Matías Estrada

José Lombardi

Lazaro Pereira

Nicolás Furquez

Informe Técnico

Trabajo realizado para la participación en el marco de SUMO-UY

mt-robótica - 2013

Docente responsable: Gonzalo Tejera

1. Índice

2. Introducción

El presente documento contiene el diseño del software y hardware para la resolución del desafío IEEE Open presentado en el LARC2013 [1], el cual fue propuesto como tarea obligatoria del módulo de taller de robótica 2013. La primer parte del documento se registra el paradigma usado y los comportamientos encontrados. Después se pasará a realizar una descripción del diseño físico y el hardware utilizado.

3. Descripción del problema

El problema es el desafío de IEEE Open 2013 se trata de recolectar latas (basura) en una playa (escenario) sin colisionar con los obstáculos dispuestos en el mismo y luego depositar los objetos recogidos en un depósito.

3.1. Ambiente

El ambiente es parcialmente observable, ya que el agente no puede sensar el estado total del mismo, sólo se tiene acceso a lo que está frente a los sensores situados sobre el agente. Se puede decir también que el ambiente es estocástico, porque no se puede determinar el estado resultante luego que el agente actúa. Como la tarea que realiza el robot forma parte de un juego, que tiene principio y fin, y que se puede volver a empezar, el ambiente se puede calificar como episódico.

Otras características son la continuidad del espacio y que el ambiente es semi dinámico, porque las latas, la arena y la silla no cambian de lugar sin la intervención del agente, pero a medida que pasa el tiempo, se pierde la posibilidad de levantar latas si no se actúa.

4. Paradigma reactivo

Un sistema de control para un robot completamente autónomo debe ejecutar varias tareas complicadas de procesamiento de información en tiempo real. Estos robots operan en ambientes en los cuales las condiciones constante y rápidamente. A su vez el sensado se realiza por sensores ruidosos que dificultan la tarea de mapear directamente lo sensado a un modelo en el cual se pueda operar [10].

A partir de Brooks y su Subsumption se vio que se puede lograr una arquitectura que reaccione bien a esos cambios y cumpla con los objetivos planteados si se divide el problema horizontalmente según capas de competencia. En esas capas se encuentran los comportamientos, que son bloques de sensado-actuado que cumplen un propósito específico.

Este tipo de arquitectura es llamada reactiva, por la característica de que los comportamientos reaccionan a estímulos externos, emergiendo la conducta global por la composición de los comportamientos.

4.1. Comportamientos

Un comportamiento es una red de modulos de sensado y actuación que realizan una tarea que son implementados como máquinas de estado finito. Éstos se agrupan en capas de competencia que pueden sobrescribir o subsumar la salida de un comportamiento de una capa inferior [11]. Una arquitectura reactiva debe proveer mecanismos para resolver como se activan los comportamientos y determinar que sucede cuando se activan múltiples comportamientos en un instante dado [11].

4.2. Caso particular

Para la resolución de este problema se utilizó un paradigma reactivo basado en comportamientos[9], donde cada comportamiento tiene una determinada prioridad de ejecución sobre el resto. Esto determina que todos los comportamientos estarán sensando en forma paralela, pero solo el de mayor prioridad podrá tener control sobre los actuadores del robot y es interrumpido si y sólo si, un comportamiento más prioritario, necesita ejecutarse. Que el paradigma sea reactivo hará que el robot actúe en base a estímulos sensoriales, sin planificar sus movimientos y sin considerar para un determinado movimiento estímulos anteriores o información del estado del ambiente que no sea la que lo hizo moverse.

5. Experiencias anteriores

El trabajo realizado en este taller no es nuevo, sino que forma parte de un proceso en el cual el grupo vino trabajando durante un semestre en la materia Robótica Embebida. Aquí se vió el desafío LARC simplificado, de forma que sirvió como introducción y acercamiento al problema.

