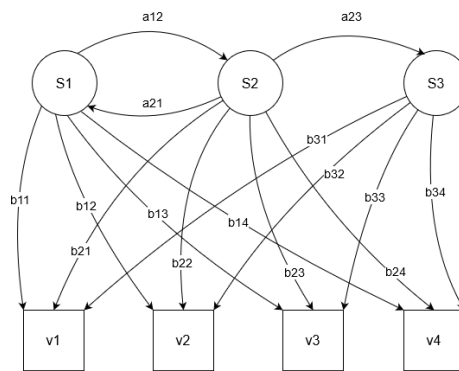


Figura 2.7: Ejemplo de HMM

Un HMM se utiliza para resolver alguna de las siguientes 3 problemáticas:

- *Evaluación*: Dados los parámetros del modelo  $\lambda$  y una secuencia de observaciones  $O$  se busca la probabilidad de generar dicha secuencia dado el modelo.
- *Decodificación*: Dados los parámetros del modelo  $\lambda$  y una secuencia de observaciones  $O$  se busca la secuencia de estados  $\hat{q}$  más probable que pueda haber generado la secuencia de observaciones.
- *Aprendizaje*: Dado un conjunto de observaciones, encontrar los conjuntos de probabilidades de transiciones  $A$ , de probabilidades de observar los símbolos  $B$  y de probabilidades iniciales  $\Pi$  más probables, es decir entrenar los parámetros de  $\lambda$ .

### 2.3.2. Modelo Oculto de Márkov con salidas

En el desarrollo de este trabajo se desea que el modelo oculto de Márkov sea equivalente a una máquina de estados finitos probabilística con salidas, para lograr esto a la estructura  $\lambda = (Q, V, A, B, \Pi)$  se le agrega otro par de elementos:

- $U = \{u_1, u_2, \dots, u_P\}$  un conjunto finito donde los elementos de  $U$  se conocen como símbolos de salida.
- $C = \{c_{ij}(u_k)\}$  es el conjunto de probabilidades de las salidas donde cada elemento  $c_{ij}(u_k) = P(u_t = k | s_t = j, o_t = i)$

Siendo en este caso la notación compacta  $\lambda = (A, B, C, \Pi)$ , esta variante se conoce como modelo oculto de Márkov con salidas o extendido (EHMM).

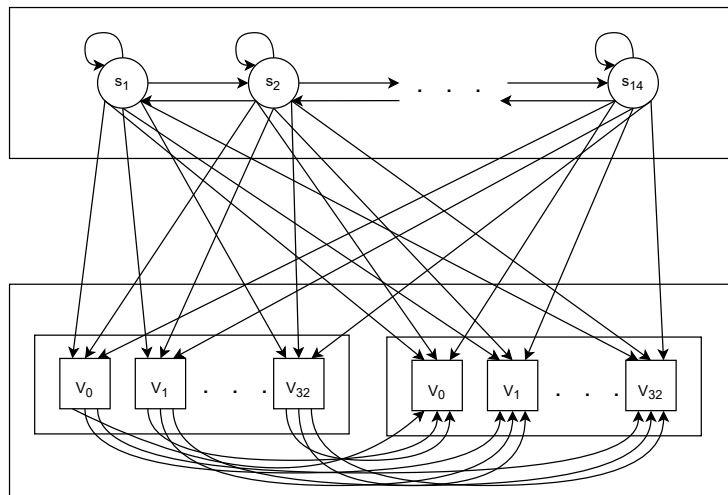


Figura 2.8: Ejemplo de EHMM

### 2.3.3. Representación como EHMM: Robot que evade obstáculos

Consideremos el mismo problema presentado en la sección 2.2.3, agregando que en este caso se tiene una incertidumbre en las lecturas de los sensores y las salidas de los actuadores, por lo que no podemos tener una completa certeza de en que estado se encuentra el robot. Considerando el mismo conjunto de estados  $Q = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$ , de símbolos de entrada  $V = \{00, 01, 10, 11\}$  y de símbolos de salida  $U = \{0, 1, 2, 3, 4\}$ , los elementos del modelo  $\lambda = (A, B, C, \Pi)$  se pueden representar como matrices de distribución de probabilidad de la siguiente manera:

- $A_{[N \times N]}$ : la matriz de probabilidad de transición entre los estados.

$$A = \begin{bmatrix} P(q_t = s_0 | q_{t-1} = s_0) & P(q_t = s_1 | q_{t-1} = s_0) & \dots & P(q_t = s_N | q_{t-1} = s_0) \\ P(q_t = s_0 | q_{t-1} = s_1) & P(q_t = s_1 | q_{t-1} = s_1) & \dots & P(q_t = s_N | q_{t-1} = s_1) \\ \vdots & \vdots & \ddots & \vdots \\ P(q_t = s_0 | q_{t-1} = s_N) & P(q_t = s_1 | q_{t-1} = s_N) & \dots & P(q_t = s_N | q_{t-1} = s_N) \end{bmatrix}$$

donde:

$$\sum_{j=1}^N a_{ij} = 1 \quad \forall i$$

- $B_{[N \times M]}$ : la matriz de probabilidad de generación de observaciones.

$$B = \begin{bmatrix} P(o_t = v_0 | s_t = 0) & P(o_t = v_1 | s_t = 0) & \dots & P(o_t = v_M | s_t = 0) \\ P(o_t = v_0 | s_t = 1) & P(o_t = v_1 | s_t = 1) & \dots & P(o_t = v_M | s_t = 1) \\ \vdots & \vdots & \ddots & \vdots \\ P(o_t = v_0 | s_t = N) & P(o_t = v_1 | s_t = N) & \dots & P(o_t = v_M | s_t = N) \end{bmatrix}$$

donde:

$$\sum_{j=1}^N b_{ij} = 1 \quad \forall i$$

- $C_{[NM \times P]}$ : la matriz de probabilidad de generación de salidas.

$$C = \begin{bmatrix} P(u_t = 0 | G_t = 0) & P(u_t = 1 | G_t = 0) & \dots & P(u_t = P | G_t = 0) \\ P(u_t = 0 | G_t = 1) & P(u_t = 1 | G_t = 1) & \dots & P(u_t = P | G_t = 1) \\ \vdots & \vdots & \ddots & \vdots \\ P(u_t = 0 | G_t = NM) & P(u_t = 1 | G_t = NM) & \dots & P(u_t = P | G_t = NM) \end{bmatrix}$$

donde:

$$\sum_{j=1}^N c_{ij} = 1 \quad \forall i$$

$G_t$  está formado por las parejas de estados y símbolos de entrada.

$$G_t = [o_t = 0, s_t = 0 \quad \dots \quad o_t = 0, s_t = N \quad o_t = 1, s_t = 0 \quad \dots \quad o_t = M, s_t = N]$$

- $PI_{[1 \times N]}$ : el vector de probabilidad de estado inicial.

$$\Pi = [P(s_0 = 0) \quad s_0 = 1 \quad \dots \quad s_0 = N]$$

donde:

$$\sum_{i=1}^N \pi_i = 1$$

En este caso primero es necesario obtener el modelo  $\lambda = \{A, B, C, \Pi\}$  dado un conjunto de observaciones, lo cual corresponde al tercer problema de los HMM mencionados anteriormente: *Aprendizaje*. El aprendizaje del modelo se puede realizar utilizando el algoritmo *Baum-Walch*, técnicas basadas en descenso por gradiente o métodos de búsqueda meta heurísticos, como los algoritmos genéticos [8].

Una vez obtenido el modelo  $\lambda = \{A, B, C, \Pi\}$  con  $N$  estados se sigue el proceso descrito en [9], el estado inicial se obtiene mediante la ecuación:

$$\delta_1(i) = \pi_i b_i(O_1), \quad 1 < i < N$$

Donde  $\delta_1(i)$  guarda las probabilidades mas altas de que el estado inicial sea  $i$ . Siendo el estado inicial más probable el dado por:

$$S_1 = j = \arg \max_{1 < i < N} [\delta_1(i)]$$

Una vez obtenido el primer estado  $S_1$  el símbolo de salida  $u_1$  se obtiene de manera aleatoria, proporcionalmente a las probabilidades  $P(u_k | o_1 = O_1, s_1 = j)$   $1 < k < P$  de la matriz  $C$ , mediante la generación de un número aleatorio  $x \in [0, 1)$  con distribución uniforme, observando el rango en que se encuentra de acuerdo al Cuadro 2.2, si  $r_i \leq x < r_{i+1}$  la salida sera  $U_i$

Salida $i$	Rango	$r$
0	$[0, c_{ij,1})$	$r_1$
1	$[c_{ij,1}, c_{ij,1} + c_{ij,2})$	$r_2$
$\vdots$	$\vdots$	$\vdots$
$k$	$[\sum_{m=1}^k c_{ij,m}, \sum_{m=1}^{k+1} c_{ij,m})$	$r_{k+1}$
$\vdots$	$\vdots$	$\vdots$
$K - 1$	$[\sum_{m=1}^{K-1} c_{ij,m}, 1)$	$r_K$

Cuadro 2.2: Tabla de generación de símbolo de salida

Para obtener el estado siguiente  $S_t$  se utilizan los valores de  $\delta_{t-1}(i)$  del estado actual, de esta manera se realiza de manera recursiva el cálculo de  $\delta_t(j) = \max_{1 < i < N} P[S_1, \dots, S_{t-1} = i, O_1, \dots, O_t | \lambda]$ , utilizando las ecuaciones siguientes:

$$\delta_t(j) = \max_{1 < i < N} [\delta_{t-1}(i) a_{ij}] b_j(O_t), \quad 1 < j < N$$

$$S_t = j = \arg \max_{1 \leq i \leq N} [\delta_t(j)]$$

Y el símbolo de salida se calcula de la misma manera que el primero generado con  $i$  el índice de la observación actual  $O_t$  y  $j$  el índice del estado actual  $S_t$ .

## 2.4. Procesos de Decisión de Márkov

### 2.4.1. Definición

Un proceso de decisión de Márkov (MDP) es un proceso de control estocástico de tiempo discreto, Los MDP son una herramienta para la toma de decisiones en situaciones donde el resultado es parcialmente aleatorio y parcialmente bajo control del agente que toma la decisión.

Un MDP se define formalmente por una 4-tupla  $(S, A, P, R)$  donde:

- $S$  es un conjunto finito, donde los elementos de  $S$  se llaman estados.
- $A$  es un conjunto finito, donde los elementos de  $A$  se llaman acciones.
- $P : S \times A \times S \rightarrow [0, 1]$  es la función de transición probabilística  $P(s'|s, a)$ .
- $R : S \times A \rightarrow \mathbb{R}$  es la función de recompensa

Una política estacionaria  $\pi : S \times A \rightarrow [0, 1]$  define la probabilidad  $\pi(s, a)$  de que el agente realice la acción  $a$  en el estado  $s$ . En un MDP el objetivo es encontrar una política  $\Pi$  que maximice una función de utilidad  $V_\pi(s)$  definida utilizando un criterio de descuento en la recompensa:  $V_\pi(s) = E_\pi[\sum_{t=0}^{\infty} \gamma^t \cdot r_t | s_0 = s]$ , donde  $0 \leq \gamma < 1$  es el factor de descuento,  $r_t$  la recompensa en el tiempo  $t$  y  $s_0$  es el estado inicial [10].

Los MDPs asumen observabilidad completa, es decir, siempre se conoce el estado en el que se encuentra el agente, dado a que esta suposición no se cumple en la mayoría de los problemas, existe una gran diversidad de variantes de MDP conocidos como *Procesos de Decisión de Márkov Parcialmente Observables* (POMDP).

### 2.4.2. Política Óptima

Dadas las funciones  $R$  y  $P$  completamente definidas existen dos métodos generales, basados en la programación dinámica, para encontrar la política óptima: *Iteración del Valor* e *Iteración de la Política*.

La función de valor-acción  $Q_\pi(s, a)$  que da el valor esperado de realizar  $a$  en el estado  $s$  se define como:

$$Q_\pi(s, a) = \sum_{s' \in S} P(s'|s, a)(R(s', a) + \gamma V_\pi(s'))$$



La idea básica del método de iteración por valor es que dado un estimado de la función de valor tras  $k$  pasos, determinar la función para el paso  $k + 1$ . Para eso se realiza el siguiente proceso:

1. Se inicializa  $V_0 = 0$  para cada estado.
2. Se calcula  $Q_{i+1}$  y  $V_{i+1}$  de  $V_i$  para cada estado:

$$Q_{i+1}(s, a) = \sum_{s'} P(s'|a, s)(R(s', a) + \gamma V_i(s'))$$

$$V_{i+1}(s) = \max_a Q_{i+1}(s, a)$$

3. Si  $\forall s |V_i(s) - V_{i+1}(s)| < \theta$  para cada estado:

$$\pi^*(s) = \operatorname{argmax}_a \sum_{s'} P(s'|a, s)(R(s', a) + \gamma V_i(s'))$$

Donde  $\theta$  es un umbral que determina el error mínimo aceptado en la política óptima obtenida.

### 2.4.3. Representación como MDP: Robot que evade obstáculos

Debido a que los comportamientos reactivos solo conocen su entorno inmediato, no es posible obtener una política óptima global, sin embargo se puede obtener una política óptima a nivel local, y una vez realizado cierto número de pasos  $t$  volver a calcular la política óptima de acuerdo al nuevo entorno. Para ejemplificar esto se considera un robot con dos motores como en los comportamientos anteriores, pero con 8 sensores de distancia distribuidos cada  $45^\circ$  de manera que pueda conocer su entorno inmediato. El entorno se representará como una cuadrícula de  $6 \times 6$  donde en cada casilla se representa si existe un obstáculo de acuerdo a la lectura de los sensores, el robot se encuentra inicialmente en la posición central y mirando hacia el norte. Siendo los componentes del MDP los siguientes:

- Conjunto de estados:  $S = \{s_1, s_2, \dots, s_{36}\}$  correspondientes a cada casilla de la cuadrícula que representa el entorno inmediato.
- Conjunto de acciones:  $A(s) = \{\text{detenerse, avanzar, retroceder, girar } 45^\circ \text{ a la derecha y avanzar, girar } 45^\circ \text{ a la izquierda y avanzar}\}$
- $P(s'|s, a)$  probabilidad de transición para cada acción  $a \in A(s)$
- $R(s)$  la función de recompensa definida como:

$$R(S) = \begin{cases} 1.00 & \text{Si no hay obstáculo y se encuentra en los extremos del} \\ & \text{entorno observado} \\ -0.04 & \text{Si no hay obstáculo y no se encuentra en los extremos} \\ & \text{del entorno observado} \\ -1.00 & \text{Si hay obstáculo} \end{cases}$$

Esto debido a que se busca el robot tenga una navegación activa al evitar obstáculos.

Para cada entorno se calcula la política óptima con el proceso de iteración por valor. Por ejemplo para el entorno mostrado en la Figura 2.9a se tiene la representación de los estados y la función de recompensa mostrados en la Figura 2.9b, en la Figura 2.9c se muestra una política de movimientos  $\Pi$ , dada tal política y considerando el estado equivalente a la tercer fila y tercer columna como estado inicial y  $t = 3$ , el robot realizaría las acciones *retroceder*, *retroceder* y *detenerse*, y posteriormente se obtendrá la representación del espacio de estados para el nuevo entorno, se volverá a obtener la política óptima y se realizarán las acciones de acuerdo a esta.

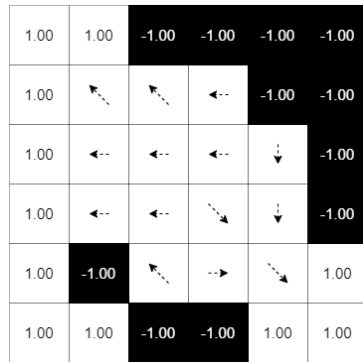
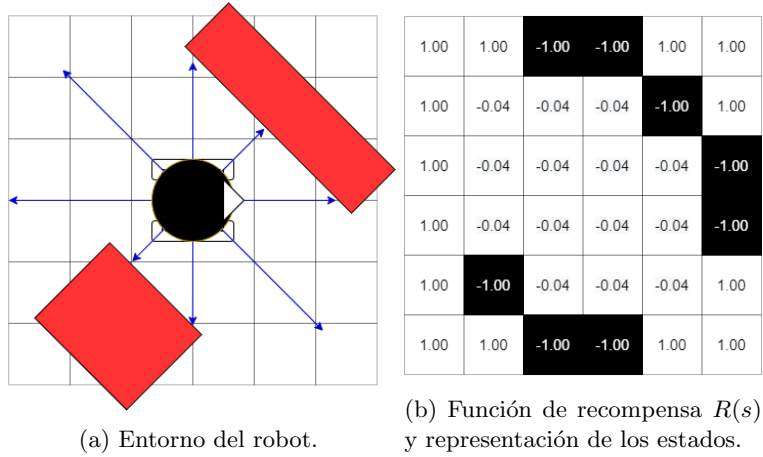


Figura 2.9: Robot representado como MDP

## Capítulo 3

# Robótica Evolutiva

Este capítulo es una introducción de la robótica evolutiva, donde se presenta su definición, sus beneficios y sus problemáticas. Se hace un énfasis en uno de sus componentes principales: *los algoritmos genéticos*. También se profundiza en la problemática del *problema simulador a realidad* dada su importancia en este trabajo de investigación.

