Contenido

[Software 1](#_Toc42795893)

[Instalación 1](#_Toc42795894)

[Driver 1](#_Toc42795895)

[Giroscopio 1](#_Toc42795896)

[Direcciones 2](#_Toc42795897)

[Permisos 2](#_Toc42795898)

[Equilibrio 2](#_Toc42795899)

[Ros 3](#_Toc42795900)

[Hardware 3](#_Toc42795901)

[Pines 3](#_Toc42795902)

[Función salida pwm 5](#_Toc42795903)

[Raspberry pi 5](#_Toc42795904)

[Movimiento 5](#_Toc42795905)

[Inteligencia Artificial 6](#_Toc42795906)

[Simulación 7](#_Toc42795907)

[Diseño 7](#_Toc42795908)

[Referencias 7](#_Toc42795909)

# Software

## Instalación

Para instalar las librerías necesarias es necesario tener Python3 instalado y pip3.

Un enlace para poder instalar pip3, se adjunta el siguiente tutorial:

<https://tecnonucleous.com/2018/01/28/como-instalar-pip-para-python-en-windows-mac-y-linux/>

## Driver

Como estamos usando el driver de la marca Adafruit modelo pca9685, la librería que hay que usar es: Adafruit\_PCA9685.

El enlace de la librería es el siguiente: <https://github.com/adafruit/Adafruit_Python_PCA9685>

## Giroscopio

Para el giroscopio mpu6050 la librería a utilizar será mpu6050-raspberrypi. El enlace de la librería es el siguiente: <https://github.com/Tijndagamer/mpu6050>

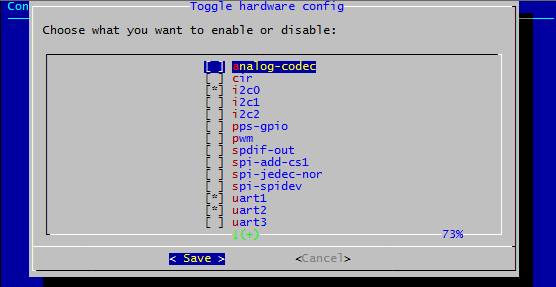
En versiones posteriores se comenzará a utilizar la librería creada por nosotros, que está en el repositorio con el nombre de mpu6050

## Direcciones

Como vamos a usar diferentes dispositivos conectados al puerto I2c de nuestra placa, vamos a necesitar saber qué dirección se asigna a cada uno.

Antes de comprobar en qué dirección se encuentra el driver hay que activar el protocolo i2c en la placa ya que por defecto viene desactivada.

Comprobamos si está activado con armbian-config. Menú “system settings” -> “Hardware” ->., en el caso de la Orange pi.



En este ejemplo se han activado i2c0 e i2c1, pero podemos activar i2c2 también.

Al tener que utilizar dos drivers, para el control de los servos, es necesario cambiar la dirección de cada uno con un soldador, de forma que no interfieran con el giroscopio, ni entre ellos.

Además, una vez hecho esto hay que actualizar el objeto robot en el código de forma que coincidan.

## Permisos

A no ser que ejecutemos nuestros programas desde el usuario root tendremos que darle permiso sobre los puertos i2c al resto de usuarios que no son root puesto que por defecto viene desactivado, una buena guía de cómo hacerlo es la siguiente <https://www.abelectronics.co.uk/kb/article/1061/i2c--smbus-and-armbian-linux>

## Equilibrio

Para poder equilibrar el robot cuando se encuentre en una posición neutral, se necesita un controlador pid. Debido a que el robot se puede inclinar en casi infinitas direccion, estas se van a reducir inicialmente dos, la primera va a ser de “adelante a atrás” y la otra “izquierda-derecha”. Al pid, se le va a pasar la inclinación deseada, en el caso de que se quiera equilibrar esta va a ser 0, y la inclinación actual, y va a proporcionar un ángulo que será el deseado.

Después dicho angulo hay que transformarlo a angulo de los servos, acción que se lleva a cabo en el módulo de control *equilibrar*

## Ros

Para una mayor facilidad se va a realizar el proyecto en ros

Clases:

Main: nodo encargado de controlar lo que se publica desde el nodo de comunicación, en el que se publican todos los mensajes recibidos por socket. Además se encarga de filtrar la información y publicar todo lo relacionado con el movimiento en el topic de control

Control: nodo encargado de mandar ordenes a los servos y de recibir la información del nodo equilibrio, y actuar en consecuencia.