Del curso de Robemb se rescataron varios aspectos que fueron de fundamentales para la realización de este taller, y de los cuales se pueden destacar:

- Trabajo con la placa usb4butia [2], además de su fabricación y conocimiento de funcionamiento interno, funcionamiento de sensores, y puertos para actuadores.
- Prototipo de chasis diferencial simple y manejo de motores AX-12.
- Diseño, implementación y prueba de la arquitectura de comportamientos que se utilizó en el taller [14].

6. Descripción de la solución

6.1. Plataforma robótica

- BeagleBoard: utilizada como centro de cómputo del robot, hay que tener en cuenta sus limitaciones con lo que el desarrollo estará limitado por su capacidad de procesamiento.
- Motores AX-12 y AX-18[3]: actuadores que nos permite movilizar al robot en el escenario y realizar las acciones de recoger latas y luego tirarlas al recipiente.

- kit bioloid [3]: partes de los kits que creemos necesarias para el armado del robot.
- Base de acrílico, reutilizando piezas de viejos kits Butiá[2].
- Placa de Entrada salida USB4Butiá[2]

6.2. Sensores

Los sensores son una parte importante en la construcción de un robot situado, ya que de estos dependen las acciones que se lleven a cabo. Los sensores que se tienen en cuenta son los siguientes:

- Cámara
- Sensor infrarrojo

Cada sensor físico tiene una clase que lo representa y se encarga de procesar la información brindada por el sensor. Todas estas clases tienen una misma estructura, heredan de la clase Sensor, de esta forma podemos agregar sensores son provocar demasiados cambios a nivel de código.

Las clases que se encuentran implementadas son:

- SensorCamaraWhite: encargada de obtener una imagen de la cámara y procesarla.
- SensorDistancia: se encarga de tomar los valores que retorna el sensor infrarrojo.

Los sensores pueden ser ejecutados en forma de hilos, independientes entre sí o de forma secuencial. Por simplicidad decidimos realizar el sensado en forma secuencial, cada sensor recoge información del escenario en forma independiente y en simultáneo, los sensores no deciden qué comportamiento se debe ejecutar, solamente recogen información.