### 3.1. Definición y Antecedentes

En la robótica evolutiva (RE) se busca la generación y evolución automatizada de modelos tanto físicos como de controladores, para que un robot realice una tarea determinada. La RE utiliza herramientas de inteligencia artificial y cómputo evolutivo para lograr este objetivo, basándose en la premisa de selección natural diferencial propuesta por Darwin, donde los individuos mejor adaptados a su medio ambiente tienen mayor probabilidad de un éxito reproductivo, lo que permite que la *información genética* de estos individuos se herede a la siguiente generación de individuos. Este éxito evolutivo también es conocido como *aptitud*.

La RE se ha aplicado en diversas problemáticas relacionadas en la robótica, por mencionar algunos ejemplos se tienen la evasión de obstáculos [11][12], la visión activa [13] y la detección de objetos [14]. La RE ha mostrado poder resolver tareas que son complicadas modelar manualmente y además ha logrado la evolución de mecanismos de control con un mejor desempeño que aquellos diseñados manualmente.

Los enfoques de RE se clasifican según cuatro dimensiones [15]:

- **Simulador - Realidad**

Depende del sistema utilizado para la experimentación. Por un lado dada la gran cantidad de pruebas necesarias y los daños que el robot puede llegar a sufrir el uso de simuladores se ha extendido dado que resultan

mas rápidos, económicos y no comprometen al robot real; sin embargo los resultados obtenidos en los simuladores no siempre son aplicables a robots reales (dados modelos inadecuados del robot o del ambiente) lo que se conoce como *problema simulador a realidad*, del que se hablará con más detalle en la sección 3.3, por lo que un enfoque más apropiado involucraría una combinación de experimentación mixta entre simuladores y el mundo real.

- **En línea - Fuera de línea**

Ésta relacionada con el tiempo en el que se realiza la evolución. En los métodos fuera de línea los controladores evolucionan en ambientes artificiales (simulaciones o experimentos de laboratorio) antes de ejecutarse en el lugar real de aplicación, en este caso también se presenta la problemática simulación a realidad en menor medida. En los métodos en línea la evolución de los controladores se realiza al ejecutar la tarea en el ambiente real.

- **Centralizado - Descentralizado**

Se relaciona con la presencia o ausencia de una unidad central de cómputo durante la evolución. En los métodos descentralizados los robots evolucionan por sí mismos o por la comunicación de una vecindad de robots, mientras que en los métodos centralizados se tiene una computadora que observa y controla los procesos evolutivos.

- **Encapsulado - Distribuido**

Divide los métodos entre secuenciales y paralelos. En los métodos encapsulados, todos los individuos son representados por un único robot y se evalúan secuencialmente. En los métodos distribuidos cada robot se representa individualmente, realizándose la evaluación de la aptitud en paralelo.

Los experimentos realizados en este trabajo se realizarán en ambos extremos de la dimensión simulador - realidad, ya que se busca que el controlador obtenido en la simulación pueda ser adaptado en el robot real mediante un re-entrenamiento menos intensivo, buscando un entrenamiento inicialmente acelerado y sin comprometer al robot real y posteriormente afinarlo en el robot real. Respecto a las demás dimensiones se realizaría Fuera de línea, Centralizado y Encapsulado, ya que se utilizará un único robot, y Fuera de línea dado que las pruebas tanto en el simulador como con el robot real son en ambientes controladas para poder tener una comparación de la aptitud de cada individuo dadas las mismas condiciones.

## 3.2. Algoritmos Genéticos

Como se menciona en el capítulo anterior la RE es un método para la evolución automática de modelos, para lograr esto se utilizan conceptos básicos del cómputo evolutivo, como son los algoritmos genéticos. Los algoritmos genéticos (AG) desde su propuesta inicial por Holland [16] han tenido una gran evolución

y diversificación para la solución de problemas de optimización. Sin embargo a pesar de la gran variedad de AG tienen como base el ciclo de evolución básico mostrado en la Figura 3.1, donde los cambios surgen en la implementación de cada paso, los cuales cambian de acuerdo al problema de optimización. Las etapas del algoritmo genético básico son [17]:

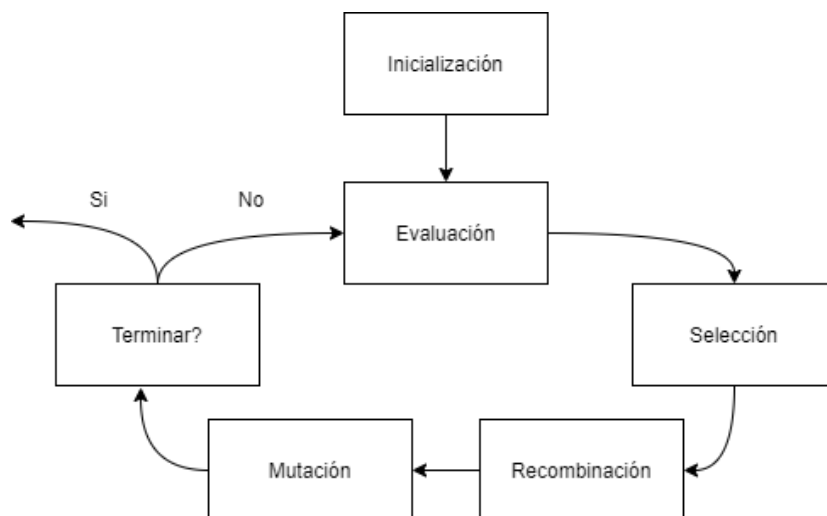


Figura 3.1: Ciclo evolutivo básico

### 1. Inicialización

La población inicial generalmente se genera de manera aleatoria sobre todo el espacio de búsqueda, aunque es posible agregar cierto conocimiento a priori para generar la población inicial. Cada individuo se distingue entre sí por su *genotipo*, el genotipo es una codificación del individuo y este está relacionado con el comportamiento de este (*fenotipo*).

### 2. Evaluación

Al generarse la población inicial o una nueva generación de individuos se debe calcular la aptitud para cada solución candidata, la aptitud se calcula mediante una función de aptitud, la cual está relacionada con la tarea a realizar, que determina que tan apto es un individuo para realizar la tarea. La función de aptitud puede considerar un solo objetivo (mono-objetivo) o con una serie de distintos objetivos (multi-objetivo), posiblemente en conflicto, donde se deberá ponderar la importancia de cada uno y definir la función de acuerdo a esto.

### 3. Selección

Se busca que los individuos con mayor aptitud sean los que tengan un mayor éxito reproductivo, los métodos de selección más comunes son: selección por ruleta, selección por estocástico universal, selección por ranqueo y selección por torneo.

#### 4. Recombinación

Se combina la información genética de los individuos seleccionados en la etapa anterior, mediante esto se busca explotar los genes de los mejores individuos y obtener individuos con mayor aptitud al combinarlos. Esta etapa se determina por el operador *cruza*.

#### 5. Mutación

La *cruza* permite explotar los genes de los mejores individuos, sin embargo es posible que un individuo aún mejor se encuentre genéticamente muy cerca de un individuo existente, la mutación realiza una búsqueda local en la vecindad de los individuos mediante una ligera modificación en su genotipo.

#### 6. Reemplazo

La nueva población (generada en las etapas de Selección, Recombinación y Mutación), reemplaza a la población actual de manera total o parcial, en el caso de un reemplazo parcial se mantiene un porcentaje de la población actual de acuerdo a su aptitud, esta técnica es conocida como *elitismo*.

### 3.3. Problema simulador a realidad

Como se mencionó anteriormente en la sección 3.1 el problema simulador a realidad (PSR) se presenta al transferir modelos evolucionados en simuladores a robots reales [18]. Esta problemática tiene diversas causas entre ellas se encuentran: falta de conocimiento tanto del robot real como del entorno real, propiedades conocidas que resultan inviables de simularse debido al costo computacional, diferencias impredecibles en los sensores y actuadores entre distintos robots, fallas mecánicas o de software [15].

Dentro de los trabajos para tratar el PSR se distinguen tres enfoques principales : *Optimización basada en la realidad*, *Optimización basada en el simulador* y *Optimización basada en robot dentro del ciclo*[19].

#### 3.3.1. Optimización basada en la realidad

En este enfoque la optimización se realiza total o parcialmente en el robot físico. Ejemplos de optimización realizada totalmente en el robot real se encuentran en [20] [21] sin embargo resulta en extensos tiempos de entrenamiento (en comparación a simulaciones) incluso para tareas relativamente sencillas.

Respecto a la optimización parcial, los modelos son optimizados en un simulador la mayor parte y solo en las últimas generaciones se optimizan en el robot real, utilizando transferencia de conocimiento y reduciendo el tiempo de entrenamiento requerido si se utilizara el robot real únicamente. Sin embargo este enfoque asume que la solución óptima en el simulador y en la realidad se encuentran suficientemente cercanas, lo cual en muchas ocasiones no se cumple

debido a inexactitudes o errores en el modelo dinámico del simulador, y en estos casos no sea posible o resulte muy largo el re-entrenamiento.

### **3.3.2. Optimización basada en el simulador**

En este enfoque la optimización se realiza completamente en simulaciones. Se busca la creación de modelos más precisos bajo la premisa de que si los modelos simulados son adecuados no debería existir una brecha significativa entre la simulación y la realidad. Sin embargo, entre más se parece a la realidad el modelo este es más complejo y tiene un mayor costo computacional, por lo que se debe tener un balance entre la complejidad del modelo y el costo computacional.

Para lidiar con la problemática del costo computacional de modelos precisos, han surgido distintas técnicas para la creación de controladores robustos que puedan transferirse al robot real. Dentro de estas técnicas se encuentra la adición de ruido para compensar dinámicas no consideradas en el simulador, técnicas adaptativas para tratar con cambios en el ambiente como modelos con plasticidad [22] o la creación de un modelo al encontrarse en un ambiente nuevo y ajustar los parámetros de acuerdo a este modelo [23].

### **3.3.3. Optimización basada en robot dentro del ciclo**

En este enfoque se optimiza completamente en simulaciones, pero se realizan algunos experimentos de transferencia en el proceso. Esto se realiza al evolucionar dos poblaciones: el controlador como en todos los enfoques y adicional el simulador. Al evolucionar poblaciones de simuladores se puede observar que tan preciso es un simulador con respecto a la realidad, esta evaluación de los simuladores se realiza a través de experimentos en el robot real [24]. Sin embargo este enfoque presenta el supuesto de que el simulador puede optimizarse suficiente para permitir la transferencia de sus modelos a la realidad, sin embargo este puede no ser el caso para ambientes complejos.

### **3.3.4. Solución al problema**

El problema simulador a realidad sigue siendo un problema abierto, si bien los enfoques mencionados han dado resultados positivos para ciertas tareas, cada uno dependiendo de la complejidad de la tarea y realizando un balance entre costo computacional y complejidad del simulador. Cada enfoque, además considera ciertos supuestos (algunos más restrictivos) que no se cumplen para todas las tareas y todos presentan dificultades a medida que las tareas a optimizar se vuelven más complejas: la optimización en robots físicos resulta lenta y puede comprometer la integridad del robot, la optimización en simuladores depende de simuladores con un alto costo computacional y la creación de modelos robustos gracias a ruido o adaptabilidad no tienen garantía de un desempeño óptimo en la realidad, la evolución de simuladores es dependiente del modelo del simulador

y que los parámetros de este puedan adecuarse a la realidad.

En este trabajo se utiliza el primer enfoque *Optimización basada en la realidad*, donde se utiliza un simulador que busca replicar las características esenciales del robot a un bajo costo computacional, y posteriormente se utiliza el modelo obtenido en este simulador como punto de partida para la optimización en el robot real.



## Capítulo 4

# Herramientas de desarrollo

### 4.1. Software

Para el desarrollo de este trabajo de investigación se utilizaron los lenguajes de programación C/C++ y Python. Las simulaciones se realizaron en un simulador realizado en el laboratorio de Biorobótica de la Facultad de Ingeniería de la U.N.A.M. El marco de trabajo *ROS* se utiliza para el intercambio de información entre el modulo de navegación donde se ejecutan los algoritmos para los distintos modelos y los módulos encargados de la adquisición de datos mediante los sensores y de la respuesta de los actuadores.

#### 4.1.1. Simulador

El simulador está diseñado para la evolución de comportamientos de robots utilizando algoritmos genéticos, donde el comportamiento se codifica como una secuencia de bits para las operaciones de cruce y mutación de los algoritmos genéticos. La evaluación se realiza de acuerdo a los algoritmos implementados de acuerdo a cada modelo (FSM, EHMM, MDP, etc.), se utilizan distintos entornos de navegación creados tanto manualmente como de maneras aleatoria (Figura 4.1), al igual que distintos pares de puntos iniciales y finales del robot. El robot se representa de una manera abstracta mediante sus dimensiones geométricas básicas (en el caso de un robot circular el radio), la distribución de sus sensores y las características de estos (rangos de operación).

El simulador cuenta con una interfaz gráfica donde una vez evolucionado un comportamiento se puede ejecutar el modelo para observar su desempeño en alguno de los entornos de navegación, un ejemplo se observa en la Figura 4.2.

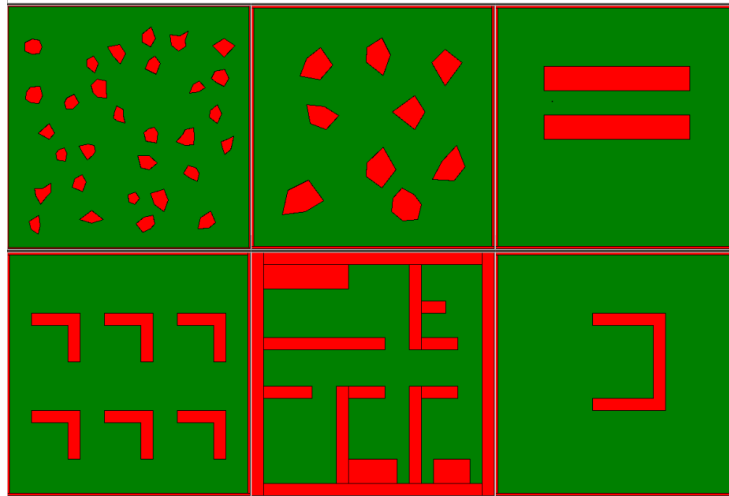


Figura 4.1: Ejemplos de entornos de navegación utilizados

#### 4.1.2. ROS

El Sistema Operativo de Robot (ROS) es un marco de trabajo flexible para escribir software para robots. Es una colección de herramientas, librerías y convenciones que buscan simplificar la tarea de crear comportamientos de robots complejos y robustos a través de una amplia variedad de plataformas de robótica [25]. Este sistema permite el desarrollo de proyectos de robótica de manera distribuida, permitiendo la integración de distintos módulos mediante la transferencia de mensajes.

Se utilizó la distribución ROS Kinetic (compatible con Ubuntu 16.04) para el desarrollo del planeador de movimientos, en el cual es donde se ejecutan los algoritmos de cada uno de los comportamientos (FSM, EHMM y MDP), y la integración con los módulos de movimiento y lectura de sensores del robot educativo utilizado.

## 4.2. Hardware

Se busca que los comportamientos derivados inicialmente en el simulador se apliquen en la realidad, para realizar las pruebas se utilizó un robot educativo cuyas características generales se describen a continuación. Se utiliza una computadora personal para visualizar y controlar la configuración del robot.

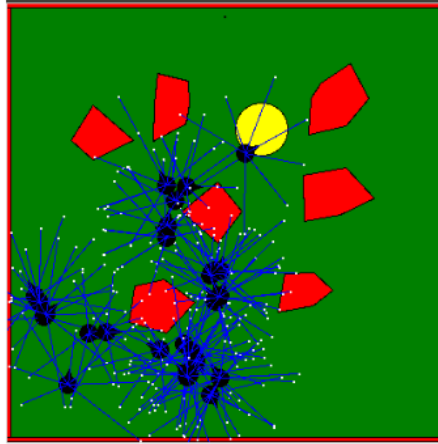


Figura 4.2: Ejemplo de ejecución en el simulador

#### 4.2.1. Robot

El robot educativo cuenta con un microcontrolador *Raspberry Pi 3* encargado del almacenamiento y procesamiento de información de los sensores, el control de los puertos salida (actuadores).

