

Proyecto Final
Robot recolector de residuos
Arquitectura de Comportamientos y Algoritmos

Guillermo Campelo
Juan Ignacio Goñi
Diego Nul

31 de julio de 2010

Resumen

Palabras clave: *Robot, recolector, residuos, robótica, comportamientos, visión, MR-2FA, L298, FR304, HX5010, GP2D120, BC327, SRF05, CNY70, motor, servo, telémetro, daisy chain, RS232, subsumption, We-bots, OpenCV, detección, objetos*

Índice

1. Arquitectura de Comportamientos	4
1.1. Introducción	4
1.2. Investigaciones previas	5
1.2.1. Desarrollo de comportamientos no triviales en robots reales : Robot recolector de basura - Stefano Nolfi [4]	5
1.2.2. Arquitectura de control para un Robot Autónomo Móvil - Neves And Oliveira [3]	6
1.2.3. Path Planning usando Algoritmos Genéticos - Salvatore Candido [1]	7
1.2.4. Navegación predictiva de un robot autónomo - Foka And Trahanias [2]	7
1.2.5. Algoritmo de navegación y evitamiento de obstáculos en un entorno desconocido - Clark Et. al [5]	8
1.3. Arquitectura propuesta	9
1.4. Comportamientos e implementaciones	11
1.4.1. Wandering	12
1.4.1.1. Detalle del comportamiento	12
1.4.1.2. Implementación del comportamiento	12
1.4.2. Enfocar Basura	13
1.4.2.1. Detalle del comportamiento	15
1.4.2.2. Implementación del comportamiento	15
1.4.3. Ir a Basura	16
1.4.3.1. Detalle del comportamiento	16
1.4.3.2. Implementación del comportamiento	16
1.4.4. Recolectar Basura	17
1.4.4.1. Detalle del comportamiento	17
1.4.4.2. Implementación del comportamiento	17
1.4.5. Ir a zona de descarga de basura	18
1.4.5.1. Buscar línea	18
1.4.5.1.1. Detalle del comportamiento	19
1.4.5.1.2. Implementación del comportamiento	19
1.4.5.2. Entrar a línea	20
1.4.5.2.1. Detalle del comportamiento	20
1.4.5.2.2. Implementación del comportamiento	21
1.4.5.3. Seguir línea	21
1.4.5.3.1. Detalle del comportamiento	23
1.4.5.3.2. Implementación del comportamiento	23
1.4.6. Descargar Basura	23
1.4.6.1. Detalle del comportamiento	23
1.4.6.2. Implementación del comportamiento	23
1.4.7. Ir a base de recarga de batería	24
1.4.8. Cargar Batería	24
1.4.8.1. Detalle del comportamiento	24
1.4.8.2. Implementación del comportamiento	24
1.4.9. Evitar Obstáculos	24
1.4.9.1. Detalle del comportamiento	24
1.4.9.2. Implementación del comportamiento	24

1.4.10.	Salir de situaciones no deseadas	25
1.4.10.1.	Detalle del comportamiento	26
1.4.10.2.	Implementación del comportamiento	26
1.5.	Odometría	27
1.5.1.	Problemas con la odometría	28
1.6.	Interfaces con hardware y módulo de reconocimiento de objetos .	29
1.6.1.	Interfaz con hardware	29
1.6.2.	Interfaz con módulo de reconocimiento de objetos usando Visión	29
1.7.	Resultados obtenidos	31
1.7.1.	Tiempo promedio en recolectar basura	34
1.7.2.	Tiempo promedio de wandering	34
1.7.3.	Distribución de tiempos entre los diferentes comportamien- tos	34
1.7.4.	Relación $(t_{\text{cargando}} + t_{\text{ir a base}}) / \text{tiempo total}$	34
1.7.5.	Porcentaje de espacio cubierto	34
1.7.6.	Tiempo cubrir toda la arena	34
1.8.	Conclusión	36
A.	Primer apéndice Comportamientos	37
B.	Segundo apéndice Comportamientos	37
C.	Tercer apéndice Comportamientos	37
D.	Cuarto apéndice Comportamientos	37
E.	Quinto apéndice Comportamientos	37

1. Arquitectura de Comportamientos

1.1. Introducción

Como se dijo en las secciones anteriores, el objetivo de éste trabajo es desarrollar un robot autónomo y reactivo que recolecte basura en un entorno estructurado pero dinámico, debido a que la arena (lugar donde se mueve el robot) es transitado por personas y es al aire libre.

En ésta sección de *Arquitecturas de Comportamientos* se van a detallar las acciones que deberá hacer el robot y cómo organizar las mismas para cumplir la meta de recolectar basura y ser autónomo, es decir, decidir por sí mismo las acciones a realizar en cada momento y ser capaz de mantenerse cargado para poder continuar recolectando.

En la sección 1.2 se analizarán papers relacionados con este desarrollo, desde la forma de organizar los comportamientos, la definición y composición de los mismos, hasta su implementación. En la sección 1.3 se detallará la arquitectura elegida para llevar a cabo y organizar los comportamientos elegidos, y las ventajas y desventajas de usar la misma. La sección 1.4 detalla uno por uno los comportamientos elegidos que tendrá el robot, y sus correspondientes implementaciones en pseudocódigo. En la sección 1.5 se explicará que es la odometría, para qué sirve y qué ventajas y desventajas tiene usarla. También se detallará el test utilizado para mejorar la eficiencia de la misma. La sección 1.6 se describirá cómo se estructuró el controlador del robot para que se pueda ir desarrollando y probando el mismo con un simulador y minimizar el trabajo al pasarlo a el robot físico. Los resultados obtenidos de performance y eficiencia del controlador desarrollado se obtuvieron de la simulación, y se pueden ver en la sección 1.7. Finalmente, en la sección 1.8 se sacan conclusiones de los resultados obtenidos y las dificultades encontradas a lo largo del desarrollo de éste trabajo práctico final.

1.2. Investigaciones previas

A continuación se van a presentar trabajos realizados por otros autores relacionados en alguna forma con el trabajo que nosotros realizamos. Primero se da una breve descripción del trabajo del otro autor y luego se comparan con nuestro trabajo, viendo similitudes y diferencias entre ambos.

1.2.1. Desarrollo de comportamientos no triviales en robots reales : Robot recolector de basura - Stefano Nolfi [4]

En éste paper se muestra el uso de un Khepera con el módulo Gripper en una arena delimitada por paredes para la recolección de "basura". Dicho robot se puede ver en la figura 1. La base tiene una fuente de luz asociada y el

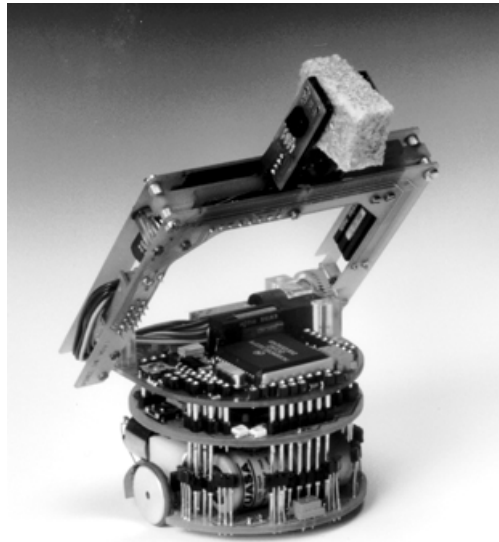


Figura 1: Robot Khepera con el módulo Gripper

objetivo del robot es llevar pequeños objetos a un lugar de depósito. Se utiliza *aprendizaje por refuerzo* para asociar las velocidades angulares a los diferentes tamaños de las "basuras". Para obtener el comportamiento deseado se hace uso de *algoritmos genéticos* y *redes neuronales*. Una vez obtenido el mismo mediante simulación en Webots, se lo probó en el Khepera físico. Comparando el trabajo de Stefano Nolfi con el nuestro podemos observar que:

- Existencia de un depósito: Ambos tienen un depósito en el cual dejar la basura recogida y una forma de identificarla: una fuente de luz usada por Nolfi y en nuestro trabajo, al llegar al final de una determinada línea de la arena.
- Autonomía: Ambos son autónomos en el sentido que no son manejados por un ente externo. En cuanto a la recarga de la batería, Nolfi no explicita

si es asistida o no; en nuestro caso, el robot se encarga de ir a la base de recarga una vez detectada la falta de energía.

- Método de Recolección: En el trabajo de Nolfi se utiliza un módulo agregado al Khepera, que simula un brazo con dos dedos, la recolección se realiza juntando ambos y la liberación se realiza separando ambos. Nuestro trabajo se basa más en la actividad humana que en el comportamiento humano para realizar este comportamiento, ya que podría compararse con un recolector de basura que primero guarda lo encontrado usando una pala en su tacho y luego descarga su tacho en un depósito de mayor tamaño.
- Método de aprendizaje: Nuestro trabajo no utiliza alguna forma de aprendizaje por refuerzo o algoritmo evolutivo. No es el caso de Nolfi, que utiliza ambas técnicas para evolucionar el comportamiento deseado.
- Robot Utilizado: Como ya se dijo, Nolfi utilizó un Khepera en la simulación. En nuestro trabajo utilizamos un E-puck para la simulación, una versión nueva del Khepera, pero sin el módulo Gripper.



