



Universidad
Carlos III de Madrid

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

TRABAJO FIN DE GRADO

DESARROLLO DE UNA PLATAFORMA
ROBÓTICA MINI-HUMANOIDE CON
VISIÓN ARTIFICIAL TODO

Autor: Javier Isabel Hernández

Director: Felix Rodríguez Cañadillas

Tutor: Alberto Jardón Huete

Leganés, Septiembre 2014

Copyright ©2014. Javier Isabel Hernández
Esta obra está sujeta a la licencia
Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional de Creative
Commons. Para ver una copia de esta licencia, visite
<http://creativecommons.org/licenses/by-nc-sa/4.0/>.

Título: TODO

Autor: Javier Isabel Hernández

Director: Félix Rodríguez Cañadillas

Tutor: Alberto Jardón Huete

EL TRIBUNAL

Presidente:

Vocal:

Secretario:

Realizado el acto de defensa y lectura del Trabajo Fin de Grado el día de en, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

VOCAL

SECRETARIO

PRESIDENTE

Agradecimientos

En primer lugar me gustaría agradecer a mi tutor, Alberto Jardón, y a mi director Félix Rodríguez, el haberme dado la oportunidad de realizar un trabajo de final de grado que me apasiona.

Quiero dar las gracias a Miguel González-Fierro, por darme tan buenos consejos TODO

Quiero dar las gracias de todo corazón a mi familia, a mi padre Javier, a mi madre Miriam y a mi hermana Marta. A ellos que siempre me han apoyado en todas las decisiones que he dedicado tomar, me han animado a perseguir mis objetivos y me han consolado en los momentos mas difíciles.

Por último, quiero dar las gracias a mi novia Silvia, que ha estado a mi lado animándome siempre que lo he necesitado, y a quien he prometido que llevaré a la playa en cuanto termine este proyecto.

Resumen

En este proyecto se ha desarrollado la plataforma robótica mini-humanoide RAIDER, (Robot Antropomórfico para la Investigación y Desarrollo en Entornos Reales) con capacidad para actuar de forma autónoma basándose en algoritmos de visión por computador. Para ello, se ha escogida una base comercial sobre la que se ha realizado un rediseño mecánico completo con la ayuda de una impresora 3D. Sobre el robot, se ha montado un SBC (Single Board Computer), que ha permitido desarrollar algoritmos de visión por computador con las librerías de OpenCV y ZBar. También se han añadido diferentes sensores y actuadores adicionales para aumentar la versatilidad del robot.

Con el robot, se han programado las funciones de locomoción necesarias para dotar a la plataforma robótica de movilidad absoluta, incluyendo marcha bípeda y control de caídas.

Adicionalmente, se han realizado funciones basadas en visión como la búsqueda de trayectorias de navegación en entornos complejos, la detección de líneas y el análisis de códigos QR. Estas funciones han servido como base para el diseño aplicaciones de competición orientadas a presentar a RAIDER al concurso nacional de robots mini-humanoides CEABOT.

Palabras clave: robótica, visión artificial, mini-humanoide.

Abstract

(El resumen en inglés)

Keywords: keyword1, keyword2, keyword3.

Índice general

| | |
|---|------------|
| Agradecimientos | v |
| Resumen | vii |
| Abstract | ix |
| 1. Introducción | 1 |
| 1.1. Línea de investigación de robots Mini-Humanoides | 1 |
| 1.2. Objetivos | 2 |
| 1.2.1. Desarrollar una plataforma robótica | 2 |
| 1.2.2. Puesta en marcha y programación | 4 |
| 1.3. Estructura del documento | 5 |
| 2. Marco de trabajo | 7 |
| 2.1. Campeonato CEABOT | 8 |
| 2.1.1. Carrera de obstáculos | 9 |
| 2.1.2. Escalera | 9 |
| 2.1.3. Sumo | 10 |
| 2.1.4. Visión | 11 |
| 3. Estado del arte | 13 |
| 3.1. Plataformas robóticas mini-humanoides | 13 |
| 3.2. Competiciones | 17 |
| 3.3. Locomoción | 20 |
| 3.4. Procesamiento de imágenes | 21 |
| 4. Elección de componentes | 25 |
| 4.1. Plataforma robótica | 25 |
| 4.2. Modificaciones estructurales | 26 |
| 4.2.1. Cabeza móvil | 26 |
| 4.2.2. Cintura móvil | 27 |
| 4.3. Sensorización | 28 |
| 4.3.1. Sensores de distancia | 28 |

| | |
|--|-----------|
| 4.3.2. Sensores inerciales | 31 |
| 4.3.3. Cámara | 33 |
| 4.4. Elección del controlador | 33 |
| 4.4.1. Controlador de locomoción | 34 |
| 4.4.2. Controlador de visión | 36 |
| 4.4.3. Arquitectura hardware | 39 |
| 4.5. Alimentación | 41 |
| 4.5.1. Batería | 41 |
| 4.5.2. Regulador de tensión | 43 |
| 5. Descripción de las herramientas a utilizar | 47 |
| 5.1. Herramientas de diseño y fabricación de piezas | 47 |
| 5.1.1. OpenSCAD | 47 |
| 5.1.2. Impresión 3d | 48 |
| 5.2. Herramientas de diseño de circuitos | 49 |
| 5.2.1. KiCad | 49 |
| 5.3. Herramientas de programación | 50 |
| 5.3.1. OpenCV | 51 |
| 5.3.2. Qt Creator | 51 |
| 5.3.3. CMake | 52 |
| 5.3.4. CM9 IDE | 52 |
| 5.3.5. Git | 53 |
| 6. Diseño de las partes mecánicas | 55 |
| 6.1. Cabeza | 55 |
| 6.2. Cintura | 56 |
| 6.3. Tronco | 57 |
| 6.4. Brazos | 57 |
| 6.5. Piernas | 58 |
| 6.6. Pies y tobillos | 59 |
| 7. Desarrollo y montaje | 61 |
| 7.1. Adecuación de sensores | 61 |
| 7.1.1. Infrarrojos Sharp | 61 |
| 7.1.2. Brújula y sensor inercial | 64 |
| 7.2. Desarrollo de una placa de expansión | 65 |
| 7.3. Montaje | 65 |
| 7.3.1. Montaje de la placa de expansión | 65 |
| 7.3.2. Integración de la cámara en la cabeza | 67 |
| 7.3.3. Integración de los sensores infrarrojos en los brazos | 69 |

| | |
|---|------------|
| 8. Programación | 71 |
| 8.1. Configuración de la BeagleBone Black | 71 |
| 8.1.1. Sistema operativo | 71 |
| 8.1.2. Instalación de librerías | 72 |
| 8.1.3. Configuración de la cámara | 73 |
| 8.1.4. Configuración de pines | 75 |
| 8.2. Sistema de locomoción TODO | 77 |
| 8.2.1. Movimiento del servo PWM | 77 |
| 8.2.2. Movimiento de los actuadores Dynamixel . . . | 78 |
| 8.2.3. Movimiento sincronizado de las articulaciones | 80 |
| 8.2.4. Funciones de movimientos combinados TODO | 83 |
| 8.2.5. Creación de movimientos completos | 83 |
| 8.3. Comunicación serie | 84 |
| 8.3.1. Comunicación serie en OpenCM | 84 |
| 8.3.2. Comunicación serie en BeagleBone | 86 |
| 8.3.3. Comunicación con módulo Bluetooth | 86 |
| 8.4. Programación de sensores | 87 |
| 8.4.1. Infrarrojos | 87 |
| 8.4.2. IMU TODO | 88 |
| 8.4.3. Brújula | 89 |
| 8.5. Algoritmos de visión | 90 |
| 8.5.1. Análisis de trayectorias en navegación | 90 |
| 8.5.2. Detección de líneas | 95 |
| 8.5.3. Lectura de códigos QR | 97 |
| 9. Aplicaciones | 99 |
| 9.1. Spain Experience | 99 |
| 9.2. CEABOT 2014 | 100 |
| 9.2.1. Algoritmo para la prueba de visión | 100 |
| 9.2.2. Algoritmo para la prueba de navegación . . . | 103 |
| 9.2.3. Algoritmo para la prueba de sumo | 105 |
| 10. Presupuesto | 109 |
| 11. Evaluación de resultados | 111 |
| 11.1. Pruebas | 111 |
| 11.2. Conclusión | 112 |
| 11.3. Desarrollos futuros | 112 |
| Bibliografía | 115 |
| Anexo: Listado de piezas | 119 |

| | |
|--|------------|
| Anexo: Fotolitos de la placa de expansión | 123 |
| Anexo: Esquemático de la placa de expansión | 127 |
| Anexo: Planos | 129 |

Índice de figuras

| | | |
|-------|--|----|
| 1.1. | Robots Nao y DARwIn-OP | 2 |
| 2.1. | Robot Sylar de la Asociación de Robótica | 8 |
| 2.2. | Prueba de la carrera de obstáculos | 9 |
| 2.3. | Prueba de la escalera | 10 |
| 2.4. | Prueba de sumo | 10 |
| 2.5. | Marcador de la prueba de visión | 11 |
| 3.1. | Hitec Robonova ©Hitec RCD | 14 |
| 3.2. | Kondo KHR-3HV | 14 |
| 3.3. | Vstone Robovie-X | 15 |
| 3.4. | Bioloid Premium | 16 |
| 3.5. | Giger, de Andrew Alter | 18 |
| 3.6. | Combate en Robo-One | 19 |
| 3.7. | Partido de futbol en RoboCup | 20 |
| 3.8. | Modulo Pixy CMUcam5 ©CMUcam.org | 22 |
| 3.9. | Robot portero con módulo HaViMo | 23 |
| 4.1. | Microservo PWM | 27 |
| 4.2. | Sensor infrarrojo | 28 |
| 4.3. | Sensor de ultrasonidos | 29 |
| 4.4. | Primer experimento. | 30 |
| 4.5. | Segundo experimento. | 30 |
| 4.6. | Tercer experimento. | 30 |
| 4.7. | Sensor inercial MPU9150 | 32 |
| 4.8. | Brújula magnética CMPS03 | 32 |
| 4.9. | Microsoft LifeCam Cinema | 33 |
| 4.10. | Placa Arbotix | 35 |
| 4.11. | Placa CM900 | 36 |
| 4.12. | Placa OpenCM9.04 | 36 |
| 4.13. | SBC Raspberry B ©Raspberry PI Foundation. | 38 |
| 4.14. | SBC BeagleBone Black | 38 |
| 4.15. | Diagrama de conexiones | 40 |

| | |
|--|----|
| 4.16. Batería LiPo | 42 |
| 4.17. Convertidor DC-DC comercial, UBEC ©HeliDirect.com | 45 |
| 5.1. Pantalla del editor de OpenSCAD | 48 |
| 5.2. Impresora 3D Prusa i2 Air | 49 |
| 5.3. KiCad | 50 |
| 5.4. Qt Creator | 52 |
| 5.5. CM9 IDE | 53 |
| 6.1. Piezas necesarias para montar la cabeza | 56 |
| 6.2. Piezas necesarias para montar la cintura | 56 |
| 6.3. Piezas necesarias para montar el tronco | 57 |
| 6.4. Piezas necesarias para montar un brazo | 58 |
| 6.5. Piezas necesarias para montar una pierna | 58 |
| 6.6. Piezas necesarias para montar un pie | 59 |
| 7.1. Gráfica de tensión entre salida para un sensor infrarrojo | 62 |
| 7.2. Esquema de entradas analógica de una BeagleBone Black | 63 |
| 7.3. Circuito de adecuación de un sensor infrarrojo | 64 |
| 7.4. Esquema de puertos I2C de una BeagleBone Black | 64 |
| 7.5. Circuito del bus I2C | 65 |
| 7.6. Placa de expansión salida de fabricación | 66 |
| 7.7. Placa de expansión con componentes soldados | 66 |
| 7.8. Placa de expansión monstada en RAIDER | 67 |
| 7.9. Cámara Microsoft LifeCam desmontada | 67 |
| 7.10. Integración de la cámara en la cabeza | 68 |
| 7.11. Cuello y soporte para la cabeza | 68 |
| 7.12. Resultado tras la integración de la cabeza en el robot | 68 |
| 7.13. Sensores infrarrojos y soportes | 69 |
| 7.14. Brazos con sensores montados | 69 |
| 8.1. Conexión de la cámara a la BeagleBone Black | 73 |
| 8.2. Salida del comando lsusb | 74 |
| 8.3. Comprobación de conexión para la webcam | 74 |
| 8.4. Comprobación de compatibilidad para la webcam | 74 |
| 8.5. Parámetros leídos de la cámara | 75 |
| 8.6. Puertos serie en una BeagleBone Black | 76 |
| 8.7. Algunas capas de Device Tree Overlays | 76 |
| 8.8. Habilitación de capas | 77 |
| 8.9. Lectura analógica del pin 4 | 77 |
| 8.10. Amplitud de giro de un AX-12A | 80 |
| 8.11. Esquema de puertos serie en una OpenCM 9.04 | 85 |
| 8.12. Teleoperación de RAIDER con módulo Bluetooth | 87 |

| | |
|---|-----|
| 8.13. Rangos de detección para un sensor Sharp | 88 |
| 8.14. findway | 90 |
| 8.15. Descomposición de la imagen en sus tres canales RGB | 91 |
| 8.16. Resta de canales verde y rojo, y de canales verde y azul | 92 |
| 8.17. Detección de superficie navegable | 92 |
| 8.18. Umbralización de obstáculos y superficie navegable . | 93 |
| 8.19. Esqueletización del contorno abierto y esquema del entorno | 93 |
| 8.20. Esqueletización del contorno cerrado y esquema del entorno | 94 |
| 8.21. Segmento de referencia para la extracción de datos (en azul) | 94 |
| 8.22. Esqueletización del contorno cerrado y esquema del entorno | 95 |
| 8.23. Esqueletización del contorno cerrado y esquema del entorno | 95 |
| 8.24. Imagen original e imagen tras aplicar la función <i>detectGreen</i> | 96 |
| 8.25. Detección de contornos | 96 |
| 8.26. Dilatación y erosión de contornos | 97 |
| 8.27. Resultado final de <i>findLine</i> | 97 |
| 8.28. Análisis de códigos QR en una imagen | 98 |
| 9.1. Partido de fútbol robótico en México | 100 |
| 9.2. Diagrama de estados simplificado de la prueba de visión | 101 |
| 9.3. Código QR de inicio (Texto: GoRaider) | 102 |
| 9.4. Diagrama de estados simplificado de la prueba de navegación | 104 |
| 9.5. Diagrama de estados simplificado de la prueba de sumo | 106 |
| 9.6. Interruptor de emergencia | 107 |

Índice de cuadros

| | | |
|------|---|-----|
| 4.1. | Especificaciones Raspberry Pi B | 37 |
| 4.2. | Especificaciones BeagleBone Black | 39 |
| 4.3. | Consumo medio | 43 |
| 8.1. | Movimientos programados | 84 |
| 1. | piezas | 119 |
| 2. | piezas | 120 |
| 3. | piezas | 121 |
| 4. | piezas | 122 |

Capítulo 1

Introducción

La Asociación de Robótica de la Universidad Carlos III de Madrid, AsRob [2], surgió en el año 2006 con el objetivo de acercar la robótica a los alumnos de la universidad que compartían inquietudes e interés por el campo de la robótica. Desde sus inicios, las actividades de la asociación han contado con el apoyo del Departamento de Ingeniería de Sistemas y Automática, y del Robotics Labs [5].

A día de hoy, la asociación cuenta con mas de cien miembros activos repartidos en cinco líneas de investigación independientes, como son: Vehículos Aéreos no Tripulados (UAVs), Robot Devastation, Robots Personales de Competición, Impresoras 3D Open-Source y Robots Mini-Humanoides. Sin embargo, cabe destacar que aunque se trata de proyectos diferentes, existen grandes sinergias entre ellos. Particularmente, los miembros de la línea de Robots Mini-Humanoides, está muy ligados al estudio de las impresoras 3D, investigando diferentes técnicas de impresión, diseño de estructuras y materiales. Prueba de ello es el proyecto MYOD [3], el cual propone la construcción de robots mini-humanoides compuestos íntegramente con piezas impresas y replicables.

1.1. Línea de investigación de robots Mini-Humanoides

La sección de la asociación que enmarca este trabajo es la línea de investigación de Robots Mini-Humanoides. Los robots mini-humanoides son robots antropomórficos con una altura menor de 50cm, tal y como indica la normativa del campeonato CEABOT [6]. De forma orientativa, tomando como referencia la Humanoid League del campeonato RoboCup [37], el tamaño de los robots mini-humanoides es ligeramente inferior al de los participantes de la división "KidSize". En la figura 1.1 aparecen dos robots de la división

KidSize, como son el DARwIn-OP de Robotis [32] o el Nao de Aldebaran [30]. Estos robots no entrarían dentro de la definición de mini-humanoide, ya que sobrepasan las dimensiones máximas estipuladas.



Figura 1.1: Robots Nao y DARwIn-OP

1.2. Objetivos

A continuación se muestran los objetivos que se persiguirán durante el desarrollo de este proyecto. Se han dividido en dos grupos: El diseño y construcción de una plataforma robótica mini-humanoide, y su programación.

1.2.1. Desarrollar una plataforma robótica

El primer objetivo de este proyecto es el desarrollo de una plataforma robótica mini-humanoide de propósito general. Para llegar a ello será necesario estudiar los componentes que forman un robot humanoide. Sus capacidades deben ser al menos suficientes para desarrollar sobre la plataforma los objetivos que se listan a continuación. Además, se pretende que la plataforma sea lo suficientemente robusta y fácil de usar como para poder servir de base para proyectos futuros.

Estudiar los componentes que necesita un robot humanoide

Se realizará un estudio de qué elementos son necesarios para formar parte de un robot mini-humanoide. Entre ellos, se evaluará su funcionamiento en base a las necesidades del proyecto y se analizará qué componentes son adecuados para integrarse en el robot.

En resumen, se diseñará un sistema completo y funcional de controlador, alimentación, sensores y actuadores.

Integrar una cámara

Otro objetivo será integrar una cámara con la que se realizarán algoritmos de visión[26]. Para ello se requiere que la cámara quede integrada físicamente en el robot. Adicionalmente, la cámara deberá poder moverse de forma independiente al cuerpo del robot, por lo que se construirá una base móvil que otorgue a la cámara capacidad para rotar e inclinarse.

Integrar un controlador

Integrar en el robot un controlador que tenga el potencial necesario para controlar la locomoción del robot, comunicarse con los diversos sensores que se monten, y procesar algoritmos de visión. También se buscará que dicho controlador permita programarse en diferentes lenguajes sin tener que depender de un entorno de desarrollo fijo. Por último, el controlador que se escoja debe tener las capacidades de procesamiento suficientes para permitir un funcionamiento fluido y, al mismo tiempo, permitir una autonomía de funcionamiento del robot razonable.

En el caso de que el controlador requiera conexiones o conectores especiales, se deberán realizar las modificaciones necesarias para que el sistema sea robusto y fiable.

Agregar sensores y actuadores

Otro objetivo será incluir sensores y actuadores para aumentar las capacidades tanto de sensorización como de locomoción del robot. Aunque el componente principal para tomar datos del entorno será la cámara, puede ser interesante incluir algunos sensores que apoyen y complementen la información recogida por la parte de visión. De este modo, obtendremos un sistema versátil que podrá suplir las carencias del procesamiento de imágenes cuando su capacidad no sea suficiente, ya sea por un entorno complejo o con características inadecuadas para la visión (escenarios con poca luz, humo, etc).

Así mismo, también se incluirán actuadores adicionales que amplien las posibilidades del robot, ya sea para soportar el movimiento de nuevos componentes (como la cámara) o simplemente para aumentar las capacidades de locomoción.

Diseñar un sistema de alimentación

Dado que se prevee la inclusión de numerosos componentes nuevos, se deberá rediseñar el sistema de alimentación de manera que cumpla dos condiciones. Por una parte, debe ser capaz de alimentar con diferentes tensiones a todos los miembros del sistema, adaptándose a los requerimientos eléctricos de cada uno. Por otra parte, debe ofrecer una autonomía total al robot razonable y suficiente para realizar pruebas de competición.

También se procurará que las baterías sean fácilmente intercambiables, sin necesidad de montar y desmontar partes mecánicas. Este objetivo se propone a partir de la necesidad de hacer cambios veloces de baterías en las competiciones de mini-humanoides.

Mejorar la estructura mecánica

Se tendrá como objetivo adecuar la estructura del robot para el montaje de nuevos dispositivos. Además, secundariamente se mejorarán diversos puntos de la estructura para mejorar las capacidades generales de la plataforma.

Las piezas deberán diseñarse siguiendo dos reglas. La primera es que puedan imprimirse en una impresora 3D de bajo coste. La segunda regla es que soporten los esfuerzos necesarios y no se rompan durante la operación del robot.

1.2.2. Puesta en marcha y programación

El robot debe diseñarse, montarse, programarse y testarse. Una vez se termine la construcción, se desarrollarán librerías para controlar las funciones del robot y con las que programar aplicaciones.

Programar la locomoción bípeda

Se deberá diseñar un sistema de locomoción completo del robot. En este punto se unen diversos objetivos, desde controlar las articulaciones por separado hasta la realización de programas de caminata. Se deberá conseguir al menos la realización de los siguientes movimientos: Avance, rotación, desplazamiento lateral y reincorporación tras una caída. Junto a esto, se deberá crear una librería que permita realizar movimientos de una forma cómoda y sencilla.

Programar sensores

Los sensores que se seleccionen deberán adecuarse para poder operar con la placa de control que se utilice. No solo deberán adecuarse electrónicamente hablando, sino que también deberán interpretarse su datos de forma correcta para extraer información útil durante el funcionamiento del robot.

Desarrollar algoritmos de visión

El comportamiento del robot se basará en la información tomada por la cámara. Para procesar esta información se utilizarán librerías de visión avanzada, por lo que será necesario familiarizarse con ellas y estudiar las diversas posibilidades que ofrece el campo de la visión por computador. Con ello, se desarrollarán programas adaptados a la plataforma robótica del proyecto. Éstos, deberán optimizarse de forma que funcionen correctamente con las capacidades de procesamiento del controlador.

Desarrollar aplicaciones de competición

Se prevee la presentación del robot a competiciones. Esto significa que como parte del proyecto se desarrollarán aplicaciones de competición de carácter diverso. Para la realización de los programas, se tendrá en cuenta principalmente el reglamento concreto de cada prueba, con sus objetivos y limitaciones.

Estas aplicaciones se apoyarán en todo lo desarrollado anteriormente, y constituirán una prueba experimental representativa del funcionamiento del robot y del grado de consecución de todos los objetivos comentados.

1.3. Estructura del documento

A continuación y para facilitar la lectura del documento, se detalla el contenido de cada capítulo.

- En el capítulo 1 se realiza una introducción a la Asociación de Robótica y la línea de investigación de robots Mini-Humanoides. Posteriormente se explican los objetivos que se perseguirán a lo largo del documento.
- En el capítulo 2 se describe el marco de trabajo sobre el que se ha desarrollado el proyecto y se explica qué es la competición CEABOT.

- En el capítulo 3 se estudia el estado de arte de cuatro secciones relacionadas con la robótica mini-humanoide: Plataformas comerciales, competiciones, locomoción y visión articial.
- En el capítulo 4 se realiza un estudio y selección de los componentes que necesitará la plataforma robótica en desarrollo para cumplir los objetivos marcados.
- En el capítulo 5 se describen las herramientas necesarias para la construcción y montaje de todos los componentes, así como su puesta en marcha y programación.
- En el capítulo 6 se listan todas las modificaciones mecánicas que se han realizado sobre la plataforma robótica.
- En el capítulo 7 se explican los pasos que se han seguido para configurar, poner a punto y conectar los dispositivos electrónicos que se han seleccionado anteriormente.
- En el capítulo 8 se desarrolla la programación de robot a diferentes niveles, como la programación de algoritmos de movilidad, técnicas de medición con sensores integrados, y la programación de algoritmos avanzados basados en técnicas de visión por computador y las librerías de OpenCV.
- En el capítulo 9 se muestran aplicaciones que se han realizado sobre RAIDER para su asistencia a exhibiciones robóticas y a la competición CEABOT en su edición de 2014.
- En el capítulo 10 se ha realizado un desglose del presupuesto total del proyecto.
- En el capítulo 11 finaliza el proyecto analizando los resultados obtenidos y proponiendo nuevos desarrollos a implementar sobre RAIDER.

Capítulo 2

Marco de trabajo

Desde el año 2006, la Asociación de Robótica ha trabajado con robots humanoides destinados a la competición e investigación. Durante su actividad, se han utilizado diferentes plataformas robóticas y modificaciones que permitían a los robots del grupo ampliar sus capacidades y su competitividad. A lo largo del tiempo, pueden diferenciarse tres etapas caracterizadas por la plataforma robótica que fué empleada.

Entre 2006 y 2010, la plataforma empleada sobre la que se centraron los estudios y desarrollos fue el Robonova de Hitec. En la figura 2.1 aparece el robot Sylar, uno de los mini-humanoides de la Asociación de Robótica basados en esta plataforma. En esta primera etapa se realizaron mejoras en el robot para alojar sensores adicionales, como brújulas y sensores infrarrojos. El funcionamiento del sistema pudo testarse en competiciones como el CEABOT y el RobotChallenge [12] con excelentes resultados. En posteriores modificaciones, se sustituyó la placa de control del Robonova por una placa Arduino [1], que permitiría una mayor libertad a la hora de programar tanto la locomoción del robot como su sensorización.

En 2010, la asociación adquirió un kit de Bioloid [13]. Esta plataforma, más moderna que el Robonova, permitiría realizar un mejor control de los movimientos del robot. Dentro de la universidad, se han realizado multiples proyectos basados en este robot. En competición, los Bioloids de la Asociación de Robótica llegaron a conseguir el segundo puesto en la edición de 2012 del campeonato CEABOT. Sin embargo, las capacidades de esta plataforma, en lo que a sensorización y programación se refiere, no son demasiado altas y con sus componentes de serie existe una gran limitación.



Figura 2.1: Robot Sylar de la Asociación de Robótica

Por esta razón, desde el año 2013, se ha estudiado el modo de modificar estos robots con el objetivo de obtener una plataforma que permita explotar todo el potencial de sus actuadores y al mismo tiempo, facilitar la inclusión de nuevos componentes. Con este objetivo, se ligó fuertemente el estudio de la fabricación de piezas mediante técnicas de impresión 3D con su aplicación en robots mini-humanoides. Con piezas mecánicas nuevas y electrónicas libres, se prevé que las capacidades del robot aumenten exponencialmente. Esto convierte a los robots de la asociación en una base con mucho potencial para la investigación.

2.1. Campeonato CEABOT

El campeonato nacional CEABOT reúne cada año a robots mini-humanoides procedentes de universidades españolas y de equipos independientes. Durante tres días, los equipos tienen la posibilidad de presentar sus robots a diferentes pruebas de habilidad en las que pueden demostrar sus capacidades. En el reglamento de la edición del 2014, existen un total de cuatro pruebas combinadas de diversa temática que ponen a prueba la locomoción, percepción y actuación sobre el entorno de los robots participantes. Las pruebas son puntuadas por separado, sumándose de forma proporcional a su dificultad en la clasificación final. A continuación se muestran las pruebas previstas para la edición de 2014.

2.1.1. Carrera de obstáculos

En la carrera de obstáculos los robots deben realizar de forma autónoma un recorrido de ida y vuelta sobre una pista de características fijas. En la figura 2.2 se muestra el campo. Éste consiste en una superficie plana de color verde en cuya zona intermedia se colocan de forma arbitraria diferentes obstáculos inmóviles de color blanco. Estos obstáculos tienen forma paralelepípeda y unas dimensiones fijas de 20x20x50cm

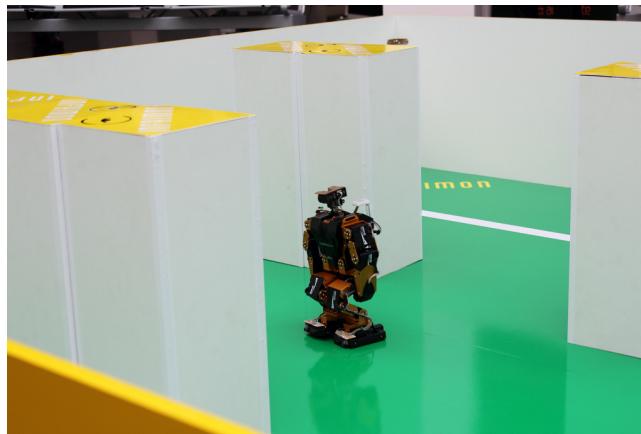


Figura 2.2: Prueba de la carrera de obstáculos

El robot participante debe cruzar el campo de extremo a extremo, y una vez haya accedido a la zona de llegada debe darse la vuelta y realizar el recorrido en el sentido contrario. En esta prueba se puntuía favorablemente el menor tiempo ocupado y la mayor longitud recorrida. Además, las caídas o bloqueos que requieran la intervención de un juez producen penalizaciones en la puntuación.

2.1.2. Escalera

La prueba de la escalera supone una combinación de las habilidades mecánicas y de sensorización de los robots. La prueba se desarrolla en un escenario formado por tres escalones de subida y tres escalones de bajada consecutivos. En la figura 2.3 se muestra un robot mini-humanoide realizando la prueba.



Figura 2.3: Prueba de la escalera

En este caso el robot debe superar escalones con una altura fija e igual a 3cm, pero con amplitud variable. El desarrollo consiste en la superación de tres escalones ascendentes, cruzar la cima de las escaleras y descender otros tres escalones hasta volver al suelo. De forma similar a la prueba de navegación, se puntúan el número de escalones superados y el tiempo utilizado; mientras que las caídas y bloqueos que el robot no sea capaz de manejar por sí mismo contarán negativamente.

2.1.3. Sumo

La prueba de sumo, que puede verse en la figura 2.4, es una de las más espectaculares del concurso. A diferencia del resto de pruebas, en el sumo los robots se enfrentan en parejas. Los duelos están constituidos por tres asaltos de dos minutos cada uno. El ring sobre el que se enfrentan los robots tiene forma circular, con un diámetro de 1.5m. Los robots compiten para derribar y/o sacar del ring a su adversario.



Figura 2.4: Prueba de sumo

2.1.4. Visión

La prueba de visión se presenta como una novedad en la edición de 2014 del concurso. Por primera vez se implanta en la competición una prueba que obliga a los robots a portar una cámara y realizar procesamiento de imágenes para su superación. El tablero de juego se comparte con el campo de la carrera de obtáculos. En esta prueba, el robot se colocará en el centro del tablero, y a su alrededor se colocarán obstáculos (los mismos que en la carrera de obstáculos) en intervalos de 45°. En la parte superior de los obstáculos se colocará un rectángulo rojo con un código QR en su interior, tal y como se indica en la figura 2.5. El robot deberá leer el código QR, en el que se le indicará una rotación que le permitirá encontrar el siguiente marcador. De esta forma, el robot deberá seguir una secuencia de rotaciones para superar la prueba.

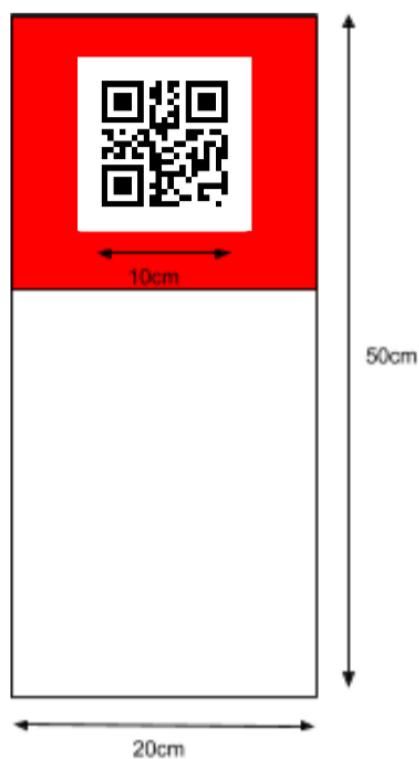


Figura 2.5: Marcador de la prueba de visión

Capítulo 3

Estado del arte

A continuación se realiza un repaso del estado del arte en robótica mini-humanoide. Nos centraremos en los campos mas relevantes del proyecto destacando cuatro bloques: Plataformas robóticas, competiciones, locomoción y sistemas de visión.

3.1. Plataformas robóticas mini-humanoides

En el mercado existe una gran variedad de robots educativos que cumplen las características antropomórficas necesarias para ser considerados robots mini-humanoides. Los robots que se muestran a continuación son una recopilación de algunos de los modelos mas accesibles y extendidos.

El Robonova, fotografiado en la figura 3.1, es uno de los mini-humanoides mas extendidos a nivel mundial. Fue uno de los primeros robots de este tipo que se fabricó comercialmente y marcó un antes y un después en su categoría. Es por esto que es muy común encontrar Robonovas en competiciones como CEABOT, ya que durante muchos años fue el robot mini-humanoide mejor equipado y mas vendido. En la Asociación de Robótica de la Universidad Carlos III, se han utilizado Robonovas en competiciones y proyectos desde su fundación.

El kit de fábrica cuenta con 16 grados de libertad. Sus actuadores son servos digitales HSR 8498HB, que desarrollan un torque de 7.4kg/cm. Cabe destacar de estos servos su función "Motion Feedback", es decir, su capacidad para leer posiciones y comunicarselas al controlador. La placa de control del Robonova está basada en un microcontrolador ATMega 128 y cuenta con hasta 40 pines GPIO (puertos binarios de entrada y salida), 8 entradas analógicas, 3 salidas PWM, puerto serie y conexión I2C. Gracias a esto, el Robonova



Figura 3.1: Hitec Robonova ©Hitec RCD

es fácilmente ampliable con sensores y actuadores, no necesariamente de la misma marca. En cuanto al software, Hitec da soporte a la programación con RoboBasic, un entorno de desarrollo completo con un lenguaje basado en Basic.

El KHR-3HV [4], mostrado en la figura 3.2, del fabricante japonés Kondo, es uno de los mini-humanoides mas avanzados actualmente. Puede presumir de ser el modelo de serie mas utilizado en el campeonato Robo-One, siendo seleccionado por los equipos por su gran agilidad y tamaño compacto.



Figura 3.2: Kondo KHR-3HV

En su configuración standard, el KHR-3HV cuenta con 17 servomotores KRS-2552HV de 14kg/cm de torque. Dichos actuadores, además, incluyen un pequeño microcontrolador, lo que les permite

conectarse en daisy chain. El robot incluye una controladora RCB-4, expandible con 10 entradas analógicas y 10 GPIOs, y con capacidad para controlar hasta 35 servos. El software de programación ofrecido por Kondo es el Heart to Heart V4, que puede ser descargado gratuitamente desde su web oficial. Es importante recalcar que gracias a su inmensa comunidad de usuarios, existen varios proyectos de código abierto con librerías que permiten programar el KHR-3HV en lenguajes más convencionales, como C y Python.

El Robovie-X [14], uno de los robots humanoides de la empresa Vstone, se presenta en tres versiones diferentes: Lite, Standard y PRO. La diferencia entre los tres modelos radica en el número y tipo de servos que montan, manteniendo comunes el resto de partes del robot. El modelo de la figura 3.3 es el modelo Standard. Posee 17 grados de libertad movidos por servos VS-S092J que desarrollan un torque de 9.2kg/cm. El controlador es un VS-C1, el cual tiene 30 canales para controlar servos. Vstone también fabrica diversas placas de expansión para la conexión de sensores en el Robovie-X.



Figura 3.3: Vstone Robovie-X

Junto al robot se suministra el programa RobovieMaker2, necesario para programarlo. El método de programación está orientado a la construcción de diagramas de flujo desde los que se controlan tanto los movimientos como la lectura de sensores externos.

La empresa coreana Robotis, comercializa un kit robótico conocido como Bioloid. Este kit proporciona una amplia gama de piezas diferentes para montar distintos modelos de robots. La modularidad

de los componentes le convierten en una base excelente sobre la que realizar modificaciones, pudiendo diseñar configuraciones alternativas con gran facilidad.

El robot Bioloid, mostrado en la figura 3.4, incluye 18 servos Dynamixel, modelo AX-12A o AX-18A dependiendo de la versión del kit. Los actuadores Dynamixel están controlados internamente por un microcontrolador ATMega8. Gracias a él, estos servos permiten realizar funciones avanzadas tales como el control de velocidad, torque, temperatura de ejecución, etc., posibilitando procesar información de bajo nivel directamente dentro del actuador y pudiendo abstraer el control de la controladora del robot a un nivel superior.



Figura 3.4: Bioloid Premium

En cuanto a su controladora, los kits proporcionan controladoras de Robotis de la serie CM, mas específicamente, la CM-5 en el caso del Bioloid Comprehensive y la CM-510 o CM-530 (según qué versión) en el Bioloid Premium y GP.

■ **Robotis CM-5.**

Cuenta con un microcontrolador ATMega128. Permite la conexión de sensores específicos de la marca en el bus TTL, como el Dynamixel AX-S1.

■ **Robotis CM-510.**

Basada en un microcontrolador ATMega2561. Además de los sensores de la propia marca (Dynamixel AX-S1, giróscopo...),

que pueden montarse conectados al bus TTL, tiene cinco puertos para la conexión de sensores de salida analógica. Adicionalmente, posee un puerto para conectar un receptor ZigBee y teleoperar el robot.