El objetivo del robot es navegar de un punto inicial a un punto final, dicho punto final es una fuente luminosa, para lograr esto el robot está equipado con los siguientes sensores:

- Arreglo de 8 fotorresistencias: las fotorresistencias están distribuidas de manera uniforme en la parte superior del robot de manera radial cada  $45^\circ$  como se muestra en la Figura 4.3 , lo que permiten identificar la región donde se encuentra la fuente luminosa.
- Arreglo de sensores de proximidad *Sharp GP2Y0A41SK0F*: los sensores cuentan con un rango de operación de  $4 - 30$  [cm], la distribución de los sensores en el robot se muestra en la Figura 4.4.
- Módulo Raspberry Pi Camera V2.1

Respecto a los actuadores el robot cuenta con dos ruedas motrices paralelas complementadas con dos ruedas multidireccionales de soporte.

Para una descripción más detallada de los componentes físicos del robot y sus conexiones véase el Apéndice A, para una descripción más detallada de los distintos nodos de ROS y la comunicación entre el robot y la computadora véase el Apéndice B.

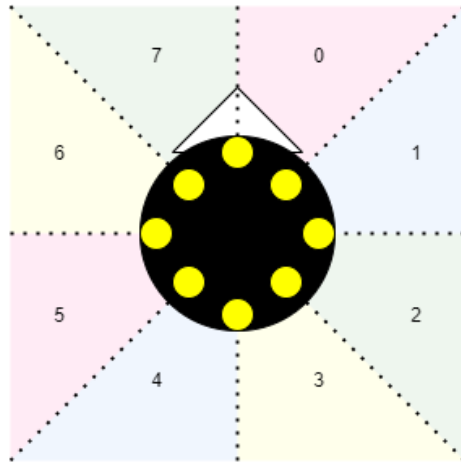


Figura 4.3: Distribución de los sensores de intensidad luminosa (amarillo) y las distintas regiones donde se puede encontrar la fuente luminosa

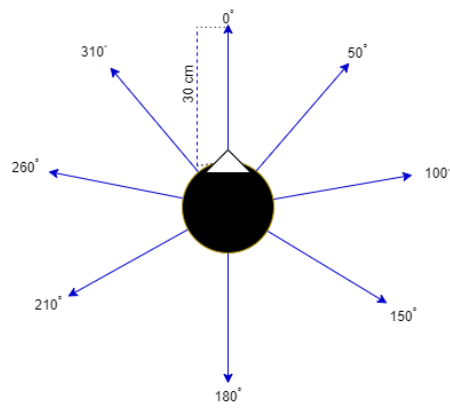


Figura 4.4: Distribución de los sensores de proximidad

# Capítulo 5

## Desarrollo

En este capítulo se explica el proceso realizado para modelar el robot en el entorno simulado, la representación del robot para cada uno de los modelos utilizados (FSM, EHMM y MDP). Así como la descripción de la función de aptitud y las características del algoritmo genético utilizado.

### 5.1. Modelado del robot móvil

El uso del entrenamiento en simulador resulta esencial dado que nos permite realizar pruebas de una manera rápida y sin comprometer al robot real, sin embargo como se explica en la sección 3.3 la aplicación de los modelos derivados en los simuladores depende de qué tan cercanos a la realidad se encuentren y la abstracción necesaria para que no se tenga una carga computacional excesiva.

El modelo del robot en el simulador se enfoca principalmente en las características geométricas del robot (distribución de los sensores, dimensiones físicas del robot) y el funcionamiento de los sensores (rangos de funcionamiento y nivel de incertidumbre en las lecturas), dejando fuera las características mecánicas de este.

#### 5.1.1. Modelado de los sensores y actuadores

Para poder representar de manera más precisa el comportamiento del robot real al simulador, consideraremos las incertidumbres que se tiene en los sensores y actuadores. Para de esta manera derivar un controlador robusto a estas perturbaciones en el mundo real.

Para modelar el ruido en las lecturas de los sensores y en las acciones de los actuadores se estimaron estadísticamente los parámetros de este a través de un procedimiento que consistió en:

- **Adquisición de valores de ruido**

En el caso de los sensores: realizar 100 lecturas de los sensores y obtener los errores  $e = |y - \hat{y}|$  entre el valor obtenido  $\hat{y}$  y con el valor ideal  $y$ .

En el caso de los actuadores: realizar 100 veces un experimento y obtener los errores  $e = |y - \hat{y}|$  entre el valor resultante  $\hat{y}$  y el valor que se esperaba  $y$ .

- **Análisis del ruido** Para cada tipo de sensor y actuador se asumió una distribución normal y de acuerdo a las mediciones registradas se obtuvieron los parámetros  $\mu$  y  $\sigma^2$  de la función normal.

$$\mu = \frac{1}{100} \sum_{i=1}^{100} e_i$$

$$\sigma^2 = \frac{1}{100} \sum_{i=1}^{100} (e_i - \mu)^2$$

Mediante el procedimiento descrito se obtuvieron los resultados mostrados en el Cuadro 5.1.

elemento	$\mu$	$\sigma$
Sensor distancia	0 [cm]	0.0054 [cm]
Avance	5.5 [cm]	0.895 [cm]
Giro	5.5 [°]	6.70 [°]

Cuadro 5.1: Parámetros del ruido de los sensores de actuadores

### 5.1.2. Representación como máquina de estado finito

De acuerdo a la definición dada en la sección 2.2.1, se definió la FSM con los siguientes elementos:

- $Q$ : es el conjunto de estados, los cuales son equivalentes a la memoria del robot.  
Se utilizaron 4 bits para el conjunto de estados resultando en un total de 16 estados posibles.
- $\Sigma$ : es el conjunto de valores de entrada de los sensores de distancia y de intensidad luminosa discretizados.  
Para discretizar las lecturas los sensores de distancia se realizó un proceso de cuantificación vectorial, con 8 vectores clave, utilizando 3 bits para representarlos.  
Para los sensores de intensidad luminosa dado que se tienen 8 sensores, se utilizaron 3 bits para codificar la posición de la fuente luminosa, y otros 2 bits adicionales se utilizaron para la intensidad luminosa máxima.  
Considerando el total de bits utilizados para los valores de entrada, se

tiene un total de 8 bits para su representación en la FSM, obteniendo un total de 256 posibles valores  $\Sigma = \{00, 01, \dots, FF\}$  en representación hexadecimal.

- $S$ : es el conjunto de acciones que puede realizar el robot, se consideraron ocho distintas:  $S = \{\text{Detenerse, Avanzar, Retroceder, Girar a la izquierda } (45^\circ), \text{ Girar a la Derecha } (-45^\circ), \text{ Girar a la izquierda y avanzar } (45^\circ), \text{ Girar a la derecha } (-45^\circ) \text{ y avanzar, Girar doble a la derecha } (90^\circ) \text{ y avanzar}\}$
- $\delta$ : es la función de transición entre estados del robot.
- $\lambda$ : es la función de salida dado el estado del robot y las lecturas actuales de los sensores.
- $s_0$ : es el estado inicial.

Los parámetros que se buscan aprender son  $\delta$  y  $\lambda$ , es decir los pares *estado siguiente-salida* para cada par *estado actual-entrada*.

Para representar esta FSM se utilizó una tabla de estado-transición en la memoria del robot donde cada fila representa un par *estado actual - entrada* y su contenido representa el par *estado siguiente - salida*. La tabla de estado-transiciones tendrá un total de  $16 \times 256 = 4096$  (estados  $\times$  entrada) filas, y el contenido de cada una de estas tendrá una longitud de  $4 + 3$  (bits de estado siguiente + bits salida), por lo que cada FSM será representada por un total de 28,672 bits. Una parte de la tabla de transiciones del robot se encuentra en el cuadro 5.2.

Estado $S_t$	Entrada	Estado Siguiente $S_{t+1}$	Salida
0000	000 000 00	1110	001
0000	000 000 01	0111	001
0000	000 000 10	1111	111
0000	000 000 11	1100	001
⋮	⋮	⋮	⋮
1111	111 111 11	0000	000

Cuadro 5.2: Ejemplo de tabla de transiciones del modelo

### 5.1.3. Ejecución de la Máquina de Estado Finito que representa al robot

Una vez generada la FSM, al iniciar la ejecución, el robot comienza en el estado inicial  $s_0$  y de acuerdo al símbolo de entrada  $e_0 \in \Sigma$  realiza la acción y cambia de estado de acuerdo a la tabla de transición que representa a la FSM. Una vez en el nuevo estado y de acuerdo al nuevo símbolo de entrada  $e_1$  vuelve a realizar la transición de estado y la acción correspondiente a la tabla de transiciones, repitiendo este proceso hasta llegar al objetivo (la intensidad luminosa

sea mayor a un umbral  $\epsilon$ ) o se realice un número determinado máximo de transiciones  $t_{max}$ . Tomando la tabla de transiciones del Cuadro 5.2 y considerando que la FSM comienza su ejecución, el estado 0000 es el estado inicial, por lo que al iniciar el robot en una posición sin obstáculos (000), con la fuente luminosa en el primer cuadrante (000) y una intensidad luminosa baja (00), se tendría  $e_0 = 000\ 000\ 00$  por lo que el robot pasaría al estado 1110 y realizaría la acción *Avanzar* (001) como se observa en la Figura 5.1.

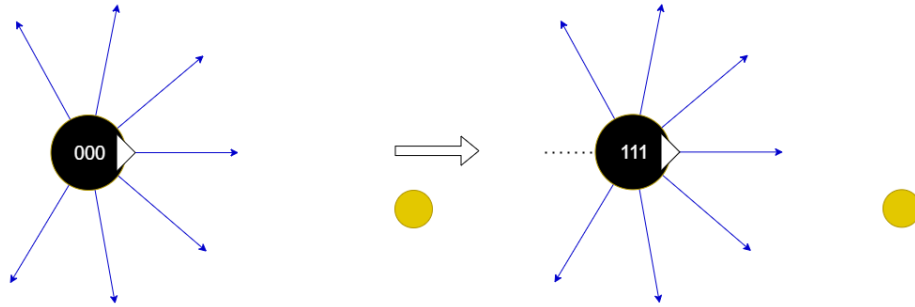


Figura 5.1: Ejemplo de ejecución FSM

#### 5.1.4. Representación como modelo oculto de Márkov extendido

Para representar el robot como un EHMM se requieren los elementos del modelo  $\lambda = (A, B, C, \Pi)$ , siendo  $A$  una matriz de  $N \times N$ ,  $B$  una matriz de  $N \times M$ ,  $C$  una matriz de  $NM \times P$  y  $\Pi$  un vector de  $N$  elementos, donde  $N = 16$  es el número de estados,  $M = 256$  el número de símbolos de entrada y  $P = 8$  el número de símbolos de salida. Los conjuntos de estados  $Q$ , de símbolos de entrada  $V$  y de símbolos de salida  $U$ , son los mismos que  $Q$ ,  $\Sigma$  y  $S$  descritos en la representación FSM. El contenido de las matrices son valores de probabilidad, por lo que por cada elemento se requieren 8 bits de memoria, siendo la longitud total del individuo la suma de los tamaños de los elementos de  $\lambda$ , siendo un total de  $8(16 \times 16 + 16 \times 256 + 16 \times 256 \times 8 + 16) = 297,088$  bits.

#### 5.1.5. Ejecución del modelo oculto de Márkov extendido que representa al robot

Con los elementos del modelo  $\lambda = (A, B, C, \Pi)$  la ejecución se realiza de acuerdo al proceso descrito en la Sección 2.3.3 para obtener el estado inicial  $S_1$ , el estado actual  $S_t$  de acuerdo al estado anterior  $S_{t-1}$  y la observación actual  $O_t$ , y los símbolos de salida  $U_t$ . El proceso se realiza hasta llegar al objetivo (la intensidad luminosa  $I_t$  sea mayor a un umbral  $\epsilon$ ) o se realice un número determinado máximo de transiciones  $t_{max}$ , la ejecución se resume con el algoritmo



siguiente:

$t = 1$   
 $O_1 =$  lectura cuantificada de los sensores  
 $\delta_1(i) = \pi_i b_i(O_1), 1 < i < N$   
 $S_1 = j = \arg \max_{1 < i < N} [\delta_1(i)]$   
 $U_1 =$  Acción( $S_1, C$ )  
 MIENTRAS  $I_t > \epsilon$  O  $t < t_{max}$   
      $t = t + 1$   
      $O_t =$  lectura cuantificada de los sensores  
      $\delta_t(j) = \max_{1 < i < N} [\delta_t - 1(i) a_{ij}] b_j(O_t), 1 < j < N$   
      $S_t = j = \arg \max_{1 < i < N} [\delta_t(i)]$   
      $U_t =$  Acción( $S_t, C$ )  
 FIN

### 5.1.6. Representación como proceso de decisión de Márkov

De manera similar a lo mostrado en la sección 2.4.3, al desconocer el entorno de manera global, se busca obtener la política óptima para el entorno local (observado por los sensores del robot) y una vez realizado cierto número de pasos volver a calcular la política óptima para este nuevo entorno de manera sucesiva hasta llegar al objetivo.

El entorno local se representa como una cuadrícula de  $6 \times 6$  donde cada cuadrícula representa un tercio del valor máximo de los sensores láser de distancia, cada celda representa un estado  $s \in S$ . El robot puede realizar una de 8 acciones posibles en cada celda  $A = \{\text{Detenerse, Avanzar, Retroceder, Girar a la izquierda (45°), Girar a la Derecha (-45°), Girar a la izquierda y avanzar (45°), Girar a la derecha (-45°) y avanzar, Girar doble a la derecha y avanzar (90°)}\}$ . El robot queda representado por sus 8 matrices de probabilidades de transición  $P(s'|s, a)$ , cada una con 9 valores de probabilidad (los cuales utilizan 8 bits cada uno), siendo la longitud total del individuo  $8 \times 9 \times 8 = 576$  bits.

### 5.1.7. Ejecución del proceso de decisión de Márkov que representa al robot

Adicional a las matrices de probabilidades de transición, el MDP requiere una función de  $R(s', a)$  la cual está definida como:

$$R(S) = \begin{cases} 2.00 & \text{Dirección donde se encuentra la fuente luminosa} \\ & \text{entorno observado} \\ -0.04 & \text{Si no hay obstáculo} \\ & \text{entorno observado} \\ -1.00 & \text{Si hay obstáculo} \end{cases}$$

La ejecución del MDP se realiza siguiendo los siguientes pasos:

1. Discretizar entorno: a través de las lecturas de los sensores se obtiene el mapa de  $6 \times 6$  de ocupación, y se asigna el valor de  $R(s)$  para cada casilla.
2. Función de recompensa: asignar los valores de recompensa a cada casilla de acuerdo a la función  $R(s)$
3. Calcular política óptima  $\pi^*(s)$ : utilizando el método de iteración por valor para 5 pasos. Realizar las acciones de acuerdo a esta política óptima.
4. Repetir: si no se ha llegado al objetivo final, regresar al paso 1.

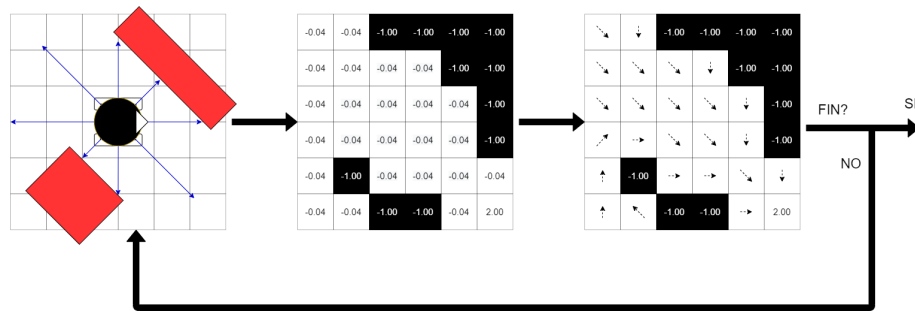


Figura 5.2: Procedimiento ejecución MDP

## 5.2. Optimización utilizando algoritmos genéticos

### 5.2.1. Evaluación de los individuos

La función de aptitud se utiliza para comparar el desempeño de los individuos en los algoritmos genéticos, siendo los individuos con mejor aptitud aquellos mejor adaptados y con mayor éxito reproductivo. Por lo que la definición de la función de aptitud es fundamental para obtener comportamientos con las características deseadas, ya que si la definición de la función de aptitud no concuerda con la problemática a resolver los individuos sin importar la complejidad del modelo, el número de generaciones y los operadores de cruce y mutación entre otras herramientas, estará buscando soluciones en una dirección equivocada.

