Proyecto Final
Robot recolector de residuos
Arquitectura de Comportamientos y Algoritmos

Guillermo Campelo
Juan Ignacio Goñi
Diego Nul

6 de agosto 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, Webots, OpenCV, detección, objetos*

Índice

1. Arquitectura de Comportamientos

1.1. Introducción

Como mencionamos 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 está al aire libre.

En esta sección de *Arquitecturas de Comportamientos* detallamos las acciones que debería llevar a cabo 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 analizamos 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 detallamos la arquitectura elegida para llevar a cabo y organizar los comportamientos elegidos, y las ventajas y desventajas de usar la misma. En la sección 1.4 detallamos uno por uno los comportamientos que elegimos para que posea el robot, y sus correspondientes implementaciones en *pseudo-código*. En la sección 1.5 explicamos que es la odometría, para qué sirve y qué ventajas y desventajas tiene usarla. También detallamos el test utilizado para mejorar la eficiencia de la misma. En la sección 1.6 describimos cómo estructuramos el controlador del robot para que podamos ir desarrollando y probando el mismo con un simulador y minimizar el trabajo al pasarlo a el robot físico. Los resultados de performance y eficiencia del controlador desarrollado los obtuvimos de la simulación, y los mostramos en la sección 1.7. Finalmente, en la sección 1.8 sacamos conclusiones de los resultados obtenidos y las dificultades encontradas a lo largo del desarrollo de éste proyecto final.

1.2. Investigaciones previas

A continuación presentamos trabajos realizados por otros autores relacionados en alguna forma con el nuestro. Primero damos una breve descripción del trabajo del otro autor y luego lo comparamos con nuestro trabajo, analizando similitudes y diferencias entre ambos.

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

En este 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 objetivo

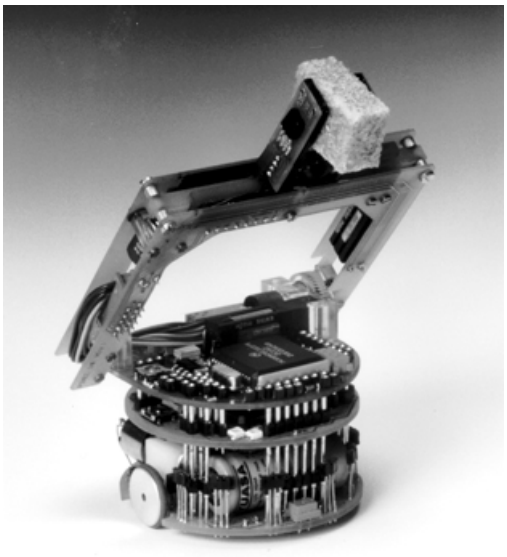

Figura 1: Robot Khepera con el módulo Gripper

del robot es llevar pequeños objetos a un lugar de depósito. Se utiliza *aprendizaje por refuerzo* para asociar las velocidades angulares de las ruedas 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 realizar las siguientes comparaciones:

- 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 el módulo Gripper" agregado al Khepera. Este módulo simula un brazo con dos dedos. De ésta forma, la recolección se realiza juntándolos y la liberación se realiza separándolos. Nuestro trabajo se basa más en la actividad humana que en el comportamiento humano para realizar la recolección, ya que podría compararse con un recolector de basura que primero guarda lo encontrado usando una pala en su tacho y luego lo descarga en un depósito de mayor tamaño.

- Método de aprendizaje: En nuestro trabajo no utilizamos aprendizaje por refuerzo o algoritmos evolutivos. No es el caso de Nolfi, que utiliza ambas técnicas para evolucionar el comportamiento deseado.

- Robot Utilizado: Como mencionamos anteriormente, 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*(Ver sección 1.3) propuesta por *Brooks* y que usamos al realizar éste proyecto.
La arquitectura propuesta en el paper, o *Control System Arquitecture*, 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 competencia particular 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, no utilizamos ésta organización de los comportamientos en capas. Sin embargo, podemos divisar que algunos de los comportamientos que propusimos corresponden a la primera capa de reflexión, como por ejemplo el evitamiento de obstáculos y otros podrían incluirse en la segunda capa de comportamientos reactivos, como sería deambular. Finalmente en la última capa estaría el reconocimiento de objectos 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 algorítmos genéticos para resolver el problema de Path Planning. Éste consiste en armar un plan, una secuencia de acciones de forma tal que a partir de un punto de origen se llegue al punto de destino, siguiendo ese plan. 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 la mejor opción a utilizar. Por el contrario, el uso de algorítmos genéticos puede ser una herramienta que se podría llegar a usar en una futura continuación de nuestro trabajo y que no utilizamos por el tiempo 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 nuestro robot.
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 que propusimos nosotros, que es tratar el *wandering* de forma separada del comportamiento de evitamiento de obstáculos. Ésto, en parte, se debió a causa de la arquitectura utilizada ya que desde ese punto de vista, ambos comportamientos tienen niveles de jerarquía muy diferentes como se puede ver en la figura 6.