Equilibrio: nodo que se suscribe a los valores del giroscopio y que publica predicciones de movimiento en función del ángulo al que se encuentre el robot

Servos: nodo que se suscribe al topic de control, y que en función del servo que se ordene mover instancia un objeto driver y otro

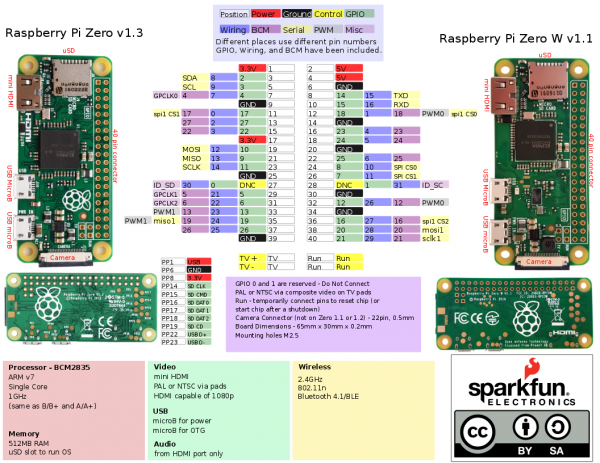
Driver: clase (que no nodo ros) que manda las ordenes directamente a los servos, nivel más bajo de movimiento

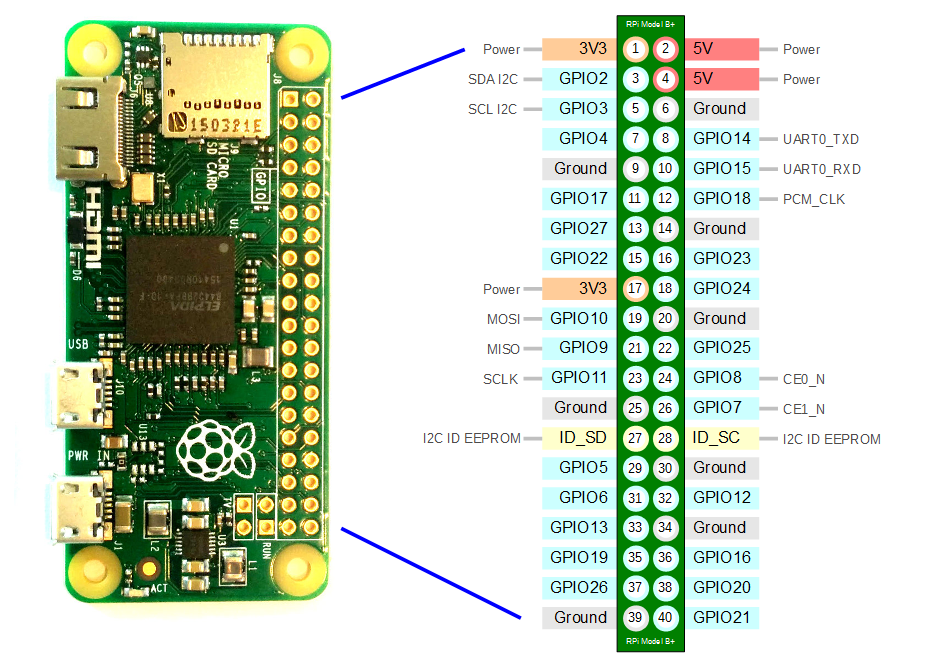
Comunicación: nodo de ros que publica en el topic comunicación los mensajes que se reciben desde el usuario. También se encarga de establecer la comunicación con el ordenador de control

# Hardware

## Pines

El driver que hemos cogido no trabaja con el puerto sda\_0 ni scl\_0, si no que la librería está preparada para trabajar con el pin scl\_1 y sda\_1. Si no la librería no encuentra el driver de forma correcta





## Función salida pwm

El driver escogido da un pulso se entre 450, que es su mínimo, mientras que 2100 parece ser un límite seguro para que los servos no se sobrecarguen. Por lo tanto, hay que hacer una función que permita recibir un ángulo deseado, y después pasarlo a un pulso pwm. E**l rango del pulso hay que comprobarlo experimentalmente.**

# Raspberry pi

USUARIO: pi

CONTRASEÑA: icai2019

Para poder utilizar la raspberry pi hay que cargarle una imagen de un sistema operativo, en este caso el elegido es raspbian por el soporte que da la comunidad, para ello hay que descargarse dicha imagen en el siguiente enlace <https://www.raspberrypi.org/downloads/>

Una vez cargada la imagen de raspbian, será necesario conectarse a la placa vía hdmi para poder configurar la conexión SSH.