#### Entorno simulado

Para la problemática de navegación reactiva se planteó la siguiente función de aptitud:

$$Aptitud = K_1 * N * D_o + K_2 * \frac{N}{D_d} + K_3 * S_d$$

La cual se divide en 3 términos cada una con su correspondiente constante  $K_1$ ,  $K_2$  y  $K_3$  las cuales se ajustaron empíricamente de acuerdo a la importancia de cada término. Cada término representa un objetivo que se desea optimizar durante la navegación

- $K_1 * N * D_o$ : Se busca que el robot llegue al punto objetivo en el menor número de pasos posibles, para evaluar esto se tiene  $N = N_k - N_s + 1$ , donde  $N_k$  es el número máximo de pasos y  $N_s$  el número de pasos requeridos para llegar al objetivo, en caso de no llegar al objetivo se tiene  $N_s = N_k$  por lo que el término  $+1$  sirve para que no sea cero en estos casos.  $D_o = \|X_o - X\|$  es la distancia entre el punto inicial  $X_o$  y el punto final del robot  $X$ , ya que se busca que el robot tenga una navegación activa.
- $K_2 * \frac{N}{D_d}$ :  $N$  al igual que en el término anterior representa la diferencia entre el número de pasos máximos y los requeridos para llegar a la posición objetivo,  $D_d$  es la distancia entre la última posición del robot  $X$  y la posición objetivo  $X_d$  más (en el caso del simulador) la distancia mas corta utilizando el algoritmo de Dijkstra sobre un mapa topológico, esta distancia  $DistDijkstra(X, X_d)$  se utiliza para que el robot aprenda a rodear obstáculos cóncavos donde se tiene un mínimo local si no se penalizan los comportamientos que no rodeen esta clase de obstáculos.

$$D_d = \|X - X_d\| + DistDijkstra(X, X_d)$$

$D_d$  divide a  $N$  pues se busca minimizar la distancia entre la posición final y la posición objetivo.

- $K_3 * S_d$ :  $S_d$  es la desviación estándar de las posiciones  $X_i$  del robot entre la posición inicial  $X_o$  y la posición final alcanzada  $X$ , en caso de caer en un mínimo local este valor tiende a ser pequeño penalizando los comportamientos con estas características.

## Entorno real

En el robot real los valores de  $D_o$ ,  $D_d$  y  $S_d$ , dado que solo se conoce el entorno inmediato, se estiman mediante la intensidad luminosa máxima en la posición inicial ( $I_o$ ), en la posición final ( $I$ ) y el valor máximo ( $I_{max}$ ), siendo:

$$\begin{aligned} D_o &= g(I_o) - g(I) \\ D_d &= g(I) - g(I_{max}) \\ S_d &= S_I \end{aligned}$$

Donde  $g(x) = \frac{1}{x}$  es una función que mapea del espacio de intensidades luminosas al espacio de distancia proyectada sobre la recta que une el punto inicial y el punto final,  $S_I$  es la desviación estándar de los valores de  $g(I_t)$  para cada paso  $t$ .

### 5.2.2. Algoritmo Genético

Como se mencionó en la sección 3.2 los algoritmos genéticos siguen un esquema general con ciertas adecuaciones de acuerdo al problema en el que se apliquen, el ciclo y las peculiaridades de cada etapa del algoritmo genético utilizado se listan a continuación:

1. *Generación de la población inicial:* la población inicial se genera de manera aleatoria con una distribución uniforme. La población cuenta con  $L$  individuos  $B_1, B_2, \dots, B_L$ , donde cada individuo es representado con una cadena de bits de una longitud fija de acuerdo al modelo utilizado  $B_i = \{01011 \dots 11011\}$ . Cada segmento de la cadena de bits representa un parámetro del modelo.
2. *Evaluación:* Las cadenas de bits que representan a cada individuo se convierten en los parámetros del modelo, se realiza la ejecución del modelo y se evalúa de acuerdo a la función de aptitud descrita en la sección anterior. La ejecución se realiza en distintos mapas, considerando distintos puntos iniciales y finales, siendo la aptitud final un promedio de la aptitud de las distintas ejecuciones.
3. *Selección, Recombinación y Mutación:* Se utiliza un método de selección determinista, con cruza anular, mutación uniforme y elitismo total, llamado algoritmo genético de Vasconcelos [8]. Este método consiste del siguiente procedimiento:
  - a) Se ordena la población actual en orden descendente de acuerdo a su aptitud  $P^t = \{B_1^t, B_2^t, \dots, B_L^t\}$ .
  - b) Se emparejan los individuos de la primer mitad de  $P^t$   $\{B_1^t, B_2^t, \dots, B_{L/2}^t\}$  con la segunda mitad de  $P^t$  invertida  $\{B_L^t, B_{L-1}^t, \dots, B_{L/2+1}^t\}$ , siendo las parejas  $\{(B_1^t, B_L^t), (B_2^t, B_{L-1}^t), \dots, (B_{L/2}^t, B_{L/2+1}^t)\}$
  - c) Para cada pareja se genera un número aleatorio  $r \in [0, 1)$  y si es menor a la probabilidad de cruza  $P_c$  se realiza la cruza anular en caso contrario no se modifican los individuos, intercambiando una sección  $\{a_{z_1}, a_{z_1+1}, \dots, a_{z_2}\}$  de cada individuo, donde  $1 \leq z_1 < z_2 \leq n$ ,  $n$  es el número de bits del individuo,  $z_1$  y  $z_2$  se generan aleatoriamente. La cruza anular se describe a continuación:

Individuos Seleccionados:

$$B_i^t = \{a_1, a_2, \dots, a_n\}$$

$$B_{L+1-i}^t = \{b_1, b_2, \dots, b_n\}$$

SI  $r < P_c$ :

$$z_1 = x \text{ y } z_2 = y$$

Nuevos individuos:

$$B_i^{t+1} = \{a_1, \dots, a_{x-1}, b_x, b_{x+1}, \dots, b_{y-1}, b_y, a_{y+1}, a_{y+2}, \dots, a_n\}$$

$$B_{L+1-i}^{t+1} = \{b_1, \dots, b_{x-1}, a_x, a_{x+1}, \dots, a_{y-1}, a_y, b_{y+1}, b_{y+2}, \dots, b_n\}$$

SINO:

Nuevos individuos:

$$B_i^{t+1} = B_i^t$$

$$B_{L+1-i}^{t+1} = B_{L+1-i}^t$$

La mutación se realiza de acuerdo a una probabilidad  $P_m$ , de manera similar a la cruce se genera un número aleatorio  $r \in [0, 1)$  y si es menor a la probabilidad de mutación  $P_m$  se realiza la mutación, esta mutación se realiza en una cantidad fija de bits para cada individuo, dichos bits se eligen aleatoriamente y se intercambia su valor por su complemento:

Individuo original:

$$B_i^t = \{a_1, a_2, \dots, a_x, \dots, a_n\}$$

SI  $r < P_m$ :

Bit a mutar =  $x$

Nuevo individuo:

$$B_i^t = \{a_1, a_2, \dots, \bar{a}_x, \dots, a_n\}$$

SINO:

Nuevo individuo:

$$B_i^t = B_i^t$$

El elitismo total se da al considerar la población anterior  $P^t$  en conjunto con sus descendientes  $P_o^{t+1}$  durante la evaluación, conservando únicamente los  $L$  mejores individuos de la unión de ambas poblaciones.  $P^{t+1} = L_{BEST}(P^t \cup P_o^{t+1})$ .

- d) *Iteración*: La evaluación, selección, recombinación y mutación se repiten hasta cumplir una condición de paro en nuestro caso cumplir un número fijo de generaciones.



## Capítulo 6

# Pruebas y resultados

En esta capítulo se describen las distintas pruebas realizadas tanto en los entornos virtuales (Simulador Tk y Simulador ROS) como en el entorno real. Se muestran los resultados obtenidos en las pruebas, así como la justificación de las pruebas adicionales dados los resultados de las primeras pruebas. Las distintas etapas de pruebas que se realizaron se muestran en la Figura 6.1, donde se tienen 3 etapas:

1. Pruebas simulador Tk: se realizó la evolución de los 3 distintos comportamientos (FSM, EHMM y MDP) y considerando tanto el ruido del robot como el ruido del robot amplificado, teniendo 6 modelos distintos. Se utilizó una población extensa y un largo número de generaciones.
2. Pruebas simulador ROS: para los 6 modelos de la última generación de la etapa previa se realiza un re-entrenamiento con los mejores individuos en un entorno similar al real. Se utilizó una población reducida y un corto número de generaciones.
3. Pruebas robot real: se realizaron pruebas en el robot real para los 6 modelos obtenidos del simulador Tk y el simulador ROS, teniendo un total de 12 modelos.

### 6.1. Pruebas simulador Tk

El simulador Tk es el primer paso de la propuesta, donde se comienza con una población inicial aleatoria, el entrenamiento consistió en dos fases dados los resultados preliminares: entrenamiento utilizando el modelo del ruido del robot y el entrenamiento utilizando un mayor nivel de ruido.

#### 6.1.1. Parámetros pruebas

Para las pruebas del simulador Tk en los 3 modelos se utilizaron los parámetros siguientes para cada uno de los 3 comportamientos (FSM, EHMM y MDP):

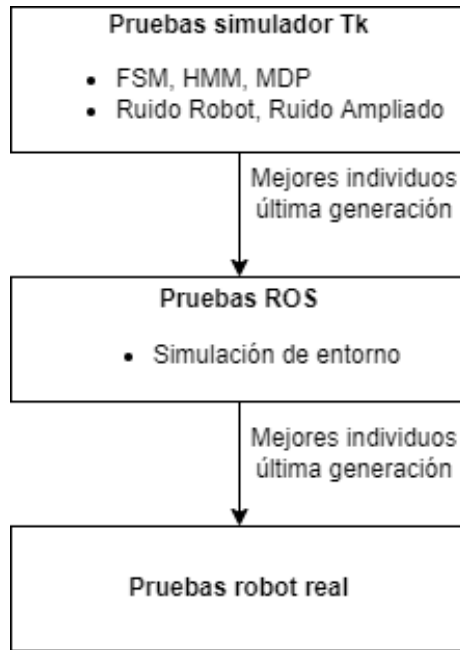


Figura 6.1: Etapas de pruebas realizadas

- 900 Generaciones
- 100 Individuos
- $P_c = 0.9$
- $P_m = 0.6/L$
- elitismo 50%
- Número de mapas = 13
- Número de ejecuciones por mapa = 4
- Número de pasos por ejecución = 300

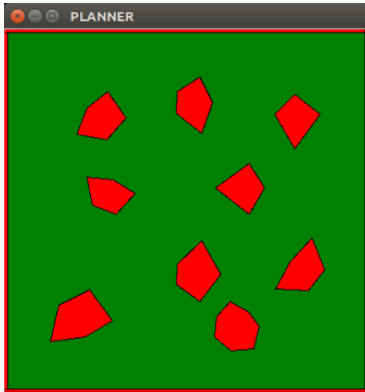
Las características y la longitud  $L$  de los individuos para cada modelo se describen en las secciones 5.1.2 (FSM), 5.1.4 (HMM) y 5.1.6 (MDP).

### 6.1.2. Ambiente simulado

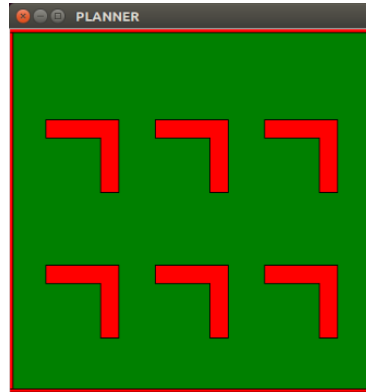
Dentro de los mapas utilizados para el entrenamiento se tienen dos tipos distintos de mapas: poligonales (Fig. 6.2a) y con paredes (Fig. 6.2b). Los mapas poligonales como su nombre lo indica se componen de obstáculos poligonales en



su mayoría convexos, los mapas con paredes contienen estructuras en forma de  $L$  e  $I$  las cuales simulan muros.



(a) Ejemplo de mapa poligonal



(b) Ejemplo de mapa con paredes

Figura 6.2: Ejemplos de mapas de entrenamiento simulador Tk

### 6.1.3. Resultados

Los valores de aptitud obtenidos utilizando el modelo de ruido del robot real se muestran en la Figura 6.3. Donde se observa que el mayor valor de aptitud en el entorno simulado se logra con el modelo MDP con un aptitud máxima de 2993, seguido del modelo FSM con 2680 y por último el modelo HMM 2046.

Los valores de aptitud obtenidos utilizando el modelo de ruido amplificado se muestran en la Figura 6.4 donde para los modelos FSM Y HMM se tiene una mejora de aptitud teniendo una aptitud máxima de 2972 y 2470 respectivamente, aunque existe una mayor varianza entre los valores de aptitud, sin embargo ponderando los valores de las últimas 20 generaciones se obtienen valores de aptitud de 2823 para FSM y de 2207 para HMM los cuales son superiores a los obtenidos con el modelo de ruido del robot.

En el Cuadro 6.1 se resumen los resultados obtenidos, considerando el mejor valor de aptitud para cada modelo, el promedio y la desviación estándar de las últimas 20 generaciones.

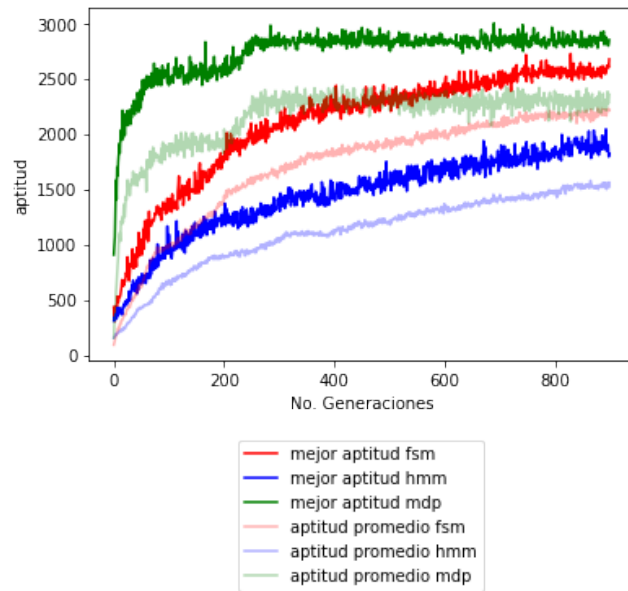


Figura 6.3: Aptitud entrenamiento Simulador Tk con modelo ruido robot real

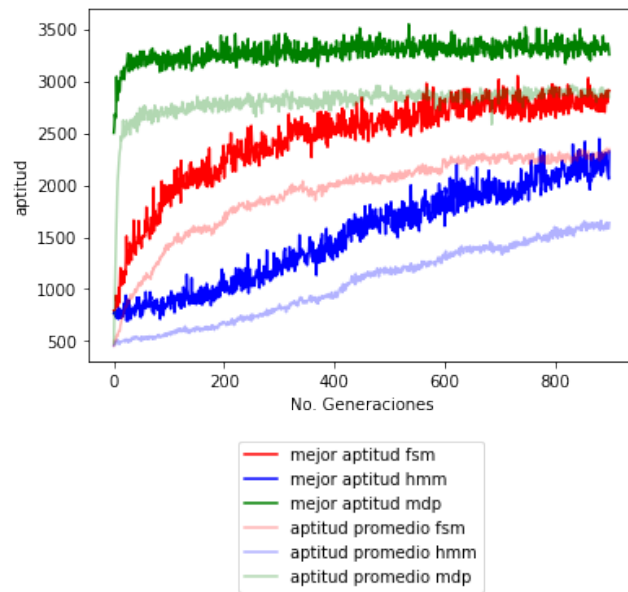


Figura 6.4: Aptitud entrenamiento Simulador Tk con modelo de ruido amplificado

Modelo	mejor aptitud	promedio 20 gen	dev std
FSM ruido robot	2680.73	2588.72	37.89
FSM ruido amplificado	2962.04	2823.78	72.4919
HMM ruido robot	2046.93	1891.07	60.59
HMM ruido amplificado	2470.43	2207.81	102.44
MDP ruido robot	2933.27	2863.42	39.08
MDP ruido amplificado	3440.27	3324.41	39.29

Cuadro 6.1: Resumen de pruebas en el simulador Tk

## 6.2. Pruebas simulador ROS

Inicialmente se realizaron pruebas en el entorno real considerando una población inicial compuesta de los mejores 10 individuos obtenidos mediante el simulador Tk, realizando 20 generaciones con estos. Sin embargo al realizar las pruebas en el entorno real, los valores de aptitud obtenidos eran volátiles y los resultados no significativos dada la aleatoriedad del proceso y el pequeño número de pruebas realizadas con cada individuo, por lo cual se consideró necesario un proceso de re-entrenamiento más extenso, dada la dificultad para realizar este re-entrenamiento en el entorno real se decidió realizarlo en un segundo *Simulador ROS*, donde el comportamiento del robot (ruido) es ligeramente distinto y se busca que los comportamientos derivados en el primer *Simulador Tk* se adecuen a este nuevo simulador. Se busca que los comportamientos sean robustos inicialmente y que esto permita adaptarlos al nuevo entorno, para esto se realizó inicialmente un análisis de diversidad genética entre los 16 mejores individuos de cada modelo obteniendo los resultados obtenidos en el Cuadro 6.2, donde se considera la similitud promedio al compararse todos los individuos entre si. Esperando que entre mayor sea esta diversidad inicial mas robusta sea la población para adecuarse a un nuevo entorno.