- **Robotis CM-530.**

Presentado como la evolución de la CM-510, en este caso el microcontrolador de la placa es un ARM Cortex STM32F103RE, de 32 bits. El resto de características son similares a las de la CM-510, tiene cinco puertos de expansión para sensores analógicos y un puerto para conectar un receptor ZigBee o Bluetooth.

En cuanto a la programación, Roboplus es una suite de programas distribuida gratuitamente por Robotis para la programación de sus robots educativos. Destaca por ser un entorno con un lenguaje de programación (R+) muy visual e intuitivo, preparado para su uso por niños o gente sin conocimientos muy avanzados de programación. Sin embargo, esto lo convierte en un lenguaje de programación muy limitado, hasta el punto que no permite utilizar todo el potencial de los actuadores Dynamixel.

3.2. Competiciones

Los robots mini-humanoides a menudo son utilizados en competiciones. A lo largo del mundo existen múltiples competiciones dedicadas a este tipo de robots. Estas competiciones suponen un escaparate de las últimas tecnologías aplicadas en robótica mini-humanoide, y suponen una mezcla de los trabajos mas avanzados realizados por ingenieros y aficionados.

Dadas sus multiples disciplinas, los RoboGames [21] son considerados la competición robótica mas grande del mundo. RoboGames es un encuentro anual para ingenieros y aficionados a la robótica en sus diferentes vertientes. En lo que a mini-humanoides se refiere, existen diversas modalidades como Kung-Fu, escaleras, carreras o sumo. Una de las particularidades de esta competición es que compiten conjuntamente robots autónomos y robots teleoperados. Quizás sea por esta razón que en este campeonato prima la realización de un control de locomoción muy avanzado respecto a una sensorización más sencilla. La programación de estos robots se centra en la realización de movimientos muy calculados y precisos. Así mismo, los robots participantes de estas pruebas suelen contar con mas articulaciones de las que suele tener un robot mini-humanoide comercial,

superando en ocasiones los 30 grados de libertad. RoboGames es uno de los mayores escaparates en los que observar los mejores algoritmos de locomoción para robots mini-humanoides. El robot de la figura 3.5, de Andrew Alter, ha sido uno de los participantes más asiduos en los últimos años.

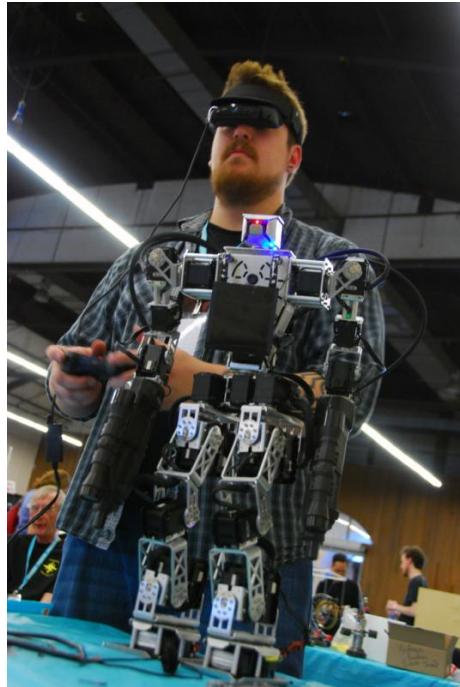


Figura 3.5: Giger, de Andrew Alter

En Japón, el Robo-One [10] es el mayor campeonato de robots mini-humanoides. Esta competición tiene unas normas más restrictivas y se centra en una única categoría, el combate cuerpo a cuerpo. Los robots que se presentan a esta competición son mayoritariamente radiocontrolados y diseñados especialmente para la competición. Estos robots suelen estar construidos con materiales de alta resistencia y bajo peso, como la fibra de carbono y el aluminio mecanizado. Además, los servos que montan estos robots suelen ser muy potentes, en ocasiones llegando a desarrollar un torque superior los 30kg/cm . Sin embargo, una diferencia importante con los robots de los RoboGames es el número de grados de libertad del robot, siendo muy inferior en este caso. Una configuración típica en la competición no supera los 18 servos. Esto se debe a su fuerte especialización para pelear con otros robots. En este caso, prima la robustez mecánica y la fuerza frente a la versatilidad. Si bien es cierto, comúnmente estos robots montan sensores iniciales que les ayudan a mantener el

equilibrio. En la figura 3.6 se muestra un combate típico por dos de los robots mas laureados en la competición: Garoo y Hammerhead.

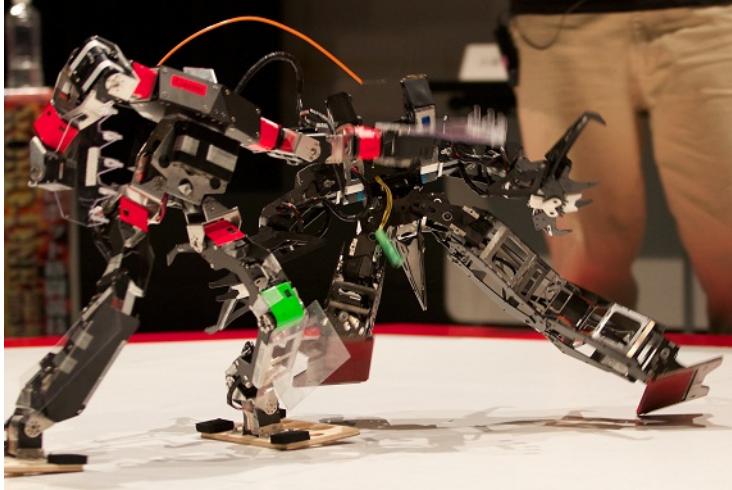


Figura 3.6: Combate en Robo-One

Sin embargo, las competiciones de robots mini-humanoides no solo se centran en pruebas individuales. En el campeonato RoboCup, los robots participan en equipos en la realización de partidos de fútbol. En este caso, los robots son exclusivamente autónomos y basan su comportamiento en algoritmos de enjambre [44]. Por tanto, en esta competición se le da una mayor importancia a la programación de los robots frente a su estructura mecánica. De hecho, comúnmente se parte de plataformas comerciales para desarrollar los robots. En cuanto a sensorización, los robots participantes montan cámaras que les permiten extraer datos del entorno mediante técnicas de visión por computador. De este modo, los robots interactúan entre ellos y los elementos del campo de forma coordinada. En la imagen de la figura 3.7 se muestran robots de diferentes equipos disputando un partido.

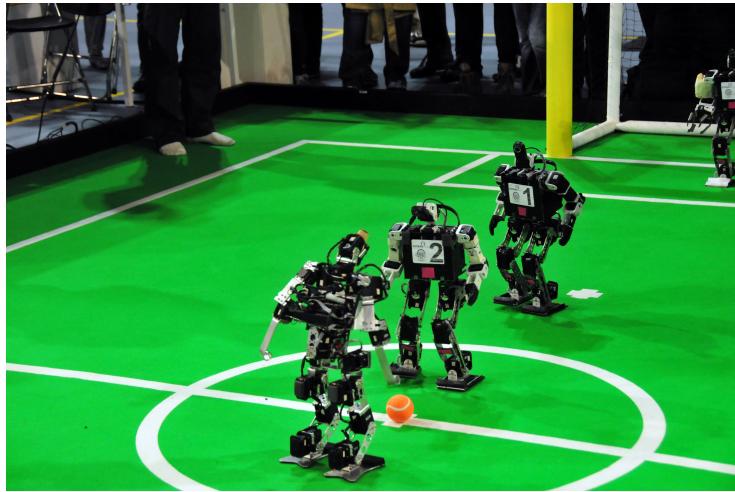


Figura 3.7: Partido de futbol en RoboCup

Por otra parte, la competición CEABOT [35] mezcla diferentes conceptos. Al ser una competición multidisciplinar y completamente autónoma, los robots necesitan una mayor versatilidad [29]. Por esta razón, un robot participante en CEABOT, y que debe realizar cuatro pruebas dentro del campeonato, no puede especializarse en una de ellas si eso supone una interferencia en su rendimiento en el resto.

3.3. Locomoción

A la hora de programar la locomoción de un robot se nos muestran diferentes opciones, por un lado existen métodos analíticos para construir movimientos en base a las propiedades geométricas del robot y el feedback que puede extraerse de sus sensores. Ejemplo de esto es la generación de algoritmos de marcha mediante el método del Zero Moment Point [36], mas conocido como ZMP. Estos métodos pasan por una extracción de la cinemática inversa [16] del robot para mover sus articulaciones de una forma controlada. Robots avanzados como el Asimo de Honda [34] usan este tipo de algoritmos.

Sin embargo, cuando se trabaja con robots de reducido tamaño, como los mini-humanoides, en muchos casos no es necesario llegar a ese nivel, sino que pueden generarse movimientos de formas más sencillas. Por ejemplo, un método muy utilizado es la programación por guiado [17]. Este método consiste en el posicionamiento manual del robot en una posición para su posterior grabación. Mediante la grabación de una serie de posiciones se compondrá una secuencia, que supondrá un movimiento completo. Los robots comerciales como el

Bioloid o el Robonova, suelen incluir interfaces de programación basadas en este método. También existen opciones open-source, como el PyPose [38]. PyPose es una aplicación escrita en Python, creada para su utilización en placas Arbotix. Esta aplicación es compatible con un gran número de robots, controladoras y servomotores del mercado. PyPose permite crear configuraciones personalizadas para adaptarse al número de grados de libertad de cualquier robot. Se permite la actuación sobre articulaciones individuales o sobre grupos de articulaciones. Esta herramienta tiene un gran potencial principalmente en robots con servomotores que ofrecen feedback sobre su posición.

3.4. Procesamiento de imágenes

Los sistemas de visión históricamente han necesitado de una gran capacidad de procesamiento. Este hecho los ha relegado a robots más grandes que puedan soportar físicamente el montaje de un ordenador. Sin embargo, a día de hoy existen opciones para dotar a los robots mini-humanoides de las capacidades que ofrece la visión por computador.

Una de estas opciones es el montaje de un módulo de placa y controlador integrado como la CMUcam [8]. En la figura 3.8 puede observarse el módulo Pixy CMUcam5, que corresponde a la quinta iteración del diseño de la CMUcam. Este módulo se diseñó con la idea de proveer con capacidades de visión a pequeños sistema embedidos. Sólo con un microcontrolador y uno de estos módulos, pueden programarse algoritmos sencillos como el tracking de objetos.



Figura 3.8: Modulo Pixy CMUcam5 ©CMUcam.org

Otra sistema similar es el módulo HaViMo. Este sistema incluye de forma integrada una cámara y un microcontrolador. Entre sus capacidades destacan su reducido tamaño y peso, su bajo consumo y una compatibilidad directa con cualquier dispositivo con comunicación serie. Este sistema ofrece la posibilidad de diferenciar regiones de una imagen basándose en su color. Puede analizar el tamaño, posición y centro de gravedad de cada región, siendo muy interesante a la hora de diferenciar objetos. Sin embargo la funcionalidad de estos sistemas es limitada, ya que no ofrecen la posibilidad de programarse con librerías de visión como OpenCV, que permiten el desarrollo de algoritmos avanzados. En la figura 3.9 puede observarse un robot portero que utiliza este módulo para localizar el balón rojo.

También existe otra posibilidad, separar la parte de visión del robot y procesarla en un sistema externo, como un PC. Para la realización de algunos proyectos puede ser interesante el montaje de una cámara en el robot que envía las imágenes crudas a un sistema más potente y después recibe los datos procesados. De este modo se consigue abstraer al controlador del robot del sistema de visión, mejorando su velocidad y enfocando su rendimiento en otras tareas. Existen dos posibilidades. La primera es conectar la cámara del robot a su placa controladora y que sea ésta la que se ocupe de enviar las imágenes a un servidor externo. El servidor devolverá la información procesada extraída de la cámara. Éste es el sistema que se ha utilizado en los desarrollos con el robot HOAP-3 del Robotics Lab [20]. La otra opción es montar en el robot una cámara inalámbrica, comúnmente una cámara IP, y que esta envíe las imágenes directamente al ordenador.

mente a un dispositivo externo, sin pasar por el robot. El robot recibirá la información que necesita desde ese agente externo.

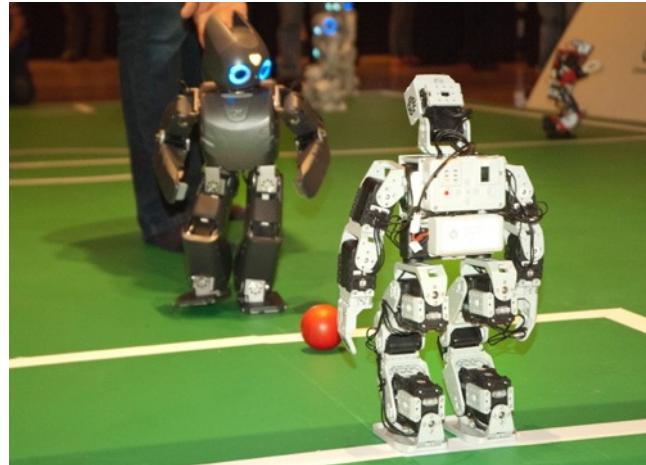


Figura 3.9: Robot portero con módulo HaViMo

El reto actual a la hora de implantar un sistema de visión en un robot mini-humanoide es conseguir integrarlo físicamente en un espacio reducido y al mismo tiempo contar con la potencia necesaria para no depender de un servidor externo. Dicho de otra forma, se persigue montar un robot lo mas independiente posible.

Capítulo 4

Elección de componentes

Como punto de partida, se han seleccionado una serie de componentes adecuados para dotar al robot con las capacidades necesarias para cumplir los objetivos. Dado que el sistema se va a rediseñar completamente, se elegirán los componentes que mejor se adapten a nuestras necesidades sin tener que ceñirse a limitaciones de compatibilidad.

4.1. Plataforma robótica

El primer paso para la realización de este proyecto fue el estudio y selección de las plataformas robóticas que se encuentran en el mercado actualmente. Dado que el objetivo es encontrar un robot humanoide sobre el que se pueda implantar un sistema de visión, es necesario analizar diversos aspectos; algunos mecánicos como el numero y fuerza de los actuadores, y otros electrónicos como la capacidad de procesamiento y velocidad del controlador. Sin embargo, dado que este proyecto es autofinanciado en su mayor medida, el factor económico también es un limitante destacable. Buscamos una plataforma que cumpla los siguientes requisitos:

- Programable en C/C++

Se requiere que sea programable en C/C++ y que además no dependa de una IDE concreta.

- Posibilidad de conectarle una cámara

Se necesita que permita conectarle una cámara y realizar programas con OpenCV.

- Expandible con sensores y actuadores

El sistema debe permitir la adición de nuevos sensores y actuadores, sin verse limitado por hardware o software.

- Servos de al menos 12kg/cm

Ya que se van a incluir nuevas partes, es absolutamente necesario que los servos puedan soportar carga adicional colocada en el robot.

- Chasis reconfigurable

Aunque no es tan importante como el resto, es posible que el robot requiera modificaciones, por lo que una base reconfigurable y versátil será apreciada.

En el siguiente apartado se presenta un estudio las principales opciones.

Comparativa y elección final

Realizando un primer análisis, ninguno de los robots candidatos cumple los requisitos. Todos ellos obligan a utilizar entornos de desarrollo y lenguajes propios para su programación. El Robonova y el Robovie tienen unos servos demasiado débiles, lo que dificultaría mucho añadir mas peso al robot. Entre el Kondo y el Bioloid, se ha elegido el Bioloid por tres razones: Es mas fácil de modificar, el kit contiene mas servos y es más barato. También, previendo las modificaciones futuras, se contempló la idea de comprar únicamente los servos Dynamixel que usa el Bioloid por separado. Sin embargo, resultó mas económico adquirir el kit completo de Bioloid Comprehensive.

4.2. Modificaciones estructurales

El Bioloid Comprehensive es un buen punto de partida, sin embargo tiene algunos puntos débiles que conviene revisar. Además, para poder implantar en el robot los dispositivos que requiere este proyecto se necesitarán mejorar las capacidades de la plataforma.

4.2.1. Cabeza móvil

Un requisito importante del proyecto es permitir que la cámara que vamos a montar pueda moverse con libertad para enfocar a diferentes zonas de su entorno. Dado que en CEABOT la mayoría de los

datos que aporta el entorno están situados en el suelo, necesitamos que la cámara al menos pueda dirigirse hacia el frente y hacia el suelo. Esto lo conseguiremos con la adición de un microservo PWM y una plataforma articulada para la cabeza. Dadas las características de este movimiento, no necesitamos un servo con grandes capacidades, ya que su rango de acción estará muy limitado y su efecto será despreciable en el reparto de pesos del robot.



Figura 4.1: Microservo PWM

Se ha elegido un microservo PWM por su bajo tamaño y peso, su bajo precio y su sencillez a la hora de montarlo y programarlo. El principio de funcionamiento de un servo PWM es muy simple. De las tres patillas de su conector, dos son de alimentación y la tercera se encarga de recibir una señal PWM que, variando la amplitud de su pulso, ordena al servo a moverse a una posición fijada.

Particularmente, se ha escogido un servo Tower Pro MG90s como el de la figura 4.1, cuyas especificaciones presentan un torque de $2.4=\text{kg}/\text{cm}$, y una transmisión metálica soportada por rodamientos. Este último dato es muy importante si tenemos en cuenta que el sistema de cabeza móvil se situará en una zona extrema del robot, y que un golpe provocado por una caída forzará de forma directa el eje del servo de la articulación. Este servo proporcionará a la cabeza la robustez necesaria para salir indemne en este tipo de accidentes. Accidentes que por otra parte son muy comunes teniendo en cuenta que el robot estará destinado a la realización de pruebas de competición.

4.2.2. Cintura móvil

Otra de las modificaciones básicas a realizar sobre la plataforma robótica, ha sido la inclusión de un servo adicional Dynamixel AX-12A para articular la cintura. A parte de la mejora de capacidades

que se produce en los movimientos del robot, servirá para girar la dirección de la cámara radialmente. Podría decirse que entre el servo de la cabeza y el de la cintura, se ha creado un sistema distribuido de pan-tilt que permitirá mover la cámara en todas las direcciones manteniendo fija la base del robot, es decir, sus piernas.

4.3. Sensorización

El funcionamiento del robot y su control va a estar basado principalmente en la cámara y los algoritmos de visión que se programarán. No obstante, la cámara consume muchos recursos del procesador y existen algunas tareas sencillas que es más fácil programar con sensores más simples. A continuación se muestra un estudio de sensores que pueden ser útiles en el proyecto.

4.3.1. Sensores de distancia

Existen diferentes tipos de sensores de distancia. El propósito de estos sensores no es otro que la medición de longitudes utilizando diferentes principios físicos. Se ha planteado el uso de sensores infrarrojos y/o de sensores de ultrasonidos.

Sensores de luz infrarroja TODO

Los sensores infrarrojos, como el que aparece en la figura 4.2 basan su funcionamiento en la reflexión de luz infrarroja sobre superficies. El sensor está formado por dos LEDs infrarrojos, uno emisor y otro receptor. El emisor emite de forma continua una luz infrarroja dirigida hacia un punto fijo. Si en ese punto fijo se encuentra un objeto físico, la luz se reflejará y será captada por el diodo receptor. La salida de estos sensores se mide mediante la diferencia de potencial que se produce en el diodo receptor. Se ha estudiado la inclusión en el proyecto de sensores infrarrojos Sharp, especialmente de su modelo GP2Y0A21YK, que tiene un rango de operación de entre 4 y 150cm.

Figura 4.2: Sensor infrarrojo

Sensores de ultrasonidos

Los sensores de ultrasonidos detectan distancias basándose en el tiempo en el que un sonido de alta frecuencia recorre el espacio. Normalmente, los sensores de ultrasonidos tienen dos focos, uno emisor

y otro receptor. Una de las diferencias de este tipo de sensores con los sensores infrarrojos es que mientras los sensores infrarrojos toman medidas de un punto, el rango de operación de los sensores de ultrasonidos se abre en un cono de 60° de amplitud desde su emisor. Se ha analizado el sensor de ultrasonidos HC-SR04 como posible candidato para formar parte del robot por su fácil accesibilidad. Otro punto importante es que en estos sensores la salida es proporcional a la distancia medida, es decir, se adecúa a una recta. Es por esto por lo que podremos realizar medidas reales directamente midiendo la salida del sensor.

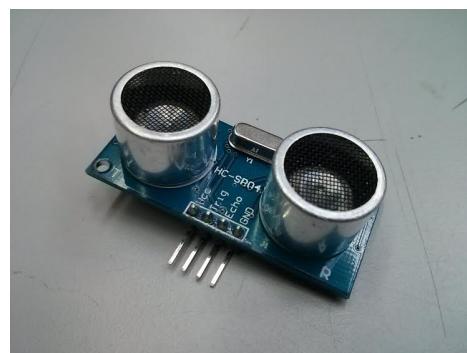


Figura 4.3: Sensor de ultrasonidos

Experimentos de medida

Para poner en práctica y observar las diferencias teóricas que existen entre el funcionamiento de ambos sensores, se ha realizado un pequeño experimento en el que se han realizado medidas de longitud entre el sensor y un obstáculo. Para ello, se han analizado tres configuraciones posibles. En cada imagen, se ha marcado en azul el rango de acción del sensor. La linea roja representa la distancia que medirá el sensor en cada situación.

En el experimento de la figura 4.4, se ha colocado un obstáculo como los que se utilizan en la prueba de navegación de CEABOT en posición frontal y recta. Puede observarse que en esta situación ambos sensores medirían de forma similar la distancia entre el sensor y el obstáculo.

En el experimento de la figura 4.5 se ha desplazado el objeto ligeramente hacia la derecha. Puede observarse cómo el rango de acción del sensor infrarrojo, al tener una forma lineal, no detecta el obstáculo. Mientras tanto, el sensor de ultrasonidos es capaz de detectarlo, ya que en esa posición sigue estando dentro de su rango



Figura 4.4: Primer experimento.

de acción.

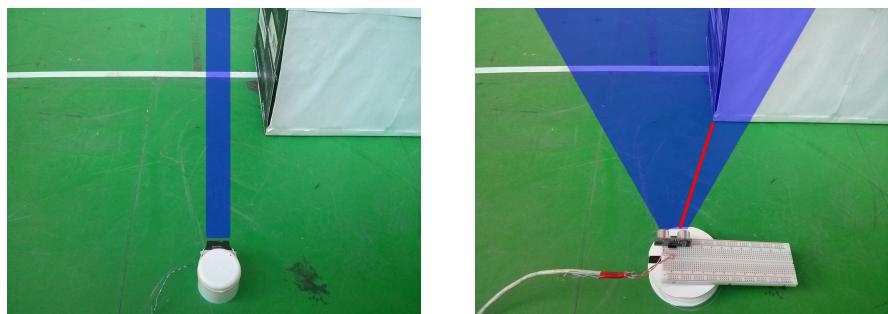


Figura 4.5: Segundo experimento.

Por último, el experimento de la figura 4.6 muestra el comportamiento de ambos sensores al enfrentarse a una superficie oblicua. Tal y como puede observarse en la imagen, el sensor infrarrojo realiza una medida puntual a la zona del obstáculo que se interpone en su rango de acción. Por otra parte, el sensor de ultrasonidos nos devuelve la distancia del punto del objeto que, encontrándose dentro de su rango, constituye la distancia mínima entre ambos.

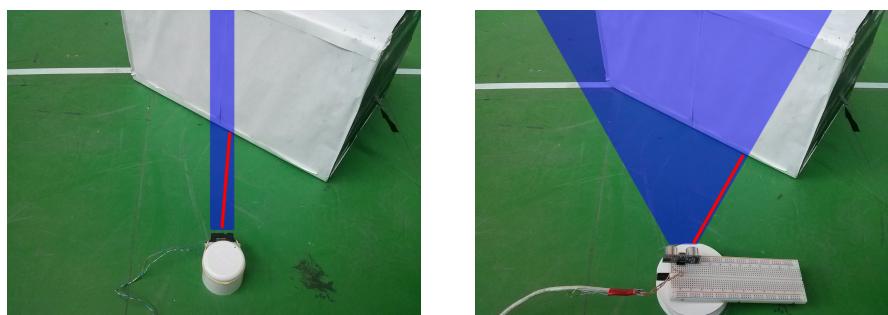


Figura 4.6: Tercer experimento.

De este experimento pueden sacarse varias conclusiones. La primera de ellas es que no existe un sensor mejor que el otro, cada uno destaca en una funcionalidad. Un sensor infrarrojo nos dará una mejor precisión a la hora de realizar medidas de distancia en un punto concreto, sin embargo, su rango de acción es limitado y será importante controlar donde está apuntando exactamente en cada momento. El sensor de ultrasonidos por su parte, detectará un obstáculo con mayor probabilidad, pero siempre existirá un incertidumbre respecto al posicionamiento exacto del obstáculo que estamos midiendo. De esta forma, un sensor infrarrojo será útil cuando necesitemos precisión en la posición del objeto y por este motivo, será el seleccionado para montarse en el robot. De todos modos, no se descarta la idea de montar también sensores de ultrasonidos en un futuro.

4.3.2. Sensores inerciales

El control del equilibrio del robot es muy importante en configuraciones bípedas, ya que estos robots son conocidos por su facilidad para tropezar y caer al suelo. Con el objetivo de analizar la posición del robot respecto al suelo se ha requerido la inclusión de diferentes sensores inerciales.

Acelerómetro y giroscopio

En el mercado existen varios modelos económicos. En este proyecto se ha utilizado el sensor MPU9150 que aparece en la figura 4.7. Se trata de un sensor inercial compuesto de acelerómetro de 3 ejes, giroscopio de 3 ejes y brújula de 3 ejes. Este sensor combina dos sensores, un MPU6050 (incluye el acelerómetro y el giroscopio) y un AK8975 (incluye la brújula). Sin embargo, en la práctica se observó que la brújula incluida en el sensor era muy propensa a sufrir interferencias, por lo que se decidió montar una brújula adicional que no tuviese estos problemas.

Brújula magnética

La inclusión de una brújula es indispensable cuando de requiere conocer la dirección en la que se desplaza un robot. En este proyecto se montará un brújula para evaluar si el desplazamiento del robot se produce de forma recta y efectuar los redireccionamientos necesarios. Existen diferentes modelos de brújulas digitales en el mercado. Todas ellas, basan su funcionamiento en la detección y medición de campos magnéticos. Para lograr conocer la orientación del robot, la brújula deberá apoyarse en su posición respecto a la del

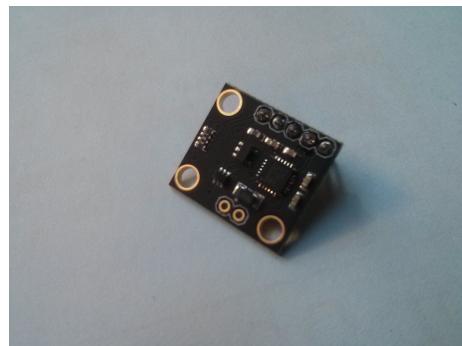


Figura 4.7: Sensor inercial MPU9150

campo magnético terrestre. El gran problema de estos dispositivos es su facilidad para modificar sus medidas cuando existe un campo magnético fuerte en su entorno. Esta interferences pueden ser causadas por aparatos electrónicos, estructuras metálicas o incluso por los propios componentes del robot.



Figura 4.8: Brújula magnética CMPS03

Para este proyecto se ha utilizado una brújula CMPS03 como la de la figura 4.8. El motivo de haber seleccionado este modelo frente a otros mas comunes, como por ejemplo la HMC5883L, se debe a que ya ha sido utilizada en anteriores proyectos robóticos en la asociación con muy buenos resultados. Esto se debe en parte a que la placa del sensor incluye un PIC que se encarga de calibrar y comunicar los datos medidos, de forma que no será necesario que el controlador del robot realice estos cálculos y podrá utilizar directamente la medida extraída.

4.3.3. Cámara

La elección de la cámara ha sido condicionada principalmente por su compatibilidad con el driver de Linux v4l2. Existe una amplia variedad de cámaras válidas en el mercado. Dado el carácter de este proyecto, se ha optado por utilizar una webcam, ya que son más accesibles que las cámaras avanzadas de investigación y los algoritmos de visión que serán programados no requieren imágenes de alta calidad.



Figura 4.9: Microsoft LifeCam Cinema

El modelo seleccionado ha sido una Microsoft LifeCam Cinema como la que aparece en la figura 4.9. Entre sus características destacan su posibilidad de grabar video en 720p HD, enfoque automático y la regulación de brillos en tiempo real. Adicionalmente, se trata de una cámara de fácil desmontaje y dimensiones contenidas, por lo que será sencillo integrarla en el robot.

Cabe remarcar que antes de utilizar esta cámara se realizaron pruebas con una Hama Pocket, pero dada su baja calidad de construcción presentó diversos problemas de fiabilidad y fue despreciada en favor de la Microsoft.

4.4. Elección del controlador

Llegamos a un punto crítico del proyecto, y es la elección de un controlador para nuestro sistema. La controladora CM-5 contenida en el kit original de Bioloid Comprehensive se encuentra muy lejos de poder soportar el conjunto de nuevos dispositivos que se usarán en el proyecto. Necesitaremos reemplazarla por un sistema más complejo que nos permita conectar y programar los sensores seleccionados, programar algoritmos de visión y comunicarse con los servos Dynamixel.

Para solucionar esto se decide colocar un SBC (Single Board Computer, ordenador de placa única), que dadas sus capacidades de procesamiento se utilizará como controlador principal. Así mismo, será el encargado de realizar el procesamiento de imágenes. Junto al SBC, se conectará un controlador más sencillo basado en un microcontrolador para controlar los Dynamixel AX-12A por separado.

De esta forma, el controlador del robot estará formado por dos elementos, el SBC que se encargará del procesamiento de imágenes y sensores, y un microcontrolador se encargará de la locomoción del robot.

4.4.1. Controlador de locomoción

El controlador de locomoción debe encargarse de mover los 20 servos del robot y de comunicarse con el controlador de visión para recibir instrucciones. A continuación se realiza un estudio de posibles placas que podrían utilizarse.

Arduino

Las placas Arduino se han hecho famosas por su fácil programación y puesta en marcha. Dada su popularidad, existe una gran comunidad de usuarios que generan documentación de forma constante. Con una placa Arduino disponemos de una amplia colección de bibliotecas para interactuar con sensores y actuadores. Sin embargo, estas placas por si solas no soportan la comunicación con los servos Dynamixel. Por esta razón, se ha desestimado su uso en el proyecto.

Arbotix

La placa Arbotix, mostrada en la figura 4.10 está basada, al igual que las placas Arduino, en un microcontrolador AVR. No obstante, esta placa de desarrollo está orientada principalmente a la construcción de pequeños robots, por lo que posee capacidades mejores que las de las placas Arduino. Entre sus especificaciones destacan: 2 puertos serie (uno de ellos reservado para el bus Dynamixel), 32 pines de entrada/salida, 8 entradas analógicas, 2 drivers de 1A para motores de continua y zócalos para la conexión de módulos XBEE.



Figura 4.10: Placa Arbotix

El problema de estas placas es que no existe un distribuidor que las venda en Europa. Eso no solo significa que sean difíciles de conseguir, sino tambien que no tienen demasiados usuarios, y por tanto la documentación es escasa y heterogenea.

Serie OpenCM TODO

Recientemente la empresa Robotis ha sacado a la venta una nueva linea de placas de desarrollo, las placas OpenCM. En la figura 4.11 se muestra una CM900. Estas placas de desarrollo están basadas en las placas Arduino, copiando sus funcionalidades y añadiendo al sistema la posibilidad de comunicarse con actuadores Dynamixel. Su entorno de programación está basado en Arduino IDE y continene bibliotecas similares a las de Arduino para la comunicación con sensores y actuadores. Su alta compatibilidad con Arduino proporciona al usuario una mayor facilidad a la hora de buscar documentación sobre cómo utilizar la placa.

TODO

Figura 4.11: Placa CM900

Si bien es cierto, es necesario comentar que la primera edición, la CM900, presentó problemas en su diseño que la convirtieron en una placa muy poco fiable y propensa a deteriorarse prematuramente. En este proyecto se utilizará la OpenCM9.04, sucesora de la CM900, con similares capacidades y tamaño reducido. Entre sus especificaciones destacan: Un microcontrolador ARM Cortex-M3, 26 pines de entrada/salida, 10 entradas analógicas de 12 bits y 3 puertos serie (uno de ellos reservado para el bus Dynamixel).

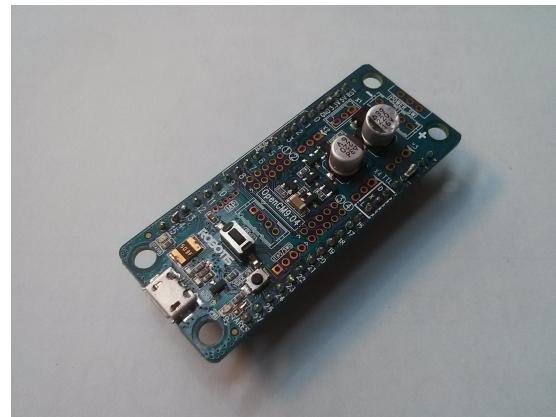


Figura 4.12: Placa OpenCM9.04

4.4.2. Controlador de visión

Como controlador principal, se utilizará un SBC (Single Board Computer). Por tamaño y capacidades existen dos alternativas di-

rectas: La Raspberry Pi B y la BeagleBone Black.

Raspberry Pi

La Raspberry Pi es un ordenador completo del tamaño de una tarjeta de crédito, dirigida a formar parte de proyectos de electrónica, informática y robótica. Dado su bajo coste y fácil adquisición, la Raspberry Pi cuenta con una comunidad de usuarios muy extensa repartida a lo largo del mundo. En internet existen colecciones de tutoriales educativos para formar al usuario y dar soporte a sus proyectos. En el cuadro 4.1 se describen las especificaciones técnicas de la placa.

| Especificaciones Raspberry PI modelo B | |
|---|---|
| CPU | ARM 1176JZFS 700 MHz |
| Memoria (SDRAM) | 512 MB (compartidos con la GPU) |
| Puertos USB 2.0 | 2 (vía hub USB integrado) |
| Almacenamiento integrado | SD / MMC / ranura para SDIO |
| Periféricos de bajo nivel | 8 x GPIO, SPI, I2C, UART |
| Consumo energético | 700 mA (3.5 W) |
| Fuente de alimentación | 5 V vía Micro USB o GPIO header |
| Dimensiones: | 85.60mm × 53.98mm |
| Sistemas operativos soportados: | GNU/Linux: Debian (Raspbian), Fedora (Pidora), Arch Linux (Arch Linux ARM), Slackware Linux. RISC OS2 |

Cuadro 4.1: Especificaciones Raspberry Pi B

En el caso que nos ocupa, sería una perfecta candidata a ocuparse del procesamiento de imágenes del robot, sin embargo hay algunos detalles que es importante conocer. El primer problema de la Raspberry Pi es que la lógica de sus pines trabaja a 5V, mientras que la placa que hemos seleccionado para la parte de locomoción, la OpenCM, funciona a 3.3V. Esto significa que para comunicar ambas placas sería necesario incluir un convertidor lógico en el sistema. Por otra parte, sus GPIOs tienen un funcionamiento puramente binario,

sin entradas analógicas ni generación de señales PWM. Por último, como puede observarse en la figura 4.13 aunque tiene un tamaño razonable, la placa cuenta con varios conectores de video, audio, conexión en red... etc que aunque podrían resultar útiles en otro tipo de proyectos, en nuestro caso estorbarán a la hora de integrar el dispositivo en la espalda del robot.



Figura 4.13: SBC Raspberry Pi B ©Raspberry Pi Foundation.

BeagleBone Black

La BeagleBone Black de Texas Instruments, fotografiada en la figura 4.14, no es tan popular como la Raspberry Pi, sin embargo posee algunas características muy interesantes.



Figura 4.14: SBC BeagleBone Black

Como puede observarse en el cuadro 4.2, sus características son ligeramente superiores a las de la Raspberry. Principalmente, el he-

cho de que tenga tal variedad de pines la hace muy adecuada para usarse como controlador principal de un robot. La BeagleBone Black actualmente no cuenta con una comunidad tan extensa como la Raspberry Pi, pero cada vez se ve mas en proyectos. El problema fundamental de la BeagleBone Black es la falta de bibliotecas en C/C++ que permitan programar sus periféricos de bajo nivel. Esto significa que en el caso de utilizar esta placa, se deberán programar bibliotecas propias para estos propósitos.

| Especificaciones BBB | |
|---------------------------------|---|
| CPU | AM335x 1GHz ARM Cortex-A8 |
| Memoria (SDRAM) | 512 MiB |
| Puertos USB 2.0 | 1x Standard A y 1x mini B |
| Almacenamiento integrado | 4GB 8-bit eMMC / microSD |
| Periféricos de bajo nivel | 4xUART, 8x PWM, LCD, GPMC, MMC1, 2x SPI, 2x I2C, A/D Converter, 2x CAN bus |
| Consumo energético | 1200 mA (6 W) |
| Fuente de alimentación | 5 V vía Micro USB, jack de 5.5mm a 5 V o GPIO header |
| Dimensiones: | 86.40 mm × 53.30 mm |
| Sistemas operativos soportados: | Fedora, Android, Ubuntu, Debian, openSUSE, Ångström, FreeBSD, NetBSD, OpenBSD, QNX, MINIX 3, RISC OS, y Windows Embedded. |

Cuadro 4.2: Especificaciones BeagleBone Black

Por otro lado, su lógica funciona a 3.3V y es mas compacta que la Raspberry Pi. Por todo lo comentado, y aunque trabajar con ella pueda resultar mas tedioso que con la Raspberry Pi, se ha seleccionado la Beaglebone Black como controlador principal del robot.

