

UNIVERSIDAD DE HUELVA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
PROYECTO FIN DE CARRERA

Banco de Calibración de Zowi

Autor:
Marcial RODRÍGUEZ

Coordinador:
Juan Manuel ENRIQUE

*Proyecto Fin de Carrera
para la titulación de Ingeniería Industrial*

realizado en

Grupo Robótica Industrial y Automatización
Extinto Dpto. Innovación y Robótica
BQ

28 de junio de 2017

Índice general

1. Introducción	1
1.1. Objetivo	1
1.1.1. Requisitos	1
1.2. Estructura del proyecto	1
2. Marco Teórico	3
2.1. Contexto	3
2.1.1. Zowi	3
2.2. Motivo del proyecto	4
2.2.1. Servos	4
2.2.2. Problema a resolver	6
2.2.2.1. Calibración	7
2.3. Posibles alternativas	8
2.3.1. Sensor IMU	8
2.3.1.1. Acelerómetro	8
2.3.1.2. Giróscopo	9
2.4. Herramientas y tecnologías utilizadas	9
2.4.1. Máquinas	9
2.4.2. Diseño	10
2.4.3. Programación	10
2.4.4. Post-desarrollo	10
3. Desarrollo	13
3.1. Introducción	13

3.2. Prototipo de validación de concepto	13
3.2.1. Cámara visión por computador	13
3.2.2. Sensores IMUs	14
3.2.2.1. Teoría	14
3.2.2.2. Prueba inicial	17
3.2.3. Conclusiones	18
3.3. Prototipo de laboratorio	18
3.3.1. Línea de evolución	18
3.3.1.1. Primeras mejoras	19
3.3.1.2. Tomando forma	21
3.3.1.3. Prototipo final: Banco de Calibración	22
3.3.2. Pruebas realizadas	25
3.4. Armarios finales	26
3.4.1. Componentes e implementación física	28
3.4.1.1. Arquitectura y conexiones	28
3.4.1.2. Armario eléctrico	28
3.4.1.3. Mega	29
3.4.1.4. Shield Mega	31
3.4.1.5. Raspberry PI 2	31
3.4.1.6. Power Boost	32
3.4.1.7. LCD Display	33
3.4.1.8. Otros componentes	33
3.4.1.9. Zowi - Zum	33
3.4.1.10. Banco soporte Final de Zowi y zapatos	34
3.4.2. Software	34
3.4.2.1. Mega	35
3.4.2.2. Zowi	38
3.4.2.3. Raspberry	39

4. Conclusiones y trabajos futuros	43
4.1. Conclusiones	43
4.2. Trabajos futuros	45
A. Manual de usuario	47
B. Planos Eléctricos: Armario 1	51
C. Planos Eléctricos: Armario 2	59
D. Mega PINOUT	67
E. Mega Custom Shield	69
F. Máquina de estados Mega	79
G. Configuración IMUS	85
H. Estructura base de datos	87
I. BOM de los armarios finales	89
J. Piezas Impresas	93
K. Fichero running.sh	97
L. Fichero ZowiInit	99
M. Autologin al inicio	101
N. Reglas puertos USB	103
Ñ. Notas de instalación Raspberry	105
O. Código Raspberry	109
P. Código Mega	125

Q. Código Zowi	157
R. Especificaciones Mega	165
S. Especificaciones Raspberry	169

Índice de figuras

2.1. Robot Zowi	4
2.2. Servo Futaba S3003	5
2.3. Diagrama de bloques servomotor	5
2.4. Posición según PWM	6
2.5. Eje dentado servo	7
2.6. Lectura acelerómetro	9
2.7. Lectura giróscopo	9
2.8. Hephestos	10
2.9. Witbox	10
2.10. Cyclone	11
3.1. BQ Zum Bluetooth empleada para mover los servos	14
3.2. Celda con cámara de Fanuc	15
3.3. Calibración tobillo con visión	15
3.4. Calibración cadera con visión	15
3.5. MinIMU-9 v3 de Pololu	16
3.6. Posiciones calibración IMU	17
3.7. Tobillo gira sobre eje Z	17
3.8. Cadera gira sobre eje Z	17
3.9. Sensor anclado con grapa impresa	19
3.10. Bancos impresos para Zowi y zapato	19
3.11. MinIMU-9 v3 Pin-out	20
3.12. Banco 1 zapato	20

3.13. Banco con inclinación regulable	21
3.14. Banco con conexión MEGA-ZUM	22
3.15. Arquitectura versión MEGA-ZUM por I ² C	23
3.16. Prototipo final Zowi SLS	23
3.17. Banco prototipo final	25
3.18. Prueba pre MP	26
3.19. Segundos por cada calibración de Zowi	27
3.20. OK:1 NOK:0 - para cada calibración de Zowi	27
3.21. Armarios finales preparados para su envío a fábrica	27
3.22. Esquema de conexiones	28
3.23. Armario final: Lateral	29
3.24. Armario final: Interior	30
3.25. Freaduino Mega2560	30
3.26. PCB Shield	31
3.27. Shield soldada	31
3.28. Raspberry PI 2 - 1GB Ram	32
3.29. Power Boost	32
3.30. Backpack para Display LCD	33
3.31. Banco soporte y zapatos finales	34
4.1. Puesto de calibración en la línea de montaje	43
4.2. Operadoras en línea de montaje	44
4.3. Línea de montaje de Zowi en Rosti	44

Índice de cuadros

3.1. Direcciones de esclavo I ² C para MinIMU-9 v3	20
4.1. Conclusiones y resultados	45

Capítulo 1

Introducción

1.1. Objetivo

Diseño y construcción de un sistema que forma parte de la cadena de montaje de un juguete comercial: el robot educativo Zowi, desarrollado por BQ. La principal función de éste sistema es la de ajustar, de forma automática, las posiciones de los servomotores del juguete. Adicionalmente, se implementan otras funcionalidades importantes tal como la descarga del software final en el controlador del robot.