Comportamiento	Promedio similitud entre individuos
FSM ruido robot	97.78 %
FSM ruido amplificado	83.77 %
HMM ruido robot	58.54 %
HMM ruido amplificado	44.66 %
MDP ruido robot	67.75 %
MDP ruido amplificado	40.49 %

Cuadro 6.2: Similitud entre la población final del simulador Tk

Los resultados del Cuadro 6.2 muestran un incremento en la diversidad genética al incrementar el ruido.

### 6.3. Parámetros de pruebas

Para el re-entrenamiento se generó en el simulador un ambiente similar al entorno real (Fig 6.5), donde se realizaron 6 ejecuciones por individuo, considerando los puntos iniciales y finales mostrados en la Figura 6.6. Además se utilizó el algoritmo genético y la función de aptitud descrito en la sección 5.2 en cual es una adecuación de la función original donde no se tiene conocimiento a priori de las distancias reales y estas se calculan a través de los sensores de intensidad luminosa.

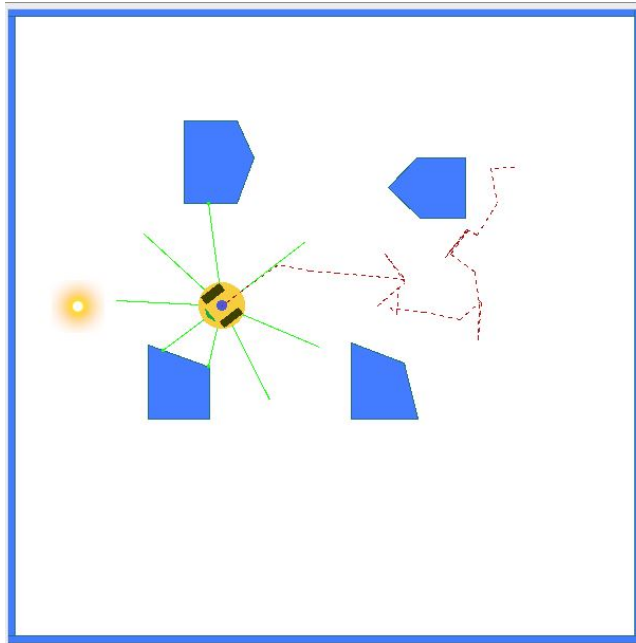


Figura 6.5: Entorno de re-entrenamiento en simulador ROS

Los parámetros de las pruebas fueron los siguientes:

- 20 Generaciones
- 16 individuos
- $P_c = 0.9$
- $P_m = 0.6/L$
- elitismo mejores 2 individuos
- Número de ejecuciones por mapa = 6
- Número de pasos por ejecución = 100

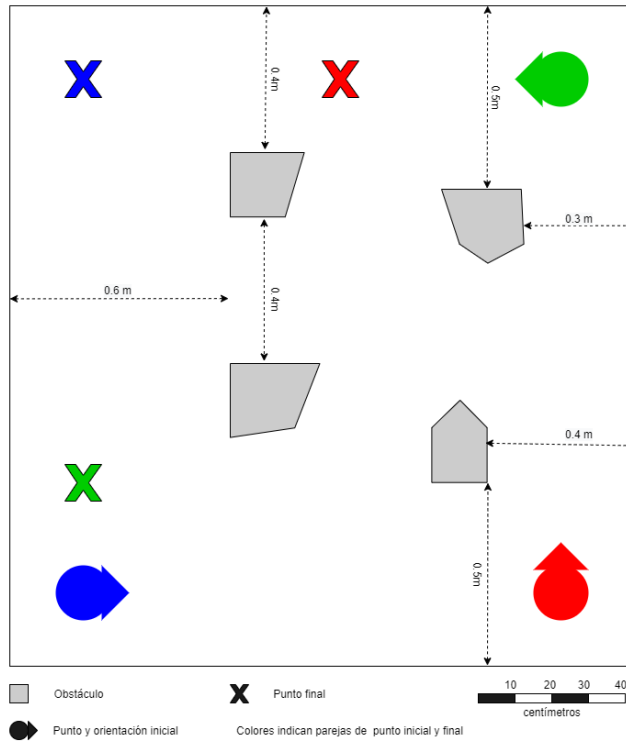


Figura 6.6: Mapa real

### 6.3.1. Resultados

A pesar del incremento tanto en generaciones como en el tamaño de la población, los resultados tampoco fueron significativos con estas pruebas y no se tuvo una mejora considerable de la aptitud a través del tiempo, sin embargo los comportamientos previamente derivados tuvieron un desempeño bastante aceptable en este nuevo simulador, los resultados obtenidos se resumen en el Cuadro 6.3, los resultados de este cuadro nos indican que en general los comportamientos con ruido amplificado tuvieron un mejor desempeño que sus contrapartes con el ruido del robot, por otro lado la mejor aptitud se obtuvo con el comportamiento *HMM con ruido amplificado*, aunque los comportamientos de FSM tuvieron un desempeño no muy inferior a los obtenidos por los HMM.

Comportamiento	mejor aptitud	promedio 20 gen
FSM ruido robot	3308.03	2922.28
FSM ruido amplificado	3542.97	3506.68
HMM ruido robot	3546.23	3324.90
HMM ruido amplificado	3552.15	3518.81
MDP ruido robot	2958.56	2280.84
MDP ruido amplificado	1950.19	1551.37

Cuadro 6.3: Resultados finales simulador ROS

Además de los resultados mostrados en el Cuadro 6.3 se observaron ciertos comportamientos peculiares para cada modelo, los cuales se describen a continuación:

### Máquinas de Estado Finito

Los modelos FSM al no contar con variables aleatorias, tienen un comportamiento determinista por lo que existe cierta posibilidad de existir bucles de estados (mínimos locales) donde el modelo caiga y no pueda salir mientras no reciba algún estímulo adicional exterior, estos bucles pueden ser de distintos tamaños un estado, dos estados, tres estados, etc. En la Figura 6.8c se tiene un ejemplo de una ejecución de un modelo FSM donde el robot se queda atorado en un bucle y no llega al destino final. Estos casos no son tan comunes sin embargo si tienen un efecto negativo en la aptitud del individuo pues el robot, en caso de entrar en algún bucle, no llegará al punto objetivo en ningún momento. En la Figura 6.7a se tiene un caso más común y deseable donde el robot llega al destino de una manera rápida y adecuada.



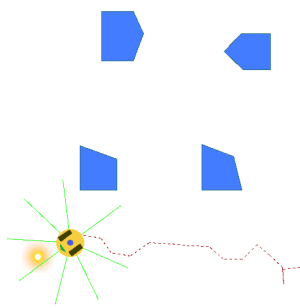
(a) Ejemplo de ejecución FSM que llega al destino

(b) Ejemplo de ejecución FSM que se queda en mínimo local

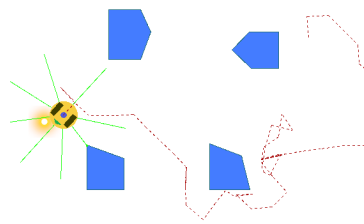
Figura 6.7: Ejemplos de ejecuciones de modelos FSM

## Modelo de Márkov Oculto Extendido

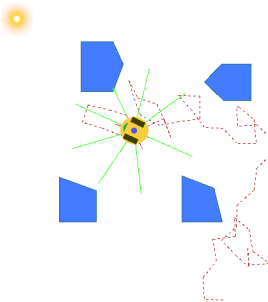
Los modelos HMM al contar con variables aleatorias no sufren del problema de los mínimos locales, causados por bucles en los estados de los modelos FSM comentados previamente, sin embargo surge otra problemática en sentido opuesto: la sobre exploración, donde se tienen casos como el mostrado en la Figura 6.8c donde el robot no llega al objetivo pues pasa gran parte de la navegación explorando en la periferia de la ruta óptima. Sin embargo este comportamiento no siempre es tan drástico y permite en la mayoría de las ocasiones lograr llegar al objetivo en el tiempo estipulado, como se observa en la Figura 6.8b e incluso, en menor medida, llegar de la misma manera o más rápido que los modelos FSM (Fig. 6.8a). Esta exploración es un comportamiento deseable si se desea librar mínimos locales o si se desea obtener un conocimiento del entorno al realizar la navegación.



(a) Ejemplo de ejecución HMM similar a FSM



(b) Ejemplo de ejecución HMM con exploración adicional



(c) Ejemplo de ejecución HMM que no llegó al objetivo

Figura 6.8: Ejemplos de ejecuciones de modelos HMM

## Proceso de Decisión de Márkov

Los modelo MDP, a diferencia de los anteriores, tuvo un desempeño muy inferior al obtenido en el primer simulador, pasando de ser los modelos con el mayor valor de aptitud a ser los modelos con el peor desempeño. En la Figura 6.9 se tiene una ejecución de un modelo MDP donde se puede observar que, si bien siempre se dirige hacia la fuente de luz, tiene grandes problemas al momento de esquivar los obstáculos por lo que en la mayoría de las ejecuciones no logró llegar al punto objetivo. Lo que parecería indicar que las políticas óptimas obtenidas por el MDP son bastantes sensibles a ligeras modificaciones en el entorno.

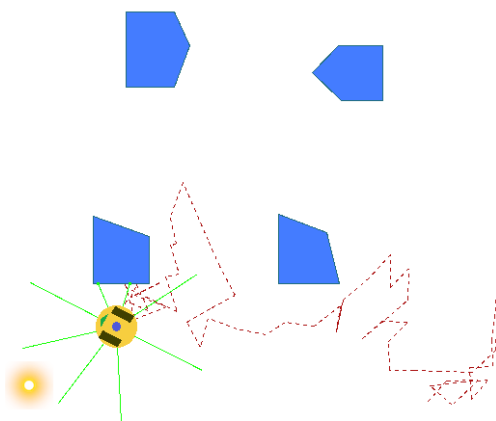


Figura 6.9: Ejemplo de ejecución de modelo MDP

## 6.4. Pruebas robot real

Inicialmente se buscaba evolucionar los comportamientos en el entorno real pero dadas complicaciones y resultados no concluyentes con una pequeña población y pocas ejecuciones se realizó el proceso descrito en la sección anterior.

En esta sección se realizaron las ejecuciones de los comportamientos evolucionados en los simuladores y se compararon sus desempeños (aptitudes) en este nuevo entorno, utilizando el robot real. Las pruebas se realizaron en un entorno de 1.8 [m]  $\times$  1.7 [m], con 4 obstáculos como se observa en la Figura 6.6. Para cada individuo se realizaron 3 ejecuciones partiendo de distintos puntos iniciales y dirigiéndose a distintos puntos finales, dichos pares se presentan con distintos colores en la Figura 6.6.





Figura 6.10: Robot en entorno real

### 6.4.1. Resultados

Los resultados obtenidos se concentran en el Cuadro 6.4, de las ejecuciones realizadas para los tres comportamientos el que mejor desempeño tuvo fueron los modelos FSM, seguidos por los HMM y por último nuevamente los modelos MDP tuvieron un comportamiento muy inferior respecto a los otros.

Los modelos FSM a pesar de haber obtenido los valores más altos de aptitud, continuaron mostrando el efecto de los mínimos locales, mencionado en las pruebas del simulador ROS, el cual se intensificó y se tuvo un mayor número de pruebas fallidas a causa de esto, teniendo algunos individuos una aptitud muy baja, los cuales igualmente son contrarrestados con individuos con una aptitud elevada ya que no sufrían de estos efectos. En cuanto a los modelos HMM igualmente el efecto adverso que conlleva la exploración a causa de su naturaleza estocástica se intensificó, por lo que igualmente se tuvo una mayor cantidad de ejecuciones fallidas, donde si bien el robot se dirigía en medida a la fuente luminosa, este no llegaba al punto objetivo en el tiempo requerido.

Comportamiento	Aptitud Mejor Individuo	Aptitud Promedio
FSM ruido robot	3854	1567
FSM ruido amplificado	2902	1123
HMM ruido robot	2815	954
HMM ruido amplificado	3276	1043
MDP ruido robot	904	412
MDP ruido amplificado	1264	525

Cuadro 6.4: Resumen de pruebas en el robot real



## Capítulo 7

# Conclusiones y trabajo futuro

### 7.1. Conclusiones

Partiendo del objetivo principal de este trabajo de investigación:

*Optimizar comportamientos reactivos estocásticos para la navegación y evasión de obstáculos de un robot móvil en un entorno desconocido*

Se logró la representación del robot móvil tanto como una máquina de estado finito (FSM), un modelo oculto de Márkov extendido (HMM) y como un proceso de decisión de Márkov (MDP). También se realizó un modelado del nivel de ruido de sus sensores.

La optimización inicial en el *simulador Tk*, dio los resultados deseados, obteniendo una mejora de la aptitud a través del tiempo para los tres modelos. Con respecto a este simulador los modelos MDP tuvieron una convergencia más rápida y un mejor desempeño global, seguido por los modelos FSM y por último los modelos HMM.

Al realizar la transferencia de los modelos al robot real e intentar realizar una fase de re-entrenamiento se identificó una gran complejidad dado a que los valores de aptitud eran volátiles y por lo tanto la pequeña muestra de ejecuciones, que era posible obtener en un tiempo razonable entrenando al robot real, no generaba resultados significativos. Posteriormente se optó por un segundo enfoque, realizar la transferencia a un segundo simulador *simulador ROS*, realizando un entrenamiento más extenso (sin ser comparable al entrenamiento inicial) que el propuesto inicialmente en el robot real en busca de alguna mejora en la aptitud de los individuos, sin embargo tampoco se obtuvieron mejoras significativas en la aptitud de estos individuos, aunque el comportamiento generado inicialmente en el *simulador Tk* tuvo un buen desempeño también en este segundo simulador

con la excepción de los modelos MDP que tuvieron un desempeño considerablemente bajo pues pasaron de ser los comportamientos con mayor aptitud en el *simulador Tk* a los de peor desempeño en el *simulador ROS*.

Otro factor que se consideró que podría ser la causa de la falta de mejora en la aptitud en el segundo simulador (además de la pequeña población y pocas generaciones por evolucionar) fue la falta de diversidad genética, en busca de ampliar esta se optó por realizar una segunda fase de generación de comportamientos en el primer simulador, pero esta vez con un mayor nivel de ruido, bajo la premisa de que se obtendrían comportamientos más generales. Los resultados de esta segunda generación de comportamientos mostraron una mayor diversidad genética para los 3 distintos modelos y al realizar el re-entrenamiento en el *simulador ROS*, si bien tampoco se tuvo una mejora del valor de aptitud, si se tuvieron en su mayoría mejores individuos, por lo que este breve incremento del ruido del robot (y por ende diversidad genética) tuvo un efecto positivo para tener comportamientos mas generales que se puedan adecuar a una mayor cantidad de ambientes.

Finalmente se probaron los comportamientos obtenidos con el robot real, donde no todos los individuos tenían un comportamiento adecuado, sin embargo para los modelos FSM y HMM se tuvieron individuos dentro de estas generaciones finales que se comportaron de una manera adecuada al navegar en el entorno real, sin embargo los comportamientos adversos (para FSM los ciclos de estados *muertos* o mínimos locales y para los HMM la *sobre exploración* que no permite llegar al punto objetivo en el tiempo deseado) se incrementaron.

Por último queda destacar que si bien no se logró brincar la brecha de la realidad de una manera completa si se lograron avances en cuanto a los resultados obtenidos respecto a la importancia del ruido y como afecta este la diversidad genética de los algoritmos genéticos y además se obtuvieron individuos que funcionaron de una manera adecuada en el entorno real.