4.4.3. Arquitectura hardware

Una vez se ha seleccionado todos los elementos que formarán parte del robot, se ha diseñado el conexionado que llevarán. Tal y como se ha ido viendo a lo largo de este apartado, por un lado tendre-

mos la OpenCM que se encargará de controlar el servo PWM y los servos Dynamixel. La BeagleBone en cambio, se encargará de recibir los datos de todos los sensores y de la cámara. Las dos placas se conectarán en serie. El diagrama de la figura 4.15 representa las conexiones de todos los dispositivos, a excepción de la parte de alimentación, que se verá en la próxima sección.

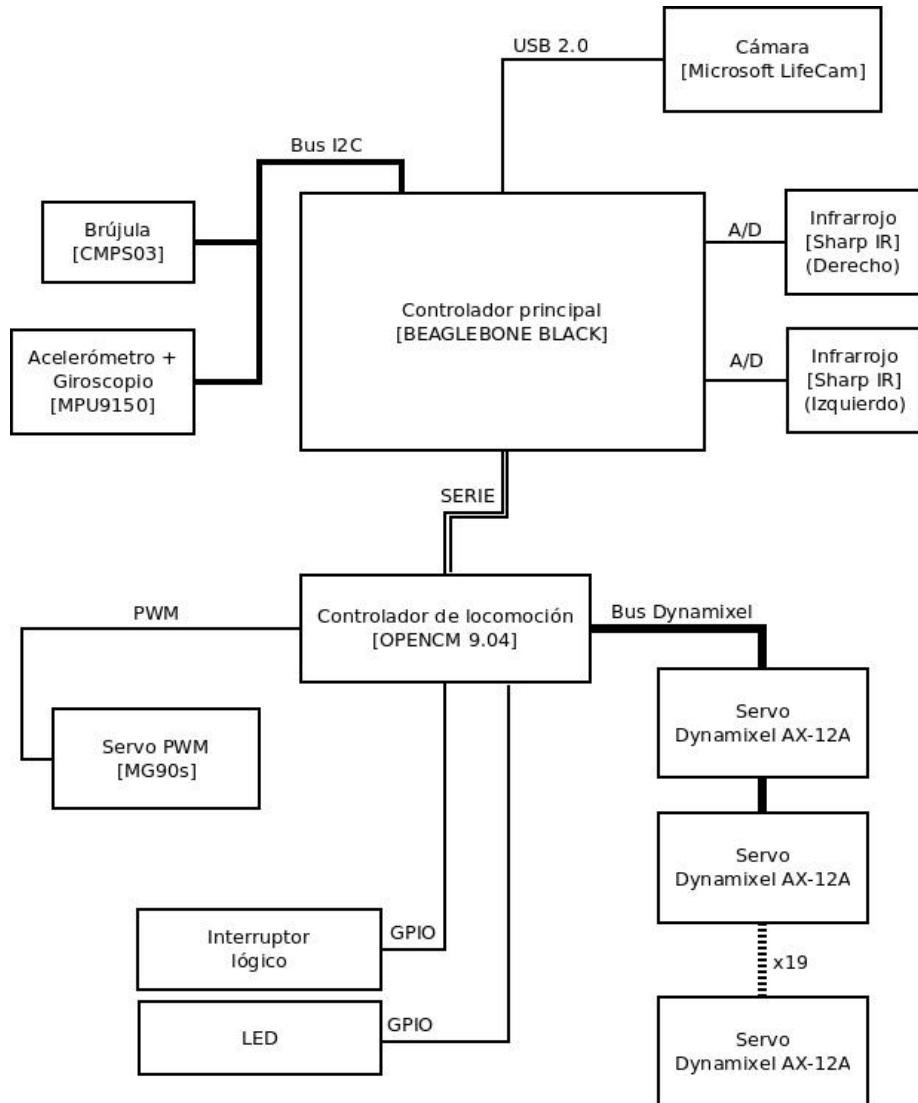


Figura 4.15: Diagrama de conexiones

4.5. Alimentación

La batería contenida en el kit es una batería NiMH de 8 elementos AA Sanyo Eneloop que desarrolla 9.6V y 1900mAh. Para el consumo de este robot es una opción aceptable que proporciona cerca de 10 minutos de autonomía. No está mal si se tiene en cuenta que pruebas con el CEABOT tiene una duración máxima limitada a 5 minutos. No obstante, tiene un inconveniente, su peso y su volumen es muy alto. Al estar situada en la espalda, el centro de gravedad del robot sube en altura de forma considerable. Esto causa inestabilidades y complica bastante la programación de desplazamientos.

4.5.1. Batería

Por las razones descritas, se ha decidido buscar una alternativa. Los requisitos para seleccionar una batería los marcan tres factores: La tensión de funcionamiento de los servos, la de las placas de control y la de los sensores. Pero dado que los servos tienen una tensión de operación en un rango de 9V a 12V, respecto al resto de elementos que serán alimentados a 5V o 3.3V, nos centraremos principalmente en encontrar una batería adecuada para los servos.

Hoy en dia, las baterías NiMH están empezando a caer en desuso a favor de las baterías de litio. Se barajaron dos posibilidades.

Baterías de polímero de litio

Las baterías de polímero de litio, mas conocidas como baterías LiPo, destacan por desarrollar una tasa de descarga bastante alta y por tener un tamaño y peso reducido. Dado su uso en automodelismo y aeromodelismo, son unas baterías muy comunes que en los últimos años han ido bajando su precio. Actualmente existe una gran variedad de baterías LiPo de diversos tamaños y precio asequible. Cada célula LiPo ofrece 3.7V, por lo que la elección correcta para este robot sería una batería de 3S, o lo que es lo mismo, 11,1V.

Sin embargo, las baterías LiPo tienen una desventaja importante, su fragilidad. El proceso de carga y descarga de una batería de este tipo no es un asunto trivial, una sobredescarga de la batería provoca un deterioro inmediato de la misma, calentando sus elementos e incluso pudiendo llegar a incendiarse. Es por ello que se necesita un cargador con capacidad para balancear las células, de forma que las tensiones de cada una de ellas se tengan bajo control durante el proceso.

Baterías de litio fosfato de hierro

Las baterías de litio fosfato de hierro, es decir, LiFe, son conocidas por su gran resistencia y larga vida útil. Cada célula de LiFe desarrolla 3.3V y no presenta problemas por sobredescargas. De hecho, estas baterías ni siquiera requieren de un cargador balanceador para su carga, sino que pueden cargarse sin problemas con un cargador normal. El gran problema de estas baterías es que sus elementos son bastante voluminosos, y, al no ser tan populares como las LiPo, tampoco es demasiado fácil encontrar modelos de tamaño y capacidad adecuados en el mercado.

Elección y diseño del sistema

Por su tamaño y prestaciones se ha elegido la batería LiPo Rhino de 3S y 1750mAh que aparece en la figura 4.16. Esta batería tiene unas dimensiones perfectas para montarse en la cintura del robot y su peso (109 gramos) es mucho menor al de la batería original (215 gramos).

A continuación se presenta una tabla (cuadro 4.3) con una estimación del consumo medio de todos los elementos seleccionados.



Figura 4.16: Batería LiPo

| Dispositivo | Número | Consumo unitario | Consumo total |
|-------------------|--------|----------------------------------|------------------------|
| Dynamixel AX-12A | x19 | $11,1V \cdot 600mA = 6660mW$ | $126540mW$ |
| TowerPro MG90s | x1 | $5V \cdot 120mA = 600mW$ | $600mW$ |
| BeagleBone Black | x1 | $5V \cdot 1200mA = 6000mW$ | $6000mW$ |
| OpenCM 9.04 | x1 | $11,1V \cdot 90mA \simeq 1000mW$ | $1000mW$ |
| Sharp IR | x2 | $5V \cdot 30mA = 150mW$ | $300mW$ |
| MPU9150 | x1 | $3,3V \cdot 15mA \simeq 50mW$ | $50mW$ |
| HMC5883L | x1 | $3,3V \cdot 12mA \simeq 40mW$ | $40mW$ |
| Microsoft LifeCam | x1 | $5V \cdot 200mA = 1000mW$ | $1000mW$ |
| Total | | | $135530mW \simeq 136W$ |

Cuadro 4.3: Consumo medio

Por tanto, la estimación de autonomía del robot en funcionamiento es la siguiente:

$$\frac{11,1V \cdot 1750mAh}{135530mW} \cdot \frac{60min}{h} \simeq 8,6min$$

Entre 8 y 9 minutos de autonomía teóricos es un buena cifra contando con el tamaño del robot y sus prestaciones. Como conclusión podemos remarcar el hecho de que gracias al nuevo sistema de alimentación hemos podido introducir los dispositivos necesarios para cumplir los requerimientos de este proyecto sin comprometer la autonomía del robot.

4.5.2. Regulador de tensión

Hasta aquí parece que es suficiente con la elección de la nueva batería, sin embargo, aún falta un detalle que deberemos resolver. La mayor parte de los dispositivos que forman el conjunto están alimentados a 5V. La batería que hemos elegido tiene una tensión

de 11.1V, y se ha observado, que cuando está cargada al máximo desarrolla hasta 12.6V. No podemos alimentar la BeagleBone Black y el resto de componentes a una tensión tan alta, por lo que será necesario colocar un regulador que nos proporcione 5V.

Regulador lineal

La primera idea fue utilizar un pequeño circuito con un regulador lineal LM7805. El LM7805 es un componente muy común en los circuitos electrónicos, ya que ofrecen una salida de 5V a partir de tensiones de entrada de hasta 35V. En principio no parecería muy descabellado conectar los servos AX-12A y la placa OpenCM directamente a la batería, y conectar el resto de componentes a través del regulador.

No obstante existen dos problemas para realizar este montaje. El primero es que la eficiencia de este componente es muy pobre, cercana al 60 % según su hoja de características. Esto provocaría una disminución drástica en el tiempo de operación del robot y una disipación de energía tan alta que sería necesario incluir un disipador. Pero el segundo problema es más importante, el componente está diseñado para aportar una corriente máxima al circuito de 1 amperio. Teniendo en cuenta los datos del cuadro 4.3, solo la BeagleBone Black ya requiere una corriente de 1.2A, y junto al resto de componentes, asciende hasta 1.7A . Un consumo superior al marcado por el fabricante puede producir fallos de funcionamiento en el componente. En este caso particular, podría producirse un calentamiento excesivo, que a su vez puede llegar a provocar cortocircuitos internos entre la patilla de entrada y la de salida. Las consecuencias de un fallo así serían nefastas, ya que deteriorarían los dispositivos conectados a él.

Convertidor UBEC

Un UBEC es un convertidor DC-DC de tipo reductor. El dispositivo se muestra en la figura 4.17. Los UBEC se utilizan normalmente con baterías de litio cuando el sistema necesita una tensión de alimentación de 5V o 6V. Es por ello que se utilizan mucho en receptores de emisoras de radiocontrol, y gracias a esto existe una amplia variedad de UBECs asequibles en el mercado. La mayor ventaja de estos circuitos convertidores es su eficiencia superior al 90 %. Gracias a esto nuestro sistema tendrá unas perdidas menores, no se calentará en exceso y gozará de una autonomía superior.



Figura 4.17: Convertidor DC-DC comercial, UBEC ©HeliDirect.com

Elección final

Tal y como se comentaba anteriormente, nuestro sistema tiene un consumo estimado de unos 1.7A en la salida de 5V, por lo que un UBEC de 3A sería suficiente para proporcionar al circuito la corriente necesaria para su funcionamiento. Adicionalmente se han incluido unos condensadores en la entrada y salida del circuito para evitar caídas bruscas de tensión debidas a requerimientos de intensidad anómalos. Este sistema nos proporcionará una distribución de alimentación correcta para todos los elementos del robot.

Capítulo 5

Descripción de las herramientas a utilizar

A continuación se presentan las herramientas básicas que serán necesarias durante el desarrollo del proyecto.

5.1. Herramientas de diseño y fabricación de piezas

Dado que la plataforma robótica seleccionada requiere modificaciones mecánicas para permitir el montaje de los componentes necesarios, se requiere construir una serie de piezas que sustituyan a las originales y que aporten al robot algunas características de las que carece. Todas las piezas del robot serán diseñadas con OpenSCAD e impresas posteriormente con una impresora 3D open-source.

5.1.1. OpenSCAD

OpenSCAD [43] es un programa destinado a la creación de objetos sólidos tridimensionales. Se trata de software libre y es compatible con Linux/UNIX, Windows y Mac OS X. A diferencia de otros programas de diseño 3D, OpenSCAD no se centra en los aspectos artísticos del diseño, sino en el aspecto técnico. Por ello, es una aplicación muy interesante cuando nuestro objetivo es crear piezas mecánicas,y en este caso particular, para un robot.

La propiedad mas característica de OpenSCAD y que le hace diferente de otros programas de diseño como SolidWorks o FreeCAD, es su interfaz, mostrada en la figura 5.1. Este programa funciona como un compilador de objetos 3D, leyendo un script que describe el objeto y renderizando el objeto a partir de ese archivo. Gracias a

48 CAPÍTULO 5. DESCRIPCIÓN DE LAS HERRAMIENTAS A UTILIZAR

esto, el usuario tiene total control sobre el proceso de modelado, permitiendo la realización de modelos variables a partir de parámetros configurables.

OpenCad también permite el diseño de modelos planos, siendo compatible con formatos como DXF. Sin embargo, para el caso de este proyecto, los archivos que nos interesa producir son los STL.

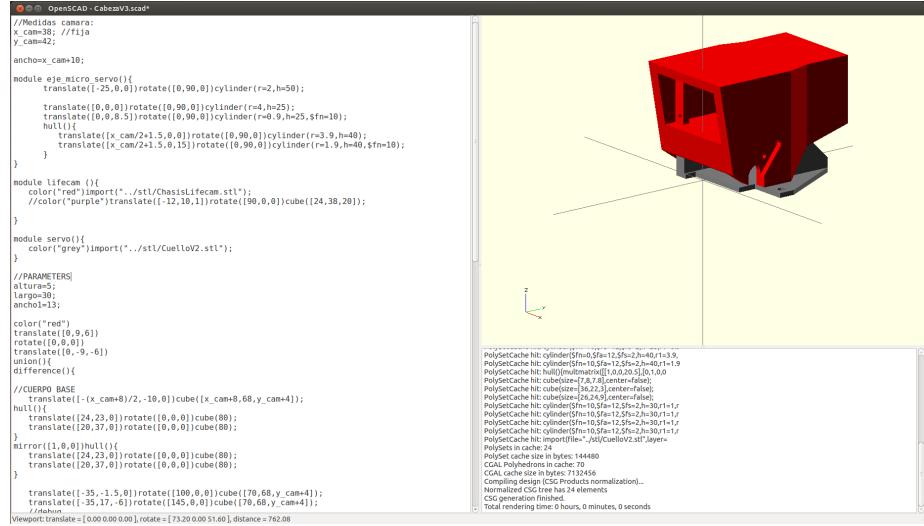


Figura 5.1: Pantalla del editor de OpenSCAD

5.1.2. Impresión 3d

Para la fabricación de las piezas que integran el robot, se ha utilizado la impresora 3D [28] replicable open-source modelo Prusa i2 Air de construcción casera de al figura 5.2. La configuración habitual de esta impresora ha sido mejorada con un extrusor Budaschnozzle 1.3 y una electrónica Sanguinololu 1.3b.

Esta impresora, dado su funcionamiento, pertenece a la familia del modelado por deposición fundida. La materia prima que utilizan este tipo de impresoras es un rollo de filamento de plástico termofusible de entre 1.5 y 3mm de diámetro. En este caso, se utilizará ABS de 3mm. El filamento de plástico es dirigido a un extrusor que empuja el material a través de un conducto caliente conocido como hotend. Al llegar a este punto, el filamento se funde y se hace pasar por un agujero de salida de tamaño muy inferior al de entrada, produciendo hilos de material fundido. Durante la impresión, el extrusor deposita plástico fundido a lo largo de diferentes trayectorias con el objetivo de formar capas horizontales sólidas. Mediante la apilación de estas capas se consigue dotar de altura al modelo y crear la pieza

requerida. A través del programa Repetier-Host, que se ocupa de gestionar el funcionamiento de la impresora desde el ordenador, y partiendo de los archivos STL que han sido generados anteriormente desde OpenSCAD, la impresora nos permite desarrollar prototipos y piezas finales para el proyecto.

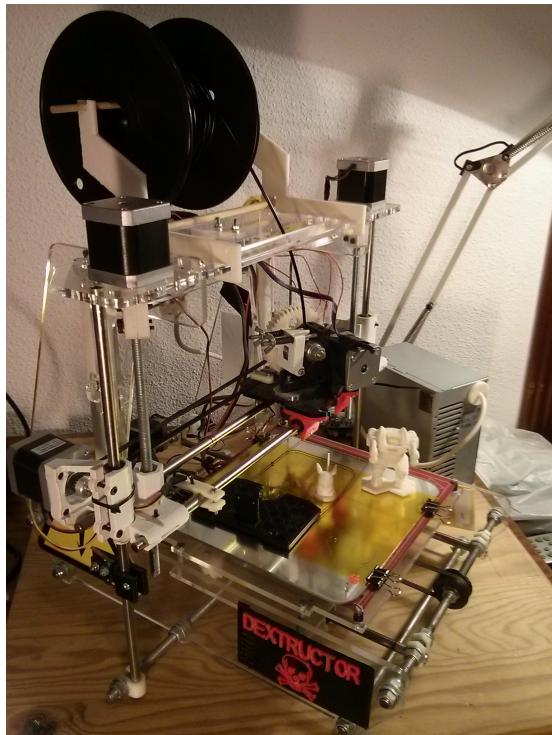


Figura 5.2: Impresora 3D Prusa i2 Air

5.2. Herramientas de diseño de circuitos

Dado que se va a necesitar expandir las conexiones físicas del controlador, se requiere diseñar una placa de expansión que permita realizar un montaje adecuado del sistema.

5.2.1. KiCad

KiCad [41] es una de las herramientas libres más avanzadas para el diseño electrónico asistido (EDA) que pueden encontrarse a día de hoy. Cuenta con un grupo de más de 150 desarrolladores y una extensa comunidad de usuarios entre quienes se pueden destacar laboratorios como el CERN[45].

50 CAPÍTULO 5. DESCRIPCIÓN DE LAS HERRAMIENTAS A UTILIZAR

KiCad permite crear diseños electrónicos pasando por todas las fases del proceso, desde la idea a los ficheros de fabricación y la simulación 3D de la placa. El entorno, mostrado en la figura 5.3, cuenta con cuatro aplicaciones independientes:

- **Eeschema.** Editor del esquemático.
- **Pcbnew.** Editor de la placa de circuito impreso.
- **Gerbview.** Visor de archivos GERBER
- **Cvpcb.** Editor de huellas para componentes.

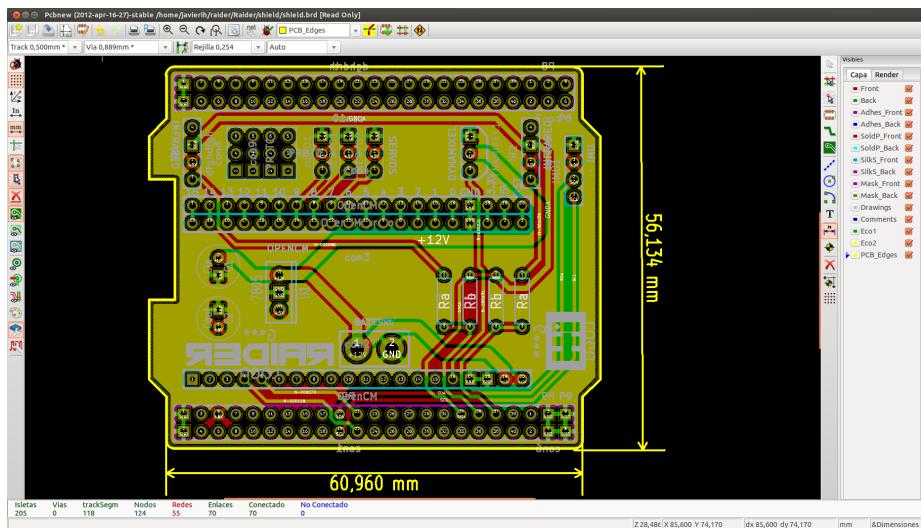


Figura 5.3: KiCad

KiCad es un programa multiplataforma, escrito con wxWidgets para ser ejecutado en FreeBSD, Linux, Microsoft Windows y Mac OS X. Existe una amplia colección de librerías de componentes, a las cuales se suma la posibilidad para el usuario de crear componentes personalizados. Además, existen herramientas para importar los componentes de otros EDA, como por ejemplo desde EAGLE.

5.3. Herramientas de programación

Este proyecto conlleva una programación a diferentes niveles, desde el control de movimientos de los actuadores desde un microcontrolador hasta el diseño de algoritmos de navegación a alto nivel.

5.3.1. OpenCV

OpenCV[18] es una biblioteca libre de visión artificial. Fue creada en 1999 por Intel y liberada un año mas tarde en la conferencia anual IEEE Conference on Computer Vision and Pattern Recognition. Existen infinidad de aplicaciones, como la programación de detección de movimiento para cámaras de seguridad o la detección visual de características importantes en diferentes etapas del proceso de fabricación de un producto. La gran popularidad de OpenCV se debe a que su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación.

OpenCV también es multiplataforma y existen versiones para GNU/Linux, Windows y Mac OS X. Entre sus funcionalidad, se abarcan campos de la visión por computador como el reconocimiento de objetos, calibración de cámaras, visión 3D y visión robótica. Además, OpenCV está programado de forma muy optimizada en C y C++, lo que le otorga una alta eficiencia y aptitud para aplicaciones en tiempo real.

5.3.2. Qt Creator

Qt Creator[25] es un entorno de desarrollo multiplataforma y open-source desarrollado por la compañía noruega Trolltech. Soporta C++, JavaScript y QML. El editor, mostrado en la figura 5.4, incluye resalto de sintaxis y funciones de autocompletado, muy útiles cuando se trabaja con proyectos de una extensión media o alta. Qt Creator utiliza el compilador de C++ de GNU Compiler Collection (o lo que es lo mismo, GCC). Qt Creator interpreta por defecto archivos de CMake.

52 CAPÍTULO 5. DESCRIPCIÓN DE LAS HERRAMIENTAS A UTILIZAR

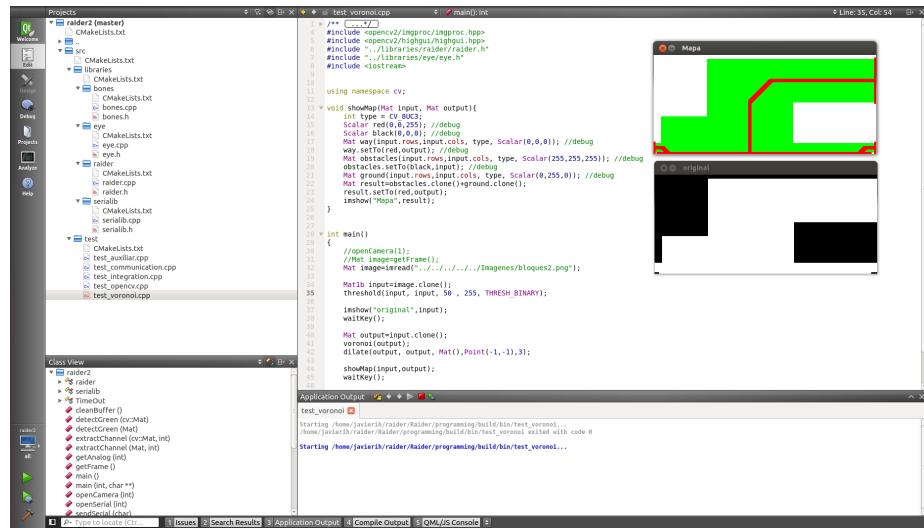


Figura 5.4: Qt Creator

5.3.3. CMake

CMake [40] es una herramienta open-source para la administración de la generación de código. El nombre CMake es una abreviatura de cross platform make (make multiplataforma).

CMake es un conjunto de herramientas creado para construir, probar y empaquetar software. Se utiliza para controlar el proceso de compilación del software usando ficheros de configuración sencillos e independientes de la plataforma. CMake genera makefiles nativos y espacios de trabajo que pueden usarse en el entorno de desarrollo deseado. CMake soporta la generación de ficheros para varios sistemas operativos, lo que facilita el mantenimiento y elimina la necesidad de tener varios conjuntos de ficheros para cada plataforma.

El proceso de construcción se controla creando uno o más ficheros CMakeLists.txt en cada directorio del proyecto.

5.3.4. CM9 IDE

CM9 IDE[42] es un entorno de programación basado en Arduino IDE, preparado para programar placas electrónicas de la serie OpenCM. Soporta programación en C/C++ y es compatible con la mayoría de las librerías de Arduino. CM9, mostrado en la figura 5.5, constituye un entorno centralizado desde el cual se puede escribir código, compilarlo y cargarlo en la placa OpenCM sin necesidad de ninguna otra aplicación adicional.

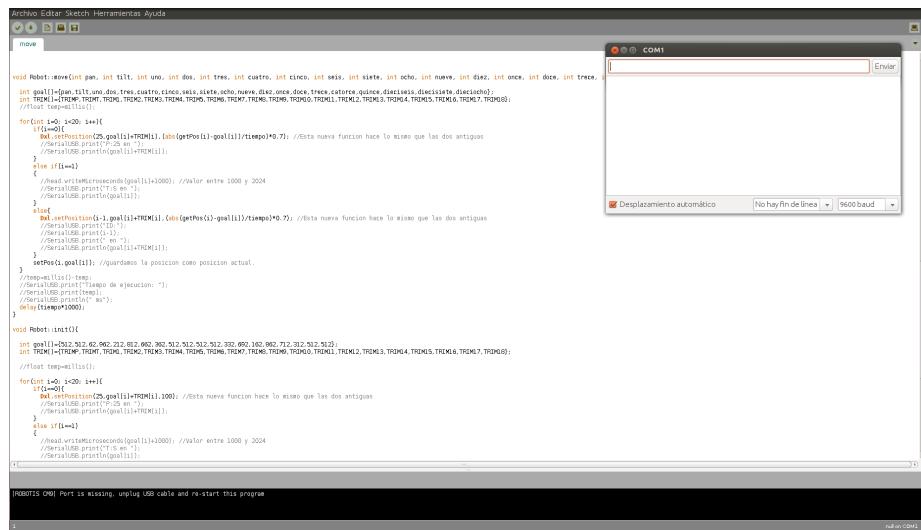


Figura 5.5: CM9 IDE

5.3.5. Git

Git[39] es un software de control de versiones diseñado para controlar el código de un proyecto en sus distintas iteraciones. Se diseñó pensando en la eficiencia y seguridad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente e involucran a varios desarrolladores. Este proyecto se ha desarrollado enteramente en un repositorio online de Github. TODO

Con todas las herramientas comentadas en este apartado, se tienen los recursos suficientes para comenzar la construcción del robot.

Capítulo 6

Diseño de las partes mecánicas

La mayor parte de las piezas que monta Raider han sido rediseñadas para mejorar su función y/o permitir el alojamiento de nuevos dispositivos. Además, al haber sido diseñadas con OpenSCAD, ha sido posible parametrizarlas para facilitar modificaciones futuras. Acciones como agregar a una pieza un soporte para un sensor determinado son ahora mucho mas rápidas.

6.1. Cabeza

Uno de los principales objetivos del proyecto, el tomar datos del entorno con una cámara, requiere el diseño y montaje se un soporte móvil en el que poder albergar la webcam. Se ha decidido colocar la webcam en la cabeza del robot por ser la zona mas alta del robot y por tener espacio para su libre movimiento. La cabeza estará formada por cuatro piezas: Bancada del servo, chasis, tapa con ranura para el cable y soporte de la cámara. En la imagen de la figura 6.1 se muestran las piezas.

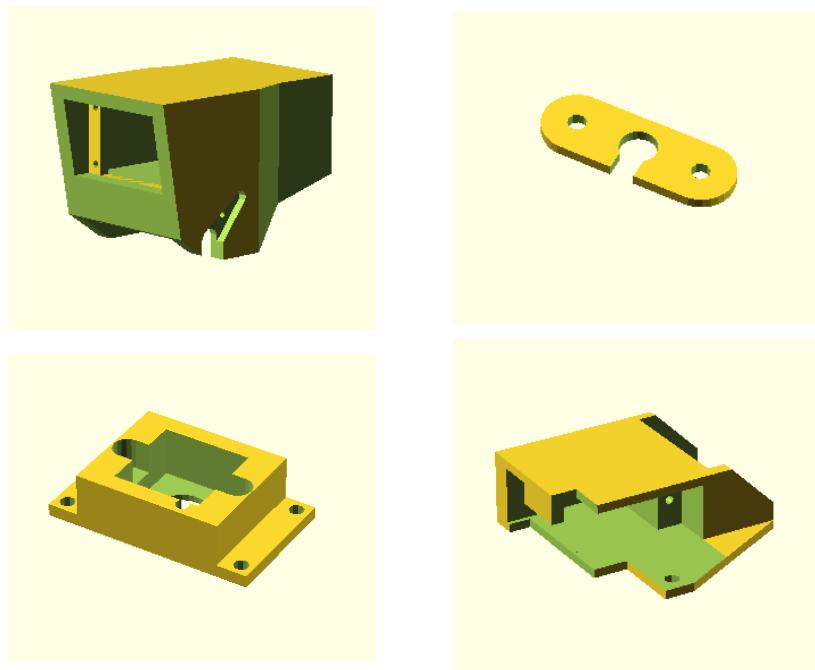


Figura 6.1: Piezas necesarias para montar la cabeza

6.2. Cintura

Como se comentó en el apartado 4.2.2, ha sido necesario incluir un servo adicional en la cintura que permita realizar movimientos horizontales del tronco del robot respecto a las piernas. Como puede verse en la figura 6.2, consta de dos piezas que aprisionan el servomotor y otra más que se utilizará como soporte de la batería.

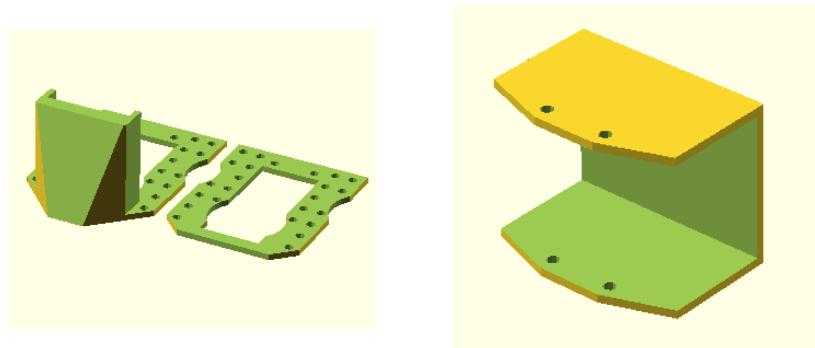


Figura 6.2: Piezas necesarias para montar la cintura

6.3. Tronco

Para diseñar el tronco existen varias premisas. Necesitamos un cuerpo que permita albergar el nuevo controlador, soportar los servos de los hombros, tener una zona adecuada para montar la cabeza y una base firme que permita atornillarlo al nuevo servo de la cintura. En la imagen de la figura 6.3 se muestran las piezas necesarias para montar el tronco: Pecho, espalda y protector.

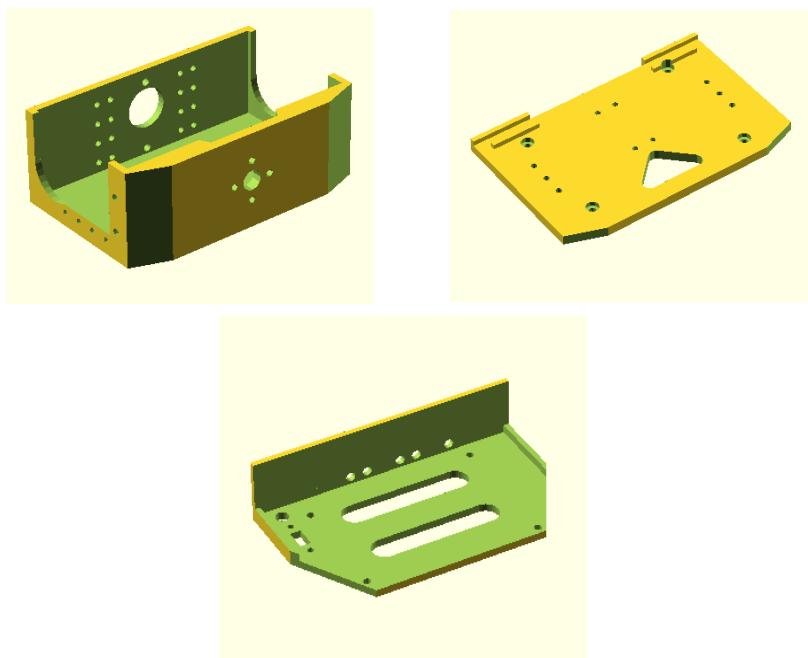


Figura 6.3: Piezas necesarias para montar el tronco

6.4. Brazos

En este apartado se han diseñado diferentes tramos de los brazos. Con ello se ha dotarle de una total modularidad, separando uniones entre servos, soportes para sensores y protectores. Gracias a esto, es muy sencillo sustituir unas piezas por otras. En este caso se han incluido sensores infrarrojos en los antebrazos, sin embargo, si se necesitase cambiarlos por sensores de ultrasonidos (o cualquier otro sensor) solo sería necesario modificar y reimprimir la bancada del sensor. En la figura 6.4

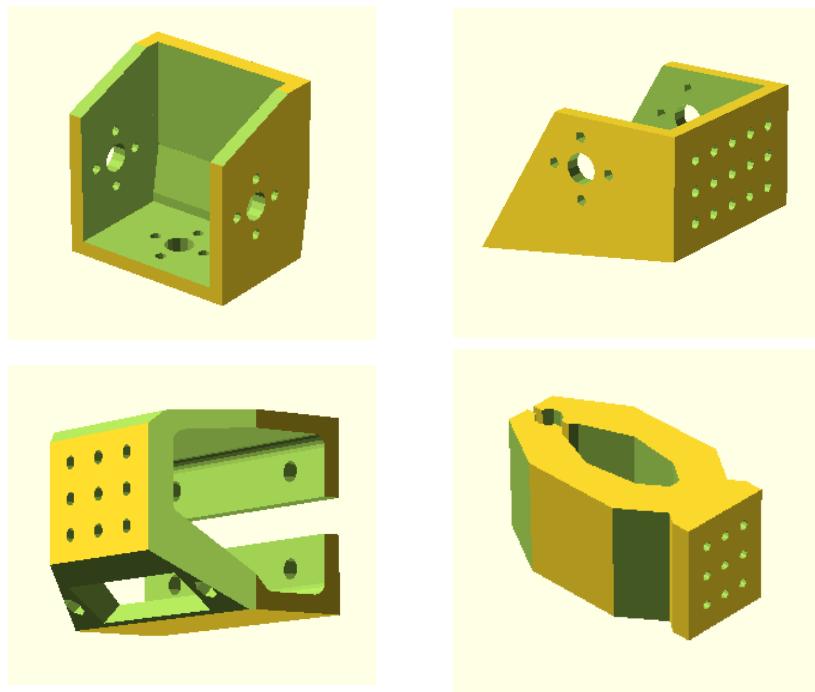


Figura 6.4: Piezas necesarias para montar un brazo

6.5. Piernas

En cuanto a las piernas, mostradas en la figura 6.5, se han modificado con dos objetivos: Rigidificar sus tramos y acortar levemente su longitud para descargar peso en las articulaciones. Al mismo tiempo, ayudarán a bajar el centro de gravedad del robot y aumentar su estabilidad.

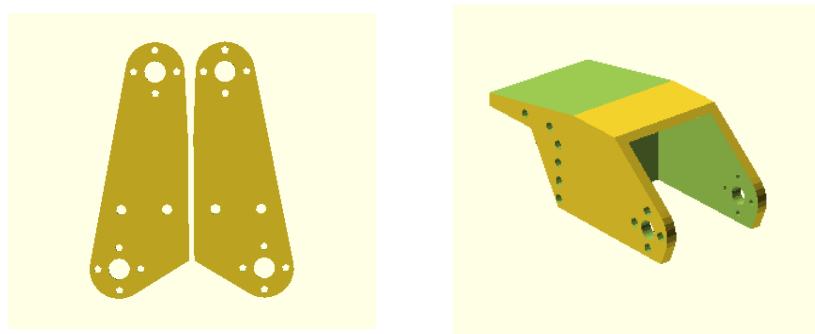


Figura 6.5: Piezas necesarias para montar una pierna

6.6. Pies y tobillos

El diseño de los pies se ha planteado de forma que cumpla las especificaciones del reglamento del campeonato CEABOT y al mismo tiempo tenga la mayor superficie posible. A priori, parece sencillo suponer que cuanto mayor sea la extensión del pie mayor será la estabilidad del robot, sin embargo, también es muy importante controlar como se distribuye el peso en la planta. Con el objetivo de concentrar el peso en el centro de la planta, se ha realizado un rediseño completo del tobillo, tal y como puede observarse en las piezas de la figura 6.6

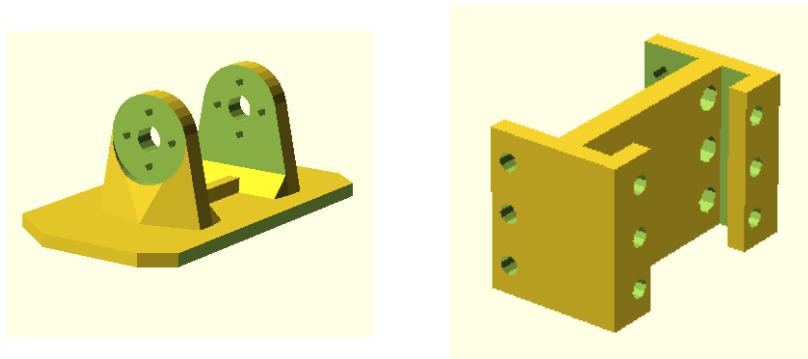


Figura 6.6: Piezas necesarias para montar un pie

Todas estas piezas serán impresas. En el próximo capítulo, se comentarán algunos aspectos constructivos del sistema, y consideraciones en el montaje de las mas importantes.