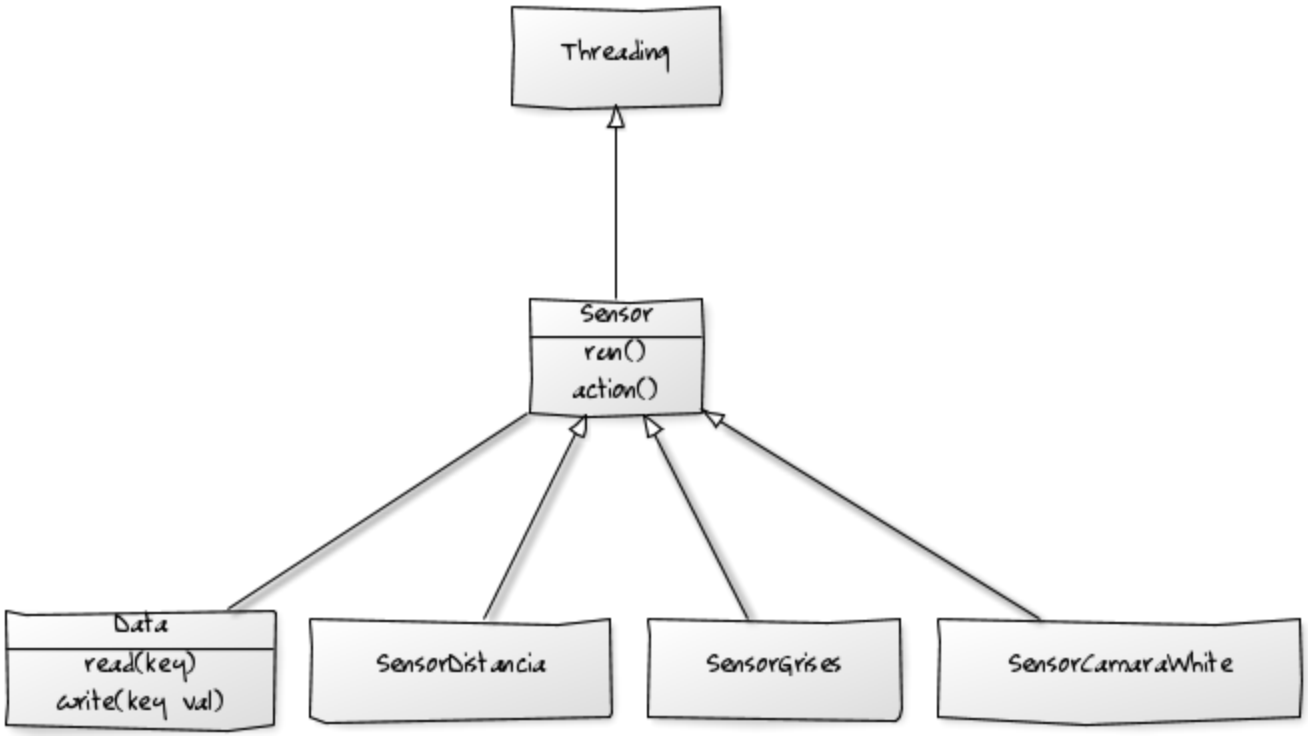

fig. 1 - Controladores de Sensores

La clase Sensor es la superclase de todas las implementaciones de los controladores de sensores particulares. Tiene asociada una estructura de datos que sirve para la transmisión de datos entre los comportamientos y los sensores particulares. Cada implementación particular puede escribir en la estructura tipo diccionario.

El propósito de esto es desacoplar los sensores de los comportamientos, y que los últimos puedan leer de la estructura de datos suponiendo que esta siempre está actualizada.

La función run() es llamada por el start del thread, por lo tanto solo es llamada si se utilizan los sensores de forma concurrente. Lo que hace esta función es, en un loop infinito llamar a action() y antes de mandar el thread a dormir. En Sensor, la función action() es abstracta, y es la única que deben implementar las subclases. En esta forma de uso cada controlador de sensor es un thread aparte.

6.2.1. SensorDistancia

Se ejecuta en un hilo aparte del hilo principal, simplemente escribe en Data su lectura cada vez que action() es llamado. Escribe bajo la clave ‘SensorDistancia::<puerto_u4b>’.

6.2.2. SensorCamaraWhite

Este es el controlador más complicado, ya que incluye el procesamiento de imágenes y escribe en la estructura de datos información procesada sobre la posición de la lata, del tacho

rojo y que tan cerca del borde se encuentra el robot.

Para el proceso de imágenes se utiliza la biblioteca libopencv [4] en su versión para python. Se eligió esta biblioteca por su gran potencia y por su popularidad, lo que hace posible encontrar grandes cantidades de artículos y ejemplos de su utilización.

El constructor de la clase inicializa todos los parámetros, como el tamaño de la captura, de forma reducir el tiempo de proceso y la memoria utilizada.

La función action() toma una imagen de la cámara, la transforma al espacio de colores HSV [5] para luego continuar con las diferentes etapas del procesamiento.

Lo primero que se hace es detectar el color blanco (arena) y quedarse con el contorno más grande y más cercano al robot (mayor eje y) y usando la función convexHull [7] definimos un nuevo contorno que nos servirá como máscara para el resto del proceso. En la figura 2 se puede ver el convexHull de la arena marcado de celeste.

La máscara es una imagen del mismo tamaño de la imagen original, solo que en blanco y negro, donde el blanco representa un uno en una máscara binaria. Para generar una máscara se imprime un polígono o contorno con todo el relleno en una imagen en blanco y negro.