Figura 2: Robot E-puck

1.2.2. Arquitectura de control para un Robot Autónomo Móvil - Neves And Oliveira [3]

Neves y Oliveira describen en su trabajo una arquitectura de control basada en comportamientos para un robot móvil en un ambiente dinámico, utilizando muchos aspectos de la arquitectura *Subsumption* propuesta por *Brooks* y que fue usada por nosotros al realizar éste trabajo explicada en 1.3. La arquitectura propuesta, o *Control System Architecture*, se basa también en la teoría de *The Society of Mind*, escrita por *Minsky*, donde el sistema es visto como una sociedad de agentes, cada uno con una particular competencia y que colaboran entre ellos

para ayudar a la sociedad a alcanzar su meta. La arquitectura está compuesta por tres niveles: un nivel reflexivo, uno reactivo, y otro cognitivo, aumentando la complejidad al igual que el orden en que fueron presentados.

El nivel reflexivo incluye aquellos comportamientos innatos, es decir, actúan directamente como estímulo-respuesta. El segundo nivel, el reactivo, está compuesto por agentes que responden rápidamente a los estímulos ya que requieren poco nivel de procesamiento. Finalmente, en el nivel cognitivo, se encuentran los agentes encargados de guiar y administrar los comportamientos reactivos de forma tal que el robot muestre un comportamiento orientado.

Aunque la arquitectura explicada es similar a la utilizada en nuestro trabajo, ésta organización de los comportamientos en capas no la utilizamos, aunque podemos divisar que algunos de los comportamientos que propusimos corresponden a la primera capa de reflexión, como por ejemplo el evitamiento de obstáculos; otros podrían incluirse en la segunda capa de comportamientos reactivos, como lo es deambular y finalmente en la última capa estaría el reconocimiento de objetos debido a su gran demanda de procesamiento.

1.2.3. Path Planning usando Algoritmos Genéticos - Salvatore Candido [1]

En este paper se describe la utilización de algoritmos genéticos para resolver el problema de Path Planning. Éste problema consiste en armar un plan, una secuencia de acciones de forma tal que a partir de un punto de origen y siguiendo ese plan, se llegue a un segundo punto, el de destino. Debido a la componente dinámica de nuestro trabajo, siempre tratamos de mantener los comportamientos del robot lo más reactivos posibles, por lo que armar un plan no sería el mejor approach para ser utilizado. Por el contrario, el uso de algoritmos genéticos puede ser una herramienta que podría llegar a ser utilizada en una futura continuación de nuestro trabajo y que no utilizamos por el tiempo que estimamos que nos hubiese demandado.

1.2.4. Navegación predictiva de un robot autónomo - Foka And Trahanias [2]

La navegación predictiva nace como una posible solución al problema de un robot navegando en un ambiente con muchas personas y obstáculos, tal como lo es el ambiente en el cual navegará el robot resultante de nuestro trabajo. La forma en que es implementada por los autores es mediante un *POMDP*, es decir, un proceso de decisión de markov parcialmente observable. Cabe destacar estas dos últimas palabras, ya que indican la naturaleza del ambiente: hay incertidumbre o falta de información acerca de ciertas variables del entorno. En este caso, se utiliza el *POMDP* para manejar tanto la navegación del robot como el evitamiento de obstáculos, un punto en que se diferencia de lo propuesto por nosotros, que es tratar el *wandering* de forma separada del módulo de evitamiento de obstáculos. Ésto, en parte, se debió a la arquitectura utilizada y que desde ese punto de vista, ambos comportamientos son muy lejanos.

1.2.5. Algoritmo de navegación y evitamiento de obstáculos en un entorno desconocido - Clark Et. al [5]

En este paper se presentan dos algoritmos complementarios para la navegación en ese tipo de entornos.

El primero consiste en la navegación y un mapeo del entorno que garantiza una cobertura completa de una arena cuyas ubicaciones de la paredes no se conocen *a priori*. Consiste básicamente en un seguimiento de las paredes complementado por una variación de *flood filling* para asegurarse la cobertura completa de la arena. El algoritmo completo se puede apreciar en la figura 3

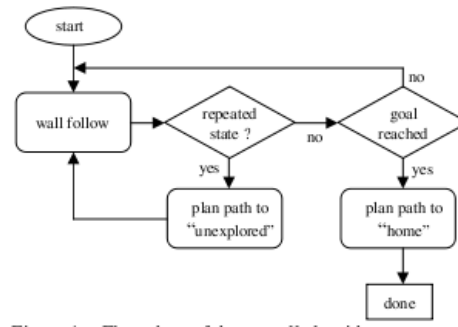


Figura 3: Algoritmo completo

Un autómata con aprendizaje estocástico es el algoritmo que complementa al primero, y su objetivo es el evitamiento de obstáculos. Para ésto, se utiliza un mecanismo de recompensa/castigo de forma tal que se adapten las probabilidades de las acciones a tomar.

En relación con nuestro trabajo, hay ciertas similitudes y diferencias, enumeradas a continuación:

- Ambos presentan un mecanismo de "seguimiento de", en nuestro caso lo hacemos con líneas y basuras, en el paper se utiliza con paredes. Sin embargo, se utilizan para objetivos diferentes. Nosotros lo usamos para dirigirnos a la base ya que tratamos de mantener el conocimiento que el robot tiene sobre el mundo lo más acotado posible, o, en el caso de las basuras, para recolectarlas. En el paper se utiliza el mecanismo de seguir las líneas para obtener un modelo del mundo, algo que puede llegar a servir mucho en ambientes no tan dinámicos como el nuestro, razón por la cual no elegimos implementarlo.
- También se coincide en la existencia de un método para evitar obstáculos. En el paper se implementa como un autómata que va aprendiendo según los premios o castigos que recibe; en nuestro caso, el comportamiento es puramente reactivo y reacciona en base a los valores de los sensores de distancia, pero no tiene memoria.

1.3. Arquitectura propuesta

Una arquitectura basada en comportamientos define la forma en que los mismos son especificados, desde su granularidad (que tan complejo o simple es un comportamiento), la base para su especificación, el tipo de respuesta y la forma en que se coordinan.

La arquitectura que elegimos para desarrollar nuestro trabajo es *Subsumption*, desarrollada por Rodney Brooks a mediados de 1980. Ésta arquitectura está basada en comportamientos puramente reactivos, rompiendo así con el esquema que estaba de moda en la época de *sensar-planear-actuar*. Algunos de los principios propuestos que fueron tenidos en cuenta durante el desarrollo de nuestro trabajo son:

- Un comportamiento complejo no es necesariamente el producto de un complejo sistema de control
- El mundo es el mejor modelo del mismo
- La simplicidad es una virtud
- Los sistemas deben ser construidos incrementalmente

Cada comportamiento es un par estímulo-respuesta. Tal como se observa en la Figura 4, cada estímulo o respuesta puede ser inhibido o suprimida por otros comportamientos activos. Además, cada comportamiento recibe una señal de reset, que vuelve al comportamiento que recibió esta señal a su estado original. El nombre *Subsumption* proviene de la forma en que los comportamientos son

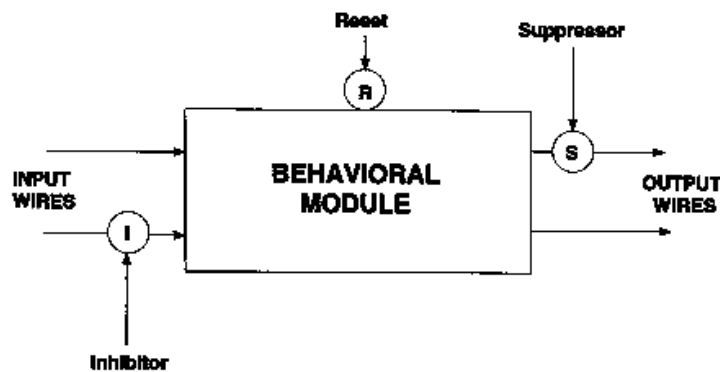


Figura 4: Esquema de comportamiento

coordinados entre sí. Hay una jerarquía donde los comportamientos de la arquitectura tienen mayor o menor prioridad según su posición. Los comportamientos de los menores niveles no tienen conocimiento de los comportamientos de las capas de mayor nivel. Gracias a esto, se puede plantear un diseño incremental, brindando flexibilidad, adaptación y paralelismo al desarrollo e implementación de los comportamientos.

La idea a seguir es que el mundo sea el principal medio de comunicación entre los comportamientos, dado que la respuesta de un comportamiento ante estímulo

resulta en un cambio en el mundo y, por lo tanto, en la relación del robot con el mismo, ya que el robot en su próximo paso sensorá otro estado del mundo. El procedimiento básico para diseñar y desarrollar comportamientos para robots con esta arquitectura es sencillo:

1. Especificar cualitativamente la forma en que el mundo responde al mundo, es decir, el comportamiento que realizará.
2. Descomponer la especificación como un conjunto de acciones disjuntas.
3. Determinar la granularidad del comportamiento, analizando en que nivel de la jerarquía existente se encontrará y cuántas acciones disjuntas es necesario llevar a cabo para el cumplimiento de la tarea.

Un ejemplo de ésta arquitectura se puede observar en 5. En la misma hay 4 comportamientos: Homing, Pickup, Avoiding y Wandering. Las líneas que entran a cada comportamiento son los estímulos ante los cuales se activan y las salidas de los mismos son las señales de respuesta correspondientes. La señal de respuesta de la arquitectura es la línea que sale por la derecha de la caja (Arquitectura) que contiene la relación entre los comportamientos. Puede verse como Homing inhibe la salida de Pickup ya que su salida entra al supresor (denotado con un círculo con una S dentro) de la salida de Pickup y por lo tanto, las salidas de los demás comportamientos, ya que su prioridad es mayor al resto. En el caso que no estén presentes los estímulos de Homing, Pickup y Avoiding, no hay inhibición en la salida de Wandering, y por lo tanto se lleva a cabo el comportamiento de Wandering.