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 asesgurarse la cobertura completa de la arena. El algoritmo completo se puede apreciar en la figura 3.

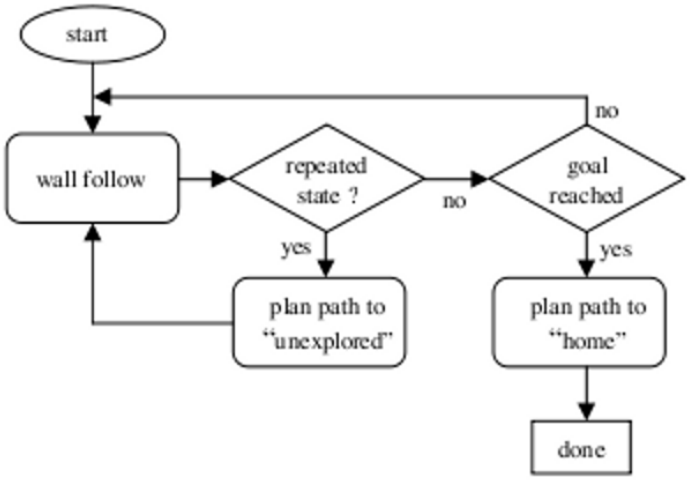

Figura 3: Algoritmo propuesto por Clark Et. al

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 conociemiento 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 coincidimos con 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 comportamien-

to es puramente reactivo y reacciona en base a los valores de los sensores de distancia y no tiene memoria.

1.3. Arquitectura propuesta

Una arquitectura basada en comportamientos define la forma en que los mismos son especificados, desde su granularidad (qué 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. Esta 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 tuvimos 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 de él mismo,
- La simplicidad es una virtud,
- Los sistemas deben ser construidos incrementalmente.

Cada comportamiento es un par estímulo-respuesta. Tal como mostramos 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 lo devuelve a su estado original. El nombre *Subsumption* proviene

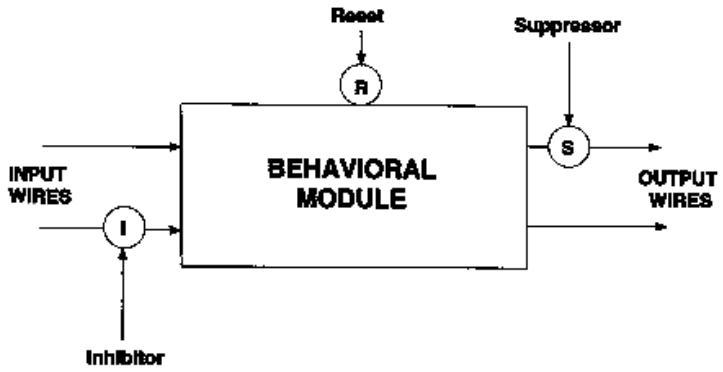

Figura 4: Esquema de comportamiento

de la forma en que los comportamientos son coordinados entre sí. Hay una jerarquía donde los comportamientos de la arquitectura tienen mayor o menor prioridad según su posición. Los comportamientos de los niveles inferiores no tienen conocimiento de los comportamientos de las capas superiores. Gracias a ésto, 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. Ésto se debe a que la respuesta de un comportamiento ante un estímulo resulta en un cambio en el mundo y, por lo tanto, en la relación del

robot con el mismo. De ésta manera, el robot en su próximo paso sensará 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 robot 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 cuantas acciones disjuntas es necesario llevar a cabo para el cumplimiento de la tarea.

Un ejemplo de esta arquitectura se puede observar en la figura 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 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.

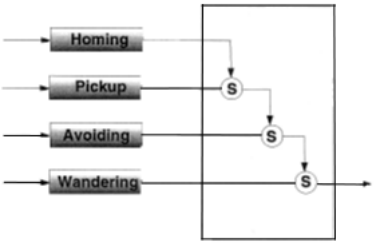

Figura 5: Ejemplo de la arquitectura Subsumption

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.

Como requerimiento de nuestro proyecto teníamos la realización de un robot autónomo que recolecte basura de su entorno dinámico pero estructuralmente estático. 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 solo para poder continuar con su actividad.
- *Wandering* (1.4.1): El robot, al ser autónomo, no es radio-controlado y debe poder recorrer el entorno por sí mismo.
- *Evitamiento de obstáculos* (1.4.9): Debido a la naturaleza dinámica del entorno, el robot debe ser capaz de navegar sin chocarse contra las paredes 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).

En la figura 6 mostramos los comportamientos implementados y su orden de jerarquía. Se puede ver que hay más comportamientos de los detallados anteriormente ya que la forma en que implementamos los comportamientos básicos del robot requirió el desarrollo de otros auxiliares.