1.1.1. Requisitos

Algunos de los requisitos marcaron el camino a seguir hasta la solución final, facilitando la toma de decisiones en diferentes puntos del proyecto. Las peticiones más relevantes para el diseño fueron las siguientes:

- Plazo de finalización fijado en 2 meses.
- Replicable fácilmente; a poder ser, por terceros.
- Utilizable por personal con poca o ninguna formación técnica.
- Cadencia aproximada de la línea de producción: 30-60 segundos.
- Deseable: fácil instalación.

1.2. Estructura del proyecto

La información se presenta de la siguiente forma:

- En éste Capítulo 1 se presenta una breve introducción del proyecto.
- En el Capítulo 2 se describe el motivo del proyecto, una pequeña descripción de las posibles soluciones y una base teórica sobre los principios y componentes más importantes del sistema, así como una pequeña mención a las tecnologías y herramientas que han sido útiles o necesarias para su desarrollo.

- El Capítulo 3 se centra en la línea de desarrollo, mostrando las diferentes etapas y prototipos por los que se ha pasado hasta llegar a la versión final, con una descripción de las funciones de los componentes electrónicos dentro del sistema y del software creado o utilizado.
- En el Capítulo 4 se sintetizan los resultados.
- En los anexos se recoge gran cantidad de la información del proyecto, conteniendo código, planos, esquemáticos o tablas de gran tamaño, entre otros. Será frecuente el uso de referencias a los anexos durante todo el documento.

Para el desarrollo de éste documento, se ha generado un repositorio en Github con dirección en <https://github.com/uborzz/proyecto-ind>. En éste repositorio se puede encontrar material adicional relacionado con el proyecto que no ha sido anexado, tal como datasheets, anotaciones, datos para análisis o vídeos de los prototipos, entre otras cosas.

Capítulo 2

Marco Teórico

2.1. Contexto

Una de las líneas de desarrollo de BQ es la robótica educativa, un conjunto de productos y servicios orientados a educar en las tres partes de todo proyecto tecnológico: Diseño, hardware y software. Cuenta para ello con diferentes productos (impresoras 3D -Witbox, hephestos-, “Mi primer kit de robótica”, o los printbots), programas (Bitbloq -basado en scratch-), así como contenido y soporte en la web (“Do it with others” - www.diwo.bq.com).

El presente proyecto surge de la necesidad de automatizar una parte del proceso de fabricación de un nuevo robot educativo, Zowi. Hasta ahora, los robots desarrollados (printbots, por su carácter de chasis y componentes no-electrónicos totalmente imprimibles en 3D) usaban para moverse servos de rotación continua y ruedas, Zowi es el primer robot bípedo desarrollado por BQ (utiliza dos servos de posición para los piés y otros dos para las caderas) y la librería de osciladores sinusoidales -Oscillator-, desarrollada por Juan “Obijuan” González, entonces jefe del departamento, para robots serpiente modulares.

2.1.1. Zowi

Zowi, también llamado el robot de Clan TVE, es un robot educativo para niños que les inicia en el mundo de la tecnología y de la programación. Se puede ver el juguete en la Figura 2.1.

Este robot, movido por una controladora basada en Arduino y diseñada a medida para él, además de los servos que lo convierten en un robot bípedo, incorpora unos sensores de ultrasonidos como ojos que le permiten detectar obstáculos, una matriz de leds como boca para expresar “emociones” y tres pequeños botones para alternar entre sus modos de funcionamiento, entre otros componentes.

Se ha creado una aplicación móvil, Zowi App, que permite controlar los movimientos de Zowi y reprogramarlo fácilmente con diferentes juegos. Zowi tiene muchas posibilidades más allá de lo predefinido en su aplicación. BQ tiene una plataforma online para programar a sus robots y controladoras, esta plataforma se llama Bitbloq. En Bitbloq se puede programar a Zowi y a otros tipos de placas Arduino de

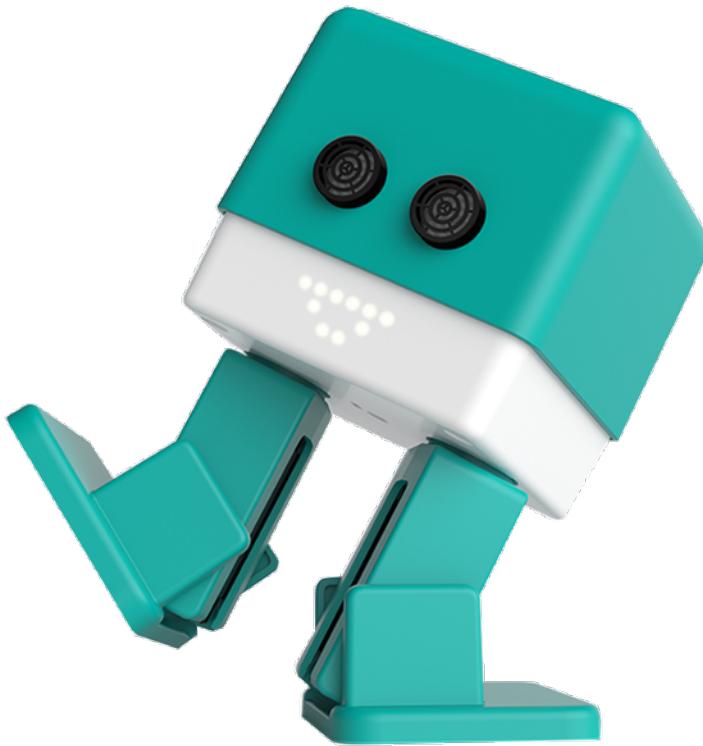


FIGURA 2.1: Robot Zowi

forma muy sencilla por medio de bloques, sin necesidad de saber ningún lenguaje de programación. La opción de emplear código directamente también está disponible.