Figura 5: Un ejemplo de Subsumption

1.4. Comportamientos e implementaciones

Una vez que decidimos utilizar la arquitectura explicada en la sección 1.3 para nuestro proyecto, tuvimos que analizar:

- Forma de implementación de la arquitectura en código
- Comportamientos a realizar
- Ordenes de inhibición y supresión entre los mismos
- Forma de implementación de los mismos
- Orden de implementación

Para implementar la arquitectura decidimos asignarle un *ID* numérico diferente a cada comportamiento. También tomamos la decisión de elegir como comportamiento activo en el instante t , aquel comportamiento que esté activo en ese instante y tenga mayor *ID*, suprimiendo así el resto de los comportamientos (con un *ID* menor) que podrían estar activos.

El requerimiento de este proyecto es la realización de un robot autónomo que recolecte basura de su entorno dinámico pero estructuralmente fijo. De aquí se infieren algunos de los comportamientos que debe tener el robot:

- *Recolectar basura* (1.4.4)
- *Recargar batería* (1.4.8): Por ser autónomo, debe poder ser capaz de recargarse sólo para poder continuar con su actividad.
- *Wandering* (1.4.1): El robot no es controlado por control, ya que es autónomo, por lo que debe poder recorrer el entorno por sí mismo.
- *Evitamiento de obstáculos* (1.4.9): Debido a la naturaleza del entorno, el robot debe ser capaz de navegar sin chocarse contra los límites del entorno ni con las personas que circulan por el mismo.

El comportamiento de recolectar basura y el requerimiento de la autonomía llevan a su vez a la aparición de más comportamientos: *Descargar basura* (1.4.6) e *Ir hacia basura* (1.4.3).

La figura 6 muestra los comportamientos implementados y su orden de jerarquía. Se puede ver que hay más comportamientos de los detallados anteriormente. Ésto se debió a que la forma en que se implementamos los comportamientos básicos del robot nos requirió el desarrollo de comportamientos auxiliares.

Primero implementamos *Wandering* debido a, en una primera aproximación, la sencillez del mismo. Luego implementamos *Evitamiento de obstáculos* para lograr que el robot pueda navegar sin problemas por el arena. Como el comportamiento de recolectar e ir hacia la basura dependía del módulo de reconocimiento de objetos y el mismo estaba siendo desarrollado en paralelo, se decidió implementar el comportamiento de *Recargar batería* y *Descargar basura*. Una vez que tuvimos la primera implementación funcional del reconocimiento de objetos, procedimos a desarrollar *Ir hacia basura* y *Recolectar Basura*.

A continuación detallamos los comportamientos indicados en la figura 6, así como la implementación en *pseudo-código* de los mismos y detalles tenidos en cuenta para la realización de los mismos.

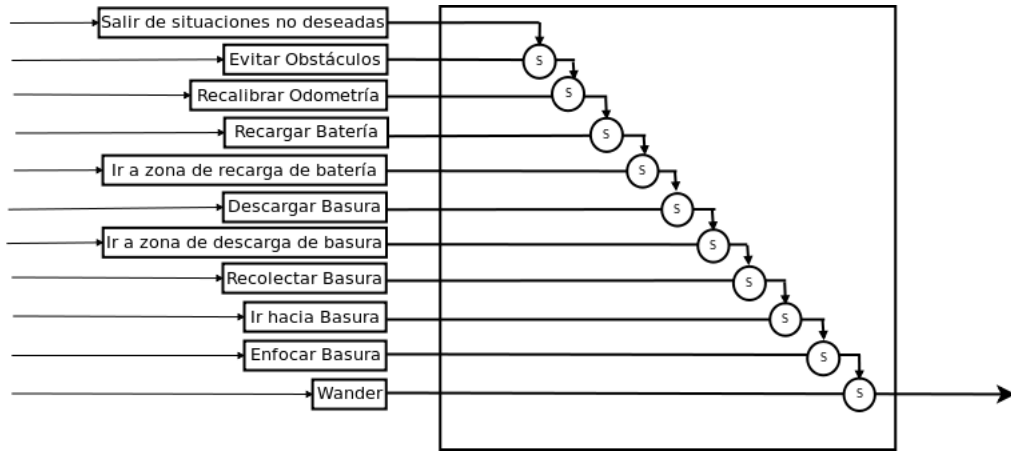


Figura 6: Arquitectura de Comportamientos

1.4.1. Wandering

1.4.1.1. Detalle del comportamiento Por ser el comportamiento que menor jerarquía tiene (Ver 6), es el único comportamiento que está activado ante la ausencia de un estímulo, asegurandonos que siempre haya por lo menos un comportamiento activo.

En una primer aproximación de *Wandering*, sólo nos preocupamos por ir hacia adelante ya que eventualmente, el robot encuentra un obstáculo y realiza un giro cambiando la dirección del robot.

Los resultados de la simulación nos indicaron que el robot no recorría ciertas zonas o las recorría después de un largo tiempo, lo que nos llevó a un segundo approach. El mismo tiene en cuenta el hecho de que el robot posee una cámara y por lo tanto se puede llevar un seguimiento de los lugares que más recientemente visitó o las zonas que hace mucho tiempo no visita.

Éste approach, en cierta forma, genera un modelo del mundo, un hecho que conflictúa con uno de los principios propuestos a seguir en la sección 1.3. Para minimizar el conflicto, decidimos mantener al mínimo la información almacenada para el funcionamiento del algoritmo, es decir, por cada zona de arena sólo mantenemos el timestamp de la última vez que el robot la visitó.

1.4.1.2. Implementación del comportamiento La implementación del segundo approach, en *pseudo-código* es la siguiente:

```

por cada paso
    zona = pedir_zona_vista(camara)
    marcar_zona_como_vista(modelodelmundo,zona)
    ultimazonavisitada = pedir_ultima_zona_visitada(modelodelmundo)
    velocidades = calcular_velocidades_de_ruedas(ultimazonavisitada)
    poner_velocidades_en_ruedas(velocidades)

```

Para obtener la zona vista por la cámara, necesitamos de la altura C_h a la cual está ubicada la cámara en el robot, el campo de visión (de ahora en ade-

lante *Field of View* o *FOV*) horizontal FOV_h o vertical FOV_v y el ángulo de inclinación de la cámara ac . Viendo la figura 7 podemos ver que:

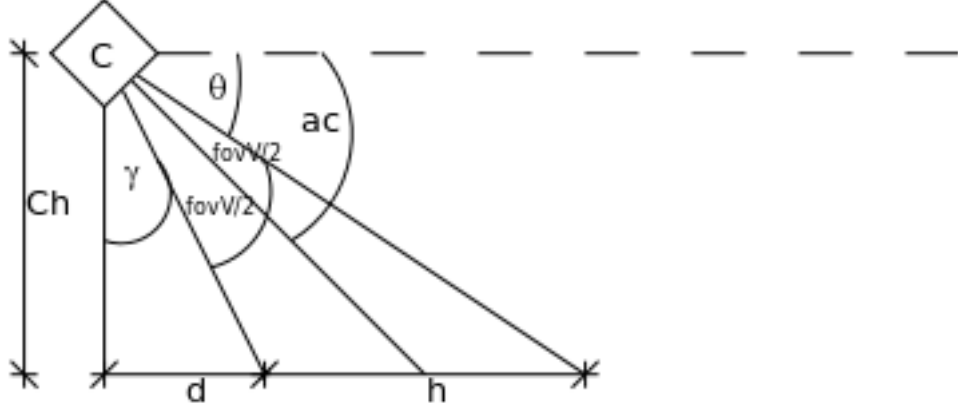


Figura 7: Diagrama de posición de la cámara

$$ac + \frac{FOV_v}{2} + \gamma = \frac{\pi}{2} \quad (1)$$

$$\gamma + FOV_v + \theta = \frac{\pi}{2} \quad (2)$$

$$\tan(\gamma) = \frac{d}{C_h} \quad (3)$$

$$\tan(\gamma + FOV_v) = \frac{d + h}{c_h} \quad (4)$$

Organizando las ecuaciones, podemos deducir que:

$$\gamma = \frac{\pi}{2} + ac - \frac{FOV_v}{2} \quad (5)$$

$$d = \tan(\gamma) * c_h \quad (6)$$

$$d + h = \tan(\gamma + FOV_v) * c_h \quad (7)$$

Por lo que obtenemos el ángulo hasta el inicio de la imagen de la cámara γ y como consecuencia, la distancia d desde la posición de la cámara hasta el inicio de la imagen y $d + h$, la distancia desde la posición de la cámara hasta el final de la imagen. Usando éstos datos, la posición del robot P y basándonos en la figura 8, podemos obtener los puntos A , B , C y D del trapecioide que determina la zona que ve la cámara.

1.4.2. Enfocar Basura

Una vez que el módulo de detección de objetos reconoce algo como basura (ver sección ??), hay que elegir la forma en que se va a la basura. En la figura 9 se puede ver un primer approach de hacer ésto. Consiste en primero enfocar la

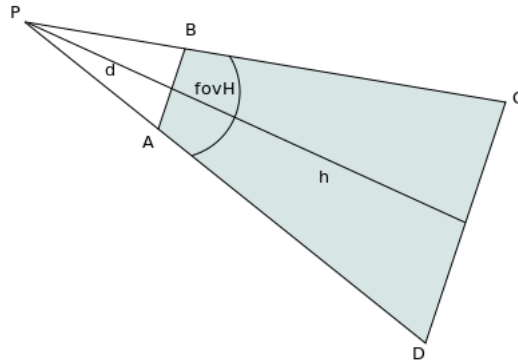


Figura 8: Zona vista por la cámara

basura de forma tal que la misma quede en el centro de la imagen de la cámara. De aquí surge el comportamiento *Enfocar Basura*.