Capítulo 7

Desarrollo y montaje

En este capítulo se mostrarán los procedimientos que se han seguido para montar algunos componentes. Hasta este punto hemos definido una colección de sensores, actuadores y piezas; el siguiente paso será integrar y conectarlos todos en el robot.

7.1. Adecuación de sensores

Los sensores utilizados, tal y como se detalló en la selección de componentes, serán: un acelerómetro y giroscopio MPU9150, una brújula CMPS03 y dos infrarrojos Sharp. A continuación se muestra cómo han sido conectados al controlador principal, la BeagleBone Black.

7.1.1. Infrarrojos Sharp

Los sensores infrarrojos modelo GP2Y0A21YK0F de la marca Sharp, permiten variar la tensión de alimentación entre -0.3 y 7V, pero experimentalmente se ha observado que la mayor precisión se alcanza a 5V. A esta tensión, podemos observar en la hoja de características[24] del sensor la gráfica 7.1, en la que se muestra el valor de la salida del sensor respecto a la distancia medida.

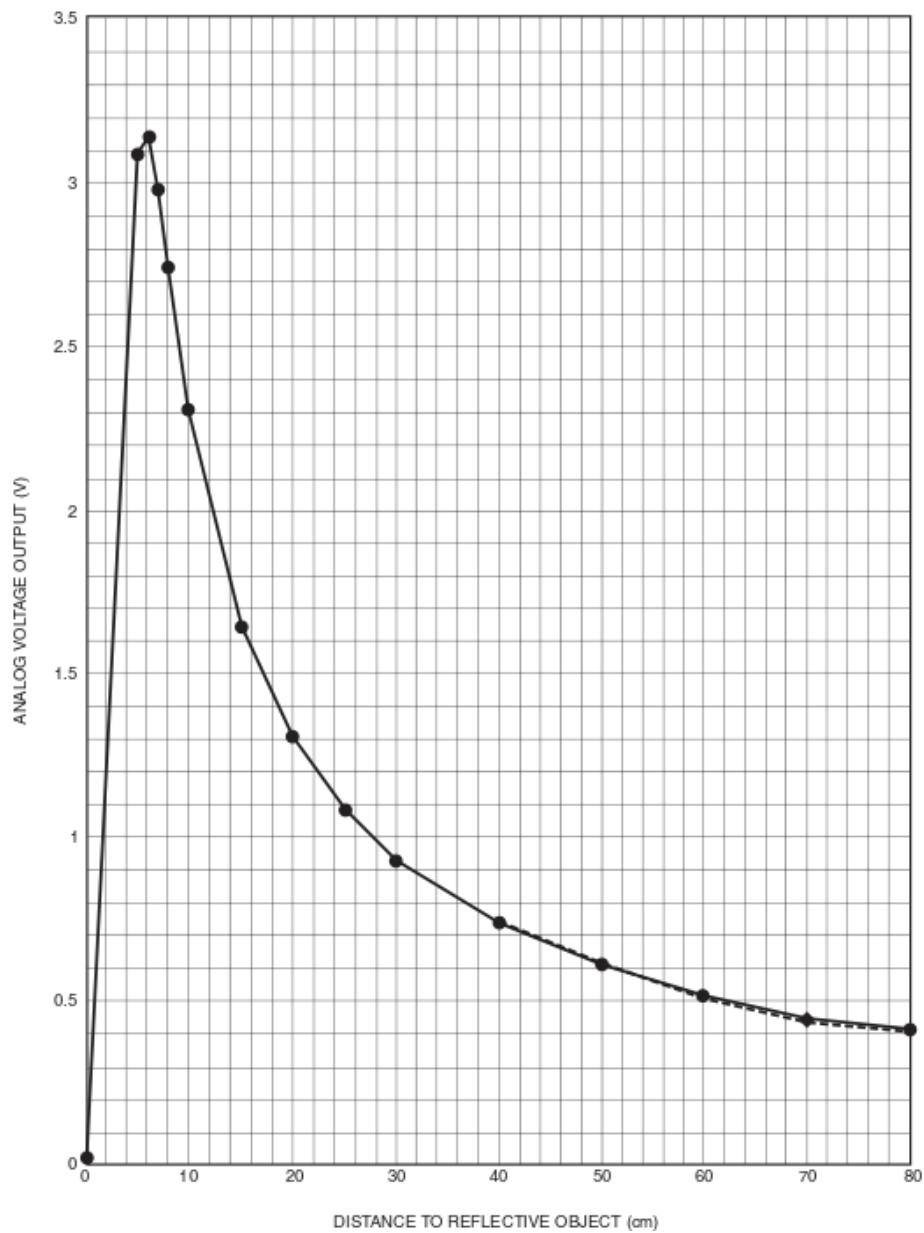


Figura 7.1: Gráfica de tensión entre salida para un sensor infrarrojo

De esta gráfica pueden sacarse dos conclusiones. La primera, ya conocida, es que la variación de la salida del sensor y su medida no son proporcionales, por lo que si fuese necesaria extraer medidas en unidades reales habría que calcular una adecuación compleja. Sin embargo, esto no será necesaria ya que la programación del sensor se basará en umbrales fijos que serán marcados de forma empírica. La segunda conclusión es que el sensor ofrecerá una tensión de salida

máxima de 3.3V.

La BeagleBone Black dispone de hasta siete entradas analógicas como puede observarse en la figura 7.2. Conectaremos las salidas de los dos sensores en dos de estos pines. Sin embargo, consultando el manual de la BeagleBone Black [23] observamos que cada entrada analógica soporta un máximo de 1.8V. Si alimentando el sensor a 5V lo conectásemos directamente, un pico de 3.3V podría deteriorar la placa. Por ello, se ha decidido aplicar una simple adecuación de la salida añadiendo un divisor resistivo. De esta forma, nos cercioraremos de que los 3.3V máximos que podrían salir del sensor se conviertan en 1.8V.

| P9 | | | P8 | | |
|----------|----|----|------------|---|---|
| | 1 | 2 | | 1 | 2 |
| DGND | 1 | 2 | DGND | | |
| VDD_3V3 | 3 | 4 | VDD_3V3 | | |
| VDD_5V | 5 | 6 | VDD_5V | | |
| SYS_5V | 7 | 8 | SYS_5V | | |
| PWR_BUT | 9 | 10 | SYS_RESETN | | |
| GPIO_30 | 11 | 12 | GPIO_60 | | |
| GPIO_31 | 13 | 14 | GPIO_40 | | |
| GPIO_48 | 15 | 16 | GPIO_51 | | |
| GPIO_4 | 17 | 18 | GPIO_5 | | |
| I2C2_SCL | 19 | 20 | I2C2_SDA | | |
| GPIO_3 | 21 | 22 | GPIO_2 | | |
| GPIO_49 | 23 | 24 | GPIO_15 | | |
| GPIO_117 | 25 | 26 | GPIO_14 | | |
| GPIO_125 | 27 | 28 | GPIO_123 | | |
| GPIO_121 | 29 | 30 | GPIO_122 | | |
| GPIO_120 | 31 | 32 | VDD_ADC | | |
| AIN4 | 33 | 34 | GNDA_ADC | | |
| AIN6 | 35 | 36 | AIN5 | | |
| AIN2 | 37 | 38 | AIN3 | | |
| AIN0 | 39 | 40 | AIN1 | | |
| GPIO_20 | 41 | 42 | GPIO_7 | | |
| DGND | 43 | 44 | DGND | | |
| DGND | 45 | 46 | DGND | | |

Figura 7.2: Esquema de entradas analógica de una BeagleBone Black

En el esquema de la figura 7.3 podemos observar las conexiones que se han realizado entre la placa y el sensor. El sensor es alimentado a 5V a partir del convertidor DC-DC y puesto a tierra común con el controlador. La salida del sensor se aplicará sobre el divisor resistivo, siendo la salida del divisor resistivo la entrada a la BeagleBone Black. Se ha colocado una resistencia de $33k\Omega$ en R_a y una resistencia de $66k\Omega$ en R_b .

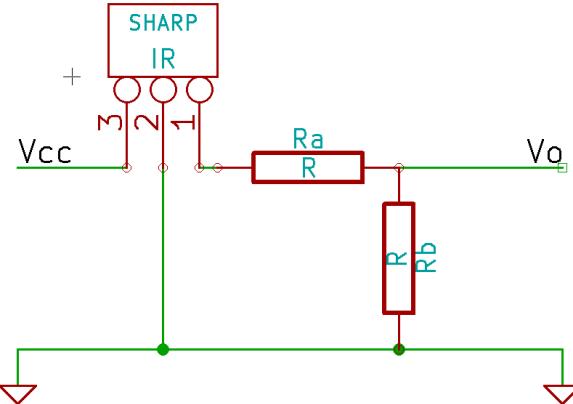


Figura 7.3: Circuito de adecuación de un sensor infrarrojo

7.1.2. Brújula y sensor inercial

Tanto la brújula CMPS03 como el sensor inercia MPU9150 han sido conectados por I2C, ambos en el mismo bus. La BeagleBone Black cuenta con dos buses I2C independientes, tal y como se observa en la figura 7.4.

| P9 | | | P8 | | |
|----------|----|----|------------|----|----|
| DGND | 1 | 2 | DGND | 1 | 2 |
| VDD_3V3 | 3 | 4 | VDD_3V3 | 3 | 4 |
| VDD_5V | 5 | 6 | VDD_5V | 5 | 6 |
| SYS_5V | 7 | 8 | SYS_5V | 7 | 8 |
| PWR_BUT | 9 | 10 | SYS_RESETN | 9 | 10 |
| GPIO_30 | 11 | 12 | GPIO_60 | 11 | 12 |
| GPIO_31 | 13 | 14 | GPIO_40 | 13 | 14 |
| GPIO_48 | 15 | 16 | GPIO_51 | 15 | 16 |
| I2C1_SCL | 17 | 18 | I2C1_SDA | 17 | 18 |
| I2C2_SCL | 19 | 20 | I2C2_SDA | 19 | 20 |
| I2C2_SCL | 21 | 22 | I2C2_SDA | 21 | 22 |
| GPIO_49 | 23 | 24 | I2C1_SCL | 23 | 24 |
| GPIO_117 | 25 | 26 | I2C1_SDA | 25 | 26 |
| GPIO_125 | 27 | 28 | GPIO_123 | 27 | 28 |
| GPIO_121 | 29 | 30 | GPIO_122 | 29 | 30 |
| GPIO_120 | 31 | 32 | VDD_ADC | 31 | 32 |
| AIN4 | 33 | 34 | GNDA_ADC | 33 | 34 |
| AIN6 | 35 | 36 | AIN5 | 35 | 36 |
| AIN2 | 37 | 38 | AIN3 | 37 | 38 |
| AIN0 | 39 | 40 | AIN1 | 39 | 40 |
| GPIO_20 | 41 | 42 | GPIO_7 | 41 | 42 |
| DGND | 43 | 44 | DGND | 43 | 44 |
| DGND | 45 | 46 | DGND | 45 | 46 |

Figura 7.4: Esquema de puertos I2C de una BeagleBone Black

De este modo el sistema ha quedado diseñado tal y como se muestra en la figura 7.5. Cabe destacar que los dos sensores se han alimentado a 3.3V, ya que el MPU9150 los requiere y el CMPS03, aunque

se recomienda conectarlo a 5V, funciona perfectamente a 3.3V. Esta tensión se ha conseguido desde el pin P9-3 del controlador.

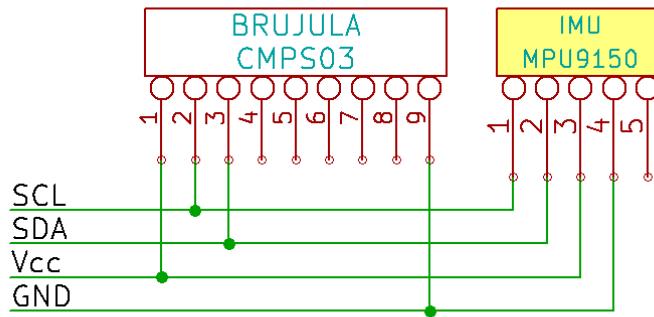


Figura 7.5: Circuito del bus I2C

7.2. Desarrollo de una placa de expansión

Para montar el esquema que se diseñó anteriormente en la figura 4.15 será necesario cablear un gran número de conexiones entre dispositivos. Para ello, se ha optado por diseñar y fabricar una placa de expansión que permita conectar todos los dispositivos de una forma limpia y ordenada. Esta placa servirá para comunicar y alimentar las dos placas controladoras, sensores, actuadores y alimentación de todos ellos. El esquema electrónico puede consultarse en el anexo 11.3. A parte, el en anexo 11.3 se ha incluído una copia de los fotolitos de la placa.

7.3. Montaje

Una vez preparados todo los componentes se ha procedido a su montaje. En los siguientes dos apartados se muestra cómo se ha montado la placa de expansión y cómo se ha integrado la webcam en la cabeza del robot.

7.3.1. Montaje de la placa de expansión

La placa de expansión ha sido fabricada mediante fresado. Tras enviar los planos a su producción se ha obtenido la placa que puede observarse en la figura 7.6.

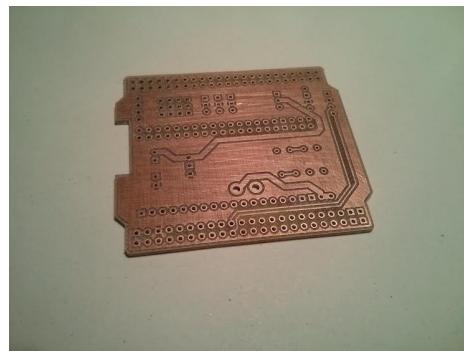


Figura 7.6: Placa de expansión salida de fabricación

Tras esto se ha procedido a soldar todos los componentes que necesita. El listado de los componentes puede consultarse en el anexo TODO. En la imagen de la figura 7.7 puede observarse cómo queda la placa una vez soldada. Además, se le ha realizado un lacado de seguridad con laca aislante. Esto nos ayudará a evitar posibles accidentes por contacto con conectores y partes metálicas del robot. Finalmente

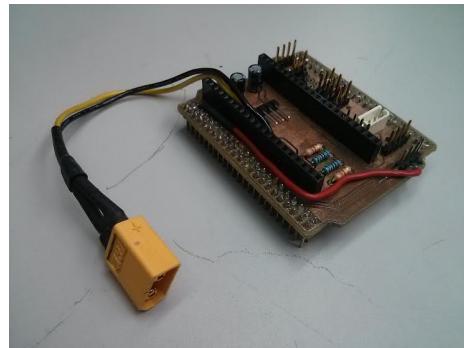


Figura 7.7: Placa de expansión con componentes soldados

En la figura 7.8 puede observarse el resultado final del sistema, montado en el robot y conectado a sus componentes.

To Do

Figura 7.8: Placa de expansión monstada en RAIDER

7.3.2. Integración de la cámara en la cabeza

Para integrar la webcam en la cabeza de RAIDER se ha procedido a desmontar la cámara Microsoft LifeCam con el objetivo de eliminar su carcasa. El resultado de extraer la cámara puede observarse en la figura 7.9.



Figura 7.9: Cámara Microsoft LifeCam desmontada

Ahora, se pasará a montarla en las piezas imprimibles que se diseñaron para tal efecto. Las piezas aparecen en la figura 7.10 junto a la cámara.



Figura 7.10: Integración de la cámara en la cabeza

Una vez montado y acortado el cable, procedemos a montar el servo que moverá la cabeza. Como se muestra en la figura 7.11, este se aloja en el cuello.



Figura 7.11: Cuello y soporte para la cabeza

Finalmente, en la figura 7.12 se muestra el resultado del montaje completo

TODO TODO

Figura 7.12: Resultado tras la integración de la cabeza en el robot

7.3.3. Integración de los sensores infrarrojos en los brazos

Tal y como se comentó en el apartado TODO, se han integrado dos sensores en los brazos de RAIDER. El motivo de montarlos en los brazos es que se esta manera gozarán de mayor libertad a la hora de apuntara obstáculos. Así mismo, se ha buscado que el sensor estuviese en una posición segura, protegido ante caídas. En la figura 7.13 se pueden observar los dos sensores infrarrojos junto a las piezas que se han diseñado para alojarlos.

[FOTO de sharps y piezas]



Figura 7.13: Sensores infrarrojos y soportes

Tras montar las piezas, se ha obtenido el resultado mostrado en la figura 7.14

[FOTO del conjunto]



Figura 7.14: Brazos con sensores montados

Capítulo 8

Programación

En este capítulo se presenta el desarrollo de todo el software que ha sido requerido preparar para la puesta en marcha del robot. El objetivo final del apartado es la programación de la librería *raider.h*, que centraliza todas las funcionalidades de RAIDER.

8.1. Configuración de la BeagleBone Black

El controlador principal, la BeagleBone Black, trae de fábrica una instalación de una distribución Ångström basada en Linux. El sistema operativo incluye: entorno gráfico, escritorio, OpenCV 2.2 y otros programas como navegadores o reproductores de archivos multimedia. Esto se debe a que la BeagleBone Black no es una placa que de fábrica esté pensada para su uso en robots, sino que constituye un ordenador completo que puede adquirir la función de centro multimedia, equipo de ofimática o incluso servidor de una red.

8.1.1. Sistema operativo

La distribución Ångström no es una de las más populares entre los sistemas operativos basados en Linux, no cuenta con una comunidad de usuarios tan amplia como Debian o Fedora. Por ello, las actualizaciones no suelen ser muy frecuentes y, en muchos casos, el sistema presenta errores de funcionamiento a la hora de realizar tareas básicas, como la instalación de nuevo software.

Por estas razones se ha decidido instalar un nuevo sistema operativo. Se pretende conseguir una distribución de Linux que solo tenga lo mínimo necesario que requiere el robot para su funcionamiento. Con este objetivo, se instala en la BeagleBone Black una distribución Debian.

8.1.2. Instalación de librerías

Para la realización del proyecto, se ha requerido instalar Git, CMake, Video4Linux2 y las librerías de OpenCV 2.4.8 y ZBar 0.1.

Antes de comenzar con la instalación de software, es recomendable realizar una actualización del sistema ejecutando los siguientes comandos:

```
sudo apt-get update
sudo apt-get upgrade
```

Lo primero que instalaremos será Git, ya que lo necesitaremos para instalar algunas librerías posteriormente. Git se instala ejecutando el siguiente comando:

```
sudo apt-get install git-core
```

Para instalar CMake lo haremos de la siguiente forma:

```
sudo apt-get install cmake
sudo apt-get install cmake-curses-gui
```

Para el control y configuración de la webcam se necesitará el driver Video4Linux. Se instalará como se muestra a continuación:

```
sudo apt-get install v4l-utils
```

Una vez hecho esto, se pasará a instalar las librerías que se utilizarán en el código del robot. La forma mas sencilla de instalar Zbar será hacerlo desde los repositorios de Debian. De este modo, ejecutaremos los siguientes comandos:

```
sudo apt-get install libzbar0
sudo apt-get install libzbar-dev
```

Por último, instalaremos OpenCV. Se comenzará por descargar todas las librerías que requiere OpenCV. Se instalarán una a una de la siguiente forma:

```
sudo apt-get install build-essential
sudo apt-get install libgtk2.0-dev
sudo apt-get install pkg-config
sudo apt-get install python-dev
```

```
sudo apt-get install python-numpy
sudo apt-get install libavcodec-dev
sudo apt-get install libavformat-dev
sudo apt-get install libswscale-dev
```

Hecho esto, se procederá a descargar la última versión estable del código, que en este momento es la 2.4.9.

```
wget http://downloads.sourceforge.net/project/opencvlibrary/opencv-unix/2.4.9/ope
```

Y posteriormente se instalará de este modo:

```
unzip opencv-2.4.9.zip
cd opencv-2.4.9
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local ..
make
sudo make install
```

8.1.3. Configuración de la cámara

A continuación se configurará la webcam en la beaglebone. Para esto se conectará por USB la Microsoft Lifecam a la BeagleBone Black tal y como puede observarse en la figura 8.1. A partir de aquí, se iniciará sesión en la BeagleBone Black y se procederá a configurar la cámara.

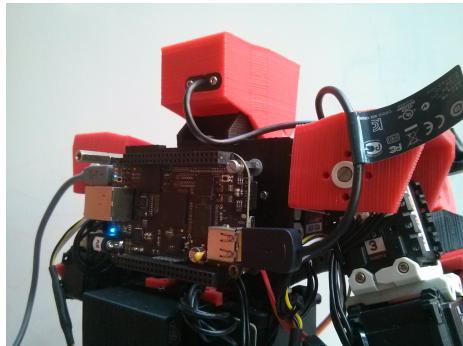


Figura 8.1: Conexión de la cámara a la BeagleBone Black

Lo primero que se hará será comprobar que la cámara se ha reconocido correctamente. Para ello ejecutaremos el comando:

```
lsusb
```

Y deberá proporcionarnos una salida parecida a la que puede observarse en la figura 8.2. La salida indica que la webcam está conectada en el bus 001

```
debian@arm:~$ lsusb
Bus 001 Device 002: ID 045e:075d Microsoft Corp. LifeCam Cinema
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Figura 8.2: Salida del comando lsusb

Para cerciorarnos de que la cámara se ha reconocido correctamente podemos buscarla en el directorio /dev. En la imagen 8.3 se observa el archivo /dev/video0, que está vinculado a la webcam.

```
debian@arm:~$ ls /dev/video0
/dev/video0
```

Figura 8.3: Comprobación de conexión para la webcam

A continuación nos ayudaremos de las utilidades de Video4Linux (una interfaz para la programación de aplicaciones con captura de imágenes para Linux) para la webcam. OpenCV se apoya en Video4Linux, por tanto, lo primero que se hará será comprobar que la cámara es compatible con Video4Linux. Para ello, se ejecutará el siguiente comando:

```
v4l2-ctl --list-devices
```

Tras ejecutarlo se deberá observar una salida similar a la de la figura 8.4

```
debian@arm:~$ v4l2-ctl --list-devices
Microsoft® LifeCam Cinema(TM) (usb-musb-hdrc.1.auto-1):
        /dev/video0
```

Figura 8.4: Comprobación de compatibilidad para la webcam

Ésto nos confirma que la cámara funciona correctamente. Para proseguir con la configuración vamos a ejecutar el comando siguiente:

```
v4l2-ctl --all
```

En la figura 8.5 podemos observar la salida del comando. En ella se pueden observar diferentes parámetros como el formato de video, la resolución o los fotogramas por segundo.

```
debian@arm:~$ v4l2-ctl --all
Driver Info (not using libv4l2):
    Driver name      : uvcvideo
    Card type       : Microsoft® LifeCam Cinema(TM)
    Bus info        : usb-musb-hdrc.1.auto-1
    Driver version: 3.8.13
    Capabilities   : 0x84000001
                      Video Capture
                      Streaming
Format Video Capture:
    Width/Height   : 640/480
    Pixel Format  : 'YUYV'
    Field          : None
    Bytes per Line: 1280
    Size Image    : 768000
    Colorspace     : SRGB
Crop Capability Video Capture:
    Bounds         : Left 0, Top 0, Width 640, Height 480
    Default        : Left 0, Top 0, Width 640, Height 480
    Pixel Aspect: 1/1
Video input : 0 (Camera 1: ok)
Streaming Parameters Video Capture:
    Capabilities   : timeperframe
    Frames per second: 312500.000 (312500/1)
    Read buffers   : 0
Priority: 2
```

Figura 8.5: Parámetros leídos de la cámara

Por último, se bajará la resolución de la cámara a un valor que nos permita tratar las imágenes con mayor rapidez cuando se programen algoritmos de visión con OpenCV desde la BeagleBone Black. Se ha elegido una resolución de 320x240, mas adelante se ajustará este valor empíricamente en función del rendimiento del robot. Para ello ejecutamos este comando de configuración:

```
v4l2-ctl --set-fmt-video=width=320,height=240
```

Tras estos pasos la cámara ya está preparada y lista para usarse.

8.1.4. Configuración de pines

Para finalizar esta sección, se muestra el desarrollo de un script para configurar los pines del controlador.

La Beaglebone Black posee 92 pines configurables de entrada y salida. Entre ellos, 7 pueden funcionar como entradas analógicas y 5 parejas de pines están preparadas como puertos serie, tal y como indica la figura 8.6.

| P9 | | | P8 | | |
|------------|----|----|------------|-------------|----|
| DGND | 1 | 2 | DGND | 1 | 2 |
| VDD_3V3 | 3 | 4 | VDD_3V3 | 3 | 4 |
| VDD_5V | 5 | 6 | VDD_5V | 5 | 6 |
| SYS_5V | 7 | 8 | SYS_5V | 7 | 8 |
| PWR_BUT | 9 | 10 | SYS_RESETN | 9 | 10 |
| UART4_RXD | 11 | 12 | GPIO_60 | GPIO_38 | 3 |
| UART4_TXD | 13 | 14 | GPIO_40 | GPIO_34 | 5 |
| GPIO_48 | 15 | 16 | GPIO_51 | GPIO_66 | 7 |
| GPIO_4 | 17 | 18 | GPIO_5 | GPIO_69 | 9 |
| UART1_RTSN | 19 | 20 | UART1_CTSN | GPIO_45 | 11 |
| UART2_TXD | 21 | 22 | UART2_RXD | GPIO_23 | 13 |
| GPIO_49 | 23 | 24 | UART1_TXD | GPIO_47 | 15 |
| GPIO_117 | 25 | 26 | UART1_RXD | GPIO_27 | 17 |
| GPIO_125 | 27 | 28 | GPIO_123 | GPIO_22 | 19 |
| GPIO_121 | 29 | 30 | GPIO_122 | GPIO_62 | 21 |
| GPIO_120 | 31 | 32 | VDD_ADC | GPIO_36 | 23 |
| AIN4 | 33 | 34 | GNDA_ADC | GPIO_32 | 25 |
| AIN6 | 35 | 36 | AIN5 | GPIO_86 | 27 |
| AIN2 | 37 | 38 | AIN3 | GPIO_87 | 29 |
| AIN0 | 39 | 40 | AIN1 | UART5_CTSN+ | 31 |
| GPIO_20 | 41 | 42 | UART3_TXD | UART4_RTSN | 33 |
| DGND | 43 | 44 | DGND | UART4_CTSN | 35 |
| DGND | 45 | 46 | DGND | UARR5_RXD+ | 37 |

Figura 8.6: Puertos serie en una BeagleBone Black

Para configurar los pines de la placa se utilizarán overlays (capas). Un overlay es un archivo que indica al sistema operativo cómo debe configurarse un pin. La BeagleBone Black se configura mediante Device Tree Overlays, un mecanismo de descripción del hardware que forma parte de un sistema. Para configurar las entradas analógicas necesarias para los sensores infrarrojos, y el puerto serie que comunicará la placa con el controlador de locomoción; se utilizarán los archivos *.dtbo* que pueden encontrarse en */lib/firmware* y aparecen en la figura 8.7.

```
debian@arm:/lib/firmware$ ls *.dtbo
am33xx_pwm-00A0.dtbo          BB-BONE-PRU-01-00A0.dtbo      BB-UART1-00A0.dtbo
BB-ADC-00A0.dtbo               BB-BONE-PRU-02-00A0.dtbo      BB-UART2-00A0.dtbo
BB-BONE-AUDI-01-00A0.dtbo      BB-BONE-PRU-03-00A0.dtbo      BB-UART2-RTSCTS-00A0.dtbo
BB-BONE-BACON-00A0.dtbo        BB-BONE-PRU-04-00A0.dtbo      BB-UART4-00A0.dtbo
BB-BONE-BACONE-00A0.dtbo       BB-BONE-PWMT-00A0.dtbo       BB-UART4-RTSCTS-00A0.dtbo
BB-BONE-BACONE2-00A0.dtbo      BB-BONE-RS232-00A0.dtbo      BB-UART5-00A0.dtbo
BB-BONE-CAM3-01-00A2.dtbo      BB-BONE-RST-00A0.dtbo       bone_pwm_P8_13-00A0.dtbo
BB-BONE-CAM-VVDN-00A0.dtbo     BB-BONE-RST2-00A0.dtbo      bone_pwm_P8_19-00A0.dtbo
BB-BONE-eMMC1-01-00A0.dtbo     BB-BONE-RTC-00A0.dtbo       bone_pwm_P8_34-00A0.dtbo
BB-BONE-GPEVT-00A0.dtbo       BB-BONE-SERL-01-00A1.dtbo      bone_pwm_P8_36-00A0.dtbo
BB-BONE-GPS-00A0.dtbo         BB-BONE-SERL-03-00A1.dtbo      bone_pwm_P8_45-00A0.dtbo
BB-BONE-LCD4-01-00A0.dtbo     BB-GPIOHELP-00A0.dtbo       bone_pwm_P8_46-00A0.dtbo
BB-BONE-LCD4-01-00A1.dtbo     BB-I2C1-00A0.dtbo        bone_pwm_P9_14-00A0.dtbo
BB-BONE-LCD7-01-00A2.dtbo     BB-I2C1A1-00A0.dtbo      bone_pwm_P9_16-00A0.dtbo
BB-BONE-LCD7-01-00A3.dtbo     BB-SPIDEV0-00A0.dtbo       bone_pwm_P9_21-00A0.dtbo
BB-BONE-LCD7-01-00A4.dtbo     BB-SPIDEV1-00A0.dtbo       bone_pwm_P9_22-00A0.dtbo
BB-BONELT-BT-00A0.dtbo        BB-SPIDEV1A1-00A0.dtbo      bone_pwm_P9_28-00A0.dtbo
```

Figura 8.7: Algunas capas de Device Tree Overlays

En concreto se utilizarán dos capas, la que activará el puerto serie

número 2: *ttyO2_armhf.com* y esta otra que nos permitirá activar las entradas analógicas: *cape-bone-iio*. Se habilitarán como se muestra en la figura 8.8.

```
debian@arm:~$ sudo su
root@arm:/home/debian# echo tty02_armhf.com > /sys/devices/bone_capemgr.9/slots
root@arm:/home/debian# echo cape-bone-iio > /sys/devices/bone_capemgr.9/slots
```

Figura 8.8: Habilitación de capas

Puede realizarse una comprobación simple de que los pines se han configurado correctamente realizando una lectura analógica. Para ello, se leerá el archivo */sys/bus/platform/drivers/bone-iio-helper/helper.15/AIN4*. En la figura 8.9 se ha realizado una lectura del pin 4, el valor de 570 indica que se está realizando una lectura de $570mV$.

```
debian@arm:~$ cat /sys/bus/platform/drivers/bone-iio-helper/helper.15/AIN4
570
```

Figura 8.9: Lectura analógica del pin 4

Para realizar estas medidas desde el programa principal escrito en C++, se ha programado la librería *bone.h*, que simplifica el uso de estos pines.

8.2. Sistema de locomoción TODO

Antes de comenzar con esta sección, es importante señalar algunos detalles importantes. Si bien es cierto, el control del robot girará en torno a un algoritmo de visión, su funcionamiento de apoyará en un control de locomoción robusto que permita al robot realizar desplazamientos seguros sobre el terreno. Por tanto, dado que se trata de un robot bípedo, la programación de movimientos no es un tema trivial como lo podría ser en un robot con ruedas.

Para conseguir que Raider pueda moverse con soltura se han seguido varios pasos, cada cual de un nivel superior al anterior.

8.2.1. Movimiento del servo PWM

El control del servo PWM de Raider, situado en su cabeza, se realiza con la biblioteca Servo.h orginalmente desarrolladas para Arduino y que posteriormente ha sido portada para su utilización en OpenCM. Dado el cometido de este servo, no será necesario realizar un control de su velocidad, por lo que un simple control de su posición será suficiente.

La biblioteca Servo.h nos permite controlar el movimiento de un servo a partir de un valor de posición, y se encarga de producir la señal PWM correspondiente. Aunque los métodos utilizados pueden encontrarse en la documentación de Arduino (TODO), a continuación introduzco una breve explicación de los que han sido utilizados en este proyecto:

Servo::attach(uint8 pin)

Con esta función configuramos un pin de la OpenCM para que actúe como emisora de señales PWM. A partir de esta inicialización ya puede moverse el servo.

Servo::writeMicroseconds(uint16 pulseWidth)

Para mover el servo se utiliza la función writeMicroseconds, cuyo parámetro define la amplitud del pulso de la onda PWM. Si bien es cierto, existe otra función paralela llamada write(int angle) que directamente utiliza como parámetro la posición en grados, se ha utilizado writeMicroseconds por su mayor precisión. Acepta valores de entre 1000 y 2000 microsegundos, pero para mantener un formato constante con el resto de articulaciones, se ha realizado una conversión matemática a un rango de 0 a 1023.

8.2.2. Movimiento de los actuadores Dynamixel

Para comunicarse con los actuadores Dynamixel (de los que hablamos en TODO) debemos utilizar su protocolo particular. Los servos son controlados mediante el envío de paquetes de datos binarios. Existen dos tipos de paquetes en el protocolo: Los paquetes de instrucciones, que son los que envía el controlador a los servos; y los paquetes de estado, que son los que los servos envían al controlador.

Cada servo tiene una ID, o dicho de otra forma, un número de identidad propio e irrepetible que identifica a un servo particular dentro del bus. La comunicación en el bus se realiza mediante el intercambio de paquetes de instrucciones y estados con una ID concreta. Por esta razón, en un mismo bus no deben existir servos con la misma ID, ya que provocarán colisiones entre los paquetes e impedirán el correcto funcionamiento del sistema. Sin embargo, estas ID son fácilmente reprogramables y pueden modificarse realizando una escritura sobre el registro 3 (0X03).

El protocolo de comunicación utilizado es una comunicación serie asíncrona de 8 bits, con 1 bit de Stop y sin paridad. La conexión, de

tipo Half Duplex, no permite la transmisión y recepción de paquetes de forma simultánea. Esto la convierte en una conexión bastante típica en los sistemas que utilizan un solo bus de comunicación. Como en el mismo bus existe más de un dispositivo, todos deben permanecer en modo de escucha salvo el que esté transmitiendo en ese instante. El controlador principal, la placa OpenCM 9.04, asigna la dirección del bus en modo escucha, y solo cambia la dirección del bus a modo de envío mientras manda un paquete. Los Dynamixel AX-12A poseen una tabla de registros sobre la cual podemos modificar varios parámetros referente a su estado y su funcionamiento. La tabla de registros puede consultarse en el manual[?].

Para realizar una rotación simple en un servomotor, sería suficiente con escribir en el registro 32 (Goal Position) un valor comprendido entre 0 y 1023, y el servo se situará inmediatamente en esa posición. Sin embargo, existen otros parámetros interesantes en el mapa de registros que conviene controlar, como la posición instantánea, la velocidad de giro, el consumo eléctrico o incluso la temperatura del dispositivo.

Para mover los 19 AX-12A de Raider se utilizan los parámetros de posición objetivo (Goal Position) y velocidad de giro (Moving Speed) de forma combinada. Dado que está programación se ha realizado desde la OpenCM 9.04 se ha utilizado la biblioteca Dynamixel.h, que funciona como una macro para leer y escribir en los registros de forma sencilla y eficiente. Dentro de la biblioteca utilizaremos la función writeWord con la siguiente sintaxis:

```
Dxl.writeWord(
    Dynamixel_Motor_Number,
    Address_Number,
    Address_Data
);
```

A modo de ejemplo, para asignar una velocidad de $3,5\text{rad/s}$ al servo con la ID 5, primero calcularíamos el valor correspondiente para una resolución de 10 bits. Según el manual de los servos Dynamixel (TODO citar) AX-12A, la velocidad máxima de estos servomotores es de 114rpm . Por tanto, se realizaría la siguiente conversión:

$$3,5 \cdot \frac{\text{rad}}{\text{s}} \cdot \frac{60\text{s}}{2\pi\text{rad}} \cdot \text{rpm} \cdot \frac{1024}{114\text{rpm}} = 300,216 \simeq 300$$

Dentro del código, utilizaremos la función writeWord para asignar este valor en el registro 32 (Moving Speed):