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

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

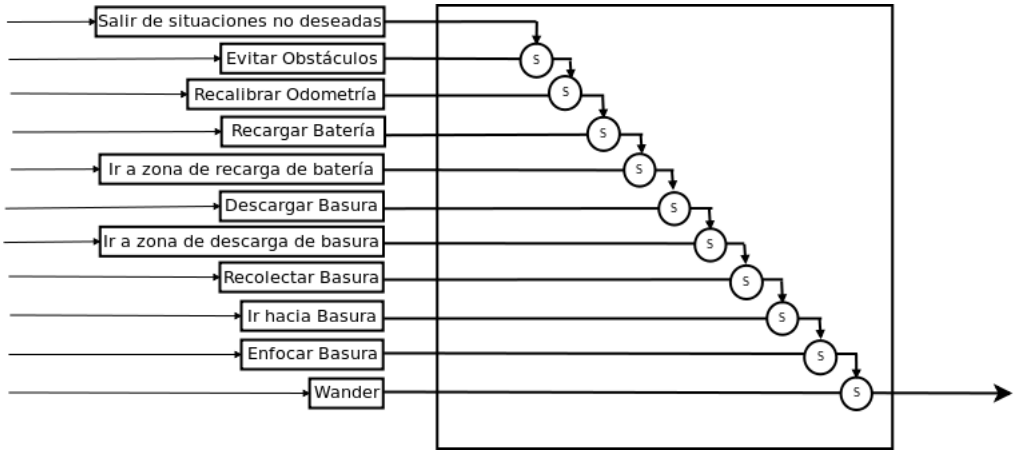

Figura 6: Arquitectura de comportamientos implementada

1.4.1. Wandering

1.4.1.1. Detalle del comportamiento

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

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

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 una segunda implementación. 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.

Ésto, 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 la arena sólo mantenemos el timestamp de la última vez que el robot la visitó.

1.4.1.2. Implementación del comportamiento

La segunda implementación en *pseudo-código* es la siguiente:

```
por cada paso
    zona = pedir_zona_vista(camara)
    marcar_zona_como_vista(modelo_del_mundo,zona)
    ultima_zona_visitada = pedir_ultima_zona_visitada(modelo_del_mundo)
    velocidades = calcular_velocidades_de_ruedas(ultima_zona_visitada)
    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*. Analizando la figura 7 podemos ver que:

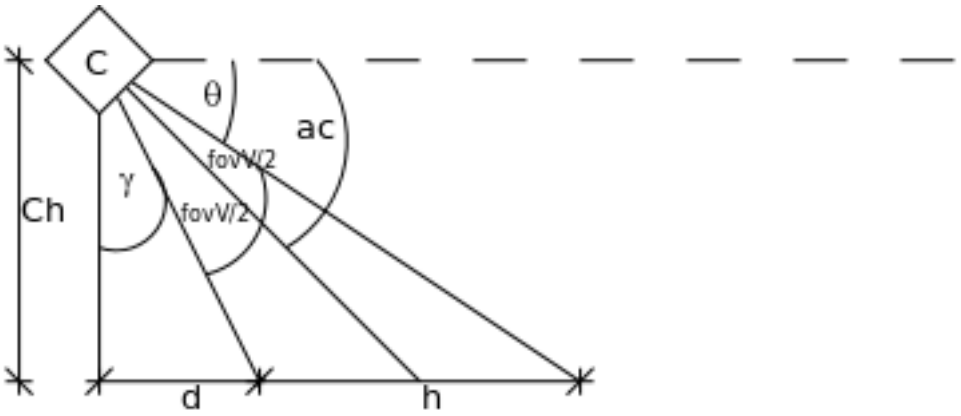

Figura 7: Diagrama de posición de la cámara

$$ac = \frac{FOV_v}{2} + \theta \tag{1}$$

$$\gamma + FOV_v + \theta = \frac{\pi}{2} \tag{2}$$

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

$$\tan(\gamma + FOV_v) = \frac{d+h}{C_h} \tag{4}$$

Organizando las ecuaciones, podemos deducir que:

$$\gamma = \frac{\pi}{2} + ac - \frac{FOV_v}{2} \tag{5}$$

$$d = \tan(\gamma) * C_h \tag{6}$$

$$d + h = \tan(\gamma + FOV_v) * C_h \tag{7}$$