Un segundo approach (Figura 10) no consiste en enfocar la basura, sino que se idea un arco hacia la misma. Ésto requiere que se seteen las velocidades correspondientes a las ruedas de forma tal que la trayectoria del robot describa dicho arco. Con este segundo approach no existiría el comportamiento que se está describiendo.

Dado que el primer approach es levemente más simple de implementar, y teniendo en cuenta el principio enunciado en la sección 1.3 “*La simplicidad es una virtud*”, elegimos implementar el mismo, a pesar de tener un posible inconveniente, como se puede ver en la figura 11.

Cuando hay una basura en alguna esquina superior de la imagen de la cámara, y el robot gira sobre sí mismo para enfocarla, la basura puede llegar a perderse por el fondo de la imagen. Ésto se debe a que la distancia hacia dichas esquinas es mayor a la distancia hacia el centro del borde superior de la imagen (Ver figura 8). Decimos que es un “posible” problema, ya que una vez que se pierde de vista la basura, el robot no enfocarás más debido a que el estímulo desapareció, pero si luego se dirige hacia adelante (por la activación de algún otro comportamiento), la basura volverá a aparecer en la imagen, sin desaprovechar la oportunidad de recogerla.

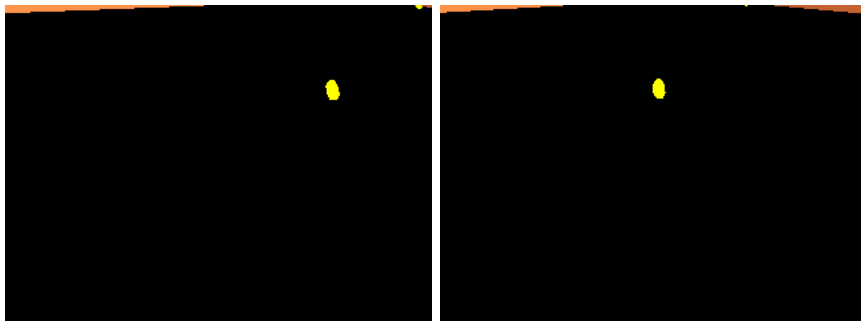


Figura 9: Primer approach de ir a la basura

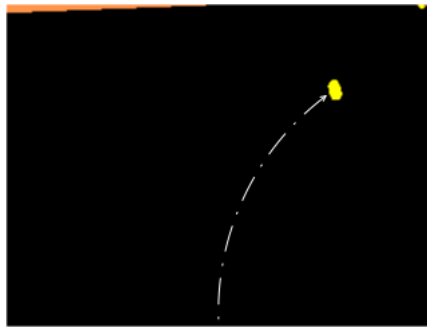


Figura 10: Segundo approach de ir a la basura



Figura 11: Posible inconveniente con el primer approach de ir a la basura

1.4.2.1. Detalle del comportamiento El approach elegido, entonces, se puede describir como:

- Si la basura está a la izquierda de la imagen, se debe girar hacia la izquierda
- Si la basura está a la derecha de la imagen, se debe girar hacia la derecha
- Si la basura está en el centro, ya está enfocada

1.4.2.2. Implementación del comportamiento El comportamiento sigue el siguiente *pseudo-código*:

```

por cada paso
    lista_de_basuras = obtener_lista_de_basuras(modulodereconocimiento)
    basura_mas_cercana = elijo_basura_mas_cercana(lista_de_basuras)
    angulo_a_basura = obtener_angulo(basura_mas_cercana)
    velocidad = VELOCIDAD_BASE * ( abs(angulo_a_basura) / (PI/2) )
                + VELOCIDAD_BASE_MINIMA

    si angulo_a_basura < 0 entonces
        veloc_izq = -velocidad
        veloc_der = velocidad
    sino
        veloc_izq = velocidad
        veloc_der = -velocidad

```

```

fin_si
poner_velocidades_en_ruedas(veloc_izq,veloc_der)

```

Se puede ver que la velocidad de giro del robot es proporcional al módulo del ángulo que hay hacia la basura, logrando enfocar más rápido cuando el ángulo es mayor y tener mayor precisión cuando el ángulo es más chico, además de tener mayor rapidez de enfoque y precisión que si la velocidad de giro fuera constante.

1.4.3. Ir a Basura

Luego de la elección de la forma que se resuelve la situación de encontrar una basura (ver 1.4.2), el comportamiento de ir a basura es trivial, ya que la basura, luego de ser enfocada, está delante del robot y lo único que basta es ir hacia adelante.

1.4.3.1. Detalle del comportamiento El estímulo necesario para que este comportamiento esté presente está dado por dos condiciones:

1. El método de reconocimiento de objetos reconoció una basura
2. La basura se encuentra en un entorno del medio horizontal de la imagen de la cámara

1.4.3.2. Implementación del comportamiento

por cada paso

```

distancia = obtener_distancia_a_basura(modulodereconocimiento)
coeff = (distancia - DIST_MIN)/(DIST_MAX - DIST_MIN)
veloc_der = VELOCIDAD_MIN*(1 - coeff) + coeff*VELOCIDAD_MAX
veloc_izq = veloc_der
poner_velocidades_en_ruedas(veloc_izq,veloc_der)

```

Al igual que en la implementación del comportamiento 1.4.2.2, las velocidades que se le otorgan a las ruedas dependen de la distancia hacia la basura, de forma tal que si una basura está muy lejos, la velocidad sea mayor y a medida que se va acercando, vaya disminuyendo linealmente.

Dado que la basura se encuentra en un entorno del eje Y en la imagen (Ver figura 12b) cometemos un error muy pequeño al estimar la distancia a la basura como si estuviera sobre el mismo (asumiendo que la coordenada X de la basura en la imagen es 0).

Como muestran las figuras 12 y 7, el ángulo vertical hacia el punto más alto de la imagen $(0, C_{rh})$ es $\gamma + FOV_v$ y hacia el más bajo, $(0, 0)$, el ángulo es γ . De las ecuaciones (6) y (7) sabemos las distancias a los puntos $(0, 0)$ (DIST_MIN) y $(0, C_{rh})$ (DIST_MAX). Entonces, para obtener la distancia dy hacia el punto $(0, y)$ basta con calcular:

$$y_{angle} = FOV_v * \frac{y}{h} \quad (8)$$

$$dy = \tan(\gamma + y_{angle}) * C_h \quad (9)$$

$$(10)$$

donde y_{angle} es la proporción de FOV_v hacia y .

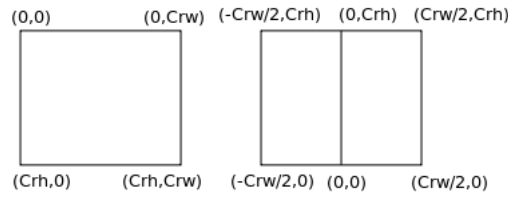


Figura 12: Conversión de coordenadas de imagen de cámara

1.4.4. Recolectar Basura

Una vez que el robot llega a estar posicionado para recolectar la basura deseada, el comportamiento de *recolectar basura* se activa. El mecanismo para esto fue cambiando a lo largo del desarrollo del proyecto.

En un principio pensamos en usar una rampa interna dentro del robot, de forma tal que la basura suba esa rampa para luego caer en un depósito de basura interno al robot. Por problemas con la simulación de este procedimiento, se buscó otro mecanismo.

El mecanismo que elegimos para usar en la simulación fue el siguiente:

- El robot tiene un servo en su parte posterior (debajo de la cámara)
- Dos paredes delimitan el espacio a lo largo de la dirección que une el centro del robot con el servo anteriormente mencionado.

1.4.4.1. Detalle del comportamiento La activación de *recolectar basura* depende de 3 condiciones:

- Las dos condiciones impuestas para *ir a basura*
- La distancia a la basura elegida para ser recolectada debe ser menor a un umbral.

Aquí se puede observar que tan importantes son los comportamientos anteriores para que el robot logre recolectar la basura.

Es importante la elección del umbral: un valor muy chico puede llevar a que el comportamiento no se active porque la basura ya no está más en la imagen. Por otro lado, un valor muy grande causaría que el robot se disponga a recolectar una basura que está muy lejos y podría llegar a moverse por cuestiones propias del ambiente, llevando así a una innecesaria activación del comportamiento.

1.4.4.2. Implementación del comportamiento El *pseudo-código* de este comportamiento es el siguiente:

```

distancia = obtener_distancia_a_basura(modulodereconocimiento)
levantar(servo_delantero)
recorrerdistancia(distancia)
cerrar(servo_delantero)

```

La distancia hacia la basura es obtenida de la misma forma que en la sección 1.4.3.2. Levantar y cerrar el servo consiste en setear su posición en $\frac{\pi}{2}$ y 0

respectivamente. Para calcular la distancia recorrida se utilizó la distancia entre la posición del robot en el instante t , $P_r(t)$ y el instante $t + 1$, $P_r(t + 1)$, ambas dadas por la odometría (Ver sección 1.5). Las diferentes etapas de la recolección de una basura se puede ver en la figura 13.

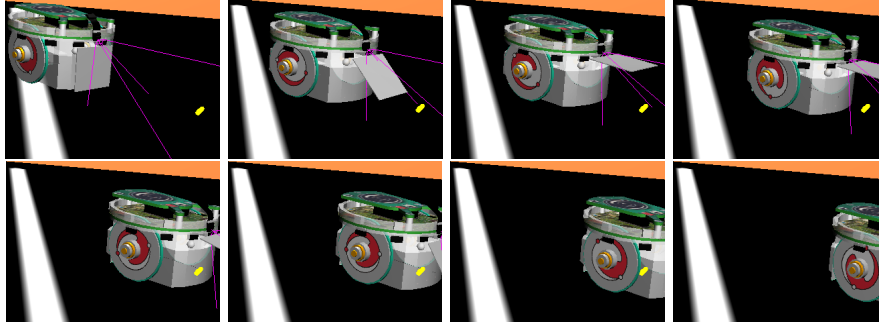


Figura 13: Etapas de recolección de basura

1.4.5. Ir a zona de descarga de basura

Como la capacidad del depósito interno de basura del robot tiene un límite, surge como necesidad que el robot sea capaz de ir hacia una zona donde descargará la basura que contiene, surgiendo así el estímulo necesario para la activación de este comportamiento.