2.2. Motivo del proyecto

La tarea asignada al autor del proyecto fue la de idear y desarrollar el prototipo de un sistema capaz de ajustar de forma automática cada uno de los cuatro servos que componen las piernas de Zowi, lo que permite al robot ejecutar sus programas de fábrica, con los que saldrá a la venta, de forma correcta; entre ellos: realizar diferentes bailes, caminar, etc. Dicho sistema debe ser utilizado por operadores sin conocimientos de Arduino ni de electrónica en general. La nacionalidad de los operarios también es desconocida por no saberse el emplazamiento de la fábrica hasta más avanzado el proyecto (razón por la que parte de la documentación anexada se encuentra en inglés: menú de la interfaz, comentarios del código, etc).

2.2.1. Servos

Los servos son un tipo especial de motor DC que se caracterizan por su capacidad para posicionarse de forma inmediata en cualquier posición dentro de su intervalo de operación. Para ello, el servomotor espera un tren de pulsos que se corresponde

con el movimiento a realizar. Están generalmente formados por el motor, un sistema reductor formado por ruedas dentadas, un potenciómetro y un circuito de realimentación, todo en una misma caja de pequeñas dimensiones. En la Figura 2.2, se muestra un servomotor del tipo empleado en Zowi.



FIGURA 2.2: Servo Futaba S3003

El circuito electrónico de realimentación del servo es el encargado de recibir la señal PWM y traducirla en movimiento del Motor DC. El eje del motor DC está acoplado a un potenciómetro, el cual permite formar un divisor de voltaje. El voltaje en la salida del divisor varía en función de la posición del eje del motor DC. En la Figura 2.3 se muestra un diagrama del funcionamiento de un servo de posición.

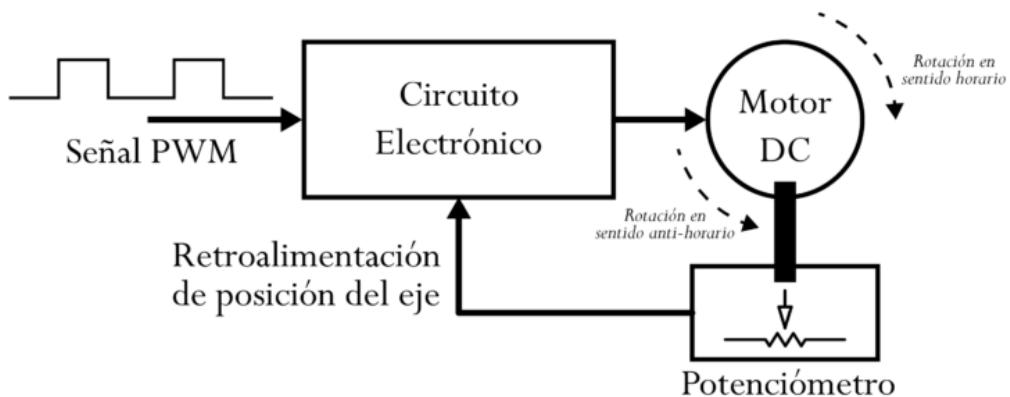


FIGURA 2.3: Diagrama de bloques servomotor

La modulación por anchura de pulso, PWM (Pulse Width Modulation), es uno de los sistemas más empleados para el control de servos. Este sistema consiste en generar una onda cuadrada en la que se varía el tiempo que el pulso está a nivel alto, manteniendo el mismo período, con el objetivo de modificar la posición del servo según se desee. Arduino permite generar PWM en algunos de sus pines de

forma sencilla, además, para hacerlo aún más fácil, tiene la librería *servo.h*, que nos permite dar un valor de posición angular (grados) a un determinado pin, y él sólo genera el PWM.

Las señales de PWM requeridas por el circuito de control electrónico son similares para la mayoría de los modelos de servo. Esta señal tiene la forma de una onda cuadrada. Dependiendo del ancho del pulso, el motor adoptará una posición fija. Un ejemplo visual en la Figura 2.4.

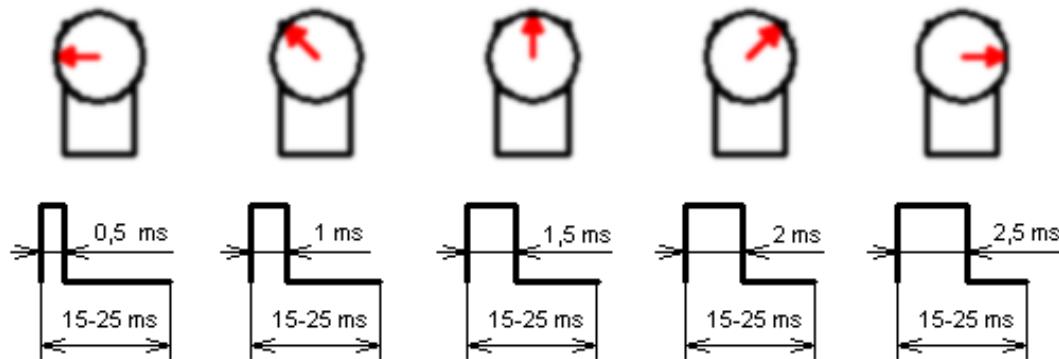


FIGURA 2.4: Posición según PWM

2.2.2. Problema a resolver

Los servos empleados son una versión clónica del Futaba S3003 que vimos en la Figura 2.2, con un rango de funcionamiento algo mayor (en torno a 193°). Para el caso de Zowi, el rango empleado será bastante menor, por cuestiones de estabilidad del juguete y por el diseño de las piezas, mecánicamente sería imposible torcer las articulaciones en todo el rango del servomotor.