```
Dxl.writeWord(5,32,300);
```

Seguidamente, asignaríamos al servo una posición final siguiendo el criterio de la figura 8.10

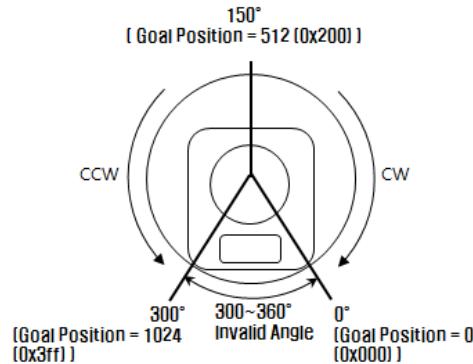


Figura 8.10: Amplitud de giro de un AX-12A

Continuando con el ejemplo, calcularemos el valor que debemos darle al registro para mover el servo a una posición de 120° , teniendo en cuenta que las especificaciones nos indican una amplitud de giro total de 300° reales con una resolución de 10 bits. De esta forma, haríamos la siguiente conversión:

$$120^\circ \cdot \frac{1024}{300^\circ} = 409,6 \simeq 410$$

En nuestro programa escribiríamos en el registro 30 (Goal Position) de la siguiente forma:

```
Dxl.writeWord(5,30,410);
```

Como resumen, con estos pasos hemos conseguido mover el servo con la ID 5 (Que corresponde al codo del brazo izquierdo de Raider), a una posición de 120° con una velocidad de $3,5 rad/s$

8.2.3. Movimiento sincronizado de las articulaciones

En el apartado anterior se ha mostrado cómo se realiza el movimiento de un servo, sin embargo, para mover el cuerpo del robot necesitaremos mover todos al mismo tiempo de una forma sincronizada. Si en el apartado anterior utilizábamos la posición objetivo y la velocidad de movimiento como parámetros, en este punto, por comodidad a la hora de programar, utilizaremos como parámetros

la posición objetivo de los 20 servos, su posición actual y el tiempo total durante el que se realizará su movimiento entre ambos puntos.

Este es quizás uno de los apartados mas críticos a la hora de diseñar las funciones que moverán el robot. Se pretende programar una biblioteca que permita mover 20 servos simultáneamente, con velocidades diferentes condicionadas por un tiempo de ejecución común. De esta forma, los servos cuya posición objetivo sea lejana a su posición actual se moverán con una velocidad mayor que la de los servos cuya posición objetivo sea cercana a su posición actual.

Las funciones pueden encontrarse en la biblioteca *motion.h*.

class Robot

La clase Robot abarca todas las funciones relativas al movimiento de Raider (aunque por el momento en esta sección solo se presenta algunas de ellas), y abstrae su controlador principal, la BeagleBone Black de la parte de locomoción. Dentro de la clase, encontramos tres variables miembros importantes:

- **int currentPosition[20].**

currentPosition es un array de 20 posiciones destinado a almacenar los valores de la posición actual de las 20 articulaciones del robot con valores comprendidos entre 0 y 1023. El primer valor es el servo AX-12A, el segundo es el servo PWM de la cabeza y a partir de ese punto están el resto de AX-12A ordenados según su ID, del número 1 al 18.

- **int targetPostion[20].**

targetPosition sigue la misma estructura de currentPosition, con la diferencia de que en este caso los valores guardados en el array corresponderán con la posición objetivo o posición final de las articulaciones.

- **int TRIM[20].**

Por último, TRIM es un array de trims. Un trims es una variable de ajuste para calibrar la posición de los servos. Tanto los servos Dynamixel como los servos PWM suelen tener un pequeño error en su posición cero. Los trimmers constituyen un offset aplicado individualmente a cada servo en absolutamente todos los movimientos que se realizarán durante el programa. Las holguras y otros factores pueden provocar el desajuste de estos valores, por lo que es necesario volver a calibrarlo cada

cierto tiempo. Una mala calibración de los trims puede radicar en problemas de asimetrías en movimientos, y por tanto, resultados inesperados.

Robot::Robot()

El constructor de la clase Robot tiene como función la apertura del bus de control para los servos AX-12A, la configuración del servo PWM y la asignación de trims en el array de trims.

void Robot::setTargetPosition(int,int,... int)

setTargetPosition accede directamente al miembro privado targetPosition[20] para asignarle nuevos valores.

void Robot::setTargetOffset(int,int,... int)

setTargetOffset varía los valores del miembro privado targetPosition[20] para sumarles un valor. La función permite variar una posición con un giro determinado sin necesidad de conocer la posición actual de la articulación.

void Robot::updateCurrentPosition()

Esta función tiene un funcionamiento sencillo, se ocupa de volcar los datos de la posición objetivo en la posición actual. Es la forma que tiene el robot de actualizar su posición actual tras un movimiento.

void Robot::move(float)

La función move es la mas importante de todas, ya que es la función que se encarga de mover las articulaciones. A esta función se le pasa un valor de tiempo expresado en segundos, y tal y como se comentó al principio de este apartado, será el tiempo en el que los servos pasarán de la posición actual (currentPosition[20]) a la posición final (targetPosition[20]).

Para ello, la función calcula la amplitud del movimiento y asigna una velocidad independiente para ese servo. Gracias a esto, todos los servos empiezan y terminan de moverse al mismo tiempo y permiten un control mas sencillo de las inercias entre movimientos consecutivos.

8.2.4. Funciones de movimientos combinados TODO

Llegado este punto se ha abordado como mover un servo y como mover los 20 servos de forma coordinada. En este apartado se presentan algunas funciones intermedias entre lo comentado y movimientos de alto nivel, como puede ser el desplazamiento bípedo.

Para facilitar la programación de movimientos mas complejos se han programado una serie de utilidades que permiten mover los servos en pequeños grupos que desempeñan una función común. Estas funciones modifican los valores del array de posiciones finales, targetPosition[20], lo que quiere decir que para efectuar el movimiento será necesario realizar una llamada a la función move(float). Por tanto, es posible la utilización de varias funciones en un mismo movimiento, dando la posibilidad de sumar sus modificaciones y superponer su utilidades.

void movHead(int)

movHead es una función que permite mover el servo PWM de la cabeza del robot. Sirve para mover la cámara independientemente de la posición instantánea del robot.

void movVertical(int,int) TODO

Meter fotos del robot con una pierna levantada

void movLateral(int,int) TODO

void movFrontal(int,int) TODO

8.2.5. Creación de movimientos completos

Encontrándonos en este punto, la programación de desplazamientos, giros y otros movimientos complejos, se ha realizado mediante la combinación de las funciones anteriormente descritas. A cada movimiento se le asignará un valor hexadecimal en forma de carácter, de forma que los caracteres comunicados a la OpenCM sirvan como identificador de el movimiento que el controlador de visión está ordenando.

Se ha creado la función void controller(char), dentro de la clase Robot, con el objetivo de indexar todos los movimientos que realizará Raider. En el cuadro 8.1 se presenta una tabla con los movimientos programados y su comando asignado.

| Comando | Función | Descripción |
|---------|--------------|---|
| W | walk(3) | Caminar 3 pasos cortos |
| A | turnL() | Rotación a la izquierda |
| D | turnR() | Rotación a la derecha |
| S | run(3) | Caminar 3 pasos largos y rápidos |
| Q | stepL() | Paso lateral a la izquierda |
| E | stepR() | Paso lateral a la derecha |
| K | kick() | Patada (para golpear una pelota) |
| Y | yes() | Movimiento intermitente de la cabeza |
| G | getUp() | Levantamiento desde una caída frontal |
| R | roll() | Rodar, se utiliza cuando el robot cae de espaldas |
| H | hello() | Saludo |
| q | miniTurnL() | Giro leve hacia la izquierda |
| e | miniTurnR() | Giro leve hacia la derecha |
| Z | punchL() | Puñetazo con el brazo izquierdo |
| B | punchR() | Puñetazo con el brazo derecho |
| C | crab() | Ataque de sumo con dos brazos |
| V | miniPunchR() | Puñetazo leve derecho |
| X | miniPunchL() | Puñetazo leve izquierdo |
| w | defense() | Posición defensiva |
| a | lookL() | Mirar hacia la izquierda |
| d | lookR() | Mirar hacia la derecha |
| L | look() | Mirar de frente |
| l | lookUp() | Mirar hacia arriba |
| f | endLookUp() | Volver a la posición de reposo tras lookUp() |

Cuadro 8.1: Movimientos programados

8.3. Comunicación serie

En la sección anterior hemos completado la programación de movimientos sobre la placa OpenCM, sin embargo, el control principal del robot se realiza desde la BeagleBone. En este apartado se comunicará la BeagleBone con la OpenCM de forma que adopten una configuración de maestro y esclavo. La estrategia consistirá en el envío de comandos hexadecimales desde la BeagleBone a la OpenCM. Cada comando servirá de identificador para un movimiento completo. Los comandos serán los descritos en el cuadro 8.1.

8.3.1. Comunicación serie en OpenCM

La OpenCM posee 3 puertos serie, de los cuales uno de ellos está reservado para el control del bus Dynamixel. En el esquema de la figura 8.11 se presenta la configuración de puertos serie de la OpenCM.

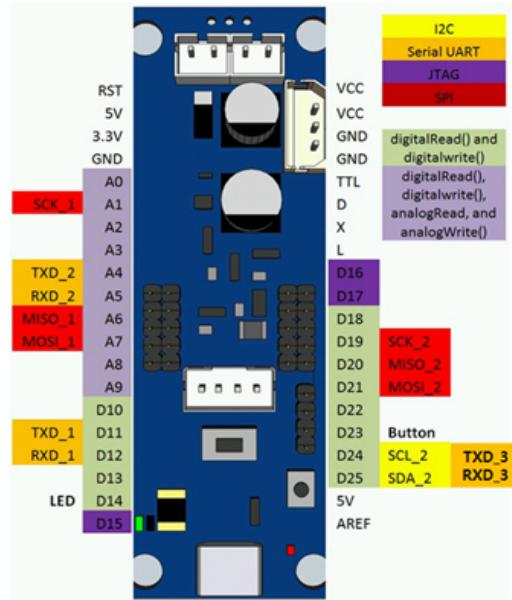


Figura 8.11: Esquema de puertos serie en una OpenCM 9.04

Para la programación de la comunicación serie en la placa OpenCM se ha utilizado la biblioteca `HardwareSerial.h`, contenida en CM9 IDE. A continuación se listan las funciones que se han utilizado junto a una breve explicación de su funcionamiento.

```
void HardwareSerial::begin(unsigned int baud)
```

Esta función habilita los pinos como puerto serie. Como parámetro se le pasará la velocidad en baudios, que deberá coincidir con la del controlador maestro. En este caso, se ha configurado el puerto 3 con una velocidad de 9600 baudios

```
unsigned int HardwareSerial::available()
```

El objetivo de esta función es consultar si existe algún mensaje en el buffer de lectura. En caso afirmativo, podremos proceder a leer el mensaje.