Para ir hacia dicha zona, decidimos imponerle una condición al entorno donde el robot actuará. Ésta condición consiste en poner líneas de forma tal que si el robot sigue la misma, lo lleve al lugar donde se encuentra la zona de descarga. Por lo tanto, para dirigirse a la zona de descarga de basura, es necesario:

- Buscar la línea e ir a la misma
- Entrar a la línea, de forma tal que el robot y la línea estén alineados
- Seguir la línea

Siguiendo con la idea que es mejor descomponer un comportamiento complejo en otros más simples, decidimos separar el comportamiento de *Ir a zona de descarga de basura* en 3 comportamientos más simples: *Buscar línea*, *Entrar a la línea* y finalmente *Seguir la línea*.

1.4.5.1. Buscar línea Inicialmente, habíamos dispuesto las líneas de forma tal que sigan los límites de la arena. Entonces para buscar la línea tuvimos que calcular para cada línea cuál es la distancia hacia la misma, luego elegir ir a la que menor distancia había. Además de ser costoso, este approach tenía un problema: a veces sucedía que al ir hacia una línea, la distancia hacia otra pasaba a ser más corta y ésta última podía estar más lejos de la zona de recarga que la primera.

Luego repensamos el problema y nos dimos cuenta que el objetivo era llegar a la zona de descarga, por lo que decidimos dejar sólo 2 las líneas que se encuentran

cerca de la misma, como se puede apreciar en la figura 14. La zona de descarga se ubica cerca de donde se encuentra el cilindro de color verde.

Para distinguir si el robot está en una línea o no, utilizamos sensores de piso dispuestos como se muestra en la figura 15. Los sensores se encuentran a una distancia Fsd del centro del robot sobre el eje Y y los de los costados a una distancia Fss del eje Y . La distancia del sensor del medio hacia el centro del robot es a , mientras que la distancia hacia un sensor del costado es $Fsl = \sqrt{Fsd^2 + Fss^2}$.

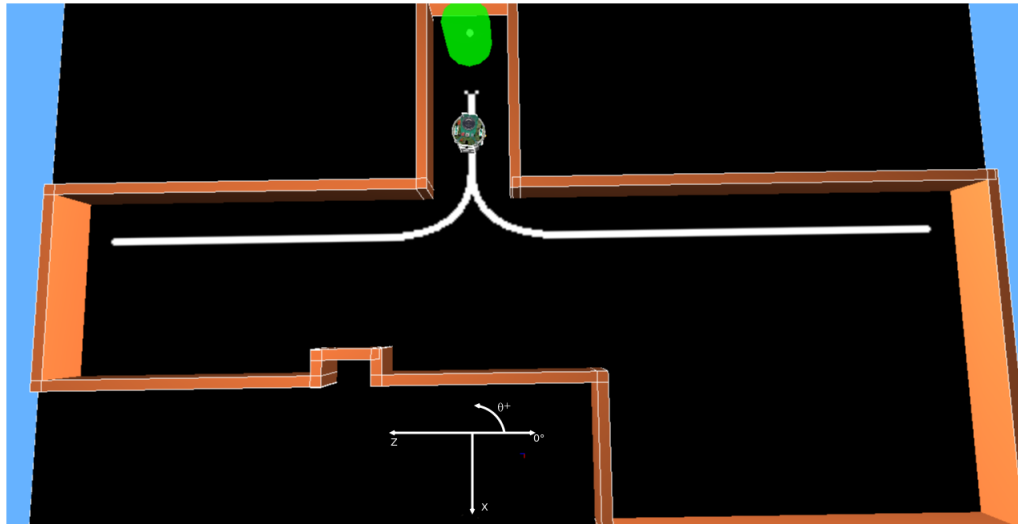


Figura 14: Arena de simulación y ejes de coordenadas

1.4.5.1.1. Detalle del comportamiento La decisión de dejar sólo dos líneas, acompañada por la elección de la ubicación de la zona de descarga, nos facilitó la composición de éste comportamiento. Como se ve en la figura 14, para ir a la línea se debe girar hasta tener un ángulo de $\frac{\pi}{2}$ y luego ir hacia adelante.

1.4.5.1.2. Implementación del comportamiento El *pseudo-código* de este comportamiento es sencillo, ya que no requiere cálculos extras:

```

por cada paso
    angulo_actual = obtener_angulo_actual(odometria)
    si esta_en_un_entorno_de(angulo_actual,PI/2) entonces
        si angulo_actual > PI/2 && angulo_actual < 3PI/2 entonces
            giro_para_la_derecha
        sino
            giro_para_la_izquierda
        fin si
    sino
        voy_hacia_linea
    fin si

```

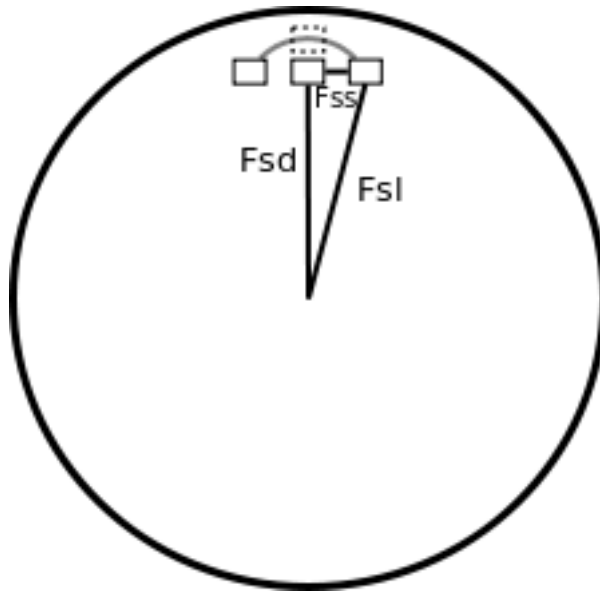


Figura 15: Disposición de sensores de piso

Hicimos la verificación de que el ángulo esté entre $\frac{\pi}{2}$ y $3\frac{\pi}{2}$ para evitar girar más de π , girando a la izquierda o a la derecha dependiendo cual sea el caso. Para ver esto más claramente, veamos que sucediese si no estuviera:

```

si esta_en_un_entorno_de(angulo_actual,PI/2) entonces
    giro_para_la_izquierda
sino

```

En el caso que el angulo actual sea π , el robot giraría un total de $3\frac{\pi}{2}$ hasta llegar al ángulo destino $\frac{\pi}{2}$, cuando en realidad girando para el sentido contrario sólo tendría que girar $\frac{\pi}{2}$.

Se puede observar en el código usamos datos calculados por la odometría, en este caso, la orientación actual del robot. El lector se podría preguntar: “Si la odometría tiene la posición actual del robot, ¿porqué no se usaron los datos de la misma para ir hacia la base?”. En principio, esto requiere que el robot tenga conocimiento acerca de la ubicación de la base. Por otro lado, se hubiera tenido que utilizar algún algoritmo de *Path Planning* para realizar el recorrido, algo que no concuerda con la arquitectura elegida. Es importante destacar el uso de datos calculados por la odometría porque si la misma llega a tener un error grande, puede llevar a una activación errónea de comportamientos. En la sección 1.5.1 se puede ver la incidencia de un error grande de la odometría en el comportamiento emergente del robot.

1.4.5.2. Entrar a línea

1.4.5.2.1. Detalle del comportamiento Para seguir la línea, es necesario que primero el robot esté posicionado sobre ella y alineado con la misma.

También se necesita que la dirección del robot sea la que lo lleve hacia la zona de descarga, por lo que una vez en la línea, el robot deberá girar dependiendo de que lado se encuentre la misma (Ver figura 14).

1.4.5.2.2. Implementación del comportamiento

```

angulo_final = obtener_angulo_final(obtener_linea(odometria))
tita = atan(dist_sens_piso_X,dist_sens_piso_Y)
si ( esta_en_la_linea(sensor_piso(DERECHA)) ) entonces
    tita = -tita
fin si
si ( esta_en_la_linea(sensor_piso(MEDIO)) ) entonces
    tita = 0
fin si

si ( tita != 0 ) entonces
    girar(tita)
fin si

distancia_a_recorrer = dist_sens_piso_Y;
si ( tita != 0 ) entonces
    distancia_a_recorrer = sqrt( dist_sens_piso_X^2 + dist_sens_piso_Y^2 )
fin si

recorrer(distancia_a_recorrer)

angulo_actual = obtener_angulo(odometria)

girar(normalizar(angulo_actual - angulo_final))

```

El primer giro del robot es para lograr que el robot quede con el sensor de piso del medio sobre la línea. Como mostramos en la figura 16, en el caso (a) y (b) el robot girará de forma tal que llegue un caso parecido al (c) ya posiblemente el sensor del medio no estará en la línea. Notar que si el sensor del medio está en la línea, entonces el robot no gira, cualquiera sea el estado de los sensores de los costados. El ángulo que debe girar está dado por $\theta = \arctan(\frac{F_{ss}}{F_{sd}})$ (Ver figura 15) o $-\theta$ según cual sea el sensor que esté sobre la línea.