Por lo que obtenemos el ángulo hasta el inicio de la imágen de la cámara γ y como consecuencia, la distancia d desde la posición de la cámara hasta el inicio de la imágen y $d + h$, la distancia desde la posición de la cámara hasta el final de la imágen. Usando estos datos, la posición del robot P y basándonos en la figura 8, podemos obtener los puntos A, B, C y D del trapezoide 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 mostramos una aproximación. Consiste en primero enfocar la basura de forma

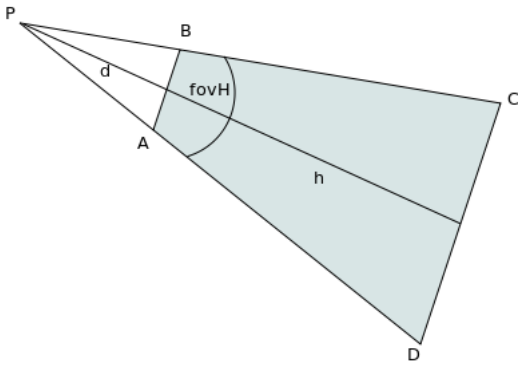

Figura 8: Zona vista por la cámara

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

Una segunda forma de lograr ésto (figura 10) no consiste en enfocar la basura, sino que se marca 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 esta alternativa no existiría el comportamiento que se está describiendo.

Dado que la primera aproximación es levemente más simple de implementar, y teniendo en cuenta el principio enunciado en la sección 1.3 *"La simplicidad es una virtud"*, elegimos implementarlo, a pesar de tener un posible inconveniente, como mostramos 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 enfocará 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.

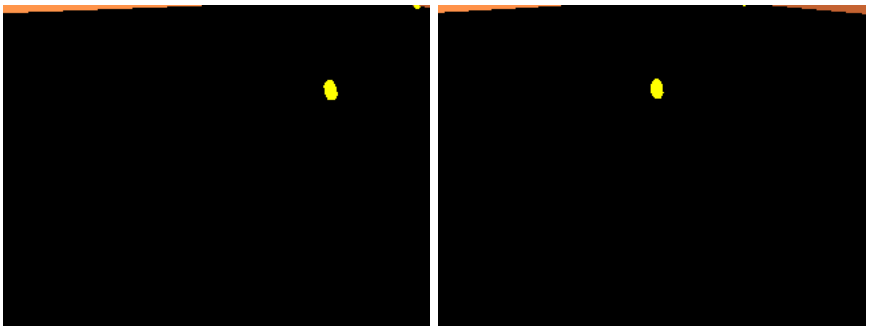

Figura 9: Primera aproximación de "Ir a la basura"

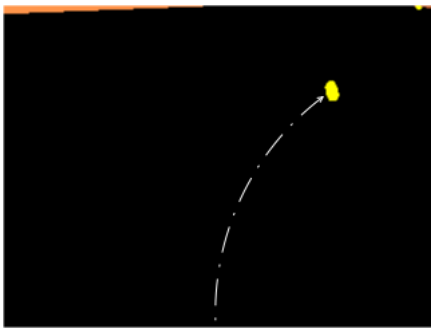

Figura 10: Segunda aproximación de “Ir a la basura”

Figura 11: Posible inconveniente con la primera aproximacíon de “Ir a la basura”

1.4.2.1. Detalle del comportamiento

La primer aproximación se puede describir como:

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

1.4.2.2. Implementación del comportamiento

Para la implementación se sigue el siguiente *pseudo-código*:

```
por cada paso
    lista_de_basuras = obtener_lista_de_basuras(modulo_de_reconocimiento)
    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 en que se resuelve el problema de encontrar una basura (ver figura 1.4.2), el comportamiento de ir a basura resulta trivial, ya que luego de ser enfocada, la basura está delante del robot y lo basta con 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 encontró una basura
2. La basura se encuentra en un entorno del centro del eje horizontal de la imagen obtenida de la cámara

1.4.3.2. Implementación del comportamiento

```
por cada paso
    distancia = obtener_distancia_a_basura(modulo_de_reconocimiento)
    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 mostramos en 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 $(0, 0)$ y $(0, C_{rh})$ son (DIST_MIN) y (DIST_MAX) respectivamente. Entonces, para obtener la distancia dy hacia el punto $(0, y)$ basta con calcular:

$$y_{angle} = FOV_v * \frac{y}{h} \tag{8}$$

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

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

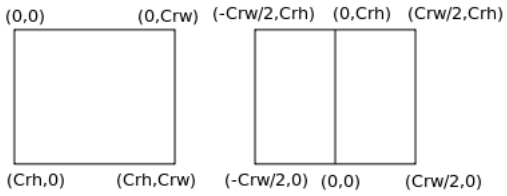

Figura 12: Conversión de coordenadas de imagen de cámara

1.4.4. Recolectar Basura

Una vez que el robot llegó a estar posicionado para recolectar la basura deseada, el comportamiento de *recolectar basura* se activa. El mecanismo para lograr ésto 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 interno. Por problemas con la simulación de este procedimiento, buscamos otro mecanismo.

El mecanismo que elegimos para usar en la simulación tiene estas componentes:

- 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(modulo_de_reconocimiento)
levantar(servo_delantero)
recorrer_distancia(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 utilizamos 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 las detallamos en la figura 13.

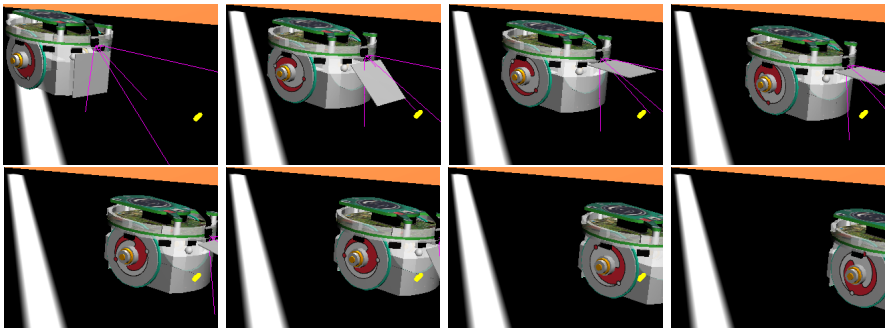

Figura 13: Etapas de recolección de basura

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. Entonces, al llenarse el depósito surge el estímulo necesario para la activación de este comportamiento.

Para ir hacia dicha zona decidimos imponerle una condición al entorno donde el robot actuará. Esta condición consiste en poner líneas de forma tal que si el robot la sigue, lo lleve a 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 queden 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 una cuál es la distancia hacia la misma, luego elegir ir a la que menor distancia se encontraba. Ésta implementación, además de ser costosa, 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 a la misma distancia, 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 aquellos de los costados a una distancia Fss del eje Y. La distancia desde el sensor del medio hacia el centro del robot es a, mientras que la distancia hacia un sensor del costado es $Fsl = \sqrt{Fsd^2 + Fss^2}$.

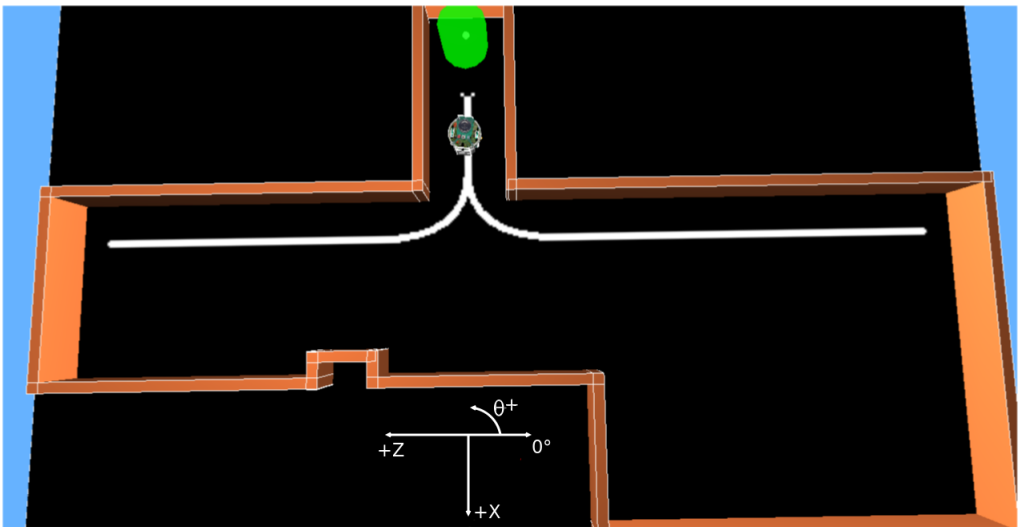

Figura 14: Arena de simulación y ejes de coordenadas

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 mostramos en la figura 14, para ir a la línea se debemos 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
```

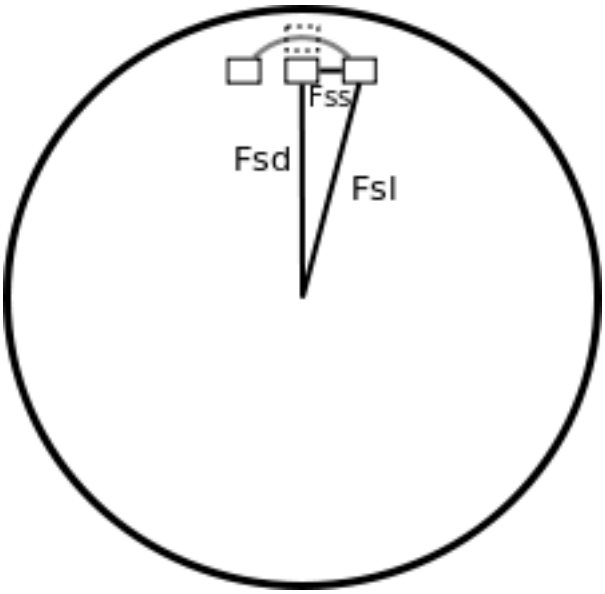

Figura 15: Disposición de sensores de piso

```
        giro_para_la_izquierda
    fin si
sino
    voy_hacia_linea
fin si
```

Hicimos la verificación de que el ángulo esté entre $\frac{\pi}{2}$ y $\frac{3\pi}{2}$ 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 $\frac{3\pi}{2}$ 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 que 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. 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 segmento que une el sensor de piso del medio con el centro del robot quede ortogonal al segmento de 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 que mostramos en la figura 17 ya posiblemente el sensor del medio no estará en la línea. Notar que si el sensor del medio está en la línea, como es el caso de la figura 16c, entonces el robot no gira, cualquiera sea el estado de los sensores de los costados. El ángulo que debe girar está dado por $\theta = \arctan(\frac{Fss}{Fsd})$ (Ver figura 15) si debe girar hacia la izquierda, o $-\theta$ en el otro caso.