```
unsigned char HardwareSerial::read()
```

La lectura de mensajes nos devolverá un valor hexadecimal. En nuestro caso, será el emitido por la BeagleBone Black y corresponderá a la ejecución de un movimiento.

void HardwareSerial::flush()

Con esta función vaciaremos el buffer de entrada. Se ejecutará cada vez que leamos un comando.

8.3.2. Comunicación serie en BeagleBone

Para poner en marcha la comunicación serie en la BeagleBone se ha utilizado como base la librería open-source Serialib [15]. Esta librería nos permite administrar un puerto serie, y está preparada para funcionar tanto en Windows como en Linux.

Se han realizado algunas modificaciones leves para ajustar su funcionamiento al requerido. Los métodos que se han utilizado se detallan a continuación.

open TODO nombre

Esta función abrirá el puerto serie. En el primer parámetro se le pasará la cadena “/dev/ttyO2”, es decir, la dirección de nuestro puerto serie. Como segundo parámetro se le pasará la velocidad del puerto, que coincidiendo con la de la OpenCM será de 9600 baudios.

writechar TODO nombre

Para mandar los comandos utilizaremos esta función. Ya que el objeto de la clase conserva las características del puerto serie, solo será necesario pasarle el valor hexadecimal que queramos transmitir al controlador de locomoción.

8.3.3. Comunicación con módulo Bluetooth

Adicionalmente, se ha incluido la posibilidad de utilizar un control auxiliar por Bluetooth desde un dispositivo externo. Para ello se ha conectado un módulo Bluetooth directamente a la OpenCM. La función de esta otra vía de control no es otra que la realización de pruebas experimentales controladas, ya que nos permite modificar el comportamiento del robot en los momentos en los que sea necesario. Por supuesto, de cara a su funcionamiento autónomo, el módulo Bluetooth se utilizará. Sin embargo, será necesario dejar preparada su conexión y programación.

Se ha utilizado un módulo Bluetooth hc05 conectado al puerto 3 del controlador de locomoción. Con esto, podremos mandar mensajes desde un teléfono móvil, un ordenador portátil o cualquier otro

dispositivo que soporte conexión Bluetooth. Para utilizar el control del robot por Bluetooth se sustituirá el puerto de lectura de la OpenCM reservado al controlador de visión (el puerto 2) por el puerto 3. En la imagen 8.12 puede apreciarse cómo se controla a RAIDER desde un teléfono móvil.



Figura 8.12: Teleoperación de RAIDER con módulo Bluetooth

8.4. Programación de sensores

En esta sección se detallará la programación que se ha aplicado a los diferentes sensores que se han montado en RAIDER.

8.4.1. Infrarrojos

Para controlar los sensores infrarrojos se ha desarrollado un algoritmo de detección, implementado en la librería *raider.h*. Dado el carácter de este sensor, se ha realizado una programación para detectar objetos en función de dos parámetros de distancia crítica. Como puede observarse en la figura 8.13 con estos dos parámetros (representados por barras verticales) se han definido tres zonas.

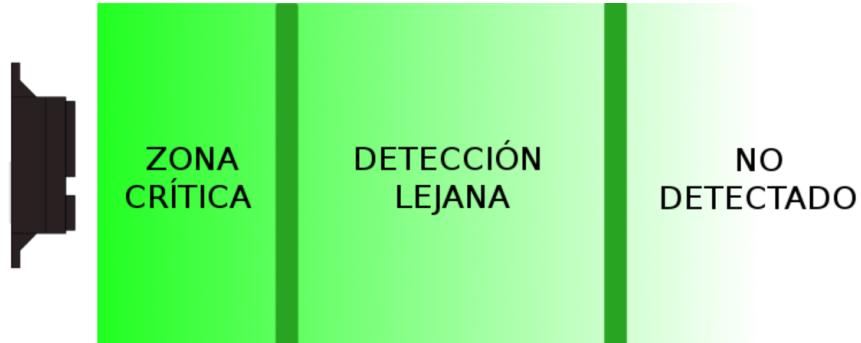


Figura 8.13: Rangos de detección para un sensor Sharp

El rango de datos que podemos extraer de la entrada analógica va de 0 a 1800mV. Teniendo esto en cuenta, se han definido experimentalmente *INFRARED_LOW* en 300mV, e *INFRARED_HIGH* en 1000mV. Estos valores corresponden aproximadamente a 12cm y 40cm respectivamente. Se han desarrollado dos funciones paralelas para medir los sensores infrarrojos izquierdo y derecho de Raider: *getLeftIR()* y *getRightIR()*. Estas funciones devolverán un valor identificativo de la zona en la que se encuentra el obstáculo.

8.4.2. IMU TODO

La programación del sensor inercial pasa por dos fases: La extracción de datos del sensor y el procesamiento de los mismos para obtener información útil.

El MPU9150 utiliza el protocolo I2C para comunicarse con el controlador. Ofrece medidas de giroscopio, acelerómetro y magnetómetro. En este proyecto se ha utilizado solamente el acelerómetro para detectar caídas. No obstante, se han programado funciones para la lectura de todos los parámetros con el objetivo de utilizarlos en futuros desarrollos. Para utilizar el bus I2C de la BeagleBone Black se ha desarrollado la librería *i2c.h*, que funciona como interfaz de la librería nativa *i2c-dev*. Para programar este sensor habrá que apoyarse en el mapa de registros [9].

El primer paso para utilizar este sensor es desactivar el modo de hibernación. Esto se consigue poniendo a 0 el registro 6B. Una vez hecho esto ya pueden leerse de forma normal los datos del giroscopio y del acelerómetro. En la librería *imu.h* pueden consultarse los registros que se están leyendo.

Para la detección de caídas, se ha utilizado el eje Y del acelerómetro. El código programado se encuentra en el método *getFall* de la

librería *raider.h*. Se sabe que en una posición vertical, el acelerómetro estará realizando una lectura de 1g provocada por la atracción gravitatoria terrestre. Al girar el robot en este eje, el valor medido irá disminuyendo. Cuando el robot se encuentre tumbado de forma perfectamente horizontal, el valor medido en el eje Y será de 0g. Dado que el incremento es proporcional se ha realizado una conversión sencilla en la que 1g corresponde con 0° de inclinación y 0g con 90°. Esta medida debe tomarse con el robot en reposo ya que de lo contrario el movimiento del robot influirá en su aceleración y podrá ser causa de falsos positivos. Se ha programado el sensor de forma que detecte caídas hacia delante cuando el robot se encuentre en una medición entre 90° y 60°, y caídas hacia atrás cuando mida entre 270° y 300° de inclinación. Este rango está parametrizado con la constante *FALL_DEGREES* que se encuentra en *raider.h*

[TODO image de caidas]

8.4.3. Brújula

Para la programación de la brújula se ha programado una biblioteca, *compass.h* que controla de forma sencilla la extracción de datos. Atendiendo a su manual [11], la CMPS03 ofrece tres formas de leer su medida:

- **Mediante PWM**

El pin 4 del sensor permite realizar una lectura con 8 bits de precisión del valor medido por la brújula.

- **Mediante I2C con 1 Byte**

Conectada por I2C, podemos realizar una lectura del registro 01 para obtener la medida de la brújula en un rango de 8 bits.

- **Mediante I2C con 2 Bytes**

Conectada por I2C, los registros 02 y 03 nos ofrecen una lectura de la medida de la brújula en un rango de 0 a 3600. De esta manera, la medida extraída tiene una precisión de 0,1° y no necesita ninguna adecuación.

Se eligió conectarla por I2C por la simplicidad en su cableado. Así mismo, se ha usado el método de lectura de dos registros por su mayor precisión. Con esto, será posible tomar una medida de la orientación del robot con solo realizar una consulta al sensor.

8.5. Algoritmos de visión

Llegados a este punto, ya pueden programarse algoritmos de visión en el robot. A continuación se mostrarán tres algoritmos inspirados en la competición CEABOT.

8.5.1. Análisis de trayectorias en navegación

El primer algoritmo que se ha programado es la búsqueda de trayectorias para la prueba de navegación. Este código puede consultarse en el método *findWay()* de *raider.h*.

El primer paso será tomar una foto del campo como la que aparece en la figura 8.14. Como se puede observar existe bastante suciedad en el suelo, hay marcas entre las tablas y algunos de los obstáculos provocan sombras.



Figura 8.14: findway

Tomando esta imagen como punto de partida, lo primero que se ha hecho es analizar cual es la superficie navegable. El código de este procedimiento se encuentra en la función *detectGreen()* de *eye.h*. En el sistema RGB cada pixel de una imagen describe su color en base a tres componentes: rojo, verde y azul. El objetivo será separar la imagen de tres canales en tres imágenes de un solo canal. En la imagen de la figura 8.15 se muestran las tres imágenes fruto de la descomposición de la imagen original (De izquierda a derecha: Canal rojo, canal verde y canal azul).



Figura 8.15: Descomposición de la imagen en sus tres canales RGB

La primera conclusión que puede sacarse al ver estas tres imágenes es que en el canal verde apenas existe contraste entre el color verde del suelo y el color blanco de los obstáculos. Esto tiene sentido ya que atendiendo a la teoría, un color verde puro (en RGB (0,255,0)) y un color blanco puro (en RGB (255,255,255)) contienen la misma cantidad de verde (255). En las otras dos imágenes, sí que se observa un mayor contraste. Si el color verde fuese puro, su componente azul y su componente roja serían nulas. Sin embargo, como se trabaja en un entorno real, los canales azul y rojo no muestran un color totalmente negro en el suelo.

El campo del campeonato CEABOT por normativa es de color verde, sin embargo no se especifica si la superficie será mate o brillante, si la iluminación estará bien enfocada... etc. Por ello, aunque a priori en el campo de pruebas de la Asociación de Robótica parezca que el canal que mejor diferencia el suelo y los obstáculos sea el canal rojo, en otros campos podremos encontrarnos que funciona mejor el canal azul.

Para solucionar esto, se ha realizado una mezcla de canales que aúne las características que nos conviene conservar de la muestra. La operación que se realiza es la siguiente. Por una parte, tomando como base la imagen del canal verde, le restaremos la imagen del canal rojo. Esto eliminará la componente de rojo residual que tiene la imagen del canal verde. Paralelamente, realizaremos la misma operación tomando como base la imagen del canal verde y restándole la imagen del canal azul para eliminar el azul residual. Las dos imágenes producto de ambas restas se muestran en la figura 8.16

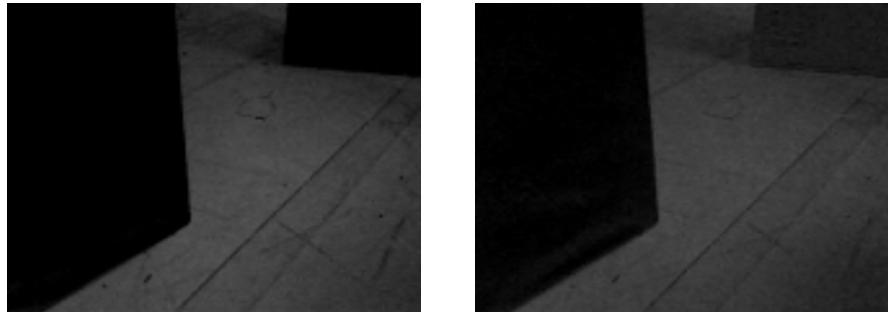


Figura 8.16: Resta de canales verde y rojo, y de canales verde y azul

Con esto, se han obtenido dos imágenes que conservan el color verde y carecen del resto. Para terminar, ambas imágenes se suman y el resultado obtenido, mostrado en la figura 8.17, supone la máxima detección de superficie navegable de la imagen original.



Figura 8.17: Detección de superficie navegable

Ahora, la imagen podrá umbralizarse con bastante seguridad, ya que existe bastante diferencia entre la zona blanca y la negra. En la imagen 8.18 se muestra cómo queda la imagen 8.17 tras aplicarle un desenfoque para suavizar la textura y una umbralización. Esta imagen supone la salida de la función *detectGreen()*.



Figura 8.18: Umbralización de obstáculos y superficie navegable

Llegado este punto, ya se ha obtenido la información de qué elementos de la imagen constituyen obstáculos (zonas negras) y qué elemento constituye el suelo (zona blanca). A priori, se sabe que el robot podrá desplazarse con seguridad por la zona blanca, no obstante, será necesario indicar un camino que el robot pueda seguir. Para definir una trayectoria que recorra la superficie blanca, se ha optado por esqueletizar el contorno blanco. Existen muchos algoritmos de esqueletización, como el Hall [33], el Guo-Hall [31] o el Zhang-Suen [46]. Se ha utilizado el algoritmo de Zhang-Suen porque es mas rápido que el resto y generalmente ofrece mejores resultados. Si aplicamos el algoritmo directamente sobre la imagen 8.18, obtendremos la imagen mostrada en la figura 8.19. Junto a la imagen esqueletizada, se muestra un esquema de la proyección de la trayectoria sobre el campo.

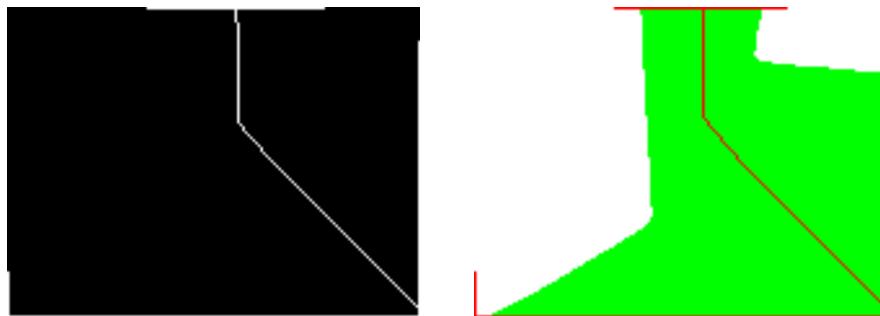


Figura 8.19: Esqueletización del contorno abierto y esquema del entorno

La esqueletización funciona bastante bien, pero hay un problema. Dado que el contorno a esqueletizar toca los bordes de la imagen, su contorno no está totalmente definido. Para evitar esto se ha realizado una pequeña transformación a la imagen antes de aplicarle el algorit-

mo. Esta transformación consiste en dibujar los bordes laterales de la imagen, de forma que corten el contorno y cierren su perímetro exterior. Una vez hecho esto, el resultado mejora notablemente, tal y como puede observarse en la figura 8.20



Figura 8.20: Esqueletización del contorno cerrado y esquema del entorno

Para evaluar la acción que deberá tomar el robot, en cada muestra se toma un segmento cuyo punto de inicio coincide con el punto de inicio de la trayectoria, y cuyo punto final estará contenido en la trayectoria, a una distancia parametrizable. Este segmento puede observarse en la figura 8.21

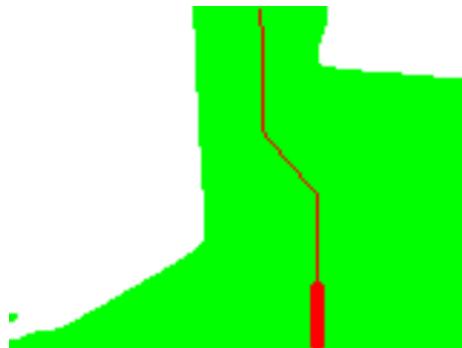


Figura 8.21: Segmento de referencia para la extracción de datos (en azul)

A partir de este segmento se han extraído dos datos. El primer dato, es la distancia horizontal que hay entre el robot y el inicio del segmento. El segundo dato es la orientación del segmento.

En la imagen de la figura 8.22 se observa una situación en la que existen dos posibles caminos.

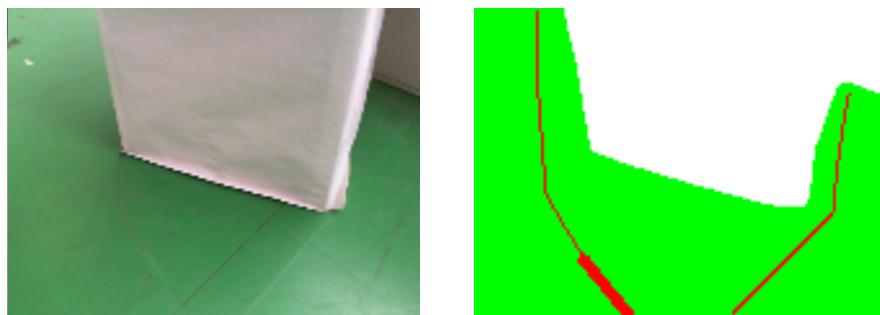


Figura 8.22: Esqueletización del contorno cerrado y esquema del entorno

Cuando esto ocurre se hace una comprobación de cual de ellos es el que permite llegar mas lejos. En caso de que ambos caminos sean igual de largos, se escogerá el que esté mas cerca del robot. Ocurrirá de igual modo cuando el camino muestre una ramificación, se tomará la que permita llegar mas lejos. Esta situación puede observarse en la figura 8.23.

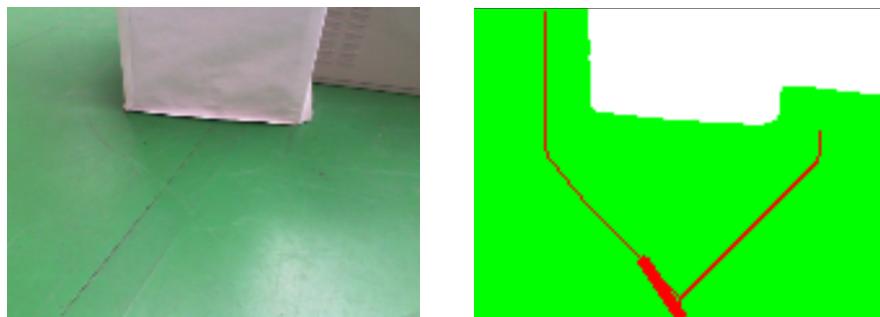


Figura 8.23: Esqueletización del contorno cerrado y esquema del entorno

8.5.2. Detección de líneas

Otra de los algoritmos que se han desarrollado es la detección de líneas. En el campo de pruebas existen dos líneas blancas. Estas líneas marcan zonas del campo relevantes durante el desarrollo de la prueba, por tanto será necesario contar con un programa que detecte las líneas y su posición respecto al robot. El código puede consultarse en el método *findLine* de *raider.h*

Para comenzar, se tomará una imagen y se pasará por la función *detectGreen*, detallada en el apartado anterior. A parte, para ahorrar recursos y aligerar el procesamiento, se recortará la imagen en altura. En la figura 8.24 se muestra la imagen original junto a la que se tomará como punto de partida.

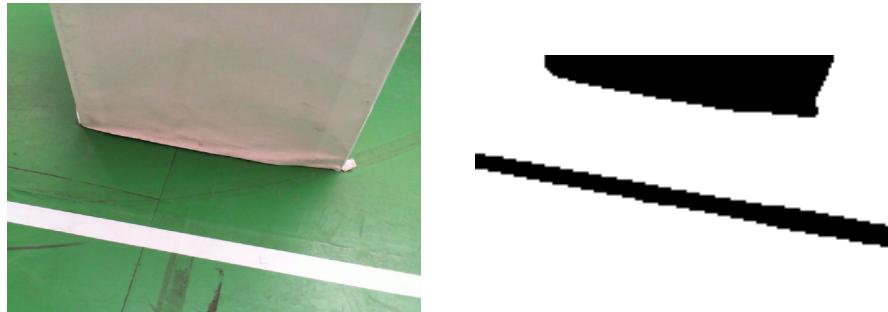


Figura 8.24: Imagen original e imagen tras aplicar la función *detectGreen*

Primero pasaremos a realizar una búsqueda de contornos [22], aplicando la función *Canny()* de OpenCV. La imagen se transforma en la figura 8.25. Como puede verse en la imagen, se han marcado los filos de los contornos de la línea y del obstáculo.

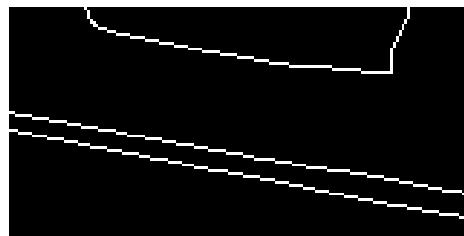


Figura 8.25: Detección de contornos

Para que el algoritmo funcione correctamente, interesaría que detecte la línea y el resto de cuerpos del campo no produzcan falsos positivos. Por tanto, el próximo objetivo es eliminar las rectas producidas por cuerpos estenos a la linea blanca. Una forma sencilla de lograr esto es el método de la dilatación y la erosión. Primero se dilatarán los contornos blancos de la imagen n veces hasta que los dos filos de la linea blanca se unan, es decir, los dos contornos se convertirán en un contorno mas grueso. Tras ello, la imagen se erosionará $n + 1$ veces. Esta erosión eliminará por completo las líneas producidas por cuerpos externos, pero mantendrá el contorno de la linea por ser mas grueso. Tras ello, el resultado es el que puede visualizarse en la figura 8.26

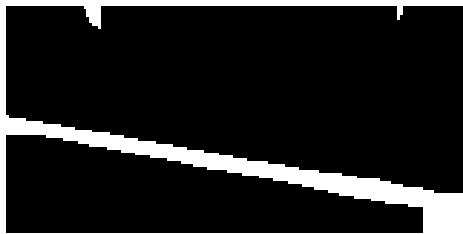


Figura 8.26: Dilatación y erosión de contornos