La continuación del eje del servo, eje dentado, engrana en la pieza de Zowi. Como podemos ver en la Figura 2.5 a modo ilustrativo, hay un determinado número de dientes en dicho engranaje; para el servo empleado en Zowi, hay 24 dientes en el engranaje, por lo que un diente supone un cambio de 15° respecto al anterior.

El ensamblaje de los servos en sus piezas correspondientes se hace de forma manual, se hace que el servo llegue a hacer tope en uno de los sentidos como referencia y es engranado de forma que tenga disponible el rango de funcionamiento necesario. Sin embargo, dentro de los 15° que se comentaban anteriormente, y por la misma fabricación de las partes del servo, las piezas podrían quedar montadas con un desfase de hasta 7.5 grados, o mayor aún si el operador no elige bien la orientación al engranar. Adicionalmente, por el fabricante del servo, la relación entre tren de pulsos PWM y la posición final del servo puede variar -estar un poco desplazada-, incluso variar muy ligeramente entre servos del mismo fabricante.

Se tienen entonces varios factores que hacen que cada servo sea montado en una posición distinta -o considerada distinta por el programa- y tenga que ser configurado por software, además, el ajuste se adaptará para cada servo, no se puede definir una regla válida para todos, de ahí que se emplee un método iterativo para solucionarlo.



FIGURA 2.5: Eje dentado servo

Lo ideal sería que la posición del servo intermedia coincidiese con la posición neutra de la articulación de Zowi, dejando la mitad del rango de giro del servomotor para cada sentido. Sin embargo, como el rango de movimiento del juguete es menor que el rango completo del servo, se tiene un margen bastante generoso que se puede corregir en el ajuste.

2.2.2.1. Calibración

La calibración de los servos es de importancia vital para un buen funcionamiento de las funciones programadas en el juguete y en cualquier aplicación que utilice servos, en general. En Arduino, utilizando la librería *srvo.h*, se consiguen generar los pulsos PWM de forma transparente, empleando directamente grados sexagesimales (números enteros). El primer paso para utilizar un servo es siempre definir qué posición es la de partida del servo, es decir, fijarle un 0° . Ésto se realiza guardando un valor (“trim” u “offset”) que podrá ser positivo o negativo y se empleará para corregir las orientaciones.

Cuando el usuario quiere mover el servo a una posición determinada utilizando la librería *srvo.h* directamente, tendrá que moverlo a la posición deseada más el valor de “offset”. El offset obviamente siempre será el mismo para cada servo, salvo que se desmonte y monte el servo en otra posición o en otra pieza y se tenga que recalibrar. Este ajuste es habitual cuando se trabaja con servos, en robótica o radiocontrol por ejemplo.

Sabemos que Zowi es un producto pensado para ser reprogramado y “trasteado”, para que el usuario aprenda sobre electrónica y programación; sin embargo, éste usuario no tiene por qué tener ningún conocimiento previo en el momento de adquirir el robot, es por ésto que Zowi sale a la venta con programas por defecto, programas que muestran sus posibilidades. Zowi tendrá los valores de “offset” de los servos, calculados para su montaje en fábrica, guardados en determinadas direcciones de la EEPROM, zona de la memoria de la placa controladora que no se borra al ser reprogramada por el usuario.

2.3. Posibles alternativas

Anteriormente, se ha explicado cuál es el problema a resolver. Para darle solución, la clave es poder leer las orientaciones reales de los servos, es decir, las orientaciones respecto al mismo juguete; en base a estas posiciones se debe actuar sobre los servos hasta colocarlos en la posición neutra y guardar en la memoria el desfase observado para cada servo. Se plantean dos posibles caminos:

- **Visión por computador:** Emplear un sistema de visión para, mediante matching y detección de líneas, observar el desvío y poder actuar sobre los servomotores hasta llevarlos a la posición deseada.
- **Sensor IMU:** Utilizar utilaje impreso y sensores de tipo inercial (acelerómetro-giróscopo) para calcular las inclinaciones producidas por la posición de los servos.

Se elige la segunda opción por tener una estimación de tiempo de desarrollo más ajustada, además de un coste considerablemente menor.

2.3.1. Sensor IMU

Se describe brevemente la utilidad del sensor IMU ya que todo el sistema se construye en torno a este componente.

Una IMU, o unidad de medición inercial, es un sensor diseñado para combinar las características de un acelerómetro y un giróscopo, y mostrar información completa sobre aceleración, posición, orientación, velocidad...

Las medidas crudas de los componentes del acelerómetro, giróscopo y magnetómetro (encapsulado junto al acelerómetro) estarán directamente relacionadas con la fuerza G, velocidad angular y campo magnético, respectivamente. Dichas medidas pueden ser procesadas y tratadas con un microcontrolador o microprocesador para obtener posiciones, orientaciones, etc.

2.3.1.1. Acelerómetro

El acelerómetro mide la aceleración en las 3 dimensiones del espacio. La gravedad de la Tierra tiene una aceleración perpendicular al suelo. Así pues, la IMU también detecta la aceleración de la gravedad terrestre. Gracias a ésta se pueden usar las lecturas del acelerómetro para saber cuál es el ángulo de inclinación respecto al eje X o eje Y. Figura 2.6.

Dado que el ángulo se calcula a partir de la gravedad, no es posible calcular el ángulo Z empleando únicamente el acelerómetro. Para hacerlo se necesita otro componente: el magnetómetro, que es simplemente un tipo de brújula digital.