Luego del posible giro para lograr que el sensor de piso del medio quede sobre la línea, se recorre una distancia de F_{sd} , en el caso que el robot no haya girado anteriormente, o F_{sl} en el caso que sí lo haya hecho. El motivo de este trayecto es que el centro del robot quede sobre la línea, como muestra la figura 17. De ésta forma, sólo queda girar nuevamente hacia el ángulo que se quiera. Si la línea es la izquierda, entonces el robot deberá girar hasta que su orientación sea 0, o π en el caso que la línea sea la derecha.

1.4.5.3. Seguir línea Una vez que el robot está posicionado y con el sensor del medio sobre la línea, sólo basta con seguirla para llegar hacia la zona deseada. Éste comportamiento es sencillo de desarrollar.

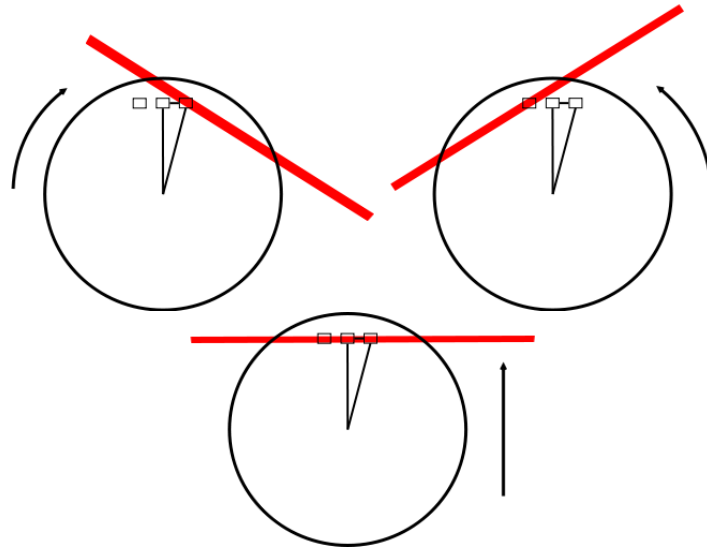


Figura 16: Posibles estados iniciales de entrar a la línea

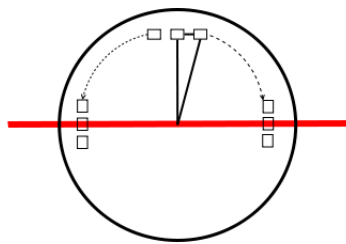


Figura 17: Centro del robot sobre la línea y posibles giros

1.4.5.3.1. Detalle del comportamiento Para lograr que el robot siga la línea el objetivo es tratar de mantener sólo el sensor del medio sobre la línea. Entonces, basta con analizar que se debe hacer en los siguientes casos:

1. El sensor de la derecha está sobre la línea
2. El sensor de la izquierda está sobre la línea

En el primer caso, se debe lograr sacar el sensor de la derecha de la línea, describiendo un pequeño arco hacia ese lado. El segundo caso es análogo: para sacar el sensor de la izquierda de la línea se debe seguir una trayectoria con un pequeño ángulo hacia la izquierda.

1.4.5.3.2. Implementación del comportamiento El *pseudo-código* es simple:

```

por cada paso
    veloc_izq = veloc_der = VELOC_SEGUIR_LINEA
    si ( esta_en_la_linea(sensor_piso(IZQUIERDA)) ) entonces
        veloc_izq *= ( 1 - FACTOR_DE_GIRO )
        veloc_der *= ( 1 + FACTOR_DE_GIRO )
    fin si
    si ( esta_en_la_linea(sensor_piso(DERECHA)) ) entonces
        veloc_izq *= ( 1 + FACTOR_DE_GIRO )
        veloc_der *= ( 1 - FACTOR_DE_GIRO )
    fin si
    poner_velocidades_en_ruedas(veloc_izq,veloc_der)

```

1.4.6. Descargar Basura

Una vez que el robot logró llegar a la zona de descarga de basura, debe descargarla. Para ésto decidimos ubicar un servo en la parte trasera del robot, con la misma idea del servo en la parte posterior utilizado para recolectar.

1.4.6.1. Detalle del comportamiento Para descargar la basura el robot debe posicionarse de forma tal que la compuerta de descarga quede adyacente a la zona. Dado que para llegar a la misma el robot siguió la línea, al finalizar va a estar orientado con un ángulo en un entorno de $\frac{\pi}{2}$, mirando la zona de descarga. Como el servo de descarga se encuentra en la parte trasera, deberá realizar un giro para luego poder descargar la basura.

1.4.6.2. Implementación del comportamiento

```

posicionarse()
levantar(servo_trasero)
recorrerdistancia(ANCHO_ROBOT)
cerrar(servo_trasero)

```

1.4.7. Ir a base de recarga de batería

Ayudados por la elección que tomamos de poner la zona de descarga de basura muy cercana a la zona donde se recarga la batería, decidimos utilizar la misma estrategia de seguir la línea para llegar hacia la misma.

La diferencia entre ambos casos es el estímulo ante el cual se activan. En el caso de ir a la zona de descarga, el estímulo proviene del sensor del depósito interno de basura que indica que el mismo está lleno. En el caso de ir a la base de recarga de batería, el estímulo para la activación depende de los valores de dos sensores de batería que indican batería baja:

- El sensor de la batería del robot, de donde se alimentan los sensores, actuadores y motores.
- El sensor de la computadora que corre el controlador.

1.4.8. Cargar Batería

1.4.8.1. Detalle del comportamiento

1.4.8.2. Implementación del comportamiento

```
posicionarse()
mientras( bateria_no_llena(BATERIA_ROBOT)
          o bateria_no_llena(BATERIA_PC) ) hacer

    esperar()
fin mientras
```

1.4.9. Evitar Obstáculos

Evitar obstáculos es uno de los comportamientos con mayor jerarquía en la arquitectura que elegimos. Su nivel se debe a la importancia que tiene en un ambiente estructurado pero dinámico como es el elegido. El objetivo del robot es facilitar una tarea, sin entorpecer el tránsito de personas.

1.4.9.1. Detalle del comportamiento Para lograr tener conocimiento sobre la proximidad de un obstáculo, utilizamos sensores de proximidad explicados en ???. Dependiendo de la proximidad sensada por un sensor, el robot deberá alejarse de ese lado para evitar un posible choque. La activación del comportamiento dependerá entonces de un valor menor del sensor que haga que la distancia hacia un obstáculo lleve al robot a quedarse estancado o le imposibilite moverse.

1.4.9.2. Implementación del comportamiento La implementación de éste comportamiento la hicimos utilizando una red neuronal sin capas ocultas (ver figura 18) usando los sensores de distancia como entradas y las 2 neuronas de salida indicando los valores a ser seteados a los motores de las ruedas. Notar que hay conexiones tanto inhibitorias como excitatorias. Como es de esperarse, ambos sensores traseros (4 y 5) exitan a ambos motores. Distinto es el caso de los sensores del lado izquierdo (5, 6 y 7) que exitan el motor ubicado

de su lado e inhiben el motor del lado opuesto, de forma tal que el robot gire para el lado opuesto de la ubicación de los sensores. La misma idea se sigue con los sensores (0, 1 y 2) ubicados en el costado derecho del robot. Luego de entrenar la red, obtuvimos los siguientes valores para los pesos de la misma:

Rueda	W_0	W_1	W_2	W_3	W_4	W_5	W_6	W_7
Izquierda	-0.9	-0.85	-0.2	0.6	0.5	0.35	0.8	0.6
Derecha	0.9	0.85	0.2	0.6	0.5	-0.35	-0.8	-0.6

Cuadro 1: Asignación de pesos para evitar obstáculos

Se puede ver que los pesos que influyen en el motor de una rueda influyen con igual fuerza pero distinto signo en el motor de la rueda contraria. Este comportamiento, en *pseudo-código* puede verse como:

```

por cada paso
  veloc_izq = suma( coeficiente(RUEDA_IZQ)(SENSOR_I) * VALOR(SENSOR_I) )
  veloc_izq *= FACTOR_DE INCIDENCIA
  veloc_der = suma( coeficiente(RUEDA_DER)(SENSOR_I) * VALOR(SENSOR_I) )
  veloc_der *= FACTOR_DE INCIDENCIA
  poner_velocidades_en_ruedas(veloc_izq,veloc_der)

```

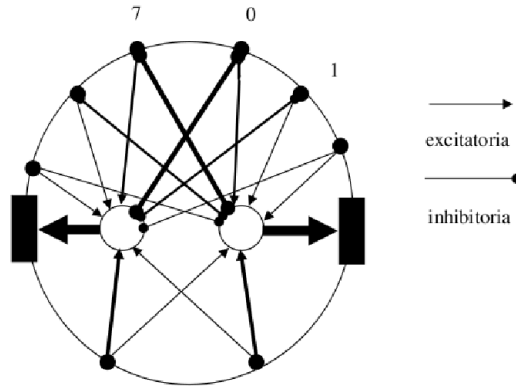


Figura 18: Red neuronal entre los sensores de distancia y los motores

1.4.10. Salir de situaciones no deseadas

Salir de situaciones no deseadas surgió como un comportamiento para ayudar al objetivo de la autonomía del robot. A medida que fuimos corriendo las simulaciones, observamos que había situaciones donde corría peligro la autonomía del robot. Un ejemplo de estas situaciones es el caso donde se “activan mutuamente” entre dos comportamientos.

Para ver esto más claramente, supongamos dos comportamientos A y B con nivel en la jerarquía $N(A)$ y $N(B)$, siendo $N(A) > N(B)$. Si la respuesta a un

estímulo de A lleva a la activación de B y a la desactivación de A y luego la respuesta de B lleva a la activación de A , podría llegar a entrarse en un ciclo si es que esta situación se da por un tiempo prolongado. El mayor peligro para la autonomía se corre cuando A es el comportamiento de *evitar obstáculos* y B es el comportamiento de *ir a zona de recarga de batería* ya que el robot podría terminar quedándose sin batería en ese ciclo.