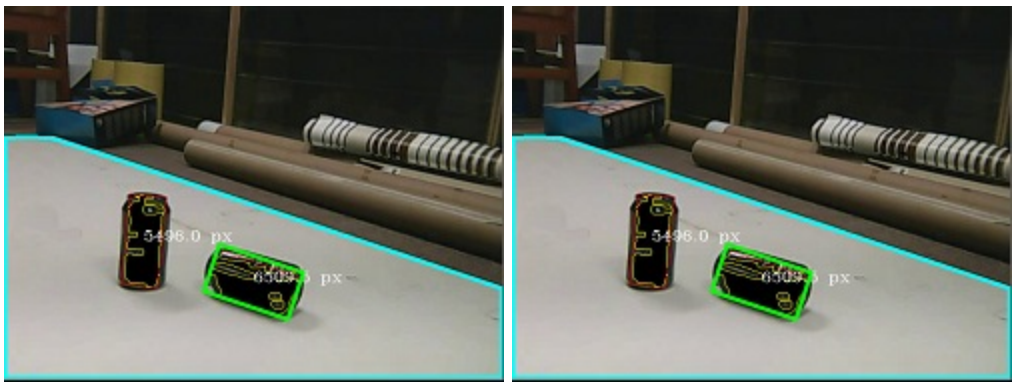

Fig 2. Imágenes después del procesamiento.

Luego se hace un threshold [8] para detectar negro, se filtra con la máscara de la arena, y se detectan los contornos (Contornos marcados en amarillo en la Fig 2). Con la máscara de la arena se asegura que lo negro que se detecta son latas y no elementos del exterior de la cancha, o sombras muy fuertes.

Luego de un filtrado por tamaño del área del rectángulo que engloba cada uno de los contornos detectados (marcado en rojo en la fig 2), se elige el rectángulo más cercano al robot (marcado con verde en la fig 2) , o sea el que tiene el mayor y.

Una vez elegido el rectángulo, se asume que es una lata y se escribe en la estructura de

datos con las claves ‘Camara::lata_x’ , ‘Camara::lata_y’ y ‘Camara::encontro’.

En caso de que no se pueda detectar un rectángulo se pone en ‘Camara::encontro’ el valor ‘FALSE’.

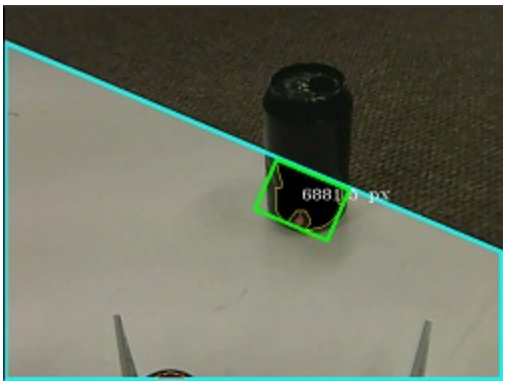

Fig. 3. Problemas de la solución.

El camino que se eligió para descartar ruidos exteriores a la arena, hace que el sensor tenga problemas para detectar ciertos casos bordes. Un ejemplo es el que se muestra en la figura 3, que se puede ver que la máscara de la arena hace que se pierda parte de la lata, haciéndose cada vez más notorio cuando el robot se acerca a la lata, porque en un momento solo se va a ver lata y alfombra, sin fondo blanco (en este caso simulando la arena), y la lata no va a ser detectada.

La detección del tacho rojo se hizo de la misma manera, solo que en vez de hacer el threshold por el negro se hizo por niveles de rojo. El controlador elige el rectángulo más grande y escribe bajo las claves ‘Camara::tacho‘, ‘Camara::tacho_x’ y ‘Camara::tacho_y’.

También se hace el procesamiento para calcular que tan cerca del agua se encuentra el robot contando la cantidad de pixeles de color arena en los bordes laterales de la imagen capturada (el ancho es ajustable) de esta manera si la cantidad baja de un margen, quiere decir que se esta acercando al borde del agua. El procesamiento y el cálculo de pixeles de las barras es guardado dentro de las variables ‘Camara::barra_der’ y ‘Camara::barra_izq’.