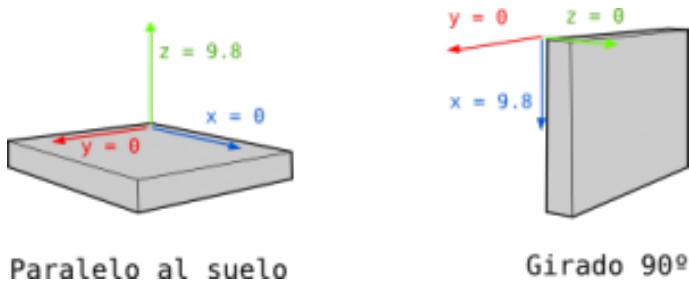


FIGURA 2.6: Lectura acelerómetro

2.3.1.2. Giróscopo

El giróscopo es un dispositivo que mide la velocidad angular en cada uno de los ejes. Figura 2.7.

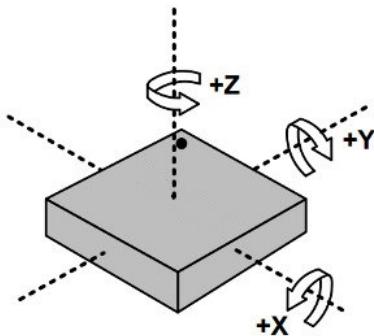


FIGURA 2.7: Lectura giróscopo

2.4. Herramientas y tecnologías utilizadas

Además de los componentes electrónicos que se han introducido anteriormente y los que se tratarán en el capítulo de desarrollo: Capítulo 3, cabe mencionar las tecnologías y herramientas que han resultado útiles durante todo el proyecto.

2.4.1. Máquinas

La tecnología de impresión 3D por FFF ha aportado agilidad al proyecto al permitir la creación de piezas en el mismo lugar en cuestión de horas. Se han empleado máquinas **Hephestos** (Figura 2.8) y **Witbox** (Figura 2.9) de BQ, ambas con firmware **Marlin**, para la creación del utilaje provisional y componentes menores como los soportes de las placas dentro del armario o la caja que contiene el display LCD y el zumbador.

Otra máquina interesante fue la fresadora **Cyclone** (Figura 2.10), entonces en fase beta. Se montó y empleó para el primer prototipo de la PCB de la shield desarrollada.



FIGURA 2.8: Hephestos

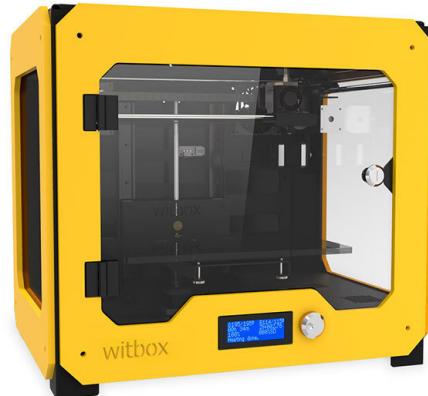


FIGURA 2.9: Witbox

2.4.2. Diseño

Para el diseño de las piezas impresas se ha utilizado **Inventor** y **FreeCad**, el laminado se ha llevado a cabo con **Cura**.

El diseño de la PCB y la generación de los gerbers se logró mediante **KiCad**. Su fresado fue posible gracias a **FlatCAM** y **CNCCodeController**.

Los planos eléctricos se han creado con **DraftSight**.

2.4.3. Programación

El desarrollo de los programas de Arduino y la programación de las controladoras se ha llevado a cabo empleando el mismo IDE de Arduino clásico: **Arduino 1.0.6**.

El sistema operativo instalado en Raspberry ha sido la versión adaptada de Debian: **Raspbian**, la versión concreta, última a la fecha: Raspbian Wheezy.

El programa principal de la Raspberry ha sido desarrollado en **Python 2.7.11**, última versión de Python 2 a la fecha. Ha sido útil el editor **Atom** y su capacidad de acceder a los directorios remotos directamente, mediante SFTP.

2.4.4. Post-desarrollo

Para acceso remoto, troubleshooting y extracción de los datos se han utilizado las siguientes herramientas:



FIGURA 2.10: Cyclone

- **Putty** para las conexiones SSH.
- **Filezilla** y **WinSCP** para acceso remoto a ficheros.
- **HeidiSQL** para acceso a la base de datos.

La documentación del proyecto se ha realizado en **L^AT_EX** con **Atom** y **TeXworks**.

Capítulo 3

Desarrollo

3.1. Introducción

En el presente capítulo se exponen las distintas vías propuestas para abordar la resolución del problema, con el desarrollo de unos prototipos para validación de concepto. Se muestra, también, la evolución del prototipo hasta la versión funcional de laboratorio, así como los detalles de la versión final desarrollada y empleada en la línea de producción.

3.2. Prototipo de validación de concepto

Para dar solución al problema planteado, inicialmente se realizan dos pruebas con tecnologías diferentes. Por un lado, se mostró una versión basada en visión por computador. Por otra parte, se presentó una solución alternativa basada en sensores giróscopo-acelerómetro.

Para ambas pruebas, se cuenta con un prototipo impreso de Zowi con una placa controladora: BQ Zum BT (ver Figura 3.1), versión derivada de Arduino UNO con bluetooth incorporado y una etapa de potencia capaz de sacar hasta 3.2 amperios por sus salidas digitales. Tanto el juguete final como su placa controladora se encuentran en desarrollo a la fecha de estas pruebas.