Luego del posible giro para lograr que el sensor de piso del medio quede sobre la línea, se recorre una distancia de Fsd en el caso que el robot no haya girado anteriormente o Fsl en el caso que sí lo haya hecho. El motivo de realizar este trayecto es que el centro del robot quede sobre la línea, como muestra la figura 18. 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. Este comportamiento es sencillo de desarrollar.

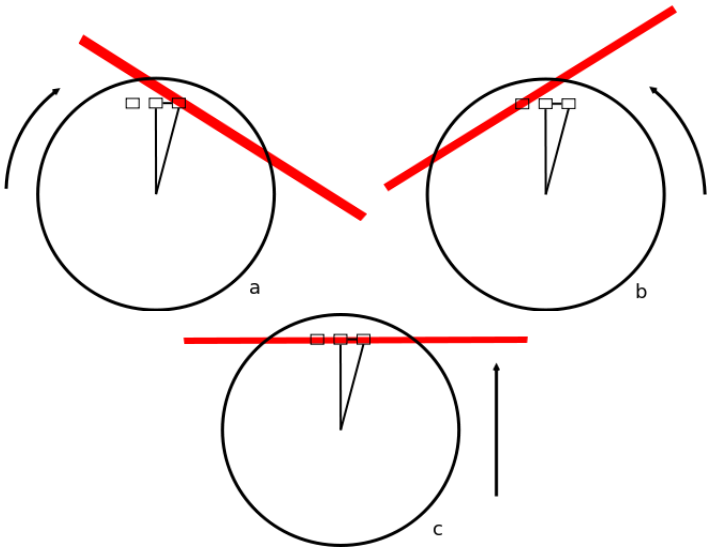

Figura 16: Posibles estados iniciales de “Entrar a la línea”. En el caso (a) el robot debe girar hacia la derecha. En el caso (b), el robot debe girar hacia la izquierda. En el caso (c) el robot no debe girar

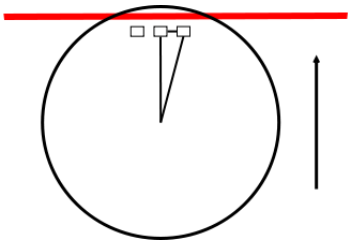

Figura 17: Posible estado final luego del primer giro del comportamiento “Entrar a la línea”

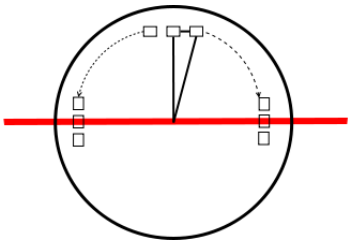

Figura 18: Centro del robot sobre la línea y posibles giros

1.4.5.3.1. Detalle del comportamiento

Para lograr que el robot siga la línea debemos es tratar de mantener sólo el sensor del medio sobre la lńea. 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 hacer ésto decidimos ubicar un servo en la parte trasera del robot, con la misma idea del servo en el frente 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 de aproximadamente π 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(time_step)
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 nuestro. 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 indicada por un determinado sensor, el robot deberá alejarse de ese lado para evitar un posible choque. La activación del comportamiento dependerá entonces del valor sensado tal que la distancia hacia un obstáculo lleve al robot a una colisión o le imposibilite moverse.

1.4.9.2. Implementación del comportamiento

La implementación de este comportamiento la hicimos utilizando una red neuronal sin capas ocultas (ver figura 19) usando los sensores de distancia como entradas y 2 neuronas de salida indicando los valores a ser seteados en los motores de las ruedas.