Ahora, una simple detección de líneas basada en la transformada de Hough [27], será suficiente para definir la linea blanca. Para ello se ha utilizado la función de OpenCV, *HoughLines*, ajustando empíricamente sus parámetros para definir una linea de una longitud adecuada y evitar falsos positivos de los restos de cuerpos externos que puedan quedar en la imagen. Tras realizar una búsqueda y proyectarla en la imagen original, se ha obtenido el resultado de la figura 8.27

Figura 8.27: Resultado final de *findLine*

8.5.3. Lectura de códigos QR

Para terminar, se ha realizado un algoritmo de detección de códigos QR basado en la librería ZBar [19]. El algoritmo se utilizará tanto para la prueba de visión del CEABOT como para dar instrucciones de propósito general al robot. Una de estas instrucciones podrá ser el inicio de una prueba, la selección de qué prueba debe realizar o cualquier otra que sea necesaria. El código programado se encuentra en la función *findQR* de *raider.h*

La librería ZBar está preparada para detectar diferentes tipos de códigos. En este caso, se ha limitado a que analice códigos QR en una imagen. A parte del código, podemos analizar su posición y su tamaño. En la figura 8.28 puede observar la detección de un marcador de la prueba de visión.

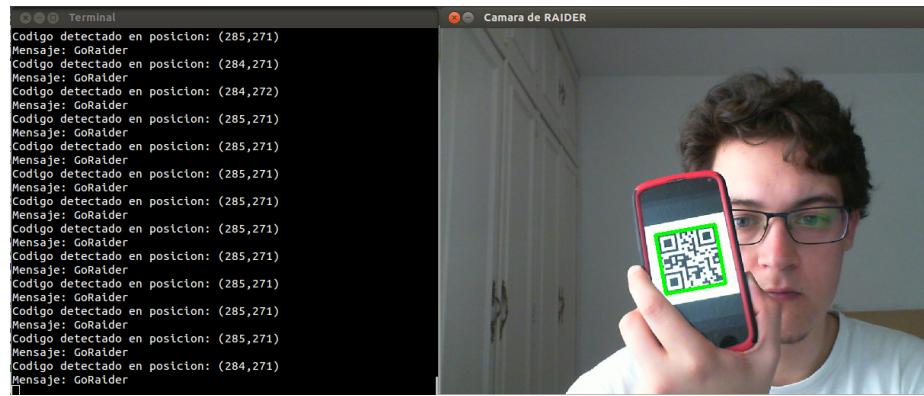


Figura 8.28: Análisis de códigos QR en una imagen

Adicionalmente, se ha programado de que en el hipotético caso de que detecte dos códigos en una imagen, sólo analice el código que se encuentre más cerca del centro de la imagen. Gracias a esto, el sistema será mas robusto de cara a la prueba de visión del CEABOT.

Capítulo 9

Aplicaciones

Llegamos a este capítulo, el estado del proyecto es el siguiente. Se ha desarrollado un robot completo tanto a nivel de hardware como a nivel de software. Así mismo, también se han creado herramientas para que el robot pueda interactuar con el entorno de forma controlada. Con los recursos que se han generado se preparará el robot para participar en dos eventos: La exhibición de futbol robótico Spain Experience, y la competición CEABOT 2014

9.1. Spain Experience

Spain Experience es un evento organizado por la Liga de Futbol Profesional en el que se unen actividades de sectores tecnológicos, empresariales y agroalimentarios. Con motivo de su celebración en México, el Centro de Investigación y de Estudios Avanzados (Cinvestav) [7] invitó a miembros del Robotics Lab y de AsRob a participar en un partido de futbol robótico inspirado en la competición RoboCup. El campo, visible en la figura 9.1 es propio de robots de mayor tamaño, lo que podría suponer una desventaja..

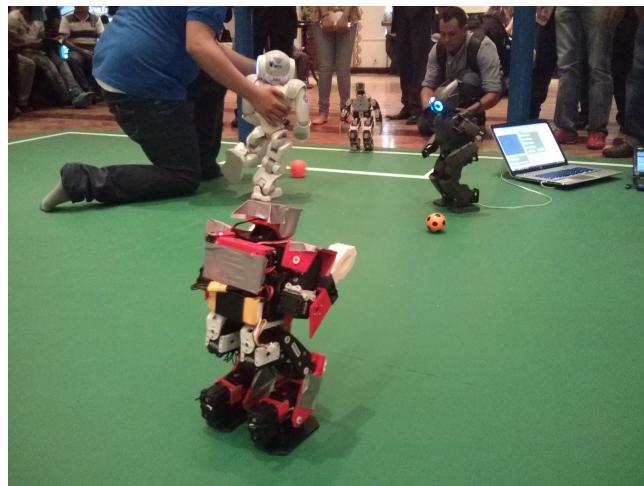


Figura 9.1: Partido de futbol robótico en México

Para preparar a RAIDER para la exhibición, se optará por montar el módulo Bluetooth que se comentó en capítulos anteriores del proyecto y teleoperarlo desde un smartphone. De este modo, el robot estaría funcionando durante mas de dos horas realizando cambios de batería, aproximadamente cada 10 minutos. Esta exhibición supone un gran paso en el desarrollo del robot, ya que es una forma excelente de someter a estress sus componentes y de operar en un escenario externo al laboratorio.

9.2. CEABOT 2014

Tal y como se ha ido diciendo a lo largo del proyecto, el robot se presenta a la edición de 2014 del CEABOT. Se propuso la participación de RAIDER en tres pruebas: Navegación, visión y sumo. Estos programas pueden encontrarse en el repositorio, en la carpeta *programming/src/apps/ceabot*. A continuación se detallan los algoritmos que se han diseñado para cada una de las pruebas.

9.2.1. Algoritmo para la prueba de visión

El algoritmo de la prueba de visión se basará en dos pilares: La cámara y la brújula. Con la cámara y la función *findQR()* de *raider.h* se realizará la detección de los marcadores. Por otro lado, la brújula se encargará de controlar el giro del robot para adecuarlo a las exigencias del marcador. A continuación, en la figura 9.2 se mostrará el diagrama de estados que se ha programado.

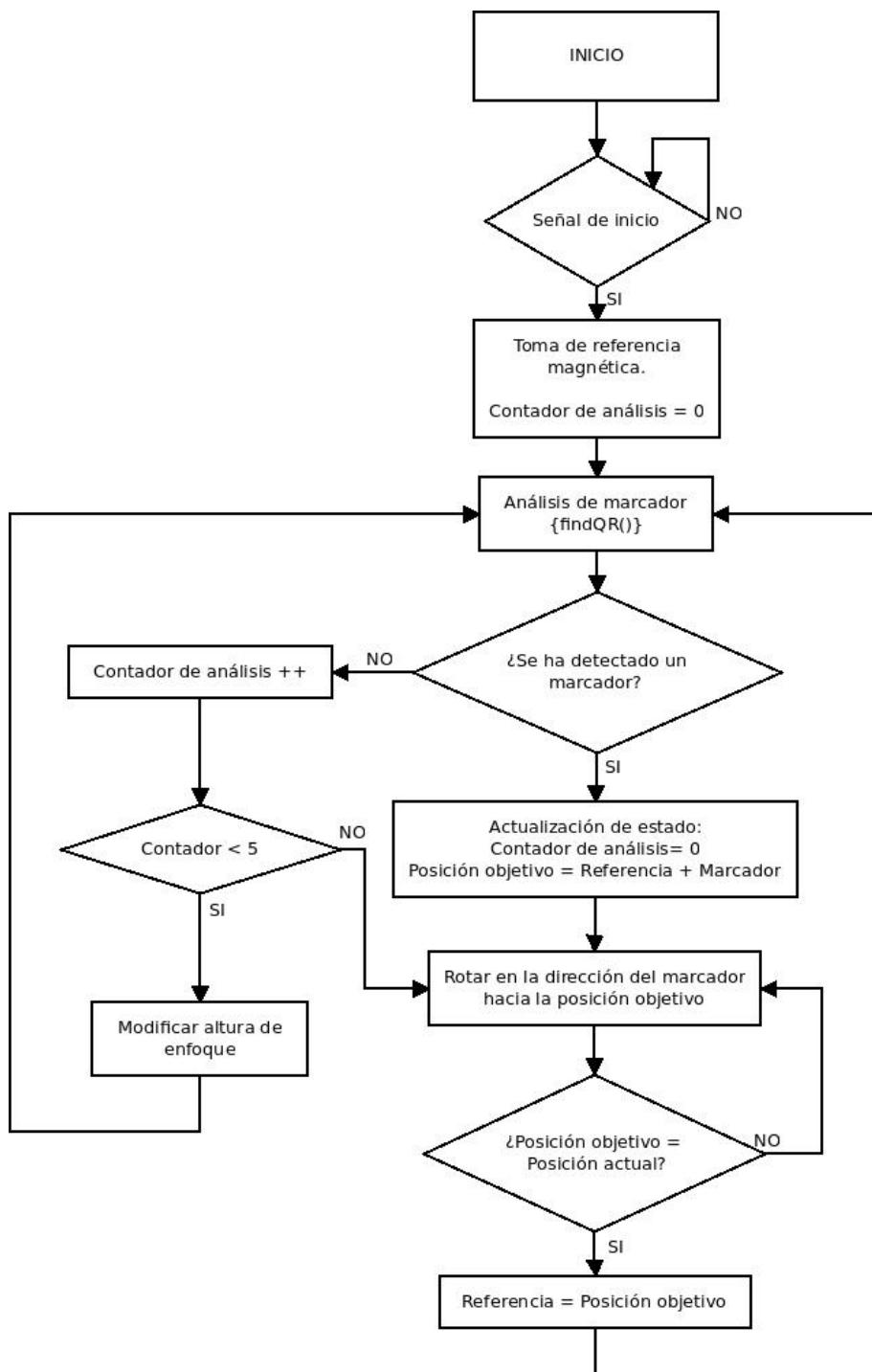


Figura 9.2: Diagrama de estados simplificado de la prueba de visión

El primer lugar el robot parte de un estado de inicio, en el que mantiene una posición de reposo. Seguirá en esta posición hasta que se le dé una señal de inicio. La señal de inicio se ha programado en la función *waitStart()* de *raider.h*. Una vez se ejecuta el robot se mantendrá realizando lecturas de códigos QR. Si se pone el marcador de la figura 9.3 en frente del robot (impreso en una tarjeta de papel, por ejemplo) el robot comenzará la prueba.



Figura 9.3: Código QR de inicio (Texto: GoRaider)

Una vez ha comenzado, guardará la medida que toma la brújula en la posición inicial. Ésta medida constituirá la primera referencia de la posición del robot. Tras ello, buscará el primer marcador, que por normativa se encontrará en frente de él. La búsqueda de códigos QR transcurre de la siguiente manera. El robot realizará un análisis visual, si éste falla, cambiará la altura de enfoque de la cámara y volverá a intentarlo. Como medida de seguridad, se ha programado que en el caso de que se produzcan 5 lecturas fallidas, el robot rote e intente recolocarse en su posición.

Tras realizar una lectura positiva de un marcador, se definirá una posición objetivo, basándose en la referencia del robot en ese momento y sumandole el incremento que indique el marcador. El robot comenzará a rotar y cuando alcance la posición objetivo tomará esa posición final como la nueva referencia. Tras esto, el algoritmo volverá a intentar detectar un código QR y se cerrará el bucle.

Cabe destacar un detalle: Las posiciones objetivo se definirán con la referencia de la posición del robot y el incremento que marque el código QR. Salvo en la primera detección (en la que la referencia se toma de una medida de la brújula), las referencias se definirán

como la posición objetivo de la búsqueda anterior y en ningún caso como una medida de la brújula al terminar los movimientos. De esta forma cada punto objetivo no estará sujeto a errores acumulativos, ya que su distancia respecto a la referencia inicial será siempre un múltiplo de 45°.

9.2.2. Algoritmo para la prueba de navegación

El algoritmo de la prueba de navegación quizás sea el mas complejo de los tres programados. En la figura 9.4 se ha realizado un diagrama de estados a alto nivel del comportamiento del robot durante la prueba.

De forma similar a la prueba de visión, el robot partirá de una posición de reposo, y se mantendrá en esa posición hasta que le indiquemos que comience la prueba con el código QR de la figura 9.3. Una vez que se da la orden, el robot toma una referencia magnética de la dirección en la que deberá recorrer el campo. Tras ello, comenzará a caminar.

Se ha implementado un control de caídas continuo durante toda la prueba. Para ello se ha utilizado la función *getFall*, detallada anteriormente en el capítulo anterior. Después de realizar cada movimiento, RAIDER evalúa si su posición es vertical respecto al suelo. En caso de que se detecte una caída, el robot se levantará de forma autónoma y continuará con su programación.

Antes de dar cada paso, RAIDER analiza la trayectoria que debe seguir con la función *findWay*. Esta función, como se vió en el apartado de programación, devolverá datos de la distancia que existe entre el robot y el punto de inicio de la trayectoria, y sobre el ángulo que forma el primer tramo de la trayectoria. Si el robot está lejos del punto de inicio, deberá acercarse con pasos laterales. Si el ángulo que forma el tramo es demasiado amplio, el robot rotará sobre sí mismo para orientarse. La definición de qué es "lejano", qué es "amplio", y su evaluación respecto al robot, se realizan con dos parámetros parametrizables dentro de *raider.h*. Por otro lado, cuando la linea es recta el robot podrá avanzar sin problemas. Existe una condición más, si el robot avanza de forma recta tres veces seguidas, se ejecutará una rutina para controlar si el robot está orientado correctamente y está siguiendo la dirección adecuada. Cuando esto ocurre, se compara la dirección medida por la brújula en ese instante con la referencia magnética que se tomó al principio de la prueba. En caso de que esta dirección sea diferente, el robot rotará sobre sí mismo para recuperar la orientación.

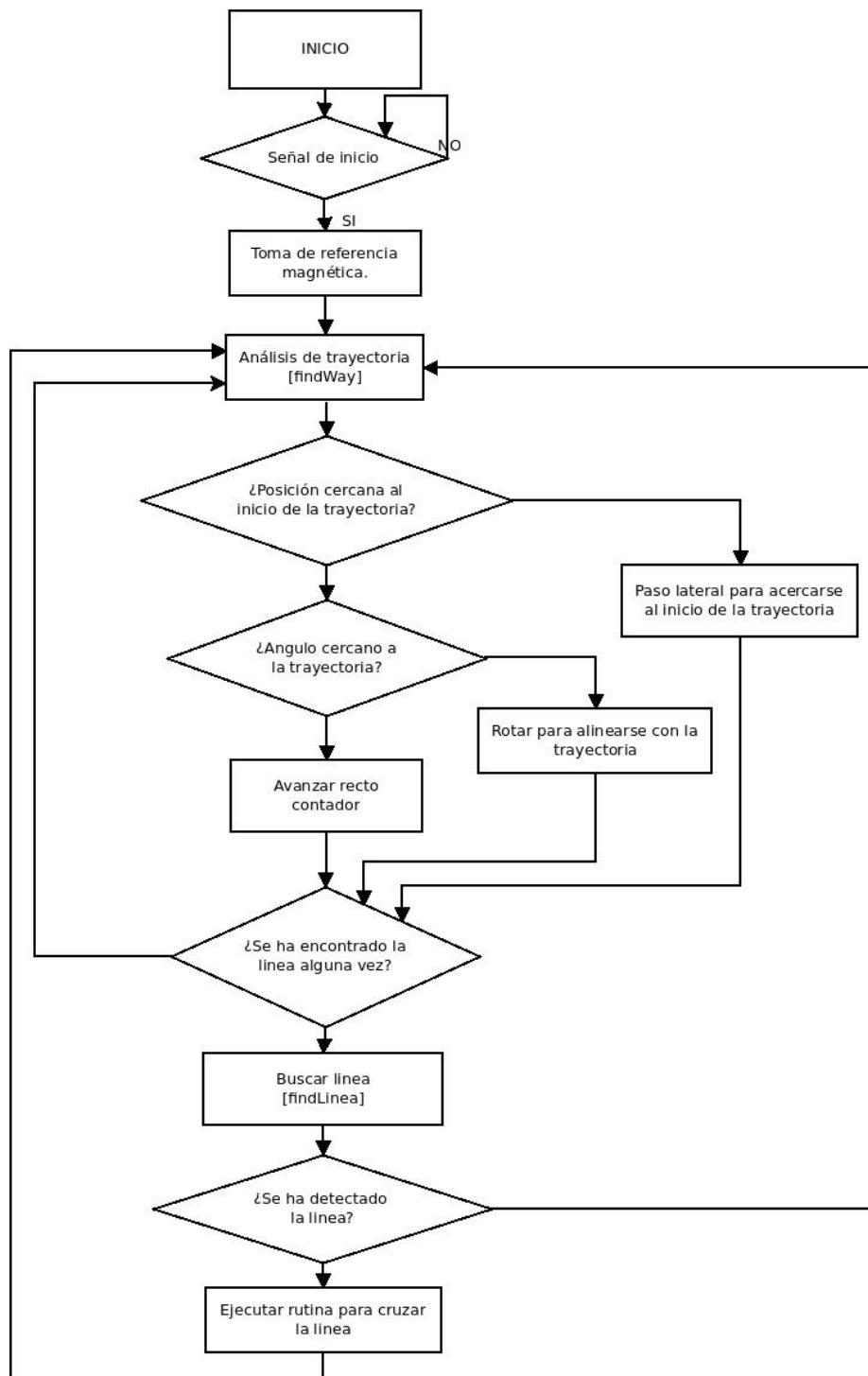


Figura 9.4: Diagrama de estados simplificado de la prueba de navegación

Tras ejecutar el desplazamiento, RAIDER intenta buscar la linea blanca mediante la función *findLine*. Si no hay ninguna detección volverá a buscar una trayectoria para seguir avanzando, sin embargo, si encuentra la linea blanca, se realizará una rutina especial. Primeramente cabe explicar qué significa esto: Que encuentre la linea blanca significa que ha llegado al ecuador de la prueba y que deberá cruzarla para posteriormente realizar el camino de vuelta. Por tanto, si el resultado de buscar la linea es positivo, la función *findLine* nos devolverá dos valores. Por una parte tendremos el valor de la distancia del robot al centro de la recta, que nos dará una idea aproximada de cuantos pasos necesitará realizar para cruzarla. Por otra parte, tendremos el valor de el angulo que forma el robot respecto a la linea. Nos interesa que RAIDER se sitúe perpendicularmente a la linea, por tanto, si este ángulo está lejos de esa posición será necesario que el robot se oriente de forma adecuada. Para definir si el robot está en un angulo lo suficientemente cerca de la perpendicularidad de la linea o debe rotarse un poco, se ha definido un valor parametrizable en la función *raider.h*.

Ya colocado correctamente junto a la linea, RAIDER caminará hacia el otro lado. Al llegar al otro lado se habrá superado la primera mitad de la prueba. Antes de emprender la vuelta, se deberá variar la referencia magnética que se tomó con la bújula al principio de la prueba. A esta referencia se le sumarán 180° , de forma que ahora el robot tratará de avanzar en esa dirección. Cuando se detecta la linea se activa un flag que indicará al robot que no debe volver a buscar lineas, sino que hasta el final de la prueba solo debe avanzar. De esta forma, RAIDER completará la segunda parte del recorrido.

9.2.3. Algoritmo para la prueba de sumo

En la prueba de sumo no se ha utilizado la cámara del robot, ya que no se ha considerado necesario. Esta prueba se basará en los sensores infrarrojos de los brazos de RAIDER. Esto se debe a que en esta prueba prima la programación de movimientos efectivos por encima de la sensorización. La programación de la prueba de sumo se ha basado en el diagrama de estados de la figura 9.5.

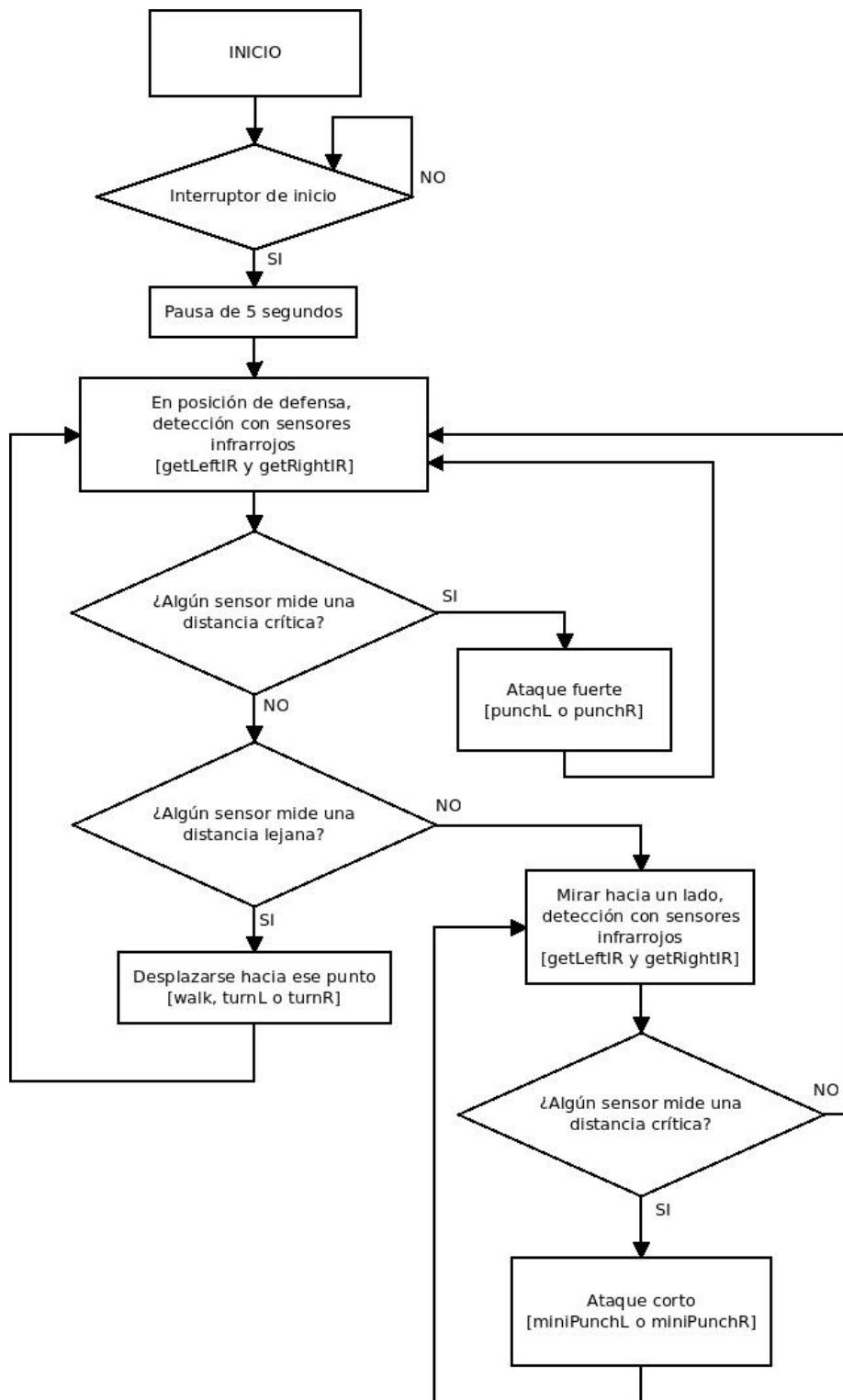


Figura 9.5: Diagrama de estados simplificado de la prueba de sumo

Como en el resto de programas, el robot permanece en reposo hasta que se le da una señal de inicio. LA diferencia es que en este caso no se utilizará la función *waitStart*, sino que se activará directamente desde el interruptor de seguridad. Este interruptor, como se aprecia en la figura 9.6, se encuentra en una zona accesible en la espalda del robot y se utiliza para cortar virtualmente la comunicación entre el controlador principal y el controlador de locomoción de este modo. Cuando el interruptor está encendido, RAIDER permanece inmóvil en una posición neutra. También, gracias a este interruptor se pueden realizar paradas de emergencia durante las pruebas.

TODO

Figura 9.6: Interruptor de emergencia

Cuando el interruptor se apaga, RAIDER realiza una pausa reglamentaria de 5 segundos. Tras ello comienza la prueba. Primero realizará una medición con los dos sensores infrarrojos apuntando hacia delante. Si alguno de estos dos sensores detecta un objeto en la zona crítica (representada en la figura 8.13) RAIDER lanzará un puñetazo con el brazo izquierdo o el brazo derecho, en función de qué medición es mayor. Si encuentra un objeto en la zona de detección lejana intentará acercarse. En este caso existirán dos posibilidades: Que ambos sensores hagan una detección lejana o que solo lo haga uno de ellos. Si ambos sensores miden la detección lejana, el robot avanzará hacia delante. Por otra parte, si solo realiza la detección uno de ellos, RAIDER rotará hacia el lado del sensor que haya tomado la medida. De este modo, se orientará hacia su objetivo.

También cabe la posibilidad de que ninguno de los dos sensores detecte nada. En este caso, RAIDER girará sobre su cintura para mirar hacia un lado. Cada vez que se llame a esta función RAIDER mirará a un lado diferente al de la última mirada, de esta forma se moverá a izquierda y a derecha intermitentemente. En el caso de que después de uno de estos giros detecte un objeto en la posición crítica, RAIDER lanzará un ataque ligero con el brazo más cercano al objetivo. Estos golpes son más débiles que los que se realizan en la posición de defensa porque en este momento el robot se encuentra en una posición inestable y podría caerse.

Paralelamente, existe un control de verticalidad. Con la función *getFall* se analiza si el robot sigue de pie después de cada movimiento. En el caso de que se haya caído, realizará la rutina de reincorporarse. Esto es muy importante en la prueba de sumo, ya que puede

suponer una perdida de puntos que el robot caiga al atacar a su oponente y no se levante.

Capítulo 10

Presupuesto

Autor:

Javier Isabel Hernández.

Departamento:

Departamento de Ingeniería de Sistemas y Automática.

Descripción del proyecto:

- Título: Desarrollo de una plataforma mini-humanoide basada en visión por computador.
- Duración (meses): 10

Presupuesto total del proyecto:

un griton de euros

Desglose presupuestario:**Personal**

| Apellidos y nombre | Categoría | Dedicación (horas) | Coste por hora (euro) | Coste (euro) |
|--------------------------|-----------|-----------------------|--------------------------|-----------------|
| Isabel Hernandez, Javier | Ingeniero | 210 | 45 | 9450 |
| Total: | | | | 9450 |

Materiales

| Descripción | Coste (euros) | % Uso dedicado al proyecto | Dedicación (meses) | Periodo de depre- ciación | Coste imputable (euro) |
|---|------------------|----------------------------------|-----------------------|---------------------------------|------------------------------|
| Bioloid Comprehensive | 620 | 80 | 10 | 60 | xxxx |
| Impresora Prusa i2 | 540 | 10 | 4 | 60 | xxxx |
| BeagleBone Black | 45 | 100 | 10 | 60 | xxxx |
| OpenCM9.04 | 12 | 100 | 10 | 60 | xxxx |
| Servo mg90s | 6.50 | 100 | 10 | 60 | xxxx |
| Servo AX-12A | 44 | 100 | 10 | 60 | xxxx |
| Microsoft LifeCam | 54 | 70 | 10 | 60 | xxxx |
| Sharp gp2y0a21yk (2) | 14 | 40 | 10 | 60 | xxxx |
| MPU9150 | 7.40 | 100 | 10 | 60 | xxxx |
| CMPS03 | 16.20 | 30 | 10 | 60 | xxxx |
| Componentes de placa de expansión | 6.40 | 100 | 10 | 60 | xxxx |
| Bobina de plástico ABS | 22 | 90 | 10 | 60 | xxxx |
| Tornillos allen M2 | 14 | 80 | 10 | 60 | xxxx |
| Tornillos y tuercas M3 | 2.40 | 60 | 10 | 60 | xxxx |
| Total: | | | | | 9450 |

Capítulo 11

Evaluación de resultados

En este capítulo se presentarán las conclusiones y consideraciones que se han podido extraer de este proyecto.

11.1. Pruebas

Llegados a este momento, RAIDER ha sido montado y puesto en marcha. Se han desarrollado aplicaciones sobre el robot que le han permitido asistir a diferentes competiciones y exhibiciones.

En Spain Experience, se puso a prueba la resistencia y autonomía de la plataforma. Durante mas de dos horas, el robot se controló de forma remota sobre el campo de juego. Durante la exhibición tuvo que dar patadas a un balón, correr al lado de otros robots (algunos de una categoría superior a los mini-humanoides), reincorporarse despues de algunas caídas y en definitiva, aguantar con una misma batería hasta diez minutos de uso.