3.2.1. Cámara visión por computador

Se realiza una prueba piloto aprovechando el sistema de visión instalado para otro proyecto, compuesto por la cámara fija de Fanuc, 2 paneles LED de luz difusa (60W) y el software de visión Fanuc: iRvision. Se elige realizar las pruebas en esta plataforma y no en un ordenador convencional, con Matlab u OpenCV, por inmediatez, ya que el sistema empleado está calibrado y en funcionamiento. El objetivo de esta prueba era comprobar la viabilidad de este tipo de solución.

El sistema de visión está conectado al controlador de un brazo robótico LRMate 200iD, con un módulo I/O digitales a 24 voltios. Por lo que se realiza una interfaz con 2 relés para comunicar con la placa Arduino (5V), encargada de calibrar los servos del juguete.

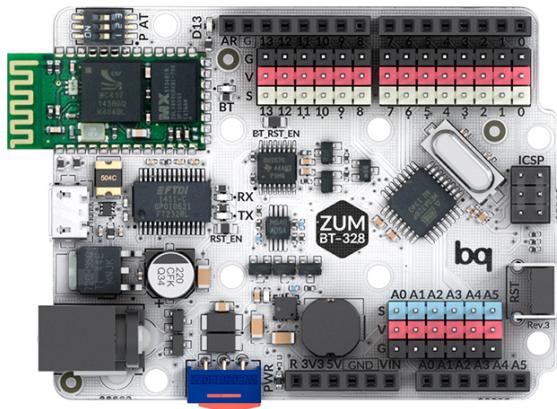


FIGURA 3.1: BQ Zum Bluetooth empleada para mover los servos

En la figura 3.2 se puede ver el área de trabajo de la celda robótica y su cámara.

El programa creado en Arduino recibe como entradas dos señales que indican en qué sentido se debe girar el servo, si es que necesita modificar su posición. Como salida, actúa sobre el servo haciéndolo corregir su posición iterativamente, 1 grado cada segundo, hasta alcanzar la posición deseada.

La posición deseada es definida utilizando iRVision (Software de visión de Fanuc), mediante técnicas de visión por computador (Localización de elementos entrelazados, medición de distancias y medición de ángulos entre los elementos). Para el caso del “tobillo”, buscando perpendicularidad con la “cadera”; y para esta, consiguiendo cierta distancia entre las dos líneas paralelas. Se pueden ver las Figuras 3.3 y 3.4 para más detalle.

3.2.2. Sensores IMUs

Para la segunda prueba se emplea una placa que combina un giroscopio, acelerómetro y magnetómetro para formar una unidad de medición inercial (de ahora en adelante IMU) con una precisión de hasta 0.1 grados.

3.2.2.1. Teoría

La MinIMU-9 v3 (Figura 3.5) fue seleccionada de entre varias opciones, como la famosa MPU6050, la GY-87 o la versión v2 de MinIMU (todas de un precio similar), por la precisión y repetibilidad de las lecturas, además de parecer un dispositivo ampliamente utilizado y probado por la comunidad.

Tras estudiar el comportamiento del sensor, y una fase de documentación acerca de las opciones de que se dispone, surgen tres posibilidades:

- Librería de Pololu “MinIMU-9 AHRS”

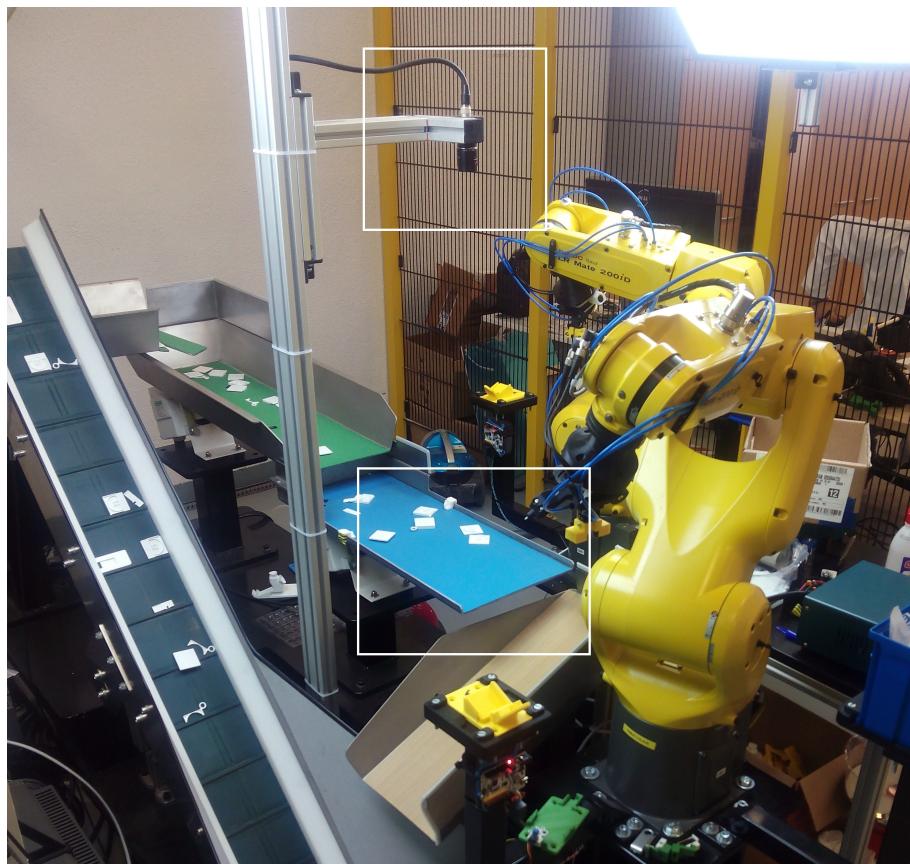


FIGURA 3.2: Celda con cámara de Fanuc



FIGURA 3.3: Calibración tobillo con visión



FIGURA 3.4: Calibración cadera con visión

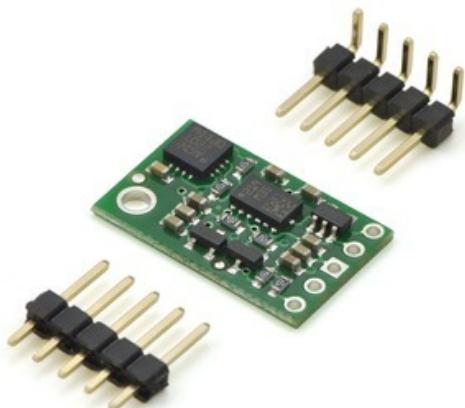


FIGURA 3.5: MinIMU-9 v3 de Pololu

- Librería “Open IMU”
- Desarrollar la matemática tomando los valores RAW del sensor