6.3. Arquitectura de comportamientos

Para resolver el problema se utilizó una arquitectura de comportamientos implementada por el grupo, inspirada en la implementación de subsumption de LeJOS [6]. Se caracteriza por estar basado en el paradigma reactivo con control basado en comportamientos. Esta fue diseñada y testeada durante el curso de Robótica Embebida por el grupo de José Lombardi y Matías Estrada ya que fue la misma que usaron para ese curso.

Al igual que la implementación de LeJOS los comportamientos son elegidos por su prioridad y esta está dada por su posición en un array en el cual se encuentran todos los

comportamientos.

Se cuenta con un Árbitro que es el que se encarga de preguntarle a cada comportamiento en su array de comportamientos si alguno está listo para ser ejecutado mediante la función takeControl():boolean. De la forma que los recorre, el primero que retorne true es el que pasa a ser ejecutado. El liberador depende de cada comportamiento y está codificado en la función takeControl, que en todos los casos corresponde a chequear la estructura de datos mencionada en la sección de sensores en una clave particular (menos en el comportamiento de wander que siempre retorna true).

El patrón fijo de acción se representa con estados dentro de cada comportamiento que en general dependen del tiempo pasado. Un ejemplo es el del comportamiento de evitar el agua, que una vez que el liberador dispara el comportamiento (la cámara detecta menos arena en los bordes de la imagen de lo aceptable) el takeControl sigue retornando true aunque ya no se esté detectando agua con la cámara durante un tiempo configurable.

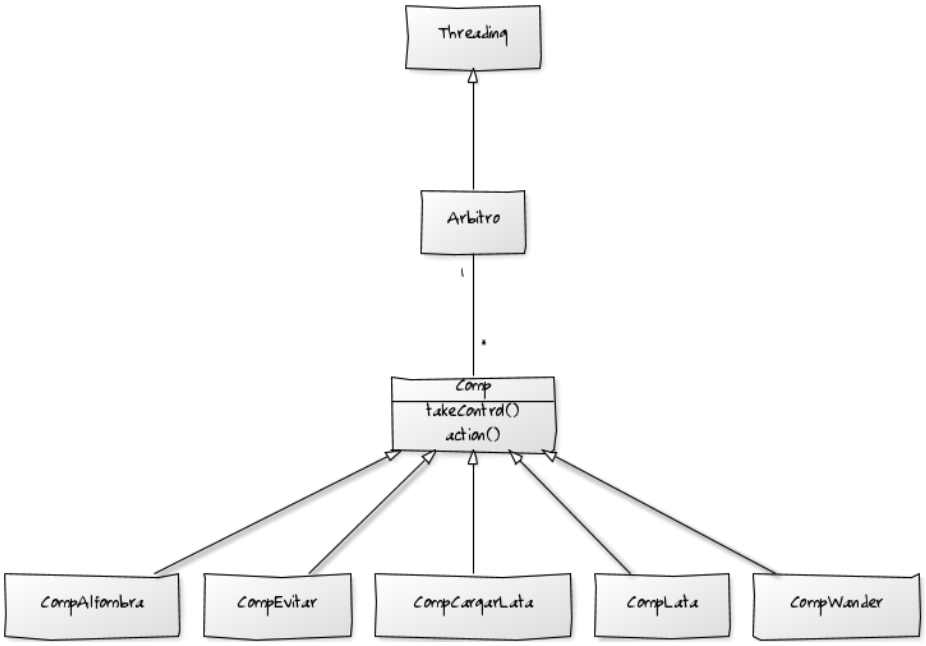

Fig. 4. Estructura de los comportamientos.

A continuación se listan y describen los comportamientos empleados:

6.3.1. CompWander

El comportamiento se basa en recorrer el escenario con el objetivo de encontrar objetos u obstáculos. Dentro del comportamiento mantenemos un estado, que nos dice que se está haciendo en cada momento, cambiando con el paso del tiempo a otros estados. Lo que se tiene en realidad es una máquina de estados no determinista para conseguir un movimiento aleatorio.