Debemos 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)
```

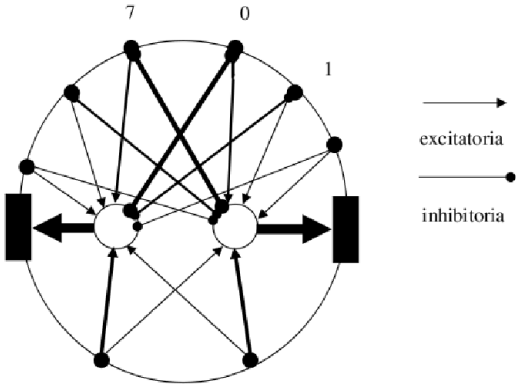

Figura 19: Red neuronal entre los sensores de distancia y los motores

1.4.10. Salir de situaciones no deseadas

Salir de situaciones no deseadas surgió como un comportamiento para ayudar al objetivo de crear un robot autónomo. Con las sucesivas simulaciones observamos que había situaciones donde corría peligro la autonomía del robot. Un ejemplo de estas situaciones es el caso donde dos comportamientos se "activan mutuamente".

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 o indeterminado. El mayor peligro para la autonomía ocurre 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. Ésto 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, girar un ángulo que cambie la dirección del robot y además que la suma de ese ángulo repetidas veces no sea periódica o lo sea luego de una gran cantidad de giros como por ejemplo, un valor de $\frac{\pi}{4} + 0{,}1$. 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 utilizamos odometría. Esta técnica se basa en la medición del encoder en cuentas realizadas por cada motor para obtener el desplazamiento realizado por la rueda asociada.
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 P_n en el instante de tiempo n y la orientación O_n en base a la posición P_{n-1} en el instante (n-1) y la correspondiente orientación O_{n-1}, usamos las siguientes fórmulas:

$$d_l = \frac{(e_l(n) - e_l(n-1)) * R_l}{EncRes} \tag{10}$$

$$d_r = \frac{(e_r(n) - e_r(n-1)) * R_r}{EncRes} \tag{11}$$

$$lc = \frac{d_r + d_l}{2} \tag{12}$$

$$P_n = P_{n-1} + (lc * \cos O_{n-1}, lc * \sin O_{n-1}) \tag{13}$$

$$O_n = O_{n-1} + \frac{d_r - d_l}{dbw} \tag{14}$$

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, la resolución del encoder $EncRes$ y es la distancia entre ruedas dbw.
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 estas fuentes de error, utilizamos el *test del camino bidireccional describiendo un cuadrado* (UMBmark) [1]. Dado que en el momento de realizar el test no teníamos 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 correción sobre la distancia entre las ruedas c_{dbw} hasta que el error sistemático máximo dejara de disminuir. En la figura 20 mostramos cómo disminuye el error a lo largo de las iteraciones.

[1] http://www-personal.umich.edu/ johannb/Papers/umbmark.pdf

Figura 20: 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 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. Esta decisión nos posibilitó 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 y 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, hicímos 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 descripto 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 quisiéramos usar otro simulador u otro tipo de hardware, sólo haría falta que implementemos 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

Como el controlador de comportamientos es **cliente** del módulo de reconocimiento, necesita conocer en el instante de tiempo t, que objetos están siendo reconocidos. Para esto el módulo debe tener una fuente de información, en nuestro caso, una cámara usada en Visión.