1.4.10.1. Detalle del comportamiento Las situaciones no deseadas se dan la mayoría de los casos cuando un comportamiento hace girar al robot hacia un lado y el otro comportamiento hacia el lado contrario, aproximadamente en la misma magnitud. Esto quiere decir que la posición del robot se mantiene alrededor de un punto por un período prolongado de tiempo, por lo que decidimos tomar este hecho como estímulo para la activación de este comportamiento. La respuesta del comportamiento es, entonces, girar un ángulo que cambie la dirección del robot y además, que la suma de esa magnitud no sea periódica. Esta última condición se pide por el siguiente escenario:

- Los comportamientos A y B se “activan mutuamente” cuando la orientación del robot es 0 .
- Los comportamientos C y D se “activan mutuamente” cuando la orientación del robot es π .
- La respuesta de *salir de situaciones no deseadas* es girar un ángulo π .

1.4.10.2. Implementación del comportamiento

```

angulo_actual = obtener_angulo_actual(odometria)
nuevo_angulo = angulo_actual + ANGULO_A_SUMAR
girar(nuevo_angulo)

```

1.5. Odometría

Para saber donde se encuentra el robot en cierto momento, se utiliza odometría. Ésta se basa en la medición de las cuentas dadas por un motor para obtener el desplazamiento realizado por la rueda asociada al mismo. Dado que el motor que usamos fue un motor diferencial para cada rueda, ésto es posible de realizar. A diferencia de los métodos de posicionamiento absoluto, la odometría da una estimación del desplazamiento *local* a la ubicación anterior del robot, por lo cual *un error* en una estimación *se propaga* hacia las siguientes estimaciones. Para calcular la posición en el instante n P_n y la orientación O_n en base a la posición en el instante $(n-1)$ P_{n-1} y la correspondiente orientación O_{n-1} , usamos las siguientes fórmulas:

$$d_l = \frac{(e_l(n) - e_l(n-1)) * R_l}{EncRes} \quad (11)$$

$$d_r = \frac{(e_r(n) - e_r(n-1)) * R_r}{EncRes} \quad (12)$$

$$lc = \frac{d_r + d_l}{2} \quad (13)$$

$$P_n = P_{n-1} + (lc * \cos O_{n-1}, lc * \sin O_{n-1}) \quad (14)$$

$$O_n = O_{n-1} + \frac{d_r - d_l}{dbw} \quad (15)$$

donde d_l y d_r son las distancias recorridas por las ruedas izquierda y derecha respectivamente. Ambas son calculadas teniendo en cuenta los valores anteriores y actuales de los encoders $e_i(n)$, el radio de la rueda R_i y la resolución del encoder $EncRes$. dbw es la distancia entre ruedas.

Los errores que influyen en el cálculo de la odometría pueden ser de dos tipos:

- Sistemáticos: Son aquellos que pueden ser corregidos o tenidos en cuenta para disminuir el error.
- No sistemáticos: Son aquellos que pueden intentarse corregir pero *no* eliminar.

Decidimos tener en cuenta dos errores sistemáticos para disminuir el error en la odometría :

- Incerteza sobre la distancia entre las ruedas (dbw)
- Ruedas con radios diferentes (R_l y R_r)

Tuvimos en cuenta estos errores dado que son los que más contribuyen al error acumulado a lo largo del trayecto del robot. Para corregir éstas fuentes de error, utilizamos el *test del camino bidireccional describiendo un cuadrado* (UMBmark)¹. Dado que en el momento de realizar el test no había un robot físico, decidimos realizar el test sobre un e-puck simulado en Webots. Así fuimos obteniendo los valores de corrección para los diámetros de las ruedas c_i y la corrección sobre la distancia entre las ruedas c_{dbw} hasta que el error sistemático máximo dejara de disminuir. En la figura 19 se puede ver cómo disminuye el error a lo largo de las iteraciones.

¹<http://www-personal.umich.edu/~johannb/Papers/umbmark.pdf>



Figura 19: Error Sistemático Máximo a lo largo de las iteraciones

En la última iteración obtuvimos los coeficientes de corrección $c_l = 0,99998703$, $c_r = 1,00001297$ y $c_{dbw} = 1,092171094$ para ruedas con radio $R_l = R_r = 0,205$, una distancia entre ruedas de $0,052$ y una resolución de encoder $EncRes = 159,23$.

1.5.1. Problemas con la odometría

A LA LA LALALA

1.6. Interfaces con hardware y módulo de reconocimiento de objetos

Decidimos hacer que el controlador encargado de realizar los comportamientos sea independiente de la forma con la que se implemente el hardware y la forma con que se implemente el reconocimiento de los objetos. Para ésto definimos interfaces, de forma tal que el controlador las mismas y tanto el hardware como el módulo de reconocimiento de objetos puedan cambiar su implementación pero brindando siempre la información que necesita el controlador para poder realizar los comportamientos. Ésta decisión nos permitió realizar un controlador que sea capaz de ser ejecutado tanto en Webots, donde se hizo el desarrollo, como en la base real del robot. Cabe aclarar que el traspaso de la simulación a la realidad no es instantánea, ya que hay que calibrar los dispositivos, realizar nuevamente la odometría, entre otras cosas, pero el trabajo demandado es mucho menor ya que una corrección en la lógica de los comportamientos se puede observar tanto en un simulador como en la realidad.

1.6.1. Interfaz con hardware

La interfaz con el hardware se basa en tener clases encargadas de obtener los valores de los sensores o del accionar de los actuadores.

En la implementación de la interfaz que corrimos en Webots, hicimos llamadas al controlador del simulador, que utiliza sensores y actuadores simulados.

En la implementación que se comunica con el robot físico, las llamadas las hicimos a un servidor encargado de enviar y recibir paquetes del protocolo descrito en la sección ?? a través del puerto serial.

De esta forma es cuestión de decidir que implementación se usa en base a si se quiere correr en un simulador como Webots, o en el robot físico. Si se quisiese usar otro simulador u otro tipo de hardware, sólo hace falta implementar la interfaz que cumpla con lo establecido e indicarle a la capa de comportamientos que la utilice para realizar su ejecución.

1.6.2. Interfaz con módulo de reconocimiento de objetos usando Visión

Dado que el desarrollo de los comportamientos lo hicimos separado del desarrollo del módulo que reconoce objetos utilizando una cámara, decidimos hacer una interfaz de forma tal que se puedan desarrollar de forma paralela. Como el controlador de comportamientos es **usuario** del módulo de reconocimiento, necesita saber en un instante de tiempo t , que objetos están siendo reconocidos. Para ésto el módulo debe tener una fuente de información, en este caso, una cámara dado que se usa Visión.

Desde el punto de vista de la anatomía, ésto se puede ver como los ojos (cámara), la parte del cerebro encargada de analizar el estímulo recibido por los mismos (módulo de reconocimiento) y la parte del cerebro encargada de analizar los estímulos y realizar las acciones (controlador de comportamientos). Si se hubiese usado algún tipo de conjunto de sensores táctiles para reconocer objetos (mano), en vez de visión, el módulo de reconocimiento de objetos bien podría analizar la información que ellos proveen e informar al controlador sobre su análisis.

Ésta analogía puede llevar a pensar que los sensores de distancia u otros dispositivos que usamos en este desarrollo bien podrían formar parte de otro módulo, y no se estaría equivocado. La razón por la cual el módulo de visión está separado y el resto de los sensores no, es que el procesamiento de una secuencia de imágenes es muy complejo y pesado, como se muestra en la sección ??.

1.7. Resultados obtenidos

En ésta sección se describen los parámetros utilizados en la simulación para la obtención de resultados sobre los diferentes comportamientos. Luego se presentan dichos resultados y se sacan conclusiones sobre los mismos.

A lo largo del proceso de desarrollo se fueron realizando diferentes simulaciones, de las cuales observamos defectos en las implementaciones de los comportamientos así como retardo y detalles que se podrían mejorar en algunos de los mismos y los fuimos corrigiendo.

La arena utilizada en dichas simulaciones se puede ver en la figura 20. A lo largo de la misma están ubicadas 15 basuras, un área de recarga de batería y un área de descarga de basura, marcadas por el cilindro naranja. Los valores de los parámetros utilizados se pueden observar en el cuadro 2. Los ángulos están expresados en radianes y las distancias están expresadas en metros, así como también los radios, correcciones y separaciones. La resolución de la cámara está expresada en pixeles, la de la grilla en unidades y los tiempos están en milisegundos.