Los movimientos realizados son: moverse hacia adelante, girar a la izquierda y girar a la derecha.

6.3.2. CompLata

El comportamiento es activado cuando el valor de la clave 'Camara::encontro' dentro de la estructura de datos está en “True”. Cuando está activo, obtiene el valor de otra clave de la estructura de datos, 'Camara::lata_x', que me dice la posición según el eje x del objeto dentro del campo visual del robot. Con este valor centramos la trayectoria del robot para poder acercarnos al objetivo. Para centrar el objetivo, se realizan giros a la derecha y a la izquierda y cuando está centrado avanza hasta tener el objetivo en una posición correcta para poder recogerlo. Para saber si el objetivo está a una distancia aceptable, necesitamos conocer la posición en el eje Y, valor almacenado en 'Camara::lata_y' dentro de la estructura de datos. Con este último valor podemos regular la velocidad con la que nos arrimamos al objetivo.

6.3.3. CompCargarLata

El comportamiento es activado cuando el sensor camara escribe en la estructura de datos que se ha encontrado una lata. El mismo activa las pinzas y agarra lo que se tenga delante las mismas. El comportamiento sigue actuando mientras se mueven las pinzas.

6.3.4. CompEvitar

Este comportamiento tiene como objetivo evitar colisionar contra obstáculos, en este caso sillas, pero también tiene la habilidad de evitar colisionar con obstáculos de altura similar a una silla. El comportamiento se puede considerar como un reflejo ante un obstáculo. El comportamiento se activa cuando el valor almacenado en alguna de las claves ‘SensorDistancia::<puerto_u4b>’ supera algún valor límite predefinido. En ese caso el robot ejecuta una secuencia de acciones que evitarían colisionar contra el obstáculo.

6.3.5. CompAgua

El comportamiento tiene el objetivo de evitar caer hacia afuera del escenario, en realidad si nos situamos en el contexto del desafío, sería no caer en el agua. También se puede considerar como un reflejo ante la presencia de agua bajo el robot, para ello se basa en lo que el SensorCameraWithe le manda por medio de las variables, ‘Camara::barra_izq’ y ‘Camara::barra_der’ las cuales indican que tan cerca esta del borde del agua, reaccionando acorde si se esta mas cerca por izquierda, derecha. Las variables están inicializadas según el área de la intersección de las barras amarillas con la zona delimitada por la línea celeste mostradas en la Figura 5.

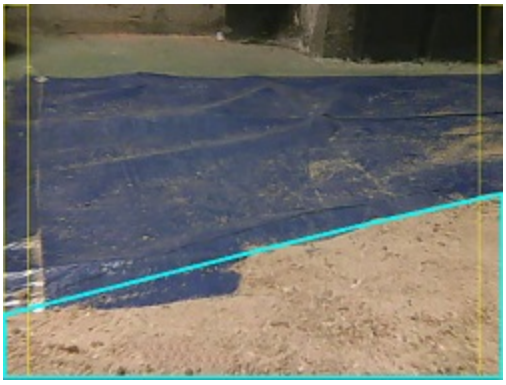

Fig 5. Detección de agua con las barras laterales en amarillo.

7. Estructura Física del Robot

7.1. Chásis

El chasis del robot esta desarrollado con acrilico y aluminio el cual le da estabilidad y a la vez ligereza para no exigir los motores AX-18 que son los encargados de mover el motor

7.2. Pinzas

Las pinzas están hechas de un brazo de aluminio y un tejido de alambre el cual permite recoger latas y filtrar la arena que podría juntarse al momento de cargar la misma, ambas pinzas están motorizadas por motores AX-12

Fig. 6. Pinzas y depósito de latas

7.3. Deposito de latas

El depósito de latas esta diseñado para que se a lo más liviano posible y tenga la mayor capacidad de latas. Para ellos se diseñó usando cartonplast, el cual es muy rígido y liviano. Para depositar las latas en el basurero, el contenedor tiene en su base un motor AX-12 el cual le permite pivotar y así descargar latas.