## 7.2. Trabajo futuro

Al analizar los resultados obtenidos se han detectado distintas actividades futuras, las cuales ampliarían el alcance de este trabajo de investigación. Estas actividades son:

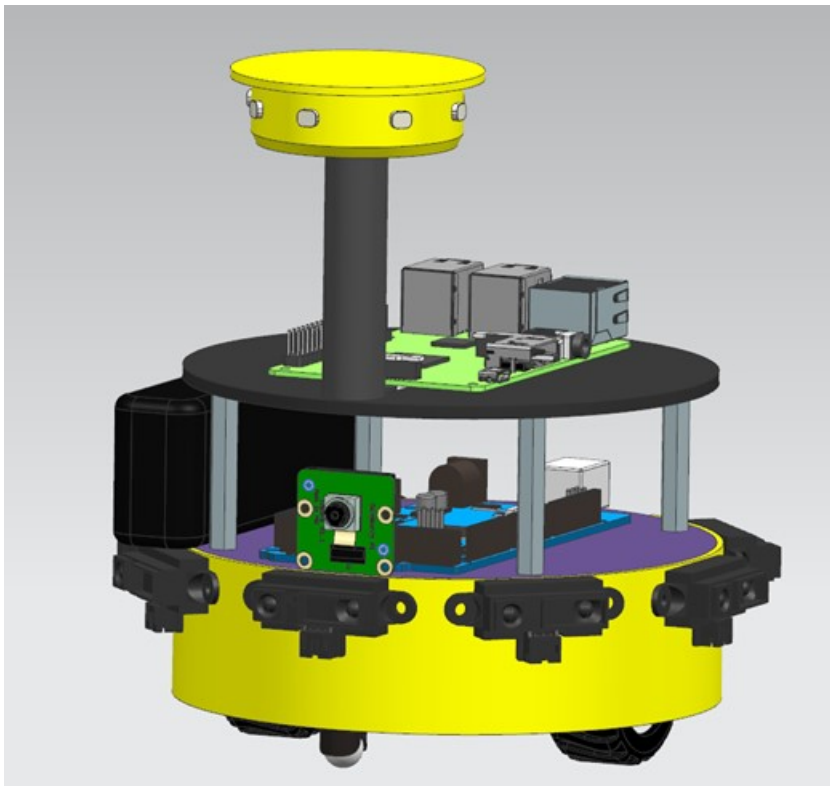
- **Automatización del proceso de pruebas en el robot real:** Las pruebas en el robot real fueron una de las complicaciones que se encontraron al realizar este trabajo de investigación principalmente por las siguiente causa: *tiempo de ejecución*, esta problemática si bien no se puede mitigar, si se puede reducir en lo que consiste al menos en la intervención del investigador al realizar estas pruebas. Para reducir el impacto de esta actividad se propone la automatización del proceso de pruebas del robot real de manera que se reduzca la participación humana en este proceso y se pueda controlar mejor el entorno de ejecución.

- **Optimización de la representación de los modelos HMM:** En los modelos HMM se tiene una convergencia lenta en el primer simulador y esto se debe principalmente a la longitud de sus individuos, si se compara la cantidad de bits requeridas para representar un HMM con los otros modelos (FSM y MDP), nos damos cuenta de que es mucho mayor por lo que es posible quede mucho espacio de búsqueda por explorar al realizar los algoritmos genéticos y esto impacte negativamente el desempeño, las alternativas de solución serían, buscar e implementar alguna representación compacta de EHMM donde los símbolos de salida solo dependan del estado actual o del símbolo de entrada y no de ambos y adicional a esto, es posible que alguna implementación de algoritmo genético real en vez de binario ayude a mejorar la convergencia.
- **Simulador más completo:** una de las principales dificultades para *saltar* la brecha de la realidad es la discrepancia que existe entre simulador y realidad, esta discrepancia se acorta al tener simuladores más complejos y similares a la realidad, por lo que una mejora futura sería utilizar un simulador con un modelos más complejo (y completo) del robot y que no solo considere su ruido normalizado y su cinemática básica, adicional se podría considerar un entorno que permita el uso de la cámara del robot para complementar la información obtenida mediante sus demás sensores (distancia y luminosos.)



## Apéndice A

# Minibot Educativo - Diagrama Eléctrico



Documento realizado por: Selk-IA Algorithms

## A.1. Diagrama eléctrico

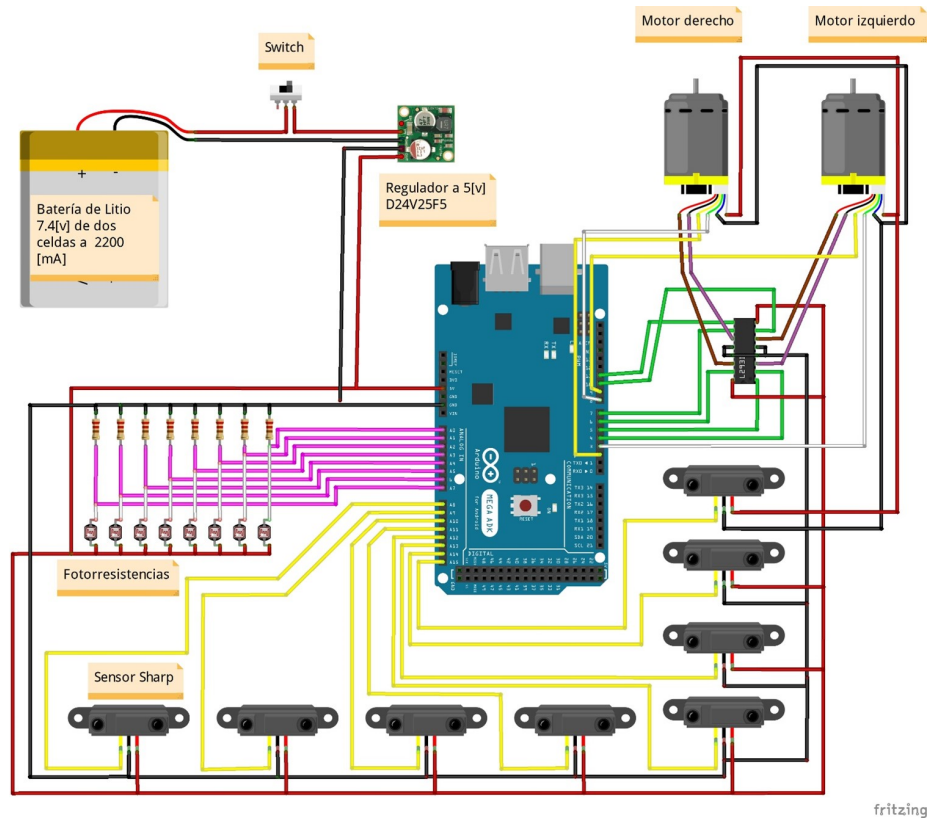


Figura A.1: Diagrama eléctrico

## A.2. Niveles

La estructura de este robot consta de dos niveles en cuanto a sus conexiones eléctricas.

El nivel inferior almacena el bloque de alimentación de todo el robot, así como los sistemas de control de actuadores y lectura de sensores.

En el nivel superior se encuentra lo que es el sistema de procesamiento de toda la información y comunicación con otros dispositivos.



### **A.2.1. Componentes principales del nivel inferior**

- Batería de 7.4[v]
- Regulador de voltaje 5[v] a 2.5 [A] D24V25F5
- Drive para motores o "Puente H" L293D
- Arduino MEGA
- 2 motorreductores de metal Pololu 100:1
- 2 encoders magnéticos para motorreductor metálico
- 8 fotorresistencias 1M[ohm]
- 8 sensores infrarojos Sharp de 10-80 cm

### **A.2.2. Componentes principales del nivel superior**

- Raspberry Pi versión 3 b+
- Cámara tipo Raspicam para Raspberry
- Tarjeta FPGA

## **A.3. Descripción**

### **A.3.1. Sistema de Alimentación**

La alimentación para los elementos de procesamiento de datos: Raspberry Pi 3 y Arduino MEGA requieren un voltaje regulado de 5[v]. Este voltaje es entregado por el regulador de voltaje hasta una tarjeta de circuitos impresos (PBC) el cual tiene forma de shield para conectarse con el Arduino. De ahí es posible hacer todas las conexiones con los sensores, los motores y la tarjeta Raspberry (en esta solo para la alimentación).

### **A.3.2. Conexión de los motores**

- 100:1 Micro Metal Gearmotor HPCB 12V with extended Motor Shaft  
Documentacion: <https://www.pololu.com/product/3052>

Con:

- Magnetic encoder pair kit with side-entry connector for micro metal gear-motors  
Documentacion: <https://www.pololu.com/product/4761>

Los cables de los encoders van conectados directamente a la Shield del Arduino para comunicarse directamente con éste (Fig. A.2).

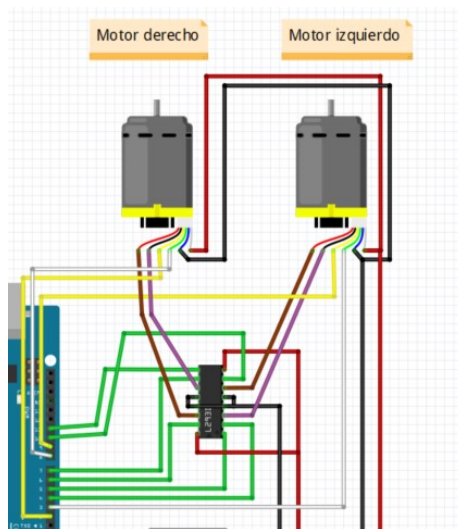


Figura A.2: Conexión de los motores con Arduino

La información de los cables que vienen desde los motores es la mostrada en la Figura A.3:



Figura A.3: Conexión de los motores

Las conexiones de los encoders y los motores para el arduino están detalladas en el Cuadro A.1.

Pin Arduino	Conexión y uso
2 (interrupción)	Encoder B, motor derecho
3 (interrupción)	Encoder A, motor izquierdo
4 (salida digital)	Salida de direccionamiento para motor izquierdo
5 (salida PWM)	Terminal para el habilitador <code>enable</code> del puente H para motor izquierdo
6 (salida digital)	Salida de direccionamiento para motor derecho
7 (salida digital)	Salida de direccionamiento para motor derecho
8 (entrada digital)	Encoder A, motor derecho
9 (entrada digital)	Encoder B, motor izquierdo
10 (salida digital)	Salida de direccionamiento para motor izquierdo
11 (salida PWM)	Terminal para el habilitador <code>enable</code> del puente H para motor derecho

Cuadro A.1: Tabla de conexiones encoders y motores

### A.3.3. Conexiones de las fotorresistencias

El robot cuenta con un arreglo de fotorresistencias en forma de anillo conectadas a las entradas con lecturas analógicas del Arduino. Estas permiten leer un voltaje de hasta 5[v] desde las resistencias, las entradas correspondientes de dichos sensores van de  $A_0, \dots, A_7$ .

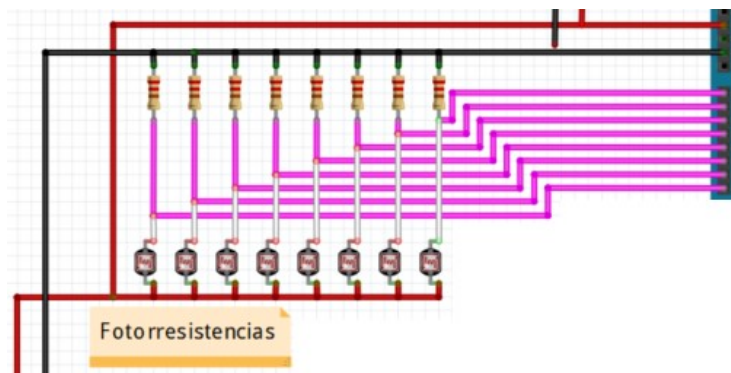


Figura A.4: Conexiones de las fotorresistencias

### A.3.4. Conexiones con los sensores infrarrojos Sharp

El robot cuenta con 8 sensores de distancia alrededor de sí, con una forma parecida a una corona. Las conexiones correspondientes a estos sensores van de  $A_8, \dots, A_{15}$  del Arduino (Fig. A.5).

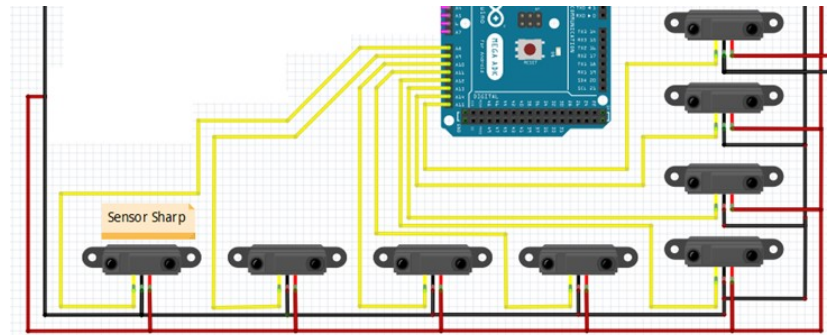


Figura A.5: Conexiones sensores infrarrojos

## Apéndice B

# Interconexión mediante ROS

### B.1. Interacción entre componentes

Para realizar la interacción entre los distintos dispositivos que conforman el robot (Raspberri Pi 3 y Arduino MEGA) y la comunicación con la computadora personal donde se tiene la interfaz gráfica donde se puede monitorear (lectura de los sensores, odometría) y modificar el comportamiento del robot (tipo de controlador, magnitud de avance y giro), se utiliza tanto ROS como conexiones físicas, en la Figura B.1 se tiene el diagrama generlr de comunicación entre los distintos componentes y los nodos de ROS que alberga cada uno.

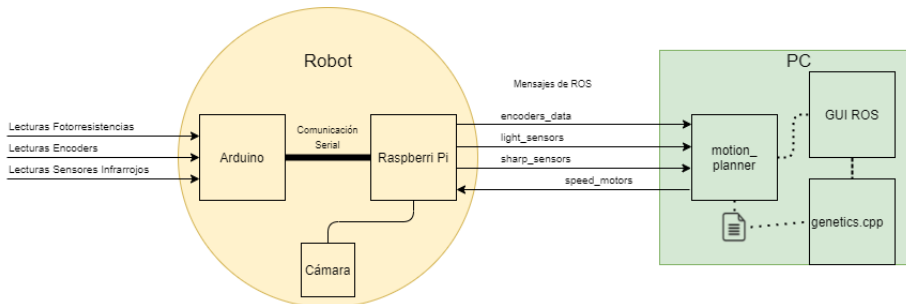


Figura B.1: Diagrama de comunicación entre componentes.

#### B.1.1. Comunicación robot - exterior

Los sensores (fotorresistencias, encoders y sensores infrarrojos) y los actuadores (salidas PWM de los motores) se conectan físicamente al Arduino. El

Arduino y la Raspberry Pi intercambian información mediante comunicación serial.

Las entradas son preprocesadas para que sus lecturas se adecuen a las unidades de medición: encoders en radianes, sensores infrarrojos en metros.

### B.1.2. Comunicación robot - PC

El controlador Raspberry Pi se comunica de manera inalámbrica con la computadora personal utilizando mensajes de ROS. Para acceder al controlador del robot se utiliza el protocolo de comunicación SSH.

Dentro de la computadora personal el nodo encargado del movimiento del robot es llamado *motion\_planner*, el cual es controlado a través de una interfaz gráfica (GUI). El programa encargado de realizar el algoritmo genético *genetics.cpp* es ejecutado desde la interfaz gráfica y este modifica los archivos que contienen los controladores con los que navega el controlador.

## B.2. Mensajes de ROS

Como se observa en la Figura B.1 se utilizan distintos mensajes de ROS para intercambiar información entre el controlador del robot y la computadora, a continuación se listan estos con una breve descripción de su contenido.

- **encoders\_data**: tipo: *std\_msgs/Int32MultiArray*  
Contiene las lecturas de los encoders que se utilizan para determinar el avance o giro del robot.
- **light\_sensors**: tipo: *std\_msgs/Int16MultiArray*  
Contiene las lecturas de los sensores de intensidad luminosa.
- **sharp\_sensors**: tipo: *std\_msgs/Int16MultiArray*  
Contiene las lecturas de distancia de los sensores sharp con valores entre 0 y 70 [cm].
- **speed\_motors**: tipo: *std\_msgs/float32MultiArray*  
Envía la velocidad de los motores para que el robot avance o gire.