A pesar de lo atractivo de la tercera opción, no se disponía de suficiente tiempo para seguir adelante con ella, se trató de comprender el funcionamiento de las 2 librerías ya creadas.

Por simplicidad, Open IMU resultaba llamativa, pero rápidamente se percibió que podía quedarse demasiado corta y que las lecturas obtenidas empleando esta librería (los ángulos euler de pitch, roll y yaw) parecían no tener la repetibilidad necesaria para nuestra aplicación, por otro lado, hacía más de 3 años que no se realizaban cambios en su repositorio GIT, por lo que posiblemente estaba preparada para funcionar con la versión anterior de la placa, MinIMU-9 v2, con una exactitud de hasta 1 grado solamente.

La línea elegida fue la de la librería recomendada por el fabricante, Pololu, dejando un poco limitado el modo de funcionar pero ofreciendo un acceso rápido a las mediciones del sensor en forma de posición en ángulos euler.

Para un correcto funcionamiento del magnetómetro, se han de configurar los valores máximos y mínimos de las lecturas del acelerómetro, estos 2 sensores se encuentran en el encapsulado LSM303, la librería de dicho chip proporciona un programa de Arduino que captura dichas mediciones y salva la mayor mientras se mueve la placa en todas las orientaciones posibles, mostrándola por el puerto serie. Los valores obtenidos se han de escribir, sustituyendo los valores “por defecto”, en la cabecera de la misma librería.

La limitación de esta librería es que, para obtener con precisión los ángulos euler en todas las orientaciones, se debe llamar a una función de la librería, **calibrateIMU0**, con la placa colocada en una de las 2 posiciones que se ven en la Figura 3.6, lo más paralelo al suelo posible. Con esta función se define en el espacio el 0 de los ejes X e Y del sistema, por lo que se corrigen desvíos si la posición no es totalmente horizontal; sin embargo, esto deriva en errores que crecen en proporción a la inclinación que se de en alguno de los ejes. Se elige una de ellas configurando ciertos parámetros de la librería de Pololu en el programa principal.

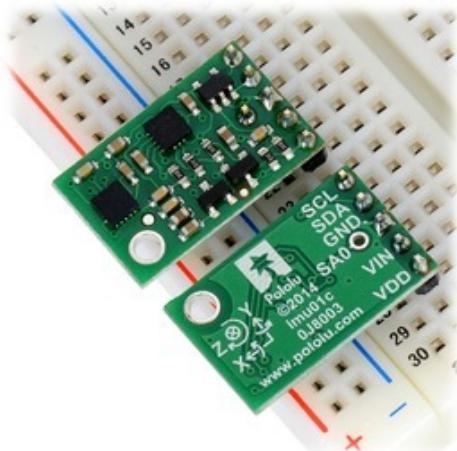


FIGURA 3.6: Posiciones calibración IMU



FIGURA 3.7: Tobillo gira sobre eje Z



FIGURA 3.8: Cadera gira sobre eje Z

La orientación en Z depende de la brújula y su tiempo de estabilización es lento. Por esta razón, se piensa en emplear los giros en los otros 2 ejes, X e Y, de manera que la posición del robot ha de ser la de acostado sobre un lado. Se descartan las posiciones que, de entrada, parecen más razonables: boca arriba/boca abajo, o con las piernas hacia arriba. Ver Figuras 3.8 y 3.7.

3.2.2.2. Prueba inicial

Se programa el algoritmo empleando una máquina de estados en la misma placa controladora de Zowi con un pulsador para pasar de un estado a otro. Versión muy básica para probar el concepto. Se emplea comunicación serie para conocer el estado del sensor y una grapa impresa para acoplarlo al pie de Zowi.

El procedimiento programado sería:

- Iniciar el sistema – Automáticamente se produce la calibración de la IMU

- Colocar el sensor en el pié de Zowi
- Pulsar el botón – Se inicia la calibración del servo

El modo de calibración del servo consiste en dar un primer valor a la posición que debería tener el servo, es decir, 0° para la cadera o 90° para el pie, entonces se lee el sensor para ver el error cometido y se ajusta la posición del servo teniendo en cuenta la diferencia del valor deseado y la nueva lectura, se repite hasta 2 veces.

Se obtienen buenas sensaciones con este método y se nota de inmediato que el éxito del mismo radica en la correcta calibración inicial del sensor y en una buena fijación al pié del robot.

3.2.3. Conclusiones

El método de calibración empleando visión muestra buenos resultados y es muy mejorable si se emplea software creado específicamente para la aplicación, sin embargo, las condiciones de iluminación y posición han de ser las mismas o muy parecidas para cada banco de trabajo para garantizar un buen funcionamiento del software, por lo que su instalación, puesta en marcha y configuración, son delicados y se deberían hacer in-situ. En el momento de las pruebas de validación aún se desconocen las ubicaciones de la fábrica y la nave de montaje. La solución se encarecería considerablemente si se han de realizar desplazamientos largos para llevar a cabo la instalación y puesta en marcha.