Para establecer una ip fija, se sigue el siguiente tutorial:

<https://www.raspberrypi.org/forums/viewtopic.php?t=191140>

Una vez hecho esto hay que activar el ssh, en el menú de interfaces, así como el i2c, en el, mismo menú.

# Movimiento

Como la dirección de los servos se encuentra invertida, hay que restar el valor máximo de los ángulos para poder mover dos servos al mismo sitio (si se manda una mano bajar, que la otra también baje), para ello hay que dividir los servos entre derecha e izquierda. Se tomará el lado derecho como referencia, por facilidad. Se pondrán en posiciones impares los servos de la izquierda, y en posiciones pares los servos de la derecha, de acuerdo con la siguiente tabla:

|  |  |
| --- | --- |
| Servo | Pin |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |
| 17 |  |
| 18 |  |
| 19 |  |
| 20 |  |

Dicha tabla hay que actualizarla en el fichero json

## Ordenes

Para el desarrollo del combate se van a necesitar una serie de órdenes, las cuales se van a codificar con números, debido a que se tiene mayor rapidez al comparar números que String

|  |  |
| --- | --- |
| Adelante | 1 |
| Derecha | 2 |
| Atrás | 3 |
| Izquierda | 4 |
| Atacar | 5 |
| Defender | 6 |

# Inteligencia Artificial

Para controlar el robot se va a utilizar una inteligencia artificial, entrenada por medio de aprendizaje reforzado.

En un inicio se va a implementar una red neuronal de aprendizaje reforzado DQL, es decir a cada iteración se va a devolver una acción a realizar que solo va a ser aumentar, disminuir o mantener el ángulo de cada servo. Más tarde se planea avanzar hacia una red neuronal de tipo “continuo”

*Dicha inteligencia artificial será de tipo DDPG es decir Deep Deterministic Policy Gradient, debido a que nuestro espacio de acciones es prácticamente infinito (se pueden tomar muchas acciones para controlar el robot).*

El entorno que se va a montar es compatible con el tipo env, es decir se compatible con el estándar que estableció Google en sus inicios. Las funciones que debe tener el entorno son:

Step: realiza un paso en la simulación

Reset: resetea el entorno de simulación y todas las variables, será llamado cada vez que el robot toque el suelo, o que se empiece de nuevo la simulación.

State: que devuelve el estado actual del objeto. Más sobre los estados posibles adelante.

Reward: función que devuelve la recompensa que se le va a pasar al algoritmo, paso clave opuesto que determina a que velocidad y de que forma el algoritmo va mejorando.

Step: función que realiza un “paso” en el entorno, y que devuelve la recompensa que se ha obtenido con el paso. Además, devuelve el nuevo estado del entorno, y una flag que indica si se ha terminado de ejecutar el código. En nuestro caso siempre va a estar a false, puesto que queremos que la red este trabajando todo el rato

El estado que se va a pasar al algoritmo va a ser 29, con lo cual la dimensión de la entrada va a ser de 29. Las variables son:

Aceleración: array de tres elementos que contiene las aceleraciones en X Y Z

Posición: array con la inclinación del robot en todo momento

Angulo servos: devuelve el valor de todos servos (18 en principio)

Velocidad servos: contiene la velocidad de los servos

Torque ángulos: devuelve la fuerza que se está aplicando en cada servo, que además se utilizará como penalización en la AI

Caído: entero que devuelve si se ha caído el robot o no

Posición relativa pies: array de dos elementos que indica si los dos pies se encuentran en paralelo o no (copiado de Boston Dynamics)

Las marcadas con rojo están pendientes de revisión o de implementación en el código

## Simulación

Para realizar la simulación del humanoide, se va a realizar con el motor de simulación Bullet. Es gratis y bastante sencillo de utilizar.

Un tutorial de cómo utilizarlo es el siguiente: <https://docs.google.com/document/d/10sXEhzFRSnvFcl3XxNGhnD4N2SedqwdAvK3dsihxVUA/edit>

El workflow será el siguiente, primero una inteligencia artificial (modelo aún por determinar), realizará una “averiguación” de los motores a mover o pasos a realizar, después se pasarán dichos actos a la simulación, en pybullet. Una vez esto se devolverán las recompensas, y se volverá al inicio del programa.

<http://wiki.ros.org/urdf/Tutorials/Create%20your%20own%20urdf%20file>

# Diseño

En el diseño 3d los siguientes

# Referencias