Los resultados fueron muy satisfactorios. Las piezas imprimibles aguantaron los golpes que recibió en robot durante su operación y el sistema electrónico no sufrió ningún tipo de calentamiento o de interrupción en su funcionamiento.

Por otra parte, RAIDER se presentó a tres pruebas del campeonato CEABOT. En la modalidad de visión, fue el único robot capaz de diferenciar más de un marcador, llegando a la cifra de cuatro marcadores detectados y ejecutados correctamente. Esta actuación supuso el primer puesto de la prueba de visión.

En la prueba de navegación, el robot fue capaz de superar la prueba completa en un tiempo de dos minutos y trece segundos, siendo el único robot capaz de realizar el camino de vuelta desde la

zona de llegada parcial. Esta marca se tradujo en el primer puesto en la prueba de navegación.

Por último, la prueba de sumo contó con siete robots participantes. Tras varios combates, RAIDER escaló en la clasificación hasta la final, pero fue vencido en el último duelo. Sin embargo, consiguió el segundo puesto de la prueba de sumo.

Tras estas tres prueba, RAIDER quedó en el segundo puesto de la clasificación global, aún contando con el handicap de no haber participado en la prueba de escaleras.

11.2. Conclusión

Una vez finalizado el proyecto, las impresiones han sido muy positivas. Considero que los objetivos propuestos se han alcanzado con un alto grado de satisfacción. RAIDER ha demostrado ser un robot con unas capacidades muy aptas para la competición y la investigación.

A parte de ello, dejando de lado el robot, pienso que este trabajo me ha aportado conocimientos en diferentes áreas de la ingeniería que no podría haber conseguido de otro modo.

11.3. Desarrollos futuros

El término de este proyecto supone un punto de partida para nuevos desarrollos basados en la plataforma robótica mini-humanoide RAIDER. Por una parte, se propone la mejora de las capacidades de desplazamiento bipedo. En su estado actual, el robot tiene la capacidad suficiente para realizar controles de estabilidad avanzados, gracias a sus sensores iniciales y a la información que ofrecen los servomotores escogidos.

Por otra parte, se prevee una mejora a nivel mecánico incluyendo piezas de fibra de carbono y aumentando el número de grados de libertad del robot con servos de mayor potencia.

También, el que seguramente sea el paso inmediatamente posterior, se integrán manipuladores en las extremidades del robot. Unas manos móviles aportarán una mayor capacidad de interacción con el entorno, y podrán programarse algoritmos de grasping. Los pies también suponen un punto importante a la hora de realizar mejoras. Unos pies con sensores serían my útiles a la hora de controlar mejor

la marcha bípeda y también para realizar tareas como la de subir y bajar escaleras.

Al mismo tiempo, se seguirán realizando algoritmos de visión sobre la BeagleBone Black, una placa que ha demostrado tener unas capacidades muy interesantes para su montaje en pequeños robots. Adicionalmente, se prevee una reutilización de las librerías desarrolladas durante el proyecto para su aplicación en otros sistemas.

Bibliografía

- [1] Arduino Official Website, septiembre 2014 [online] <http://arduino.cc/>.
- [2] Asociación de Robótica (UC3M), septiembre 2014 [online] <http://asrob.uc3m.es>.
- [3] Proyecto myod: Make your own droid, septiembre 2014 [online] <http://asrob.uc3m.es/index.php/MYOD>.
- [4] Kondo robots website, septiembre 2014 [online] <http://kondo-robot.com>.
- [5] Robotics lab, septiembre 2014 [online] <http://roboticslab.uc3m.es>.
- [6] Normativa del campeonato ceabot, septiembre 2014 [online] http://www.ceautomatica.es/sites/default/files/upload/10/CEABOT/docs/normativa_CEABOT_2014.pdf.
- [7] Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, Cinvestav, septiembre 2014 [online] <http://www.cinvestav.mx>.
- [8] Cmucam website, septiembre 2014 [online] <http://www.cmucam.org>.
- [9] Register Map for MPU9150, septiembre 2014 [online] http://www.invensense.com/mems/gyro/documents/RM-MPU-9150A-00v4_2.pdf.
- [10] Robo-One website, septiembre 2014 [online] <http://www.robo-one.com>.
- [11] Cmps03 - compass module documentation, septiembre 2014 [online] <http://www.robot-electronics.co.uk/htm/cmps3tech.htm>.
- [12] Robotchallenge website, septiembre 2014 [online] <http://www.robotchallenge.org>.

- [13] Robotis bioloid website, septiembre 2014 [online] http://www.robotis.com/xe/bioloid_en.
- [14] Vstone Robovie-X, septiembre 2014 [online] http://www.vstone.co.jp/english/products/robovie_x/.
- [15] Serialib, simple serial library v1.2, septiembre 2014 [online] serialib.free.fr.
- [16] Antonio Barrientos, Luis Felipe Peñín, Carlos Balaguer, and Rafael Aracil. *Cinemática inversa*. McGraw-Hill, Interamericana de España, 2007.
- [17] Antonio Barrientos, Luis Felipe Peñín, Carlos Balaguer, and Rafael Aracil. *Programación por guiado*. McGraw-Hill, Interamericana de España, 2007.
- [18] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. .”O'Reilly Media, Inc.”, 2008.
- [19] Jeff Brown. Zbar bar code reader, 2012.
- [20] Jorge García Bueno, Miguel González-Fierro, Luis Moreno, and Carlos Balaguer. Facial emotion recognition and adaptative postural reaction by a humanoid based on neural evolution. *International Journal of Advanced Computer Science*, 3(10), 2013.
- [21] David Calkins. An overview of robogames [competitions]. *Robotics & Automation Magazine, IEEE*, 18(1):14–15, 2011.
- [22] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.
- [23] Gerald Coley. Beagleboard system reference manual. *BeagleBoard.org, December*, page 121, 2013.
- [24] Sharp Cooperation. Gp2y0a02yk data sheet, 2007.
- [25] Qt Creator. Cross-platform qt ide, 2009.
- [26] Arturo De la Escalera and José María Armingol. Visión por computador. *Fundamentos y métodos*, 2001.
- [27] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.
- [28] Brian Evans. *Practical 3D Printers*. Springer, 2012.

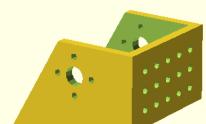
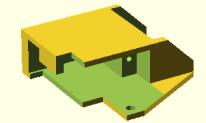
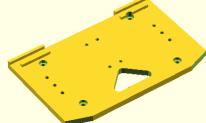
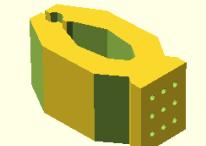
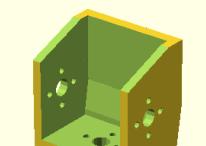
- [29] Miguel González-Fierro, Alberto Jardón, Martin F Santiago Martínez de la Casa, Juan G Stoelen, and Carlos Balaguer. Educational initiatives related with the ceabot contest. *Proceedings of SIMPAR*, pages 15–16, 2010.
- [30] David Gouaillier, Vincent Hugel, Pierre Blazevic, Chris Kilkner, Jérôme Monceaux, Pascal Lafourcade, Brice Marnier, Julien Serre, and Bruno Maisonnier. Mechatronic design of nao humanoid. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 769–774. IEEE, 2009.
- [31] Zicheng Guo and Richard W Hall. Parallel thinning with two-subiteration algorithms. *Communications of the ACM*, 32(3):359–373, 1989.
- [32] Inyong Ha, Yusuke Tamura, Hajime Asama, Jeakweon Han, and Dennis W Hong. Development of open humanoid platform darwin-op. In *SICE Annual Conference (SICE), 2011 Proceedings of*, pages 2178–2181. IEEE, 2011.
- [33] Richard W. Hall. Fast parallel thinning algorithms: parallel speed and connectivity preservation. *Communications of the ACM*, 32(1):124–131, 1989.
- [34] Kazuo Hirai, Masato Hirose, Yuji Haikawa, and Toru Takenaka. The development of honda humanoid robot. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2, pages 1321–1326. IEEE, 1998.
- [35] Alberto Jardón, Félix Rodríguez, Juan G Victores, Santiago Martínez, and Carlos Balaguer. A review of eight years of ceabot contest: A national wide mini humanoids competition. In *ROBOT2013: First Iberian Robotics Conference*, pages 41–52. Springer, 2014.
- [36] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 2, pages 1620–1626. IEEE, 2003.
- [37] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The robot world cup initiative. In *Proceedings of the first international conference on Autonomous agents*, pages 340–347. ACM, 1997.

- [38] Vanadium Labs. Software pypose, septiembre 2014 [online] <http://www.vanadiumlabs.com/pypose.html>.
- [39] Jon Loeliger and Matthew McCullough. *Version Control with Git: Powerful tools and techniques for collaborative software development*. O'Reilly Media, Inc.", 2012.
- [40] Ken Martin and Bill Hoffman. *Mastering CMake*. Kitware, 2010.
- [41] M Pareja Aparicio. Diseño y desarrollo de circuitos impresos con kicad, 2010.
- [42] ROBOTIS. OpenCM e-Manual. 2010 http://support.robotis.com/en/software/robotis_opencm.htm.
- [43] J. Russell and R. Cohn. *Openscad*. Book on Demand, 2012.
- [44] Erol Şahin. Swarm robotics: From sources of inspiration to domains of application. In *Swarm robotics*, pages 10–20. Springer, 2005.
- [45] Peter Troxler and Dannie Jost. Frictions collaborative creation of knowledge vs. practices in trade and commerce: The example of open hardware. 2014.
- [46] TY Zhang and Ching Y. Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984.

Anexo: Listado de piezas

| Imagen | Archivo | Cantidad | Observaciones |
|--------|-------------------|----------|--|
| | antebrazo.stl | x2 | Se imprimen dos iguales, son simétricas |
| | cabeza.stl | x1 | |
| | chasislifecam.stl | x1 | |
| | cintura.stl | x1 | El archivo incluye las dos partes de la pieza |

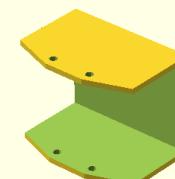
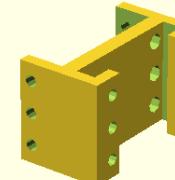
Cuadro 1: piezas

| Imagen | Archivo | Cantidad | Observaciones |
|---|-------------|----------|---|
|  | codo.stl | x2 | Incluye versión A (izquierda) y B (derecha) |
|  | cuello.stl | x1 | |
|  | espalda.stl | x1 | |
|  | gripper.stl | x2 | Se imprimen dos iguales, son simétricas |
|  | hombro.stl | x1 | Se imprimen dos iguales, son simétricas |

Cuadro 2: piezas

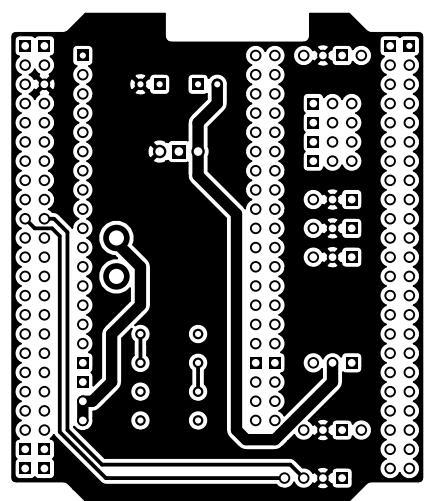
| Imagen | Archivo | Cantidad | Observaciones |
|--------|---------------|----------|--|
| | pecho.stl | x1 | |
| | pie.stl | x2 | Incluye versión A (izquierda) y B (derecha) |
| | pierna1.stl | x2 | Se imprimen dos iguales, son simétricas. El archivo incluye dos piezas |
| | pierna2.stl | x2 | Se imprimen dos iguales, son simétricas |
| | protector.stl | x1 | |

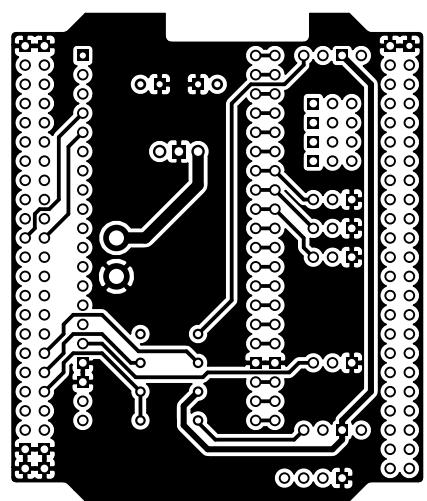
Cuadro 3: piezas

| Imagen | Archivo | Cantidad | Observaciones |
|---|----------------------|----------|--|
|  | soporteBateria.stlx1 | | |
|  | tapaCabeza.stl | x1 | |
|  | tobillo.stl | x2 | Se imprimen dos iguales, son simétricas |

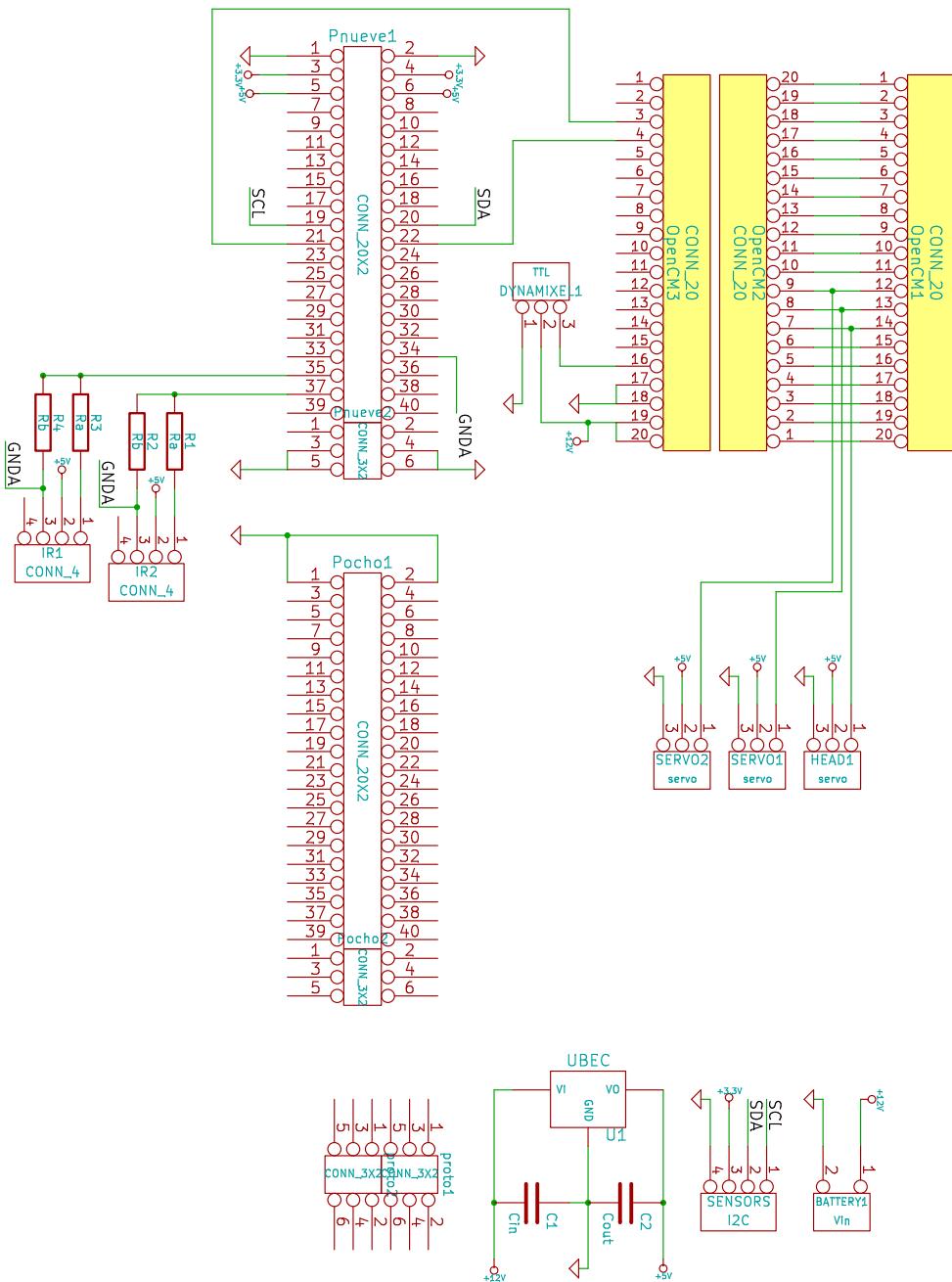
Cuadro 4: piezas

Anexo: Fotolitos de la placa de expansión



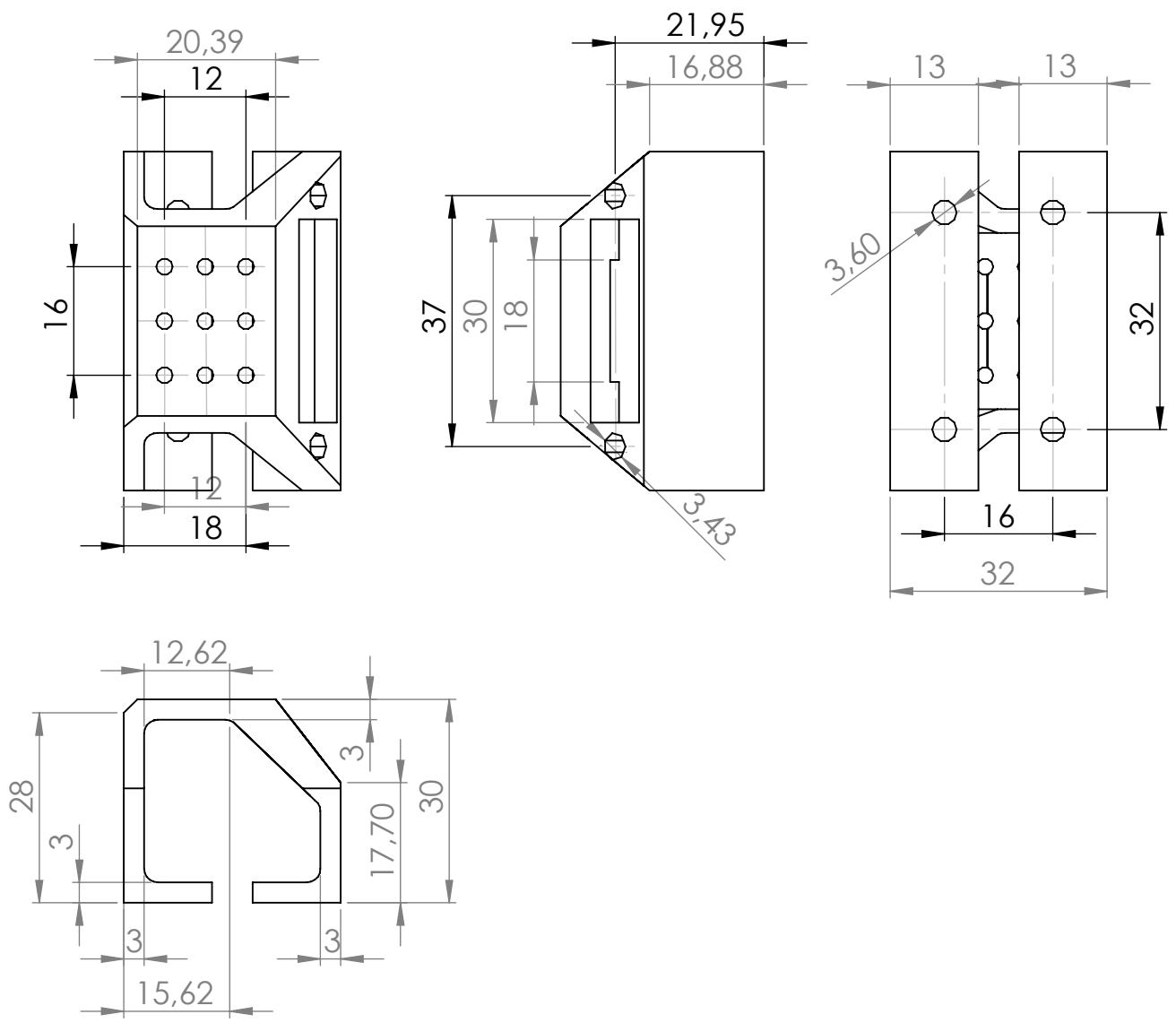


Anexo: Esquemático de la placa de expansión



File: shield.sch
Sheet: /
Title: PLACA DE EXPANSION PARA EL ROBOT RAIDER
Size: A4 **Date:** 8 sep 2014 **Rev:** 2
KiCad E.D.A. eeschema (2012-apr-16-27)-stable
4 5

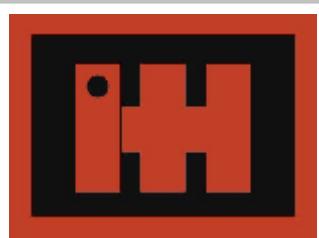
Anexo: Planos

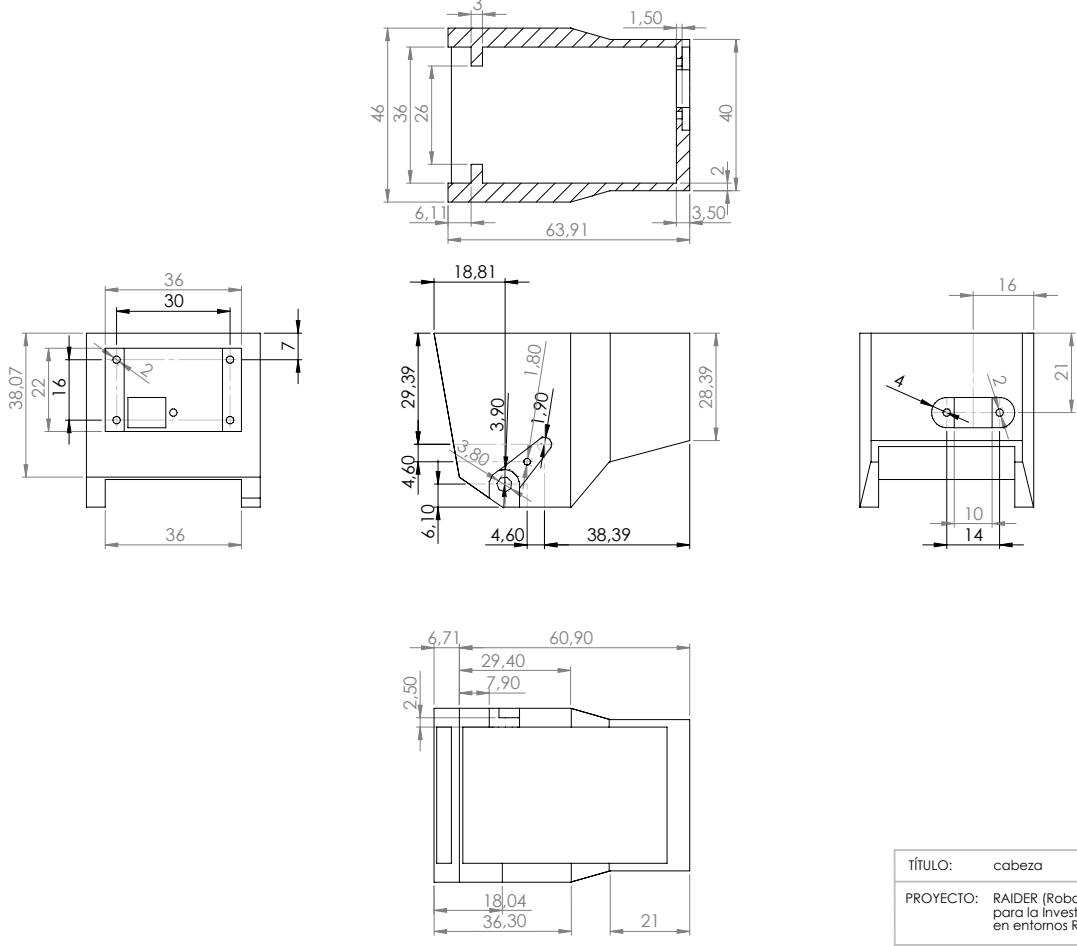


TÍTULO: antebrazo

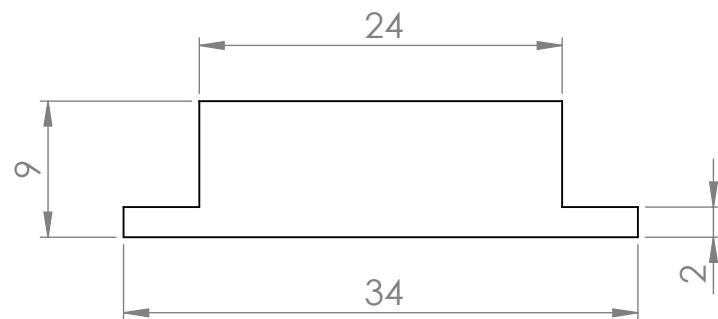
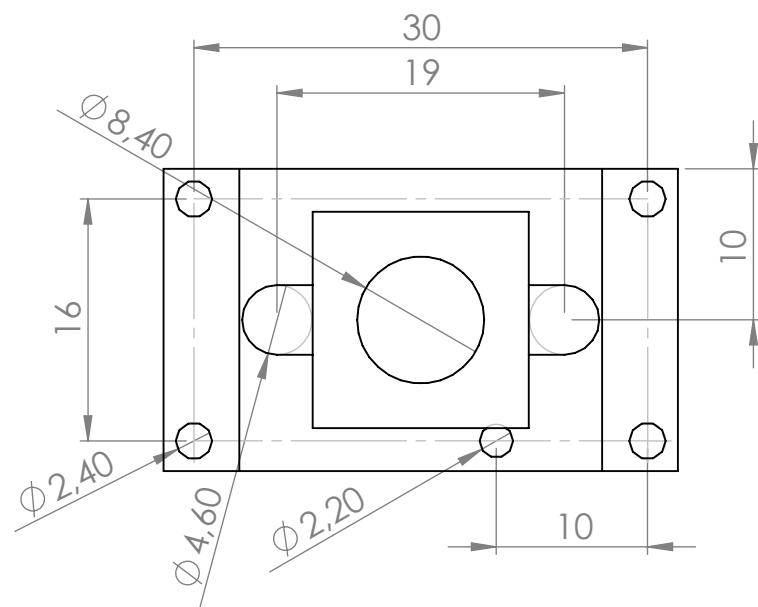
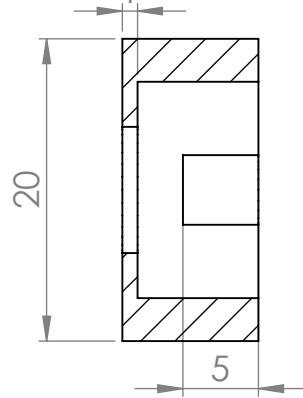
PROYECTO: RAIDER (Robot Antropomórfico para la Investigación y el Desarrollo en entornos Reales)

AUTOR: Javier Isabel Hernández

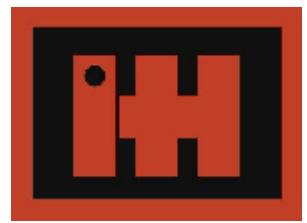




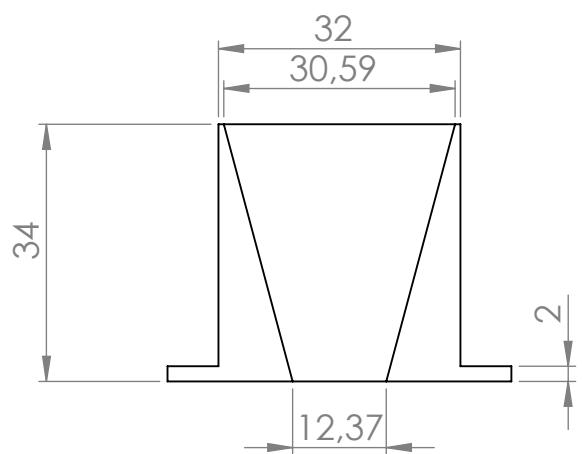
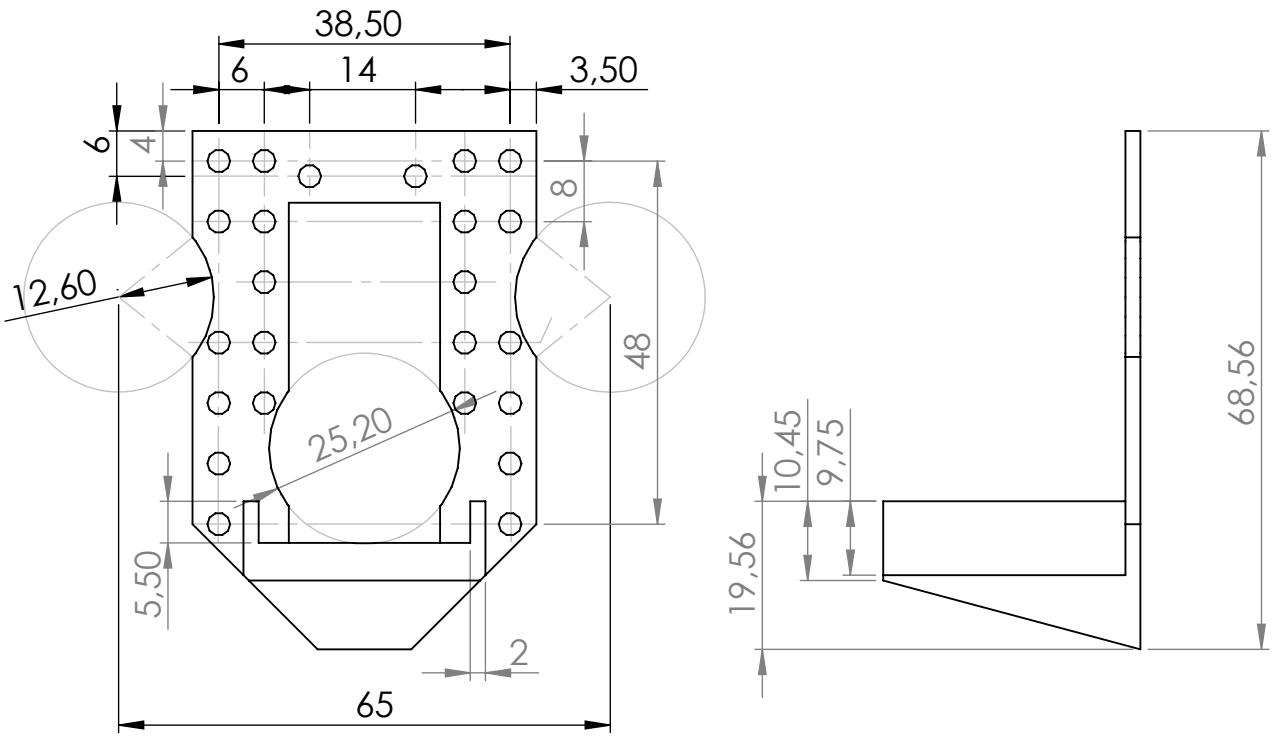
| | | |
|------------|--|----|
| TÍTULO: | cabeza | |
| PROYECTO: | RAIDER (Robot Antropomórfico para la Investigación y el Desarrollo en entornos Reales) | |
| AUTOR: | Javier Isabel Hernández | |
| PLANO 2/18 | ESCALA: 1:1 | A3 |



| | |
|------------|--|
| TÍTULO: | chasislifecam |
| PROYECTO: | RAIDER (Robot Antropomórfico para la Investigación y el Desarrollo en entornos Reales) |
| AUTOR: | Javier Isabel Hernández |
| PLANO 3/18 | ESCALA: 2:1 |



A4



TÍTULO: cintura

PROYECTO: RAIDER (Robot Antropomórfico para la Investigación y el Desarrollo en entornos Reales)

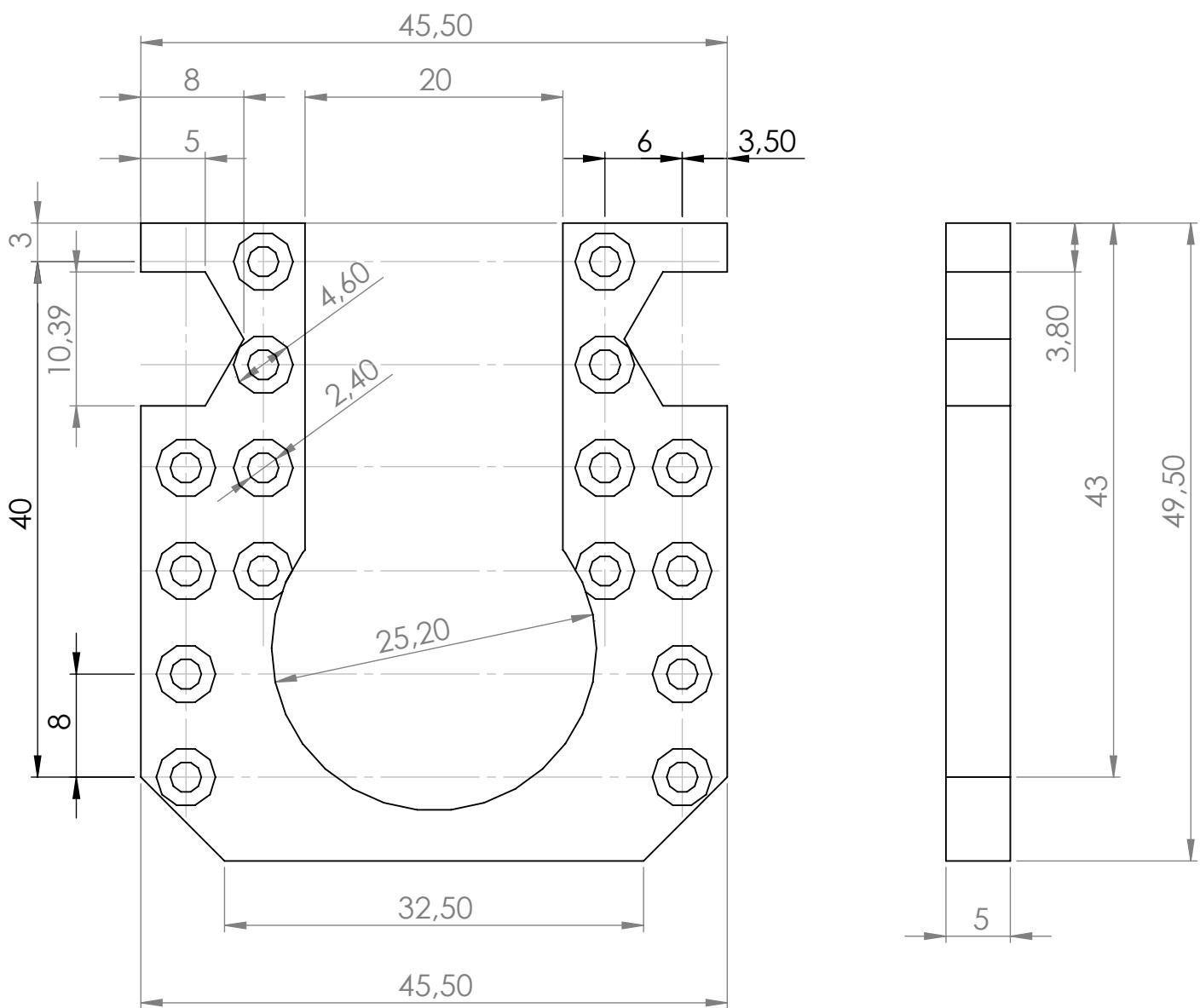
AUTOR: Javier Isabel Hernández

PLANO 4/18



ESCALA: 1:1

A4



TÍTULO: cintura2

PROYECTO: RAIDER (Robot Antropomórfico para la Investigación y el Desarrollo en entornos Reales)

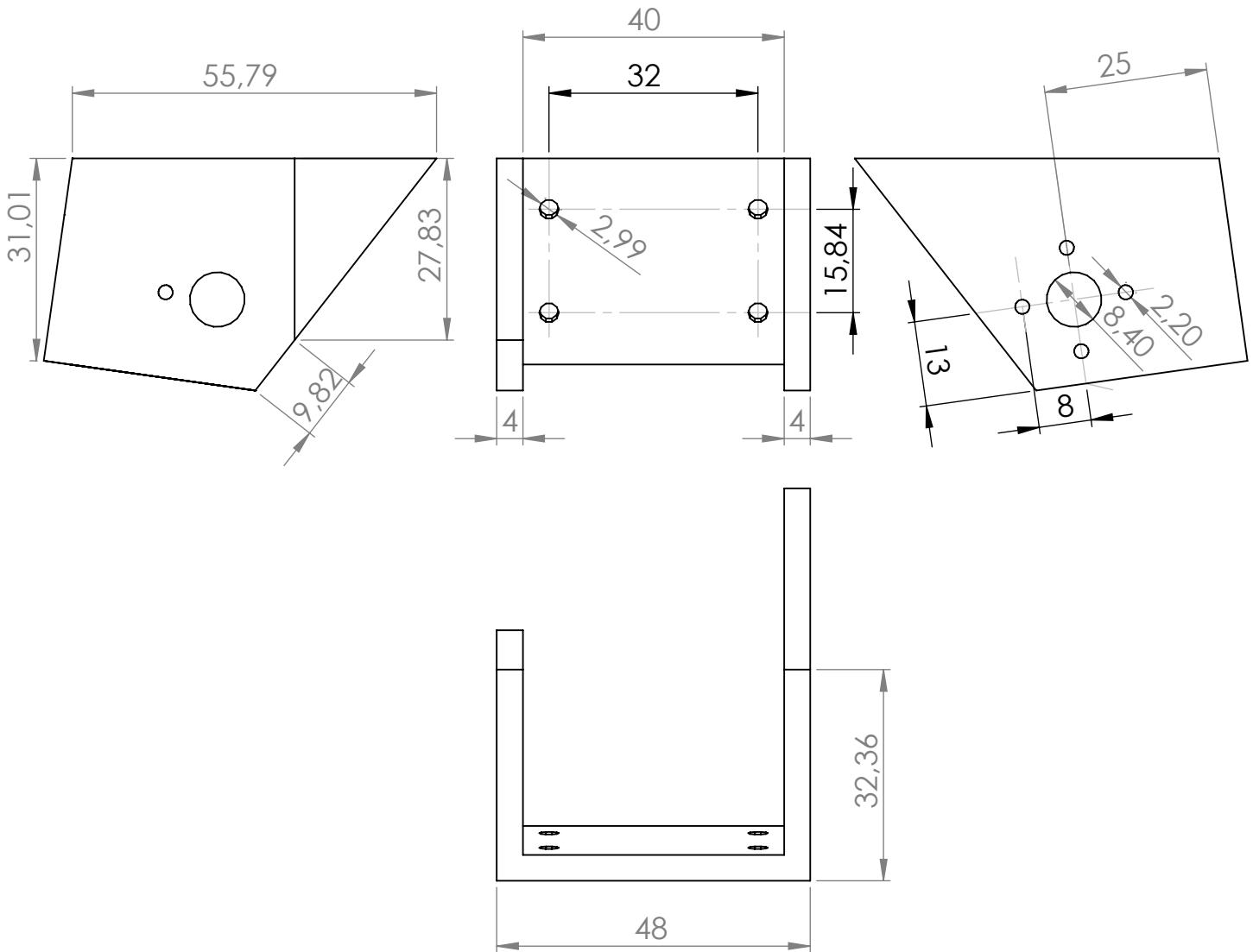
AUTOR: Javier Isabel Hernández

PLANO 5/18



ESCALA: 2:1

A4

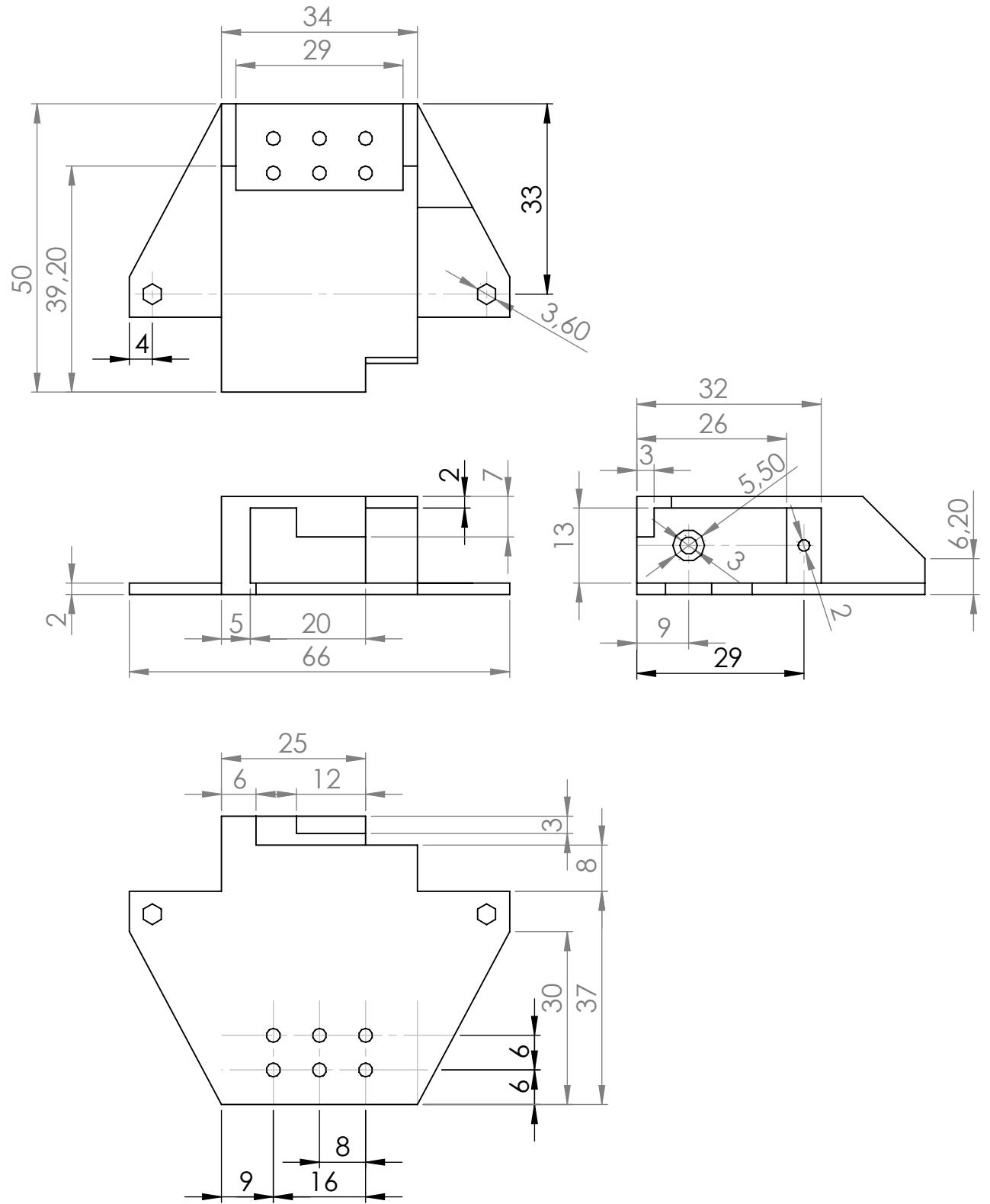


TÍTULO: codo

PROYECTO: RAIDER (Robot Antropomórfico para la Investigación y el Desarrollo en entornos Reales)

AUTOR: Javier Isabel Hernández



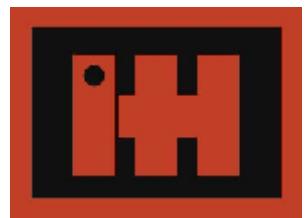


TÍTULO: cuello

PROYECTO: RAIDER (Robot Antropomórfico para la Investigación y el Desarrollo en entornos Reales)

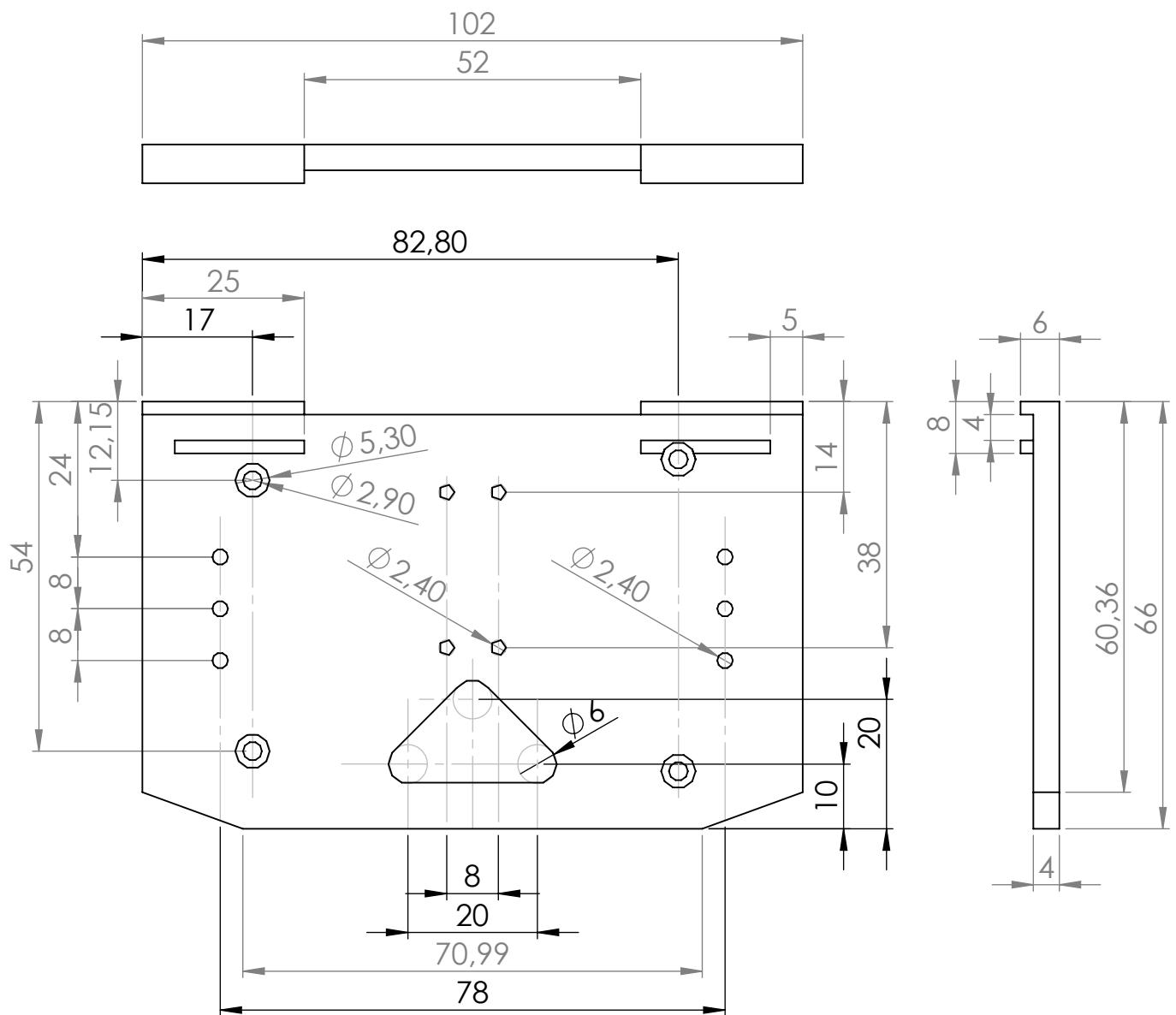
AUTOR: Javier Isabel Hernández

PLANO 7/18



ESCALA: 1:1

A4

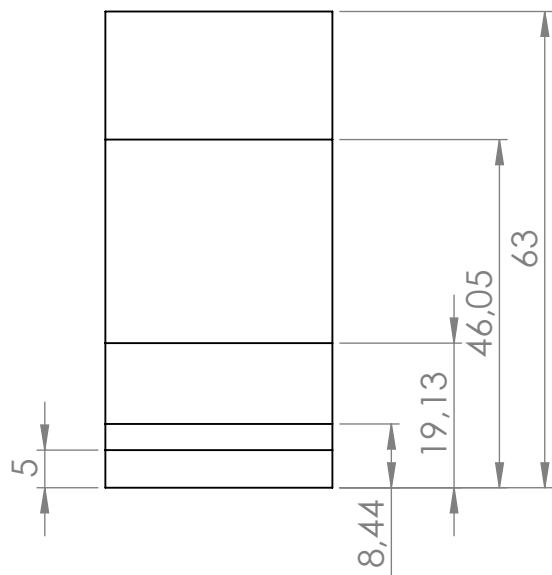
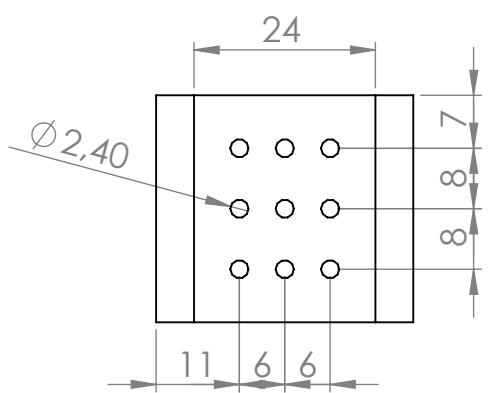
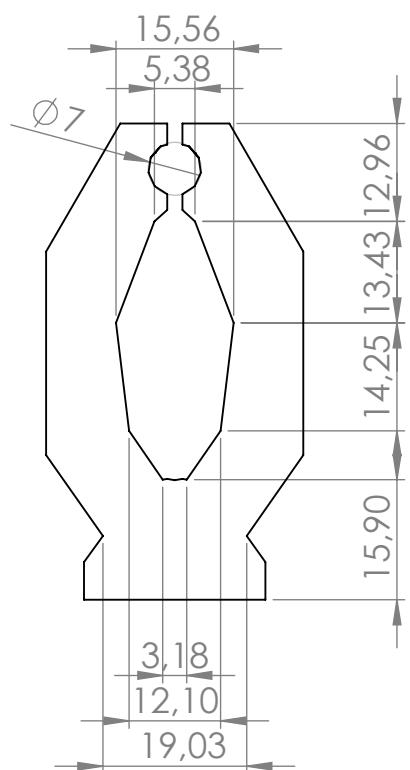
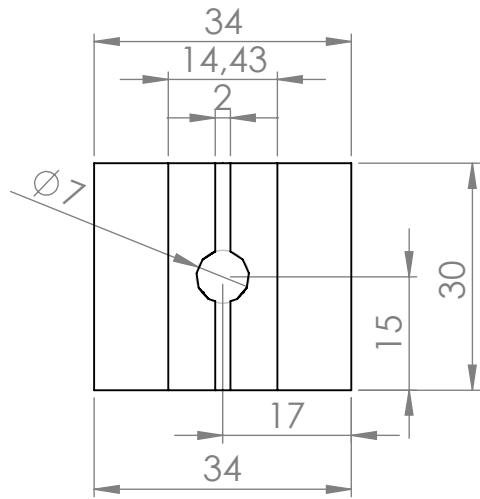


TÍTULO: espalda

PROYECTO: RAIDER (Robot Antropomórfico para la Investigación y el Desarrollo en entornos Reales)

AUTOR: Javier Isabel Hernández

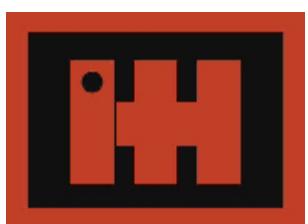


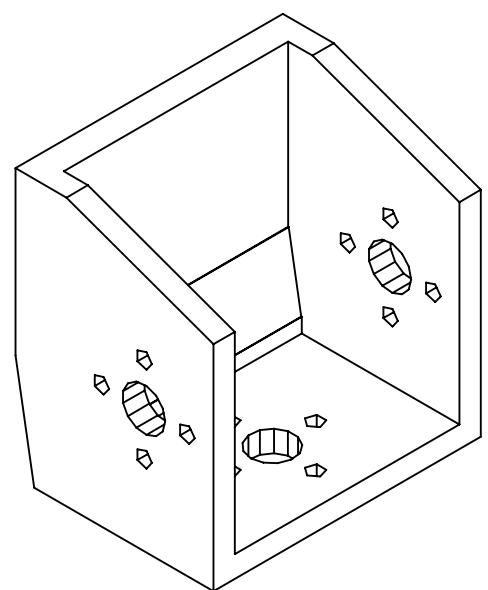
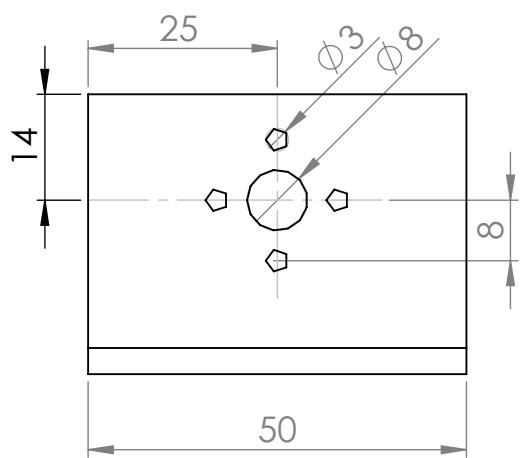
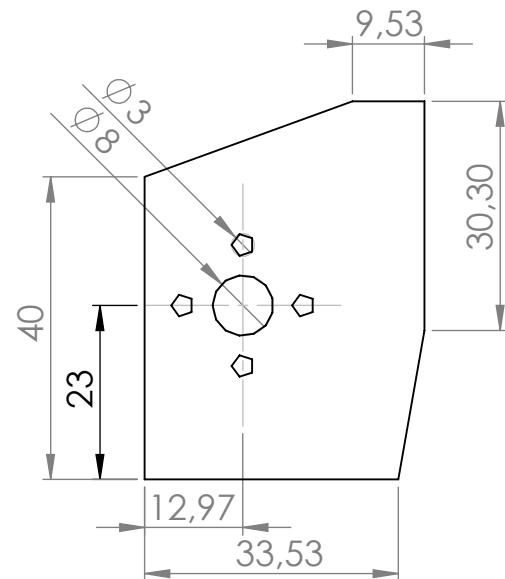
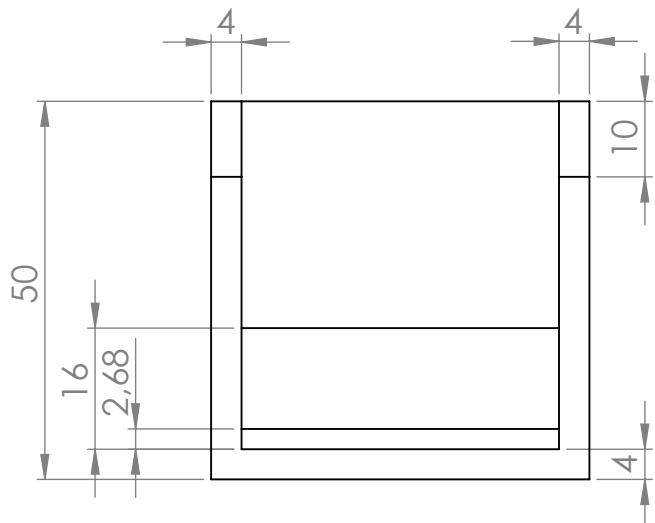
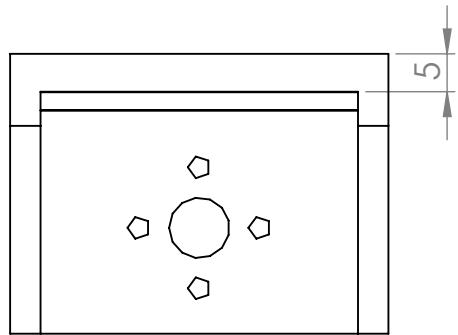


TÍTULO: gripper

PROYECTO: RAIDER (Robot Antropomórfico para la Investigación y el Desarrollo en entornos Reales)

AUTOR: Javier Isabel Hernández



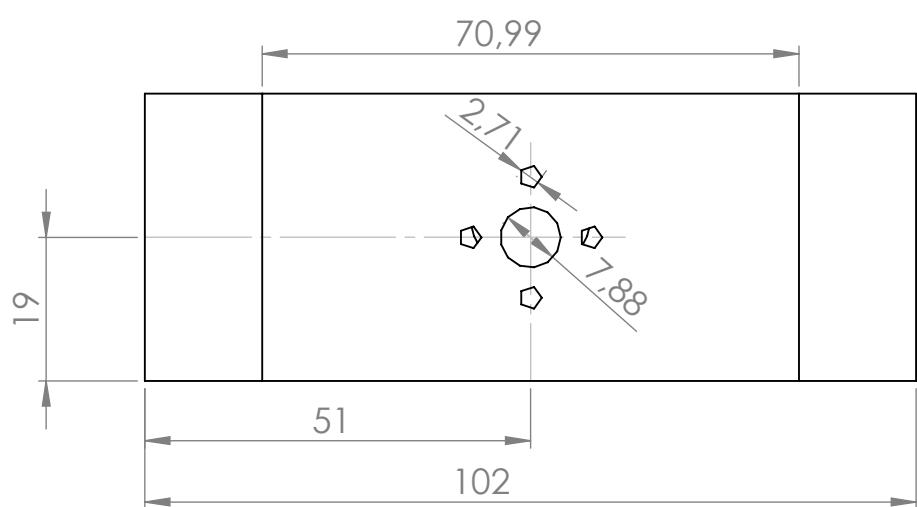
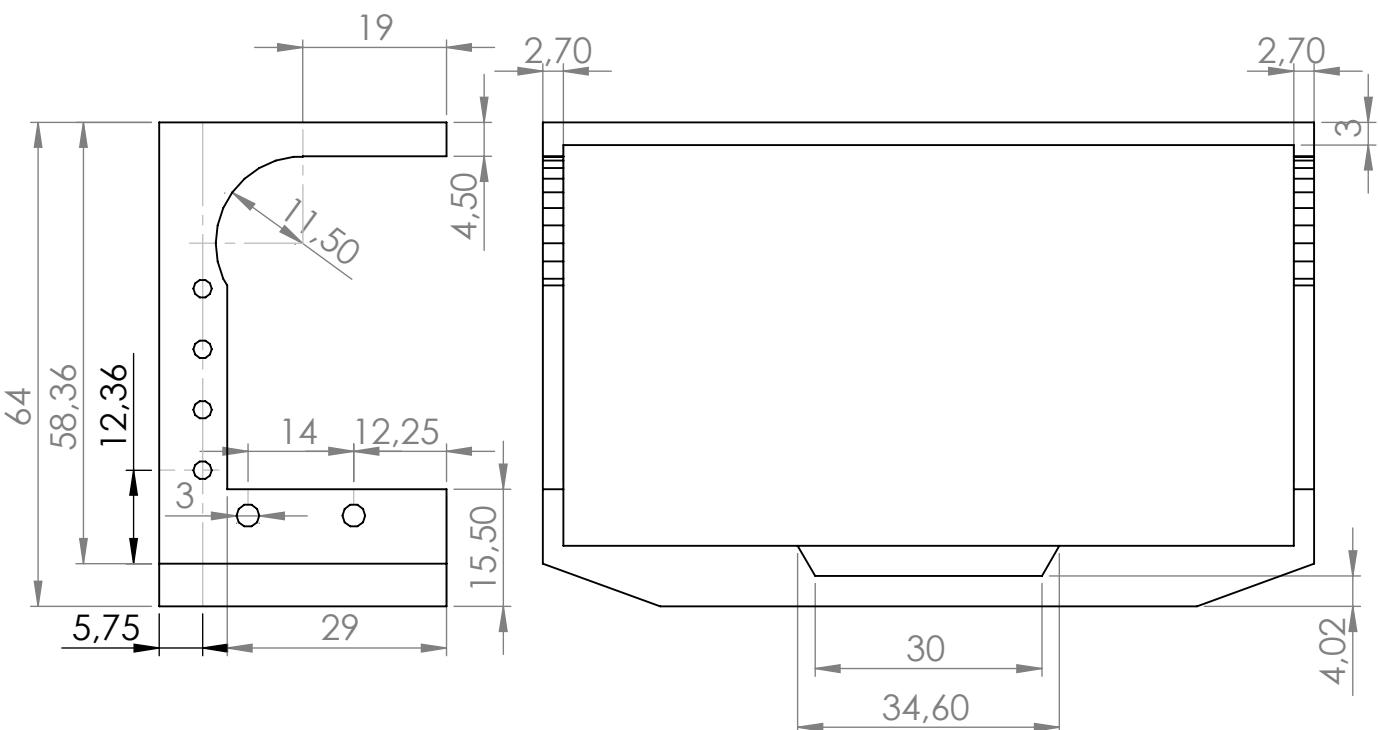
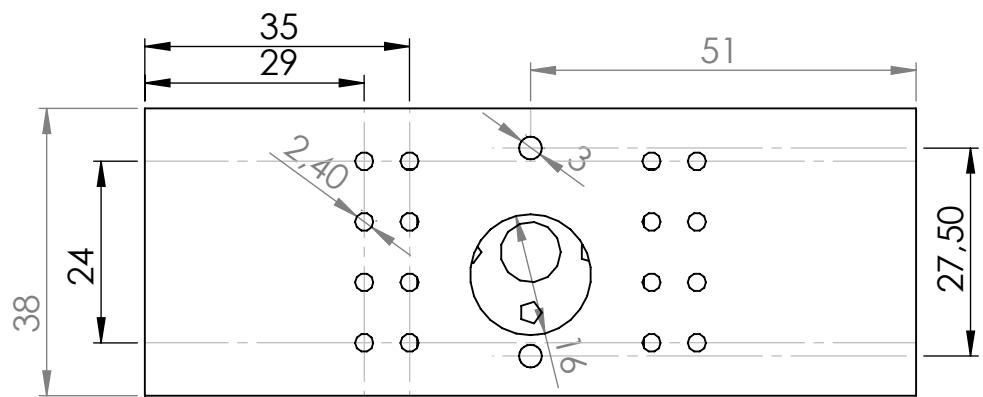


TÍTULO: hombro

PROYECTO: RAIDER (Robot Antropomórfico para la Investigación y el Desarrollo en entornos Reales)

AUTOR: Javier Isabel Hernández



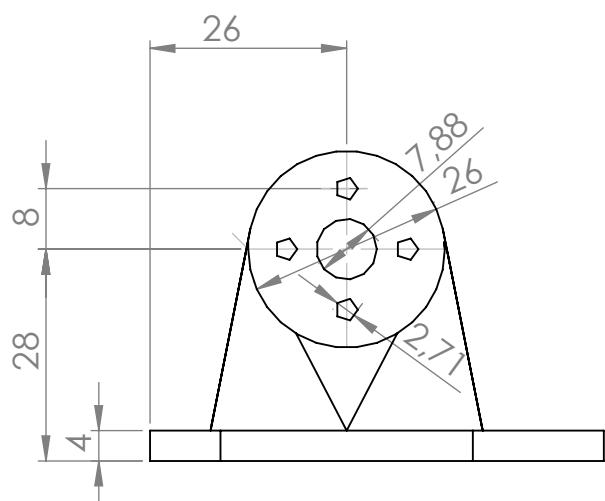
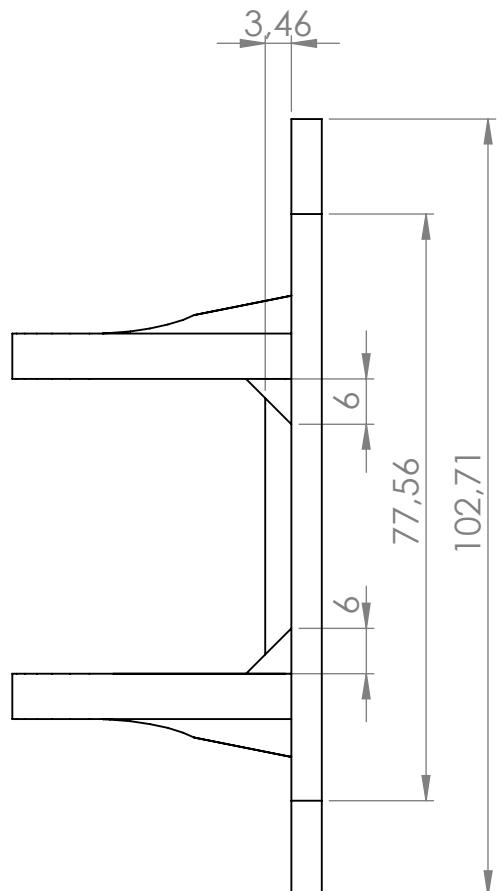
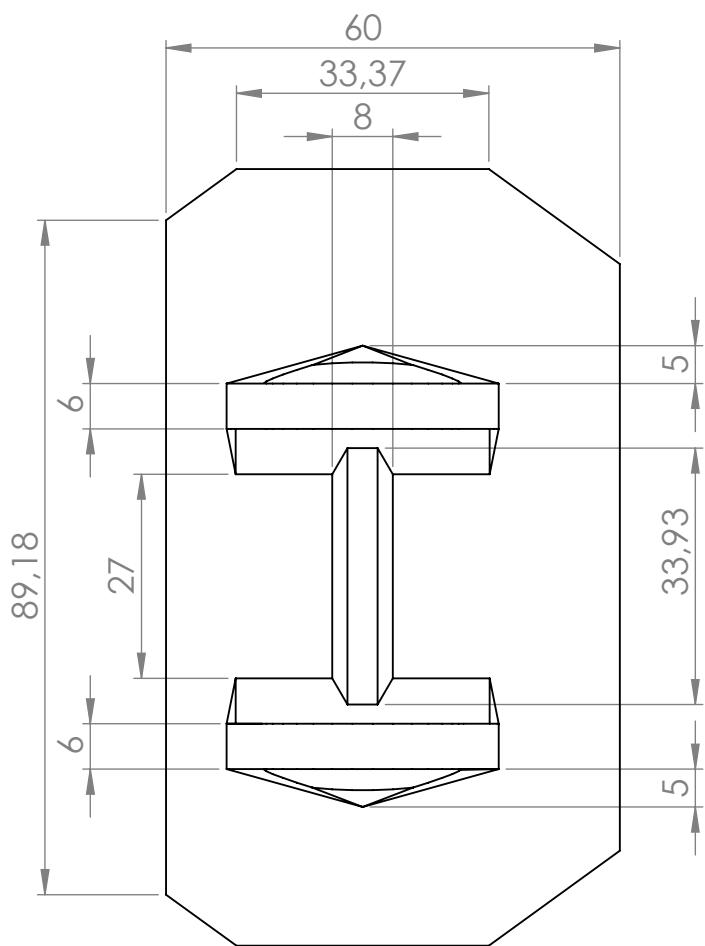


TÍTULO: pecho

PROYECTO: RAIDER (Robot Antropomórfico para la Investigación y el Desarrollo en entornos Reales)

AUTOR: Javier Isabel Hernández



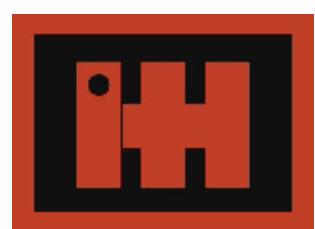


TÍTULO: pie

PROYECTO: RAIDER (Robot Antropomórfico para la Investigación y el Desarrollo en entornos Reales)

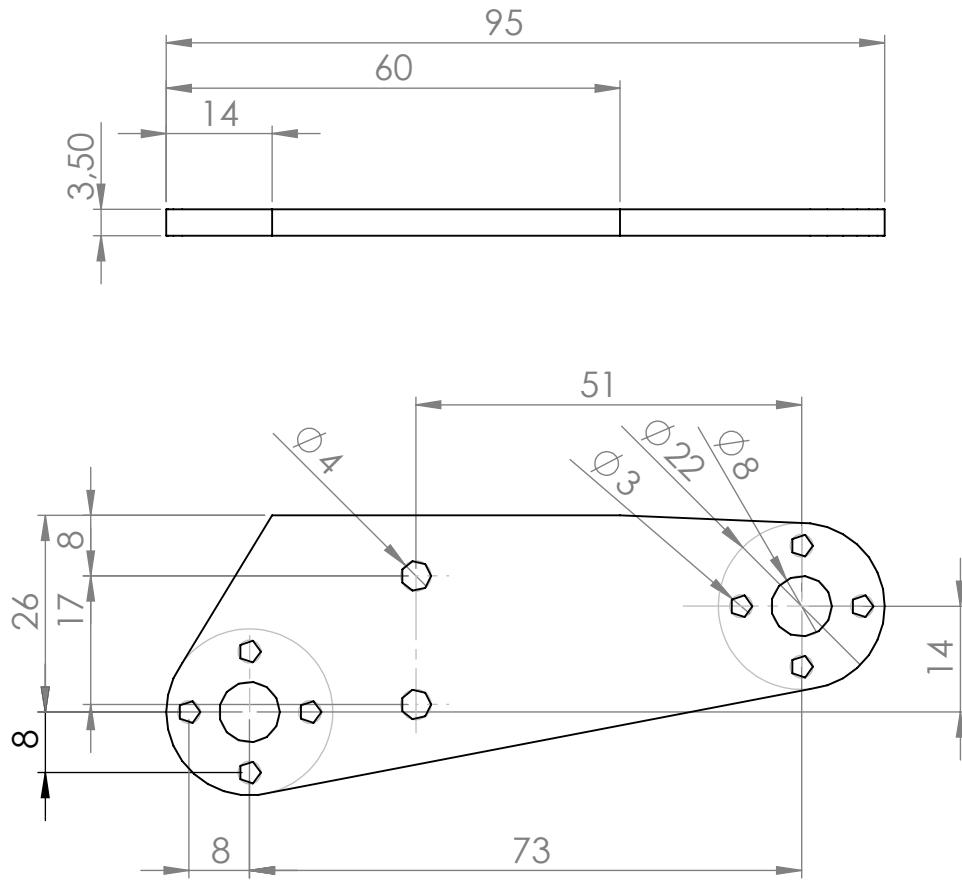
AUTOR: Javier Isabel Hernández

PLANO 12/18



ESCALA: 1:1

A4

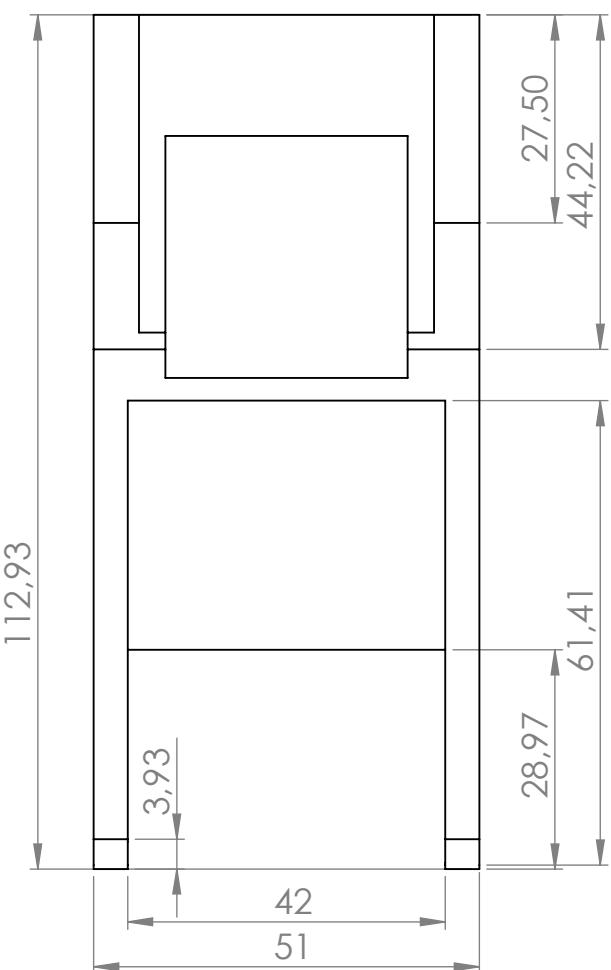
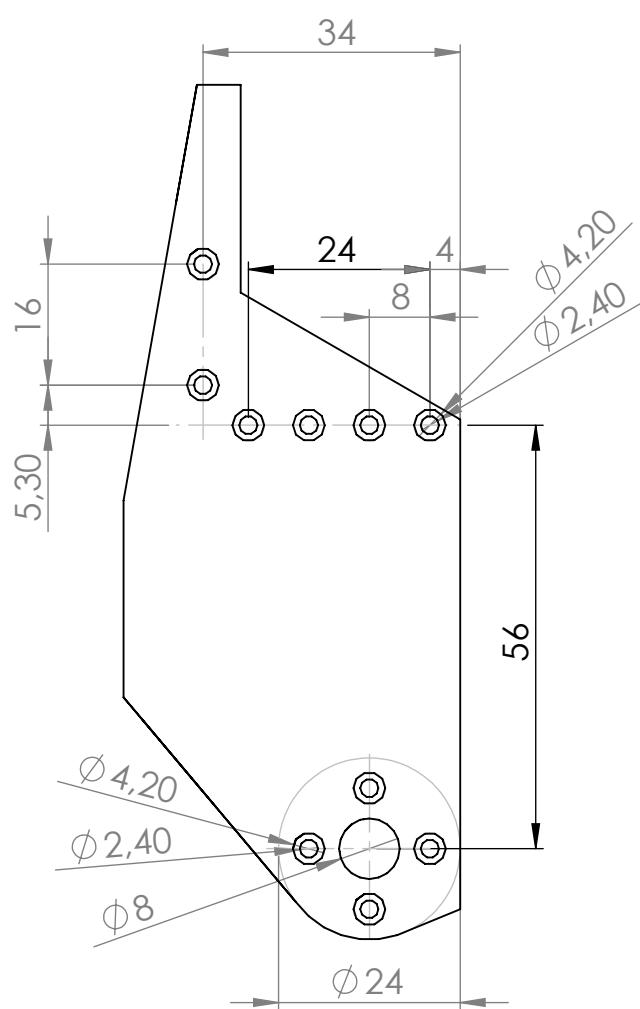
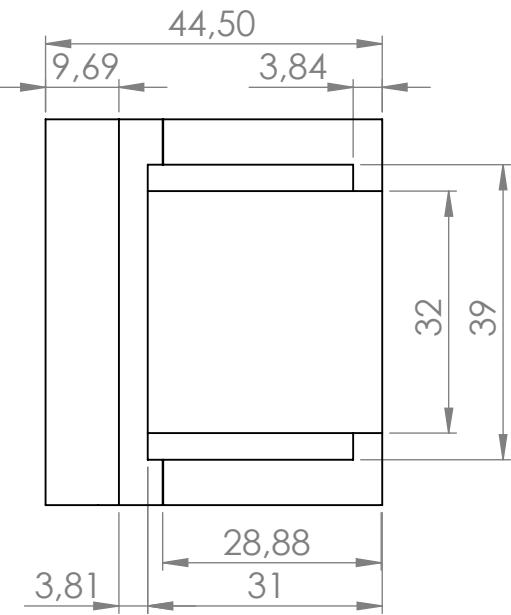


TÍTULO: pierna

PROYECTO: RAIDER (Robot Antropomórfico para la Investigación y el Desarrollo en entornos Reales)

AUTOR: Javier Isabel Hernández



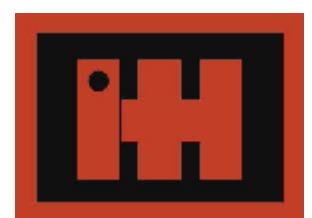


TÍTULO: pierna2

PROYECTO: RAIDER (Robot Antropomórfico para la Investigación y el Desarrollo en entornos Reales)

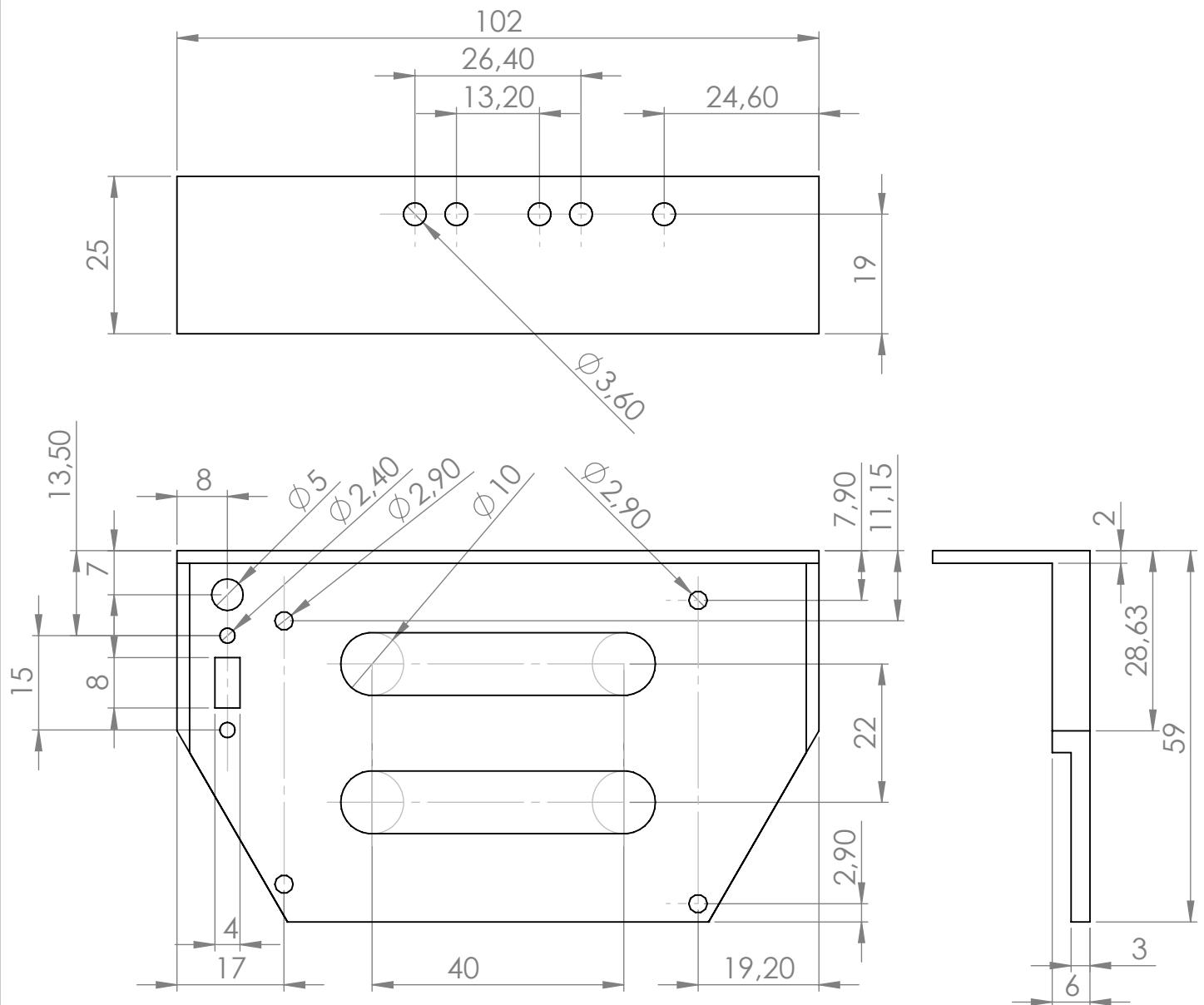
AUTOR: Javier Isabel Hernández

PLANO 14/18



ESCALA: 1:1

A4

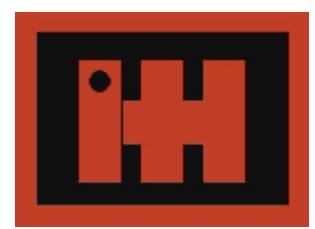


TÍTULO: protector

PROYECTO: RAIDER (Robot Antropomórfico para la Investigación y el Desarrollo en entornos Reales)

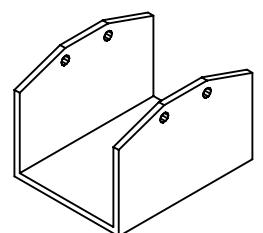
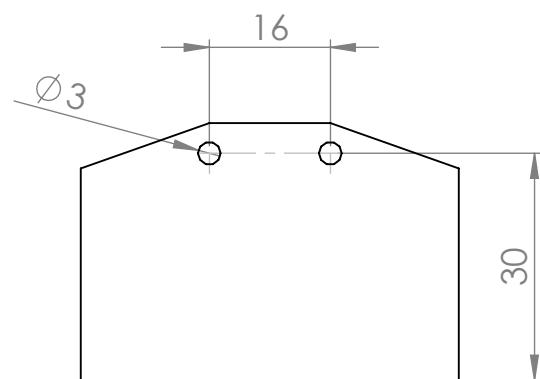
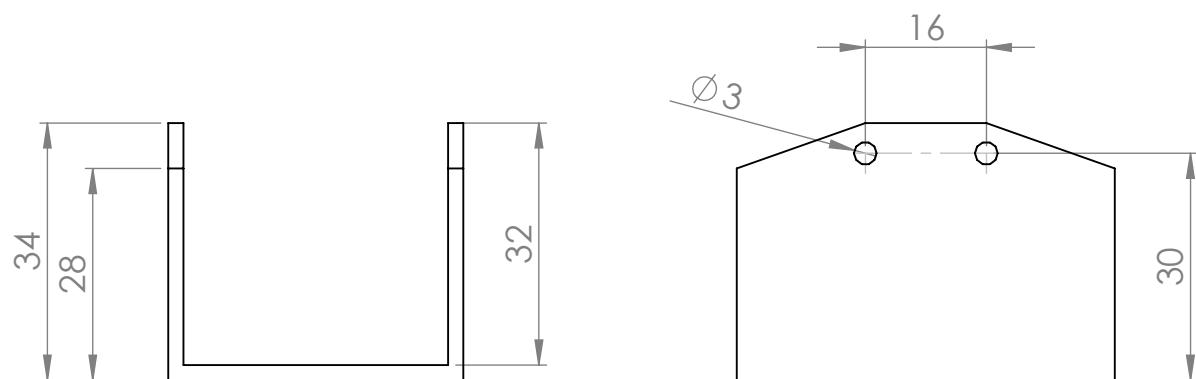
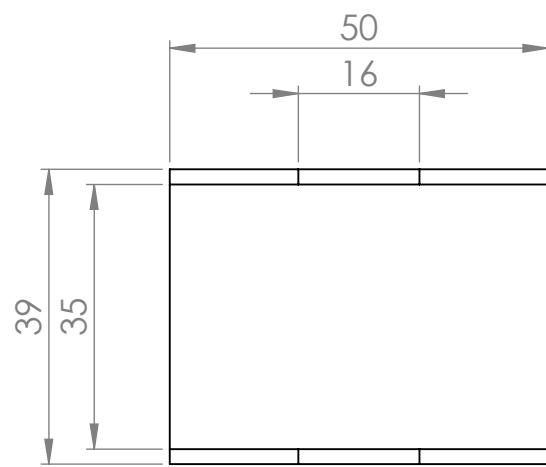
AUTOR: Javier Isabel Hernández

PLANO 15/18



ESCALA: 1:1

A4



ESCALA 1 : 2

TÍTULO: soporteBateria

PROYECTO: RAIDER (Robot Antropomórfico para la Investigación y el Desarrollo en entornos Reales)

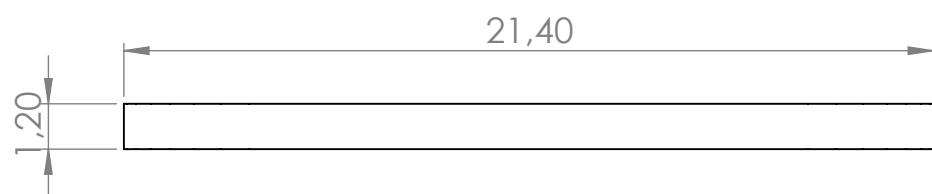
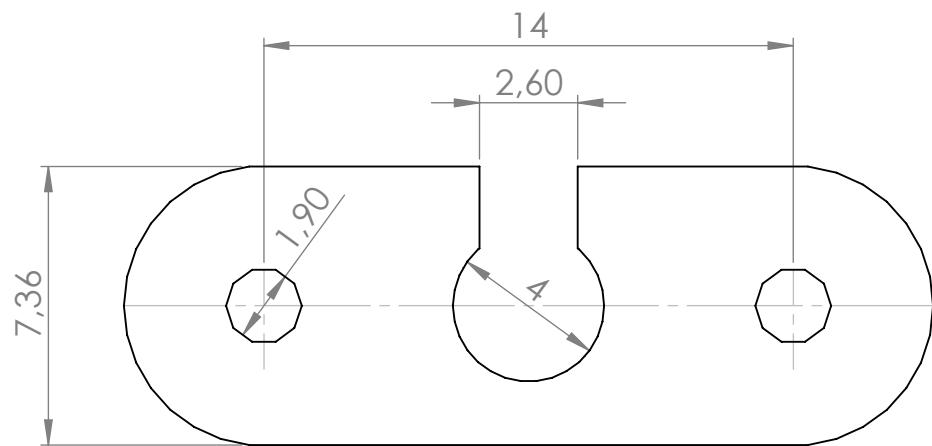
AUTOR: Javier Isabel Hernández



PLANO 16/18

ESCALA: 1:1

A4

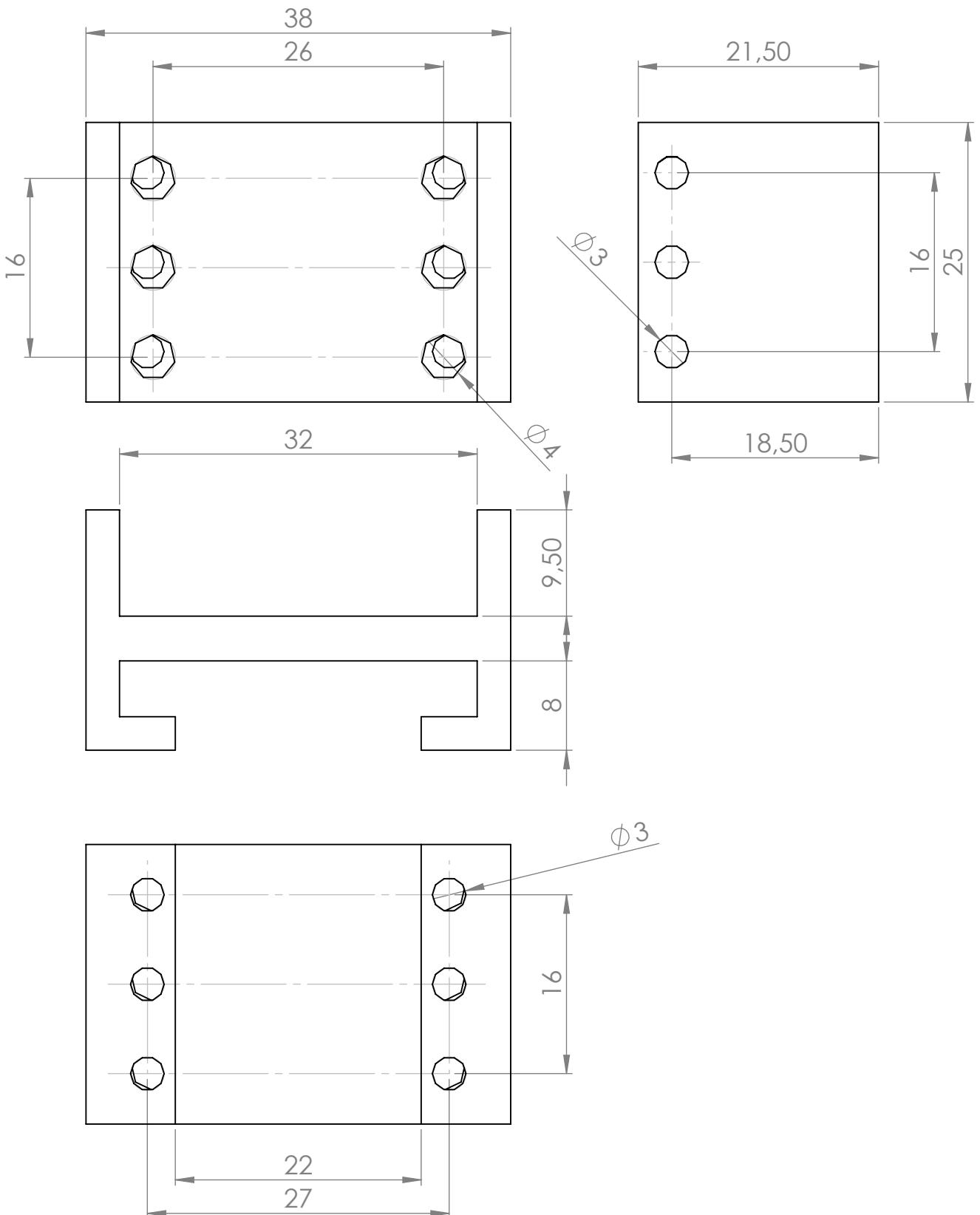


TÍTULO: tapaCabeza

PROYECTO: RAIDER (Robot Antropomórfico
para la Investigación y el Desarrollo
en entornos Reales)

AUTOR: Javier Isabel Hernández





TÍTULO: tobillo

PROYECTO: RAIDER (Robot Antropomórfico para la Investigación y el Desarrollo en entornos Reales)

AUTOR: Javier Isabel Hernández

PLANO 18/18



ESCALA: 1:1

A4