Desde el punto de vista anatómico, ésto se puede ver como los ojos (cámara), la parte del cerebro encargada de analizar el estímulo recibido (imágenes) por los mismos (módulo de reconocimiento) y la parte del cerebro encargada de analizar los estímulos recibidos y realizar las acciones (controlador de comportamientos).
Si hubiésemos 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.
Esta 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 sepa-

rado y el resto de los sensores no, es que el procesamiento de una secuencia de imágenes es muy complejo y demandante, tal como detallamos en la sección **??**.

1.7. Resultados obtenidos

En esta sección describimos los parámetros que utilizamos en la simulación para obtener los resultados sobre los diferentes comportamientos. Luego presentamos dichos resultados y sacamos conclusiones sobre los mismos.
A lo largo del proceso de desarrollo fuimos realizando diferentes simulaciones, de las cuales observamos defectos en las implementaciones de los comportamientos así como retardos y detalles que podíamos mejorar en algunos y que fuimos corregiendo.
La arena utilizada en dichas simulaciones la mostramos en la figura 21. A lo largo de la misma hay ubicados 15 objetos que simulan ser basuras, un área de recarga de batería y un área de descarga de basura, marcadas por el cilindro naranja. Los valores de los parámetros que utilizamos los detallamos en el cuadro 2. Los ángulos los expresamos en radianes y las distancias las expresamos en metros, así como también los radios, correcciones y separaciones. La resolución de la cámara la expresamos 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

1.7.1. Tiempo promedio en recolectar basura

base base posible

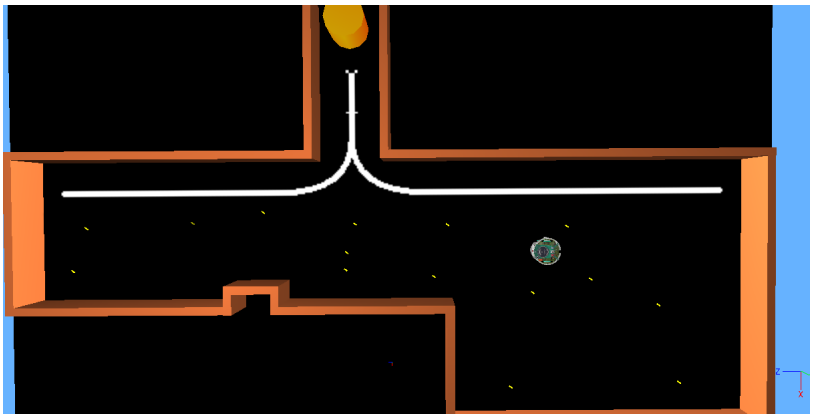

Figura 21: Arena que utilizamos para las simulaciones

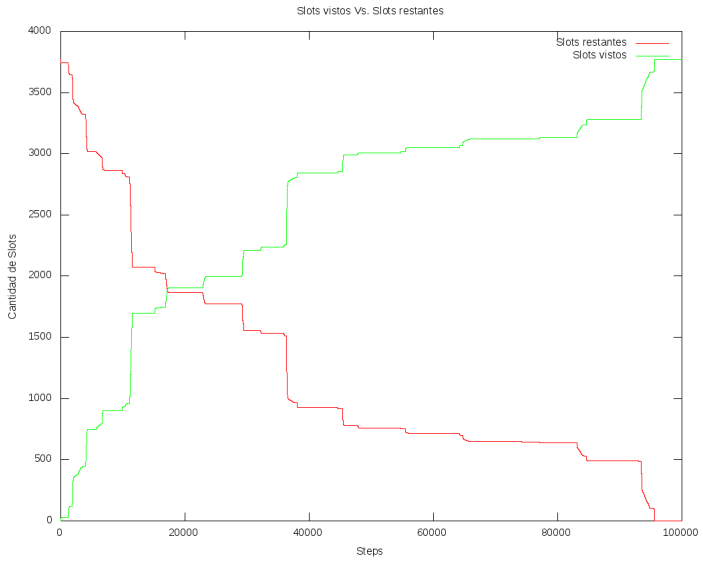

Figura 22: Área visualizada y área no visualizada a lo largo de la simulación

Figura 23: Distribución de los tiempos de los comportamientos en la simulación

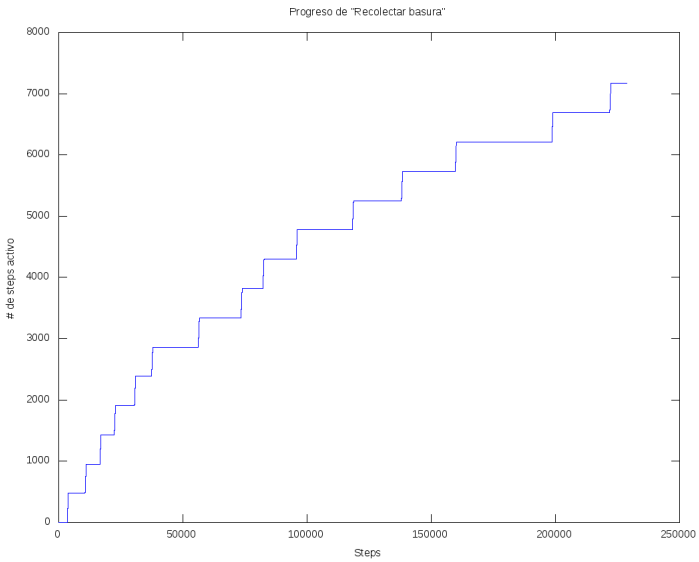

Figura 24: Progreso del comportamiento *Recolectar Basura*

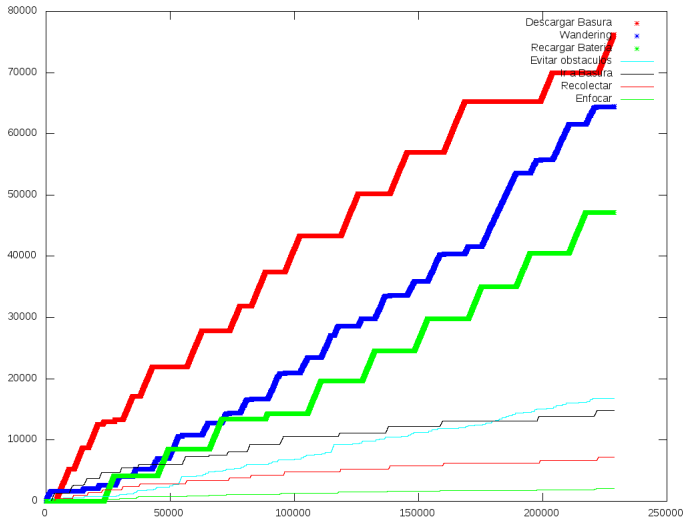

Figura 25: Evolución de los comportamientos a lo largo de la simulación

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 en los cuales los comportamientos están activos

Comportamiento	Promedio	Desvío Estándard	# máximo
Wandering	48.44	225.07	2212
Enfocar basura	1.48	3.27	67
Ir a basura	12.17	58.44	1222
Recolectar basura	477.73	0.70	478
Ir a descargar basura	237.25	840.49	6012
Ir a recargar batería	178.06	647.39	3950
Evitar obstáculos	11.17	54.72	1386
Salir situaciones indeseadas	0	0	0

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

1.7.2. Tiempo promedio de wandering

1.7.3. Distribución de tiempos entre los diferentes comportamientos

1.7.4. Relación (t_cargando + t_ir a base) / 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.