7.4. Orugas

El sistema de movimiento está compuesto por dos orugas hechas a partir de cadenas de bicicleta y ductos para cable de pvc cortados a la mitad, el diseño fue sacado desde [13]. Dos motores por oruga traccionan ruedas de plástico y goma que se apoyan en los caños permitiendo así el movimiento de la oruga. Para las orugas se usaron en un principio dos motores AX12, pero luego se modificó para usar 4 motores AX18, los cuales tienen una mayor potencia.

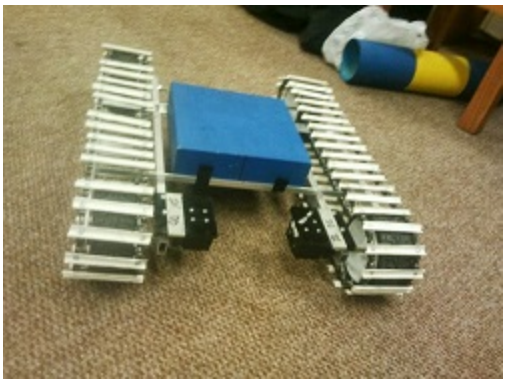

Fig. 7. Estructura de las orugas

8. Pruebas Realizadas

Se probaron los componentes de software y hardware de forma de obtener un mejor resultado. El control basado en comportamientos facilita la prueba de los módulos por separado, y se hicieron los meta-sensores de forma también se pudieran probar por separado.

Cada meta-sensor tiene un main que despliega la información de las variables que son escritas en la estructura de datos compartida. Por eso simplemente se ejecutaban y se miraba la salida. En sensores como la cámara se usaron algunas herramientas de software creadas por nosotros, que nos ayudaban a procesar el video de forma offline, logrando así no depender de la disponibilidad del laboratorio en el desarrollo de éste módulo.

```
$ python sensorCameraWhite -i video.avi
```

Ejemplo de ejecución del meta-sensor de la cámara

Las pruebas de hardware fueron más difíciles porque requerían de que la estructura estuviera relativamente estable, cosa que no siempre fue posible, debido a las demoras en la concreción de un sistema de traslado eficiente.

Para las pruebas de las orugas se armó un chasis simple (Fig. 7) comandado

directamente por el pyBotServer [12] y se probó en la arena. Ahí se detectaron serios problemas de diseño que llevaron a cubrir las orugas primero, agregarle más motores, cambiarle los motores y luego descartar el sistema de orugas.

Las pruebas de los comportamientos se hicieron activandolos de a uno de manera incremental, aprovechando el control basado en comportamientos.

9. Conclusión

La construcción de un robot requiere de una dedicación amplia, tanto para ver aspectos mecánicos, electrónicos y computacionales. Cada una de estas tareas es un área muy distinta en la que hay que investigar las posibilidades con el fin de mejorar el desempeño del robot para la tarea requerida.

Uno de los aspecto más complicados que se enfrentó el grupo es en la mecánica, sobretodo en lo respectivo a la forma de moverse, en un principio usar orugas pareció un buen camino, pero al probar las orugas en la arena se notó que el robot se enterraba al girar o entraban piedras entre las ruedas y las orugas, trancando el mecanismo. La solución que se encontró fue no girar en el lugar, sino que hacerlo en arco.

Una de las mejoras que se hicieron para ir a competir a LARC, fue cambiar las orugas por unas ruedas más grandes, se modificó el sistema que sostiene las ruedas y el motor, por uno hecho en acrílico cortado a láser, que permite que la fuerza se haga sobre estos y no sobre los motores, y mejora la calidad de las uniones anteriores hechas a mano directo en el aluminio.

Con respecto al Software, se agregaron los comportamientos de buscar el tacho cuando tiene latas y de arrojar las latas en él, así como también se modificó la forma en que el robot tiraba las latas, en vez de ser por detrás sera por delante.