## Apéndice C

# Guía de evolución de comportamientos

La evolución de los comportamientos se realiza mediante dos programas distintos:

- **Simulador Tk:** Permite la evolución de comportamientos en el simulador, se tiene un modelo del robot real y se evalúa su desempeño al simular la navegación en distintos mapas. Este programa permite evaluar distintos tipos de comportamientos y almacena los mejores individuos de cada generación, así como los valores históricos de aptitud durante el proceso de evolución.
- **Simulador ROS:** Este simulador se encuentra dentro de un entorno que permite la conexión con el robot real a través de ROS, en este simulador también se pueden evolucionar los comportamientos utilizando un modelo del robot real y entornos simulados, sin embargo su tiempo de ejecución es mucho mayor y requiere una mayor intervención del usuario. El objetivo de este simulador es tomar los comportamientos previamente evolucionados en el *Simulador Tk* y realizar un re-entrenamiento menos intensivo sobre el robot real.

### C.1. Evolución de comportamientos simulador Tk

#### C.1.1. Instalación del simulador Tk

1. Extraer el archivo `robotics.tar.gz` en el directorio base.
2. Abrir una terminal y cambiarse al directorio donde se encuentra el programa principal:

```
cd ~/robotics/motion_planner/
```

Compilar utilizando make:  
make

### C.1.2. Ejecución simulador Tk

Para acceder a la interfaz gráfica del simulador se debe ir al directorio donde esta se encuentra:

```
cd ~/robotics/gui/
```

Para ver la información de uso, dentro de la carpeta utilizar el comando:

```
python GUI_behaviors.py
```

El simulador se puede utilizar para dos funciones: *Probar Comportamientos* y

```
edujire@ubuntu:~/robotics/gui$ python GUI_behaviors.py
*****
Usage: python GUI_behaviors.py num_behavior
Where num_behavior:
0 = Tests Behaviors
1 = Stochastic FSM outputs
2 = HMM
3 = FSM
4 = Reactive
5 = Reactive Stochastic
6 = Neural Networks stochastic_outputs num_recurrent
   Example: python GUI_behaviors.py 6 0 0
   Example: python GUI_behaviors.py 6 1 0
   Example: python GUI_behaviors.py 6 0 4
   Example: python GUI_behaviors.py 6 1 4
7 = Potential fields num_bits_variables num_bits_fractions
   Example: python GUI_behaviors.py 7 16 8
   Example: python GUI_behaviors.py 7 8 7
8 = MDP
*****
```

Figura C.1: Información de uso

*Evolucionar Comportamientos*. En ambas se mostraran dos ventanas:

- **GUI\_robots**: Contiene entradas de texto y botones que permiten modificar la ejecución del programa (Figura C.2).

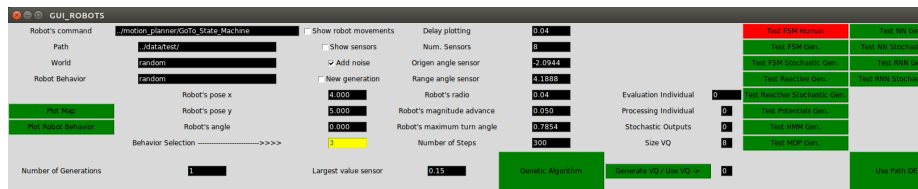


Figura C.2: ventana GUI\_robots

- **Planner**: Muestra el mapa y la navegación del robot (Figura C.3).

### Probar Comportamientos

Seleccionando la opción 0 se pueden probar los comportamientos previamente evolucionados, ejecutando el siguiente comando:



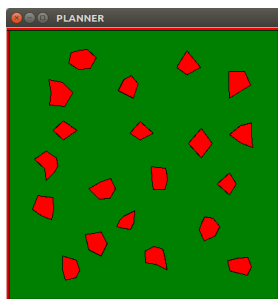


Figura C.3: ventana Planner

```
python GUI_behaviors.py 0
```

1. Seleccionar directorio: En el menú izquierdo se encuentra una entrada de texto donde se puede especificar el directorio en el cual se encuentran los archivos de los comportamientos, si se desea probar los comportamientos ya evolucionados que se tienen guardados en el simulador presionar el botón inferior derecho *Use Path DEMO*.
2. Seleccionar comportamiento: En la parte derecha de GUI\_robots se encuentran diversos botones que permiten elegir los distintos comportamientos disponibles.
3. Seleccionar mapa: Para cambiar de mapa se debe modificar las entradas de texto *World* y *Robot Behavior* con el nombre del mapa. Los mapas se encuentran en el mismo directorio que los comportamientos. Los mapas se encuentran en un archivo llamado `<nombre_del_mapa>.wrl` que contiene las dimensiones del mapa y los obstáculos que se encuentran en este.
4. Probar comportamiento: En Planner dar click izquierdo en el punto inicial del robot y click derecho en el punto final, el valor de aptitud se muestra en evaluation individual, con las opciones *Show robot movement* y *Show sensors* se modifica la visualización de navegación en la ventana Planner, con la opción *Add noise* se agrega ruido a la simulación.

### Evolucionar Comportamientos

1. Seleccionar comportamiento: Con las opciones 1–6 al ejecutar el programa (`python GUI_behaviors <opcion>`) se elige el tipo de comportamiento a evolucionar.
2. Cuantizador Vectorial: para los sensores de distancia se utiliza un cuantizador vectorial para discretizar las lecturas, al modificar la configuración de sensores es necesario crear el cuantizador vectorial, esto se realiza presionando el botón *Generate VQ / Use VQ* → , en caso de continuar la

evolución de un comportamiento es recomendable continuar utilizando los mismos vectores clave.

### 3. Parámetros de la simulación:

- Número de generaciones: En la entrada de texto *Number of Generations* se indica el número de generaciones a entrenar, para continuar evolucionando los comportamientos ya adquiridos debe estar desmarcada la casilla *New generation*, en caso de desear comenzar la evolución desde una población aleatoria marcar esta casilla.
  - Parámetros del robot: El simulador permite configurar distintos parámetros del robot simulado, como son: magnitud de avance, magnitud de giro, número de los sensores, máximo valor de los sensores, número de pasos, radio del robot por mencionar los principales.
  - Parámetros algoritmo genético: Estos parámetros se pueden modificar en el archivo `robotics/gui/initial_genetics_behaviors.py`. Algunos de estos parámetros son: número de individuos, factor de mutación, factor de cruce y número de pruebas.
4. Evolucionar individuos: Para comenzar la evolución una vez configurados los parámetros presionar el botón *Genetic Algorithm*. Para cada generación se imprime en consola el promedio, la mejor y la peor aptitud.

### 5. Archivos de salida: En cada generación se guardan los siguientes archivos:

- `avoid_<beh>_<ind>.dat`: Es el modelo de cada individuo de la última generación.
- `fitness_<beh>.dat`: Son los valores de aptitud de los individuos de la última generación.
- `best_<beh>_<gen>.dat`: Es el mejor individuo de cada generación.
- `worst_<beh>_<gen>.dat`: Es el peor individuo de cada generación.

Para cada comportamiento se tienen los siguientes archivos históricos:

- `fitness_best_<beh>.dat`: Histórico de la mejor aptitud.
- `fitness_worst_<beh>.dat`: Histórico de la peor aptitud.
- `fitness_average_<beh>.dat`: Histórico de la aptitud promedio.

Las etiquetas `<beh>`, `<ind>` y `<gen>` son remplazadas por los correspondientes comportamientos, número de individuo y número de generación respectivamente.

## C.2. Evolución de comportamientos ROS

### C.2.1. Instalación del simulador ROS

El simulador de ROS se encuentra en github, es compatible con la distribución de Linux *Ubuntu versión 16.04* y la distribución de ROS *ROS Kinect* [26]. Para instalarlo seguir los siguientes paso:

1. Clonar el repositorio: `https://github.com/edujire/MobileSimulatorThesis`
2. Moverse a la carpeta `catkin_ws` dentro del repositorio.
3. Ejecutar el script de instalación `install.sh`:  
`./install.sh`  
Compilar con el comando:  
`catkin_make`

### C.2.2. Transferir modelos derivados en simulador Tk

Para transferir los comportamientos evolucionados se utiliza un script de python, ubicado en la subcarpeta:

```
catkin_ws/src/simulator/src/genetic_behaviors/
```

abrir una terminal desde esta ubicación y ejecutar el script con el siguiente comando:

```
python get_best_N.py <N> <path> <beh>
```

Reemplazar:

- `<N>`: Con el número de mejores individuos a transferir.
- `<path>`: Con el directorio donde se encuentran los archivos creados por el simulador Tk.
- `<beh>`: El comportamiento de acuerdo a las abreviaciones mostradas en el Cuadro C.1.

Los nuevos archivos se crearán en la carpeta:

```
catkin_ws/src/simulator/src/genetic_behaviors/data/
```

Siendo estos:

- `fitness_<beh>.dat`: Valores de fitness de los mejores  $N$  individuos.
- `avoid_<beh>.dat`: Datos de los  $N$  individuos.
- `avoid_<beh>_<num_ind>.dat`: Datos de cada individuo.

### C.2.3. Derivar comportamientos en el simulador ROS

El proceso para evolucionar los individuos utilizando el robot real y el simulado son los mismos una vez realizada la conexión entre la computadora y el robot real, dicho procedimiento se describe en el apéndice D.

1. Inicializar el simulador: Existen dos maneras de iniciar el simulador desde `catkin_ws`:

- `./start.sh`

Se abren 7 terminales X mostrando un nodo de ros distinto cada una: `roscore`, `simulation_node`, `light_node`, `laser_node`, `base_node`, `motion_planner_node` y `pyclips_node`.

- `source devel/setup.bash`

```
roslaunch simulator simulator.launch
```

Solo se utiliza una terminal.

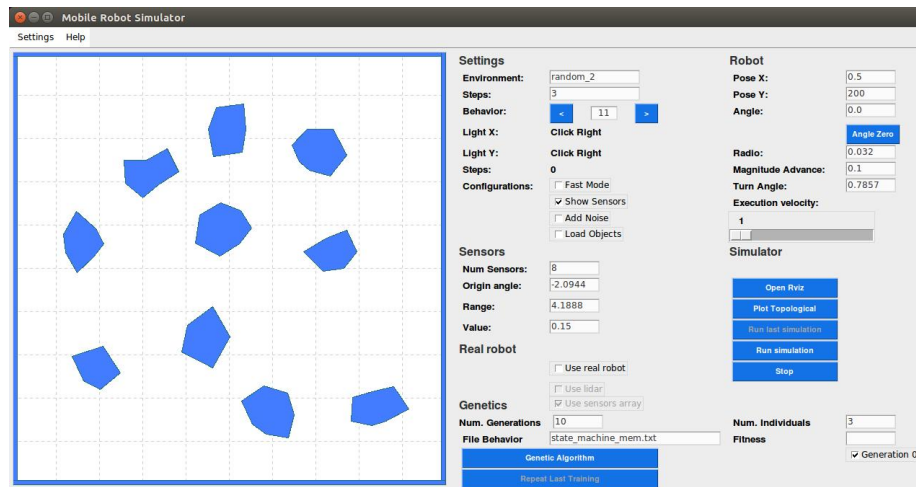


Figura C.4: Interfaz gráfica simulador ROS

2. Seleccionar comportamiento y ajustes iniciales: En el menú derecho se pueden ajustar distintos parámetros algunos que solo se utilizan al trabajar en el simulador (mapa, radio del robot, el número de sensores, entre otros). Los parámetros que se utilizan tanto en el simulador como al utilizar el robot real son:

- Número de Pasos: Número máximo de pasos en una ejecución.
- Magnitud de Avance y Ángulo de Giro: La salida deseada al avanzar o girar en centímetros y radianes correspondientemente.
- Comportamiento: Representados con un valor decimal, se tiene una variedad de comportamientos pero aquellos con los que se trabajó son los mostrados en el Cuadro C.1.

Número	Comportamiento	abrev.
11	Máquina de Estado Finito	fsm
13	Modelo Oculto de Márkov Extendido	hmm
14	Proceso de Decisión de Márkov	mdp

Cuadro C.1: Comportamientos simulador ROS

3. Evolucionar individuos: El menú para evolucionar comportamientos se encuentra en la parte inferior derecha de la interfaz gráfica, donde se elige el número de generaciones a entrenar y el número de individuos, y se muestra el valor de aptitud una vez realizada una ejecución.

Si se desea evolucionar desde los modelos previamente obtenidos en el simulador Tk (una vez importados siguiendo la metodología presentada en la sección C.2.2) Seleccionar el recuadro *Generation 0*, en caso contrario se reanudará desde los últimos archivos guardados, sobrescribiendo los históricos de aptitud y mejores/peores comportamientos por generación.

Para cada ejecución se colocará el robot y la fuente en la posición inicial y se presionará el botón *Genetic Algorithm*, en caso de que sea necesario repetir la última ejecución presionar el botón *Repeat Last Training* para eliminar los datos de la última ejecución, una vez realizado esto volver a colocar el robot y la fuente luminosa en su posición inicial y presionar el botón *Genetic Algorithm*.

4. Archivos de salida: Los archivos de salida son los mismos que los generados por el simulador Tk.

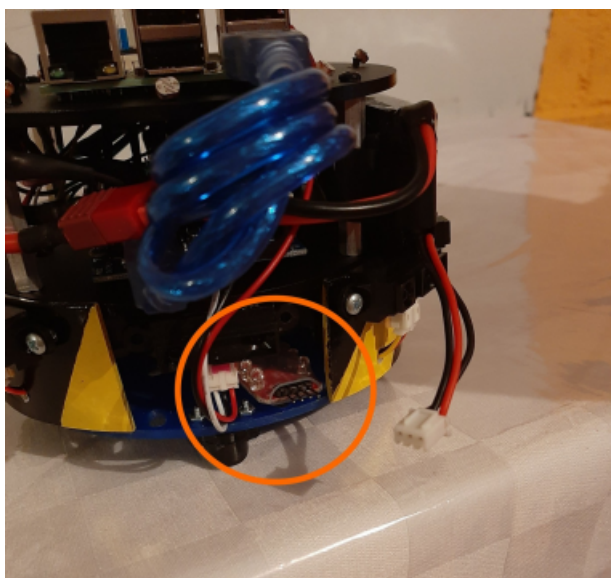


## Apéndice D

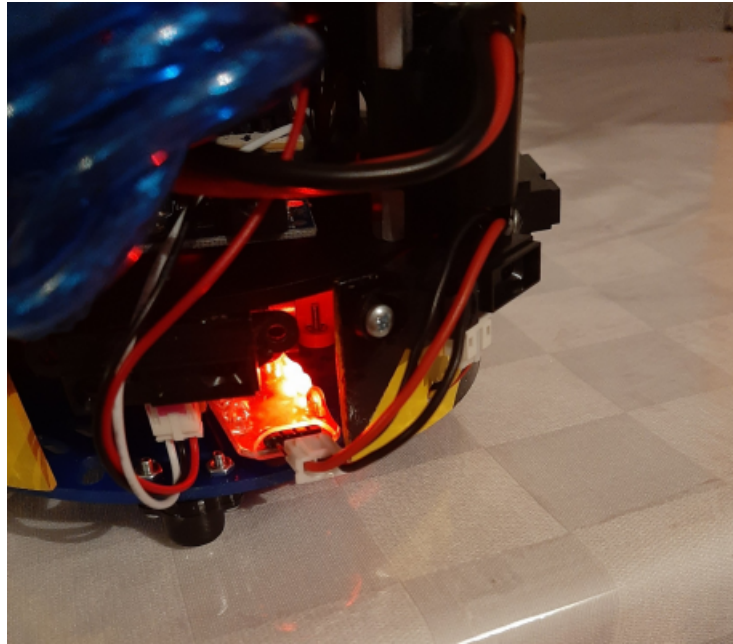
# Manual de conexión Minibot - PC

### D.1. Conexión vía inalámbrica con el robot

1. Antes de encender el robot asegúrese que la batería este bien cargada.  
Para esto ubique la alarma de batería en la parte trasera del robot.



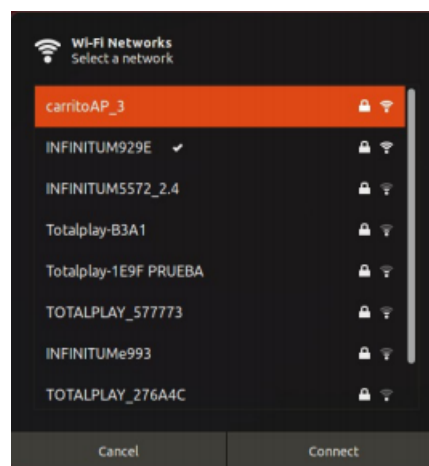
Enchufe el conector de la batería en el sensor, dejando el cable negro (tierra) a la terminal COM de la alarma.  
Una vez conectada (de manera adecuada), el sensor debe sonar y encender dos de sus leds, indicando el estado de las celdas de la batería



No olvide desconectar el sensor si el robot no esta en uso ya que consume batería.