Parámetro	Descripción del parámetro	Valor
$P_r(0)$	Posición Inicial del robot	(-0.295,-0.4)
$O_r(0)$	Orientación Inicial del robot	0
R_r	Radio del robot	0.026
dbw	Distancia entre ruedas	0.052
c_{dbw}	Corrección de la distancia entre ruedas	1.09217109
R_l, R_r	Radios de las ruedas	0.0205 0.0205
c_l, c_r	Correcciones de los radios de las ruedas	0.99998702 1.00001297
$EncRes$	Resolución del encoder	159.23
Cd	Distancia de la cámara al centro del robot	0.0355
Ch, C_h	Altura de la cámara	0.038
FOV_h	Field of view horizontal	1.1
ac	Ángulo de la cámara	-0.5
C_{rw}, C_{rh}	Resolución de la cámara	640 x 480
A_w, A_h	Dimensiones del arena	2 x 1.2
A_{rw}, A_{rh}	Resolución de la grilla	100 38
Fsd	Dist. s.de piso del medio al centro del robot	0.03
Fss	Separación entre sensores de piso	0.01
Ts	Time Step	32

Cuadro 2: Parámetros utilizados para las simulaciones

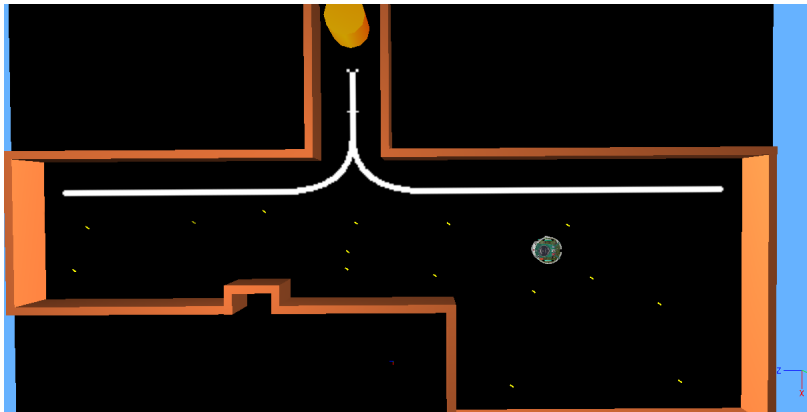


Figura 20: Arena utilizada para las simulaciones

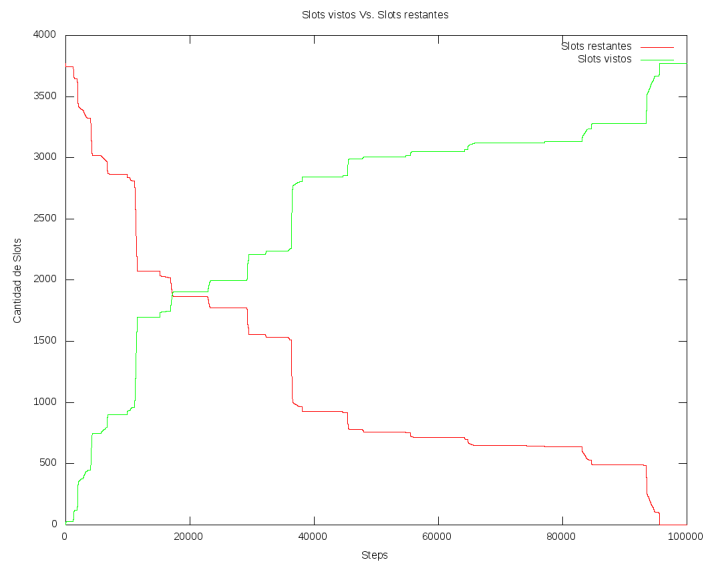


Figura 21: Área vista y Área no vista a lo largo de la simulación

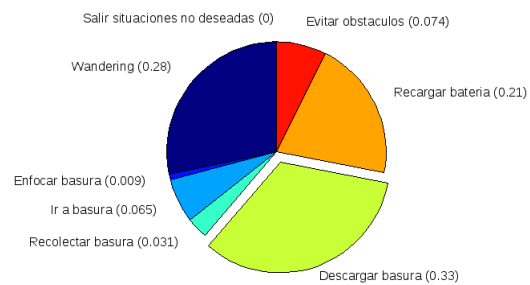


Figura 22: Distribución de tiempos de los comportamientos en la simulación

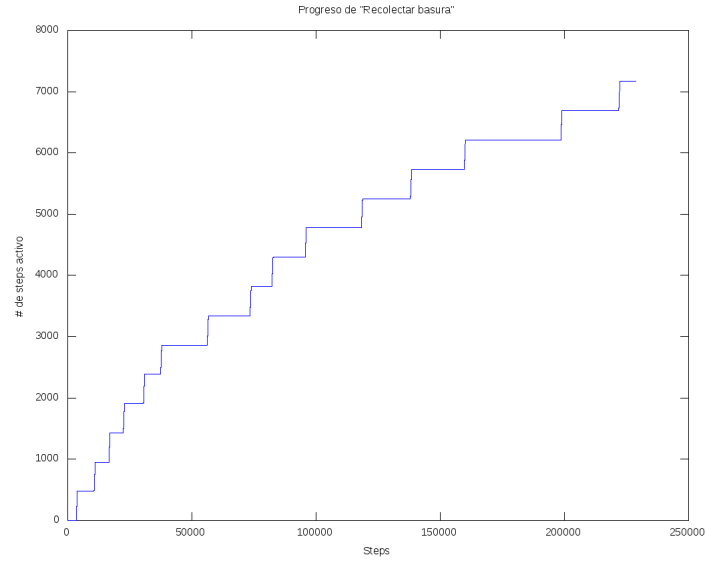


Figura 23: Progreso del comportamiento *Recolectar Basura*

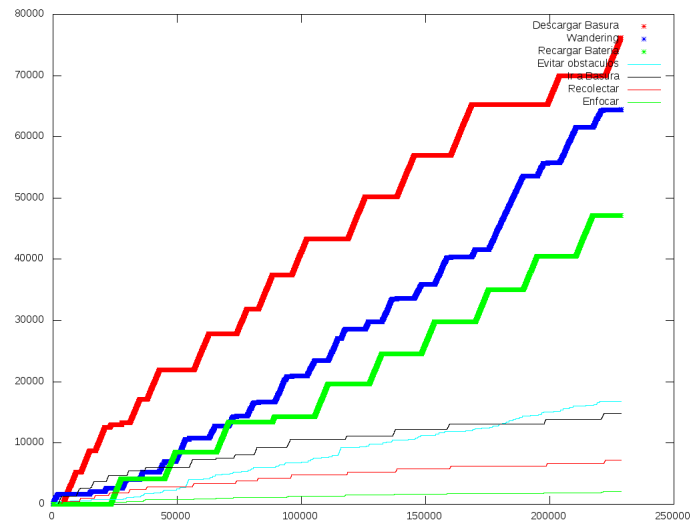


Figura 24: Evolución de los comportamientos a lo largo de la simulación

Comportamiento	# total de steps activo	# veces que se activó
Wandering	64476	1330
Enfocar basura	2057	1384
Ir a basura	14832	1218
Recolectar basura	7166	15
Ir a descargar basura	76160	321
Ir a recargar batería	47188	265
Evitar obstáculos	16843	1507
Salir situaciones indeseadas	0	0

Cuadro 3: Resultados sobre steps que los comportamientos están activos

Comportamiento	Promedio	Desvío Estándar	# máximo
Wandering	48.4410	225.07466	2212
Enfocar basura	1.4863	3.27591	67
Ir a basura	12.1773	58.44234	1222
Recolectar basura	477.7333	0.70373	478
Ir a descargar basura	237.2586	840.49848	6012
Ir a recargar batería	178.0679	647.39786	3950
Evitar obstáculos	11.1765	54.72354	1386
Salir situaciones indeseadas	0	0	0

Cuadro 4: Resultados sobre steps que los comportamientos están continuamente activos

- 1.7.1. Tiempo promedio en recolectar basura**
- 1.7.2. Tiempo promedio de wandering**
- 1.7.3. Distribución de tiempos entre los diferentes comportamientos**
- 1.7.4. Relación $(t_{\text{cargando}} + t_{\text{ir a base}}) / \text{tiempo total}$**
- 1.7.5. Porcentaje de espacio cubierto**
- 1.7.6. Tiempo cubrir toda la arena**

aaa

	W	EB	IAB	RB	DB	R	EO	SSI	Maximo	Ind Max
W	0	303	176	1	12	5	833	0	833	7
EB	238	0	1036	2	1	0	107	0	1036	3
IAB	229	967	0	12	2	1	7	0	967	2
RB	12	3	0	0	0	0	0	0	12	1
DB	14	0	0	0	0	2	305	0	305	7
R	9	0	0	0	1	0	255	0	255	7
EO	828	111	6	0	305	257	0	0	828	1
SSI	0	0	0	0	0	0	0	0	0	0
Maximo	828	967	1036	12	305	257	833	0	-	-
Ind Max	7	3	2	3	7	7	1	0	-	-

Cuadro 5: Transiciones entre estados

N° muestra	(%)	(%) Acumulado
1	0.246	0.246
2	0.258	0.505
3	0.081	0.586
4	0.167	0.754
5	0.044	0.798
6	0.011	0.809
7	0.017	0.827
8	0.003	0.830
9	0.038	0.869
10	0.130	1

Cuadro 6: Porcentaje de arena cubierta

1.8. Conclusión

Conclusión de comportamientos

A. Primer apéndice Comportamientos

Mostrar como está implementada la arquitectura como software,

B. Segundo apéndice Comportamientos

que habría que hacer para agregar un comportamiento nuevo, modificar uno existente.

C. Tercer apéndice Comportamientos

Que hay que hacer para agregar un sensor o actuador nuevo.

D. Cuarto apéndice Comportamientos

Que hay que hacer en caso que se modifique el protocolo.

E. Quinto apéndice Comportamientos

Como es la interfaz con vision y porque se hizo de esa forma.

Referencias

- [1] Salvatore Candido. Autonomous robot path planning using a genetic algorithm.
- [2] Amalia F. Foka and Panos E. Trahanias. Predictive autonomous robot navigation. In *In Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 490–495, 2002.
- [3] Maria C. Neves and R. S. Tome. A control architecture for an autonomous mobile robot. In *Proceedings of Autonomous Agents*. ACM, 1997.
- [4] Stefano Nolfi. Evolving non-trivial behaviors on real robots: A garbage collecting robot. *Robotics and Autonomous Systems*.
- [5] K. Wedeward R. Clark, A. El-Osery and S. Bruder. A navigation and obstacle avoidance algorithm for mobile robots operating in unknown, maze-type environments. December 2004.