Luego de tener hechos los cambios, notamos que uno de los problemas que íbamos a enfrentar era la falta de fuerza en los motores al moverse a poca velocidad, esto nos causó problemas al tratar de girar, ya sea en arco o en el lugar, con lo que concluimos que los motores no brindan el torque necesario para realizar todos los movimientos planificados.

Otro cambio que se hizo entre la finalización del SUMO|UY hasta el LARC, fué el mecanismo de volcado de las latas, en un principio estaba pensado para que el volcado se hiciera hacia atrás, pero finalmente decidimos realizarlo hacia delante. Este nuevo mecanismo nos evitaba giros en el lugar de 180º, y de esta forma evitar enterrarse.

Se agregaron algunas modificaciones menores en la chata para poder desarmar el robot y re armarlo fácilmente a la hora de viajar, también se muestra alguna otra modificación realizada con el fin de mejorar el desempeño del robot.

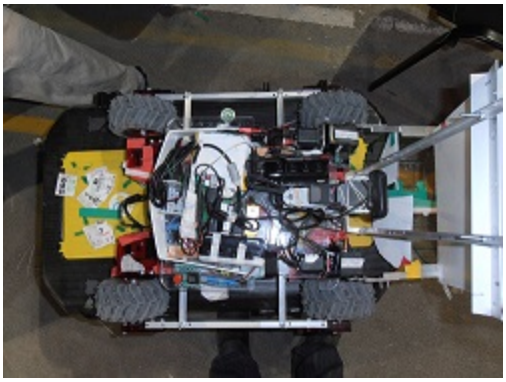

Fig. 8. Ubicación de los componentes (Beagle, USB4Butia, hubs, etc)

Fig.9 y 10 . Modificaciones estructurales al Robot para participar en el LARC

10. Referencias

Todo el código usado para la solución puede accederse desde su repositorio en:
https://github.com/tonyhawz/mt-robotica/

[1] Reglamento oficial para la categoría IEEE Open
http://ewh.ieee.org/reg/9/robotica/Reglas/LARC2012_open-rules_v1.1.pdf
visitada Setiembre 2013.

[2] Proyecto Butia
http://www.fing.edu.uy/inco/proyectos/butia/
visitada Setiembre 2013

[3] Robotis - Dynamixel
http://www.robotis.com/xe/dynamixel_en
visitada Setiembre 2013

[4] libopencv Open Source Computer Vision
http://opencv.org/
visitada Setiembre 2013

[5] Modelo de color HSV
http://es.wikipedia.org/wiki/Modelo_de_color_HSV
visitada Septiembre 2013

[6] Java for Lego Mindstorms
http://lejos.sourceforge.net
visitada Setiembre 2013

[7] Convex Hull
http://docs.opencv.org/doc/tutorials/imgproc/shapedescriptors/hull/hull.html
visitada Setiembre 2013

[8] Basic Thresholding Operations
http://docs.opencv.org/doc/tutorials/imgproc/threshold/threshold.html
visitada Setiembre 2013

[9] Robots Autónomos y Aprendizaje por refuerzo
http://www.fleifel.net/ia/robotsyaprendizaje.php
visitada Setiembre 2013

[10] Brooks, R.,"A robust layered control system for a mobile robot". Robotics and Automation, IEEE Journal,1986.

[11] Robotica Embebida “Paradigma Reactivo”. Diapositivas del curso 2013, FIng UdelaR, 2013.

[12] PyBot - Proyecto Butiá
http://www.fing.edu.uy/inco/proyectos/butia/mediawiki/index.php/PyBot
visitada Noviembre 2013

[13] Diseño de Orugas
http://www.instructables.com/id/How-to-make-custom-and-strong-tank-tracks-for-very/?ALLSTEPS
Visitado Diciembre 2013

[14] Estrada, M., Lombardi, J., Valenzani, J., “Laboratorio 2 - Comportamientos”. Informe de Laboratorio 2 de curso de Robótica Embebida curso 2013. FIng. UdelaR, 2013