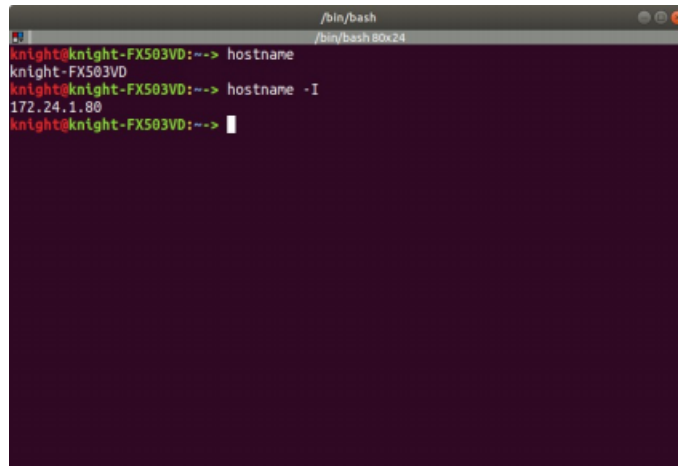
No usar el robot si la alarma está sonando o si los leds parpadean, ya que si la batería se descarga demasiado, puede quedar inutilizable.

2. Ubique el interruptor del robot en la parte superior y enciéndalo.
3. Una vez encendido el robot espere a que se levante la red del robot y conéctese a ella, la contraseña es *raspberrry*.





4. Ya conectado a la red inalámbrica del robot, abra una terminal y obtenga el nombre de su computadora y su IP respectivamente con los siguientes comandos:



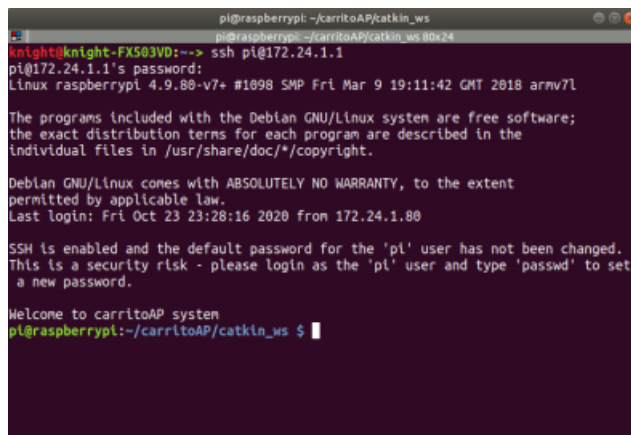
```
knights@knight-FX503VD:~$ hostname
knight-FX503VD
knights@knight-FX503VD:~$ hostname -I
172.24.1.80
knights@knight-FX503VD:~$
```

5. Conéctese a la Raspberry con el siguiente comando:

```
ssh pi@172.24.1.1
```

La contraseña de acceso es: *raspberry*

Cuando se este dentro de la raspberry la terminal se verá de la siguiente manera:



```
knights@knight-FX503VD:~$ ssh pi@172.24.1.1
pi@172.24.1.1's password:
Linux raspberrypi 4.9.80-v7+ #1098 SMP Fri Mar 9 19:11:42 GMT 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Oct 23 23:28:16 2020 from 172.24.1.80

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
a new password.

Welcome to carritoAP system
pi@raspberrypi:~/carritoAP/catkin_ws $
```

6. Ya dentro del sistema del robot ingrese el siguiente comando:

```
sudo nano /etc/hosts
```

Y agregue el nombre de la IP y el nombre de su computadora (obtenidos en el paso 4) en la última línea, tal y como se ve en la siguiente imagen:

```
pi@raspberrypi: ~/catkin_ws
pi@raspberrypi: ~/catkin_ws
127.0.0.1 localhost
::1 localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
127.0.1.1 raspberrypi
192.168.1.73 knight-FX503VD
172.24.1.80 knight-FX503VD
/etc/hosts" 9L, 195C 1,1 All
```

Para guardar teclee: *control + o*  
Para salir del editor teclee: *control + x*

- Abra una nueva terminal (sin cerrar la actual) y teclee el siguiente comando:

```
sudo nano /etc/hosts
```

Y agregue la IP y el nombre de la raspberry dentro de su computadora de escritorio de la siguiente manera:

```
/bin/bash
/bin/bash
GNU nano 2.9.3 /etc/hosts
127.0.0.1 localhost
127.0.1.1 knight-FX503VD
10.42.0.1 turtlebot
192.168.1.67 punas-ThinkPad-T560
192.169.9.148 turtle-12258
10.42.0.1 miztli
10.42.0.1 k-9
10.42.0.1 pi
192.168.1.71 miztli
192.168.1.77 raspberrypi
192.168.1.76 raspberrypi
192.168.1.64 raspberrypi
172.24.1.1 raspberrypi
# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
Read 21 lines
Get Help Write Out Where Is Cut Text Justify Cur Pos
Exit Read File Replace Uncut Text To Spell Go To Line
```

Guarde los cambios y salga.

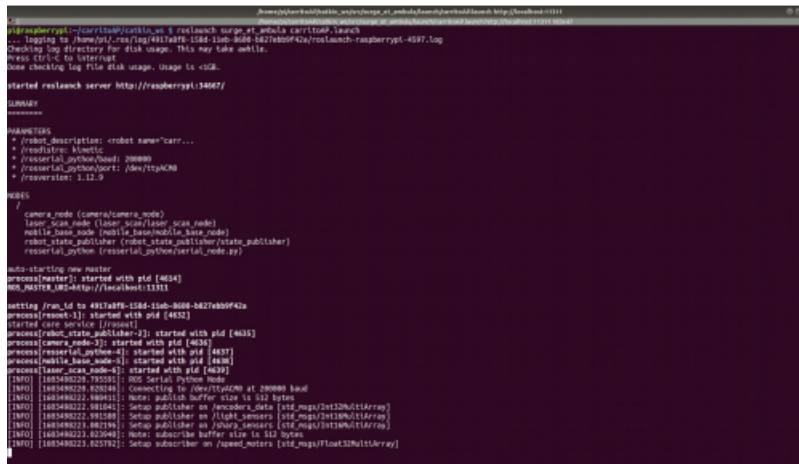
- En esta misma terminal, diríjase a la carpeta **catkin\_ws** del simulador del curso e ingrese los siguientes comandos:

```
source devel/setup.bash
export ROS_MASTER_URI=http://172.24.1.1:11311/
```

9. En la primera terminal de la que se empezó a trabajar (sistema de la raspberry) escriba el siguiente comando:

```
roslaunch surge_et_ambula carritoAP.launch
```

La terminal deberá verse como:



```
ros@raspberrypi:~/carritoAP/carrito_ws$ roslaunch surge_et_ambula carritoAP.launch
... logging to /home/ros/.ros/log/4911a18f-1262-120b-8000-8427e60942a/rosLaunch-raspberrypi-4297.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is 45G...

started roslaunch server http://raspberrypi:13407/

SUMMARY
-----

PARAMETERS
 * /robot_description: -robot name="carr...
 * /rostopic_qos: 0
 * /rserial_pythone/used: 200000
 * /rserial_pythone/port: /dev/ttyACM0
 * /rserial_pythone: 1.13.9

NODES
 /
  camera_node (camera/camera_node)
  laser_scan_node (laser_scan/laser_scan_node)
  mobile_base_node (mobile_base/mobile_base_node)
  robot_state_publisher (robot_state_publisher/robot_state_publisher)
  rserial_pythone (rserial_pythone/rserial_pythone.py)

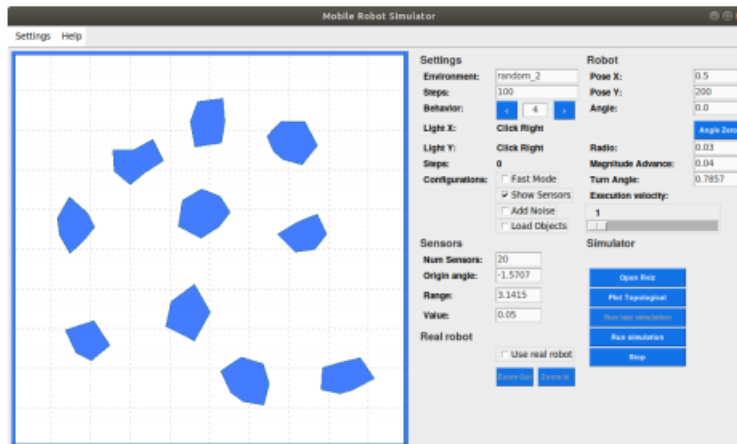
auto-starting new master
process[main]: started with pid [4054]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 4911a18f-1262-120b-8000-8427e60942a
process[roscpp-1]: started with pid [4452]
started core service [/roscpp-1]
process[robot_state_publisher-2]: started with pid [4615]
process[camera_node-3]: started with pid [4616]
process[rserial_pythone-4]: started with pid [4637]
process[mobile_base_node-5]: started with pid [4638]
process[laser_scan_node-6]: started with pid [4639]
[INFO] [1653490228.795259]: ROS Serial Pythone Node
[INFO] [1653490228.825356]: Connecting to /dev/ttyACM0 at 200000 baud
[INFO] [1653490222.880611]: Node: publish buffer size is 512 bytes
[INFO] [1653490222.981242]: Setup publisher on /robot_state [std_msgs/String[]Array]
[INFO] [1653490222.991290]: Setup publisher on /light_sensors [std_msgs/Int16[]Array]
[INFO] [1653490223.002196]: Setup publisher on /cherry_sensors [std_msgs/Int16[]Array]
[INFO] [1653490223.022048]: Node: subscribe buffer size is 512 bytes
[INFO] [1653490223.025792]: Setup subscriber on /speed_notera [std_msgs/Float32[]Array]
```

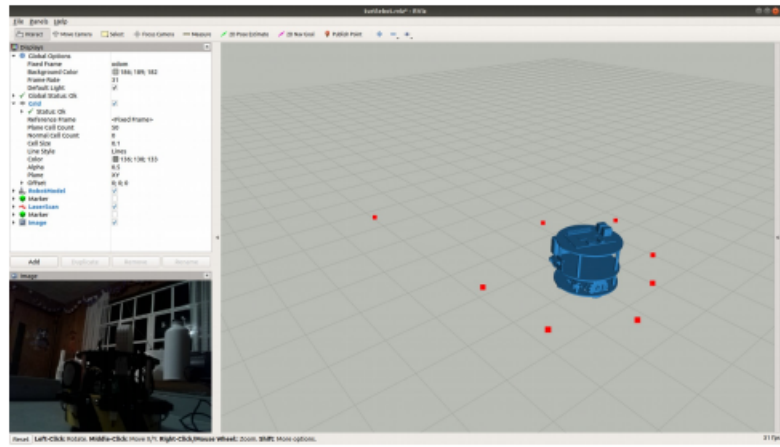
10. En la segunda terminal corra el comando que levanta al simulador de ROS:

```
roslaunch simulator simulator.launch
```

11. En la GUI de click en el espacio *use real robot* para desplegar la simulación tridimensional:



La ventana de visualización del robot tiene que verse como:



# Bibliografía

- [1] S. M. J. Jalali, P. M. Kebria, A. Khosravi, K. Saleh, D. Nahavandi, and S. Nahavandi, “Optimal autonomous driving through deep imitation learning and neuroevolution,” in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pp. 1215–1220, 2019.
- [2] P. M. Kebria, A. Khosravi, S. Nahavandi, Z. Najdovski, and S. J. Hilton, “Neural network adaptive control of teleoperation systems with uncertainties and time-varying delay,” in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pp. 252–257, 2018.
- [3] R. Murphy, R. Murphy, and R. Arkin, *Introduction to AI Robotics*. MIT Press, 2000.
- [4] R. Balogh and D. Obdržálek, *Using Finite State Machines in Introductory Robotics: Methods and Applications for Teaching and Learning*, pp. 85–91. 01 2019.
- [5] D. Kozen, *Automata and Computability*. Springer Berlin Heidelberg, 2013.
- [6] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco, “Probabilistic finite-state machines - part ii,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 7, pp. 1026–1039, 2005.
- [7] L. Rabiner and B. Juang, “An introduction to hidden markov models,” *IEEE ASSP Magazine*, vol. 3, no. 1, pp. 4–16, 1986.
- [8] Á. F. K. Morales and E. Aldana-Bobadilla, “The best genetic algorithm I - A comparative study of structurally different genetic algorithms,” in *Advances in Soft Computing and Its Applications - 12th Mexican International Conference on Artificial Intelligence, MICAI 2013, Mexico City, Mexico, November 24-30, 2013, Proceedings, Part II* (F. Castro-Espinoza, A. F. Gelbukh, and M. González-Mendoza, eds.), vol. 8266 of *Lecture Notes in Computer Science*, pp. 1–15, Springer, 2013.
- [9] J. Savage, S. Munoz Gutierrez, L. Contreras, M. Matamoros, M. Negrete, C. Rivera, G. Steinbauer, O. Fuentes, and H. Okada, “Generating reactive

- robots' behaviors using genetic algorithms," 2 2021. International Conference on Agents and Artificial Intelligence, ICAART ; Conference date: 04-02-2021 Through 06-02-2021.
- [10] T. Degris, O. Sigaud, and P.-H. Wuillemin, "Learning the structure of factored markov decision processes in reinforcement learning problems," ICML '06, (New York, NY, USA), p. 257–264, Association for Computing Machinery, 2006.
  - [11] W. Banzhaf, P. Nordin, and M. Olmer, "Generating adaptive behavior using function regression within genetic programming and a real robot," in *In 2nd International Conference on Genetic Programming*, pp. 35–43, Morgan Kaufmann, 1997.
  - [12] A. L. Nelson, G. J. Barlow, and L. Doitsidis, "Fitness functions in evolutionary robotics: A survey and analysis," *Robotics and Autonomous Systems*, vol. 57, no. 4, pp. 345 – 370, 2009.
  - [13] B. Hallam, D. Floreano, J. A. Meyer, and G. Hayes, *Active Vision and Feature Selection in Evolutionary Behavioral Systems*, pp. 247–255. 2002.
  - [14] J. C. Zagal, J. Ruiz-del Solar, P. Guerrero, and R. Palma, "Evolving visual object recognition for legged robots," in *RoboCup 2003: Robot Soccer World Cup VII* (D. Polani, B. Browning, A. Bonarini, and K. Yoshida, eds.), (Berlin, Heidelberg), pp. 181–191, Springer Berlin Heidelberg, 2004.
  - [15] L. König, *Complex Behavior in Evolutionary Robotics*. Berlin, Boston: De Gruyter Oldenbourg, 13 Mar. 2015.
  - [16] J. Holland, J. Holland, and P. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, 1975.
  - [17] K. Sastry, D. E. Goldberg, and G. Kendall, *Genetic Algorithms*, pp. 93–117. Boston, MA: Springer US, 2014.
  - [18] N. Jakobi, P. Husbands, and I. Harvey, "“noise and the reality gap: The use of simulation in evolutionary robotics,”," vol. 929, pp. 704–720, 01 1995.
  - [19] S. Koos, J.-B. Mouret, and S. Doncieux, "The transferability approach: Crossing the reality gap in evolutionary robotics," *Evolutionary Computation, IEEE Transactions on*, vol. 17, pp. 122–145, 02 2013.
  - [20] D. Floreano and F. Mondada, "Evolutionary neurocontrollers for autonomous mobile robots," *Neural Networks*, vol. 11, no. 7, pp. 1461 – 1478, 1998.
  - [21] V. Zykov, J. Bongard, and H. Lipson, "Evolving dynamic gaits on a physical robot," in *Proceedings of Genetic and Evolutionary Computation Conference, Late Breaking Paper, GECCO'04*, 2004.

- [22] D. Floreano and J. Urzelai, “Evolution of plastic control networks,” *Autonomous Robots*, vol. 11, no. 3, pp. 311–317, 2001.
- [23] C. Hartland and N. Bredeche, “Evolutionary robotics, anticipation and the reality gap,” in *2006 IEEE International Conference on Robotics and Biomimetics*, pp. 1640–1645, 2006.
- [24] S. Koos, J.-B. Mouret, and S. Doncieux, “Automatic system identification based on coevolution of models and tests,” in *2009 IEEE Congress on Evolutionary Computation*, pp. 560–567, 2009.
- [25] “ROS ROS.” <https://www.ros.org/>. Visitado 23-02-2021.
- [26] “ROS Kinectic.” <http://wiki.ros.org/kinetic>. Visitado 14-04-2021.