Con las IMUs se obtuvieron buenos resultados y se tenía bastante claro que se quería seguir adelante con ellas por su bajo coste y su fácil implementación. Pendiente quedaba afinar el algoritmo de calibración, implementando una máquina de estados mucho más robusta y endureciendo el criterio de validación automática de una buena calibración, para obtener mejores resultados. En el siguiente subcapítulo se desarrolla su evolución.

3.3. Prototipo de laboratorio

Se decide continuar con la solución de las IMUs. En la presente sección se mostrará la línea de evolución del prototipo, que pasa por diferentes fases, mencionando las características y mejoras más relevantes añadidas en cada una de ellas, así como las pruebas realizadas antes de pasar al desarrollo de la versión final.

3.3.1. Línea de evolución

Tras elegir la línea de los sensores IMU, se comienza a trabajar de inmediato en la mejora de la solución.

3.3.1.1. Primeras mejoras

La máquina de estados es mejorada, diferenciando calibraciones malas de calibraciones buenas. Se añaden un indicador led y uno acústico para resaltar el resultado con diferentes sonidos. Esta mejora se mantendrá hasta la versión final del producto.

La siguiente necesidad a cubrir sería garantizar que el robot comparte el plano con el sensor, que ha de ser calibrado antes de colocarse en el robot como se ha explicado en la sección anterior. Hasta ahora el sensor era fijado utilizando una grapa impresa (Ver Figura 3.9).

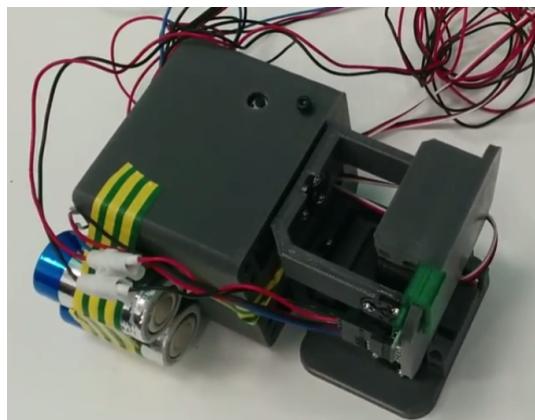


FIGURA 3.9: Sensor anclado con grapa impresa

Se monta el sensor en una pieza especialmente diseñada e impresa para colocarla en el pié de Zowi, razón por la que es llamada “Zapato”. Se crea también una base donde se coloca a Zowi, con un área para colocar el zapato en el momento de su inicialización, dónde el sensor es calibrado. Ambos soportes están integrados en la misma pieza, de esta forma se garantiza la correspondencia y paralelismo de los planos de apoyo de ambas partes (robot – zapato). Se puede ver el banco en la Figura 3.10.

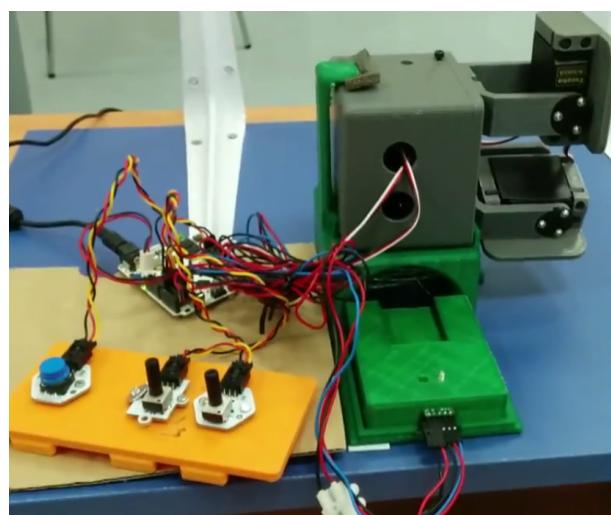


FIGURA 3.10: Bancos impresos para Zowi y zapato

El algoritmo hasta ahora pasaba por calibrar una pierna del juguete solamente, haciendo ambas partes -tobillo y cadera-. Se mejora este punto, conectando a la placa controladora 2 sensores IMUs. La comunicación con la MinIMU-9 se realiza a través de I²C, y su interfaz permite seleccionar fácilmente entre 2 direcciones distintas para el giroscopio, y lo mismo para el acelerómetro/brújula. Para ello solamente es necesario conectar SA0 a GND. Ver Figura 3.11 y Cuadro 3.1.

Sensor	Slave Add. - default	Slave Add. - SA0 driven low
L3GD20H (gyro)	1101011b	1101010b
LSM303D (acc+magnet)	0011101b	0011110b

CUADRO 3.1: Direcciones de esclavo I²C para MinIMU-9 v3

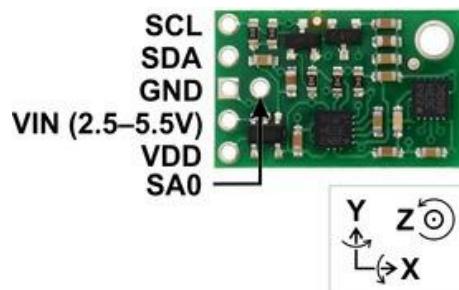


FIGURA 3.11: MinIMU-9 v3 Pin-out

Es necesario cambiar las direcciones de una de las placas, de lo contrario ambas tendrían las mismas direcciones de esclavo I²C y tratar de leer de ellas causaría un mal funcionamiento. Además, se mejora el banco impreso para poder albergar los 2 zapatos; se mejora también la máquina de estados, implementando la calibración de ambas piernas, aumentando número de iteraciones e incluyendo un filtrado de múltiples medidas para evitar leer ruido. Ver Figura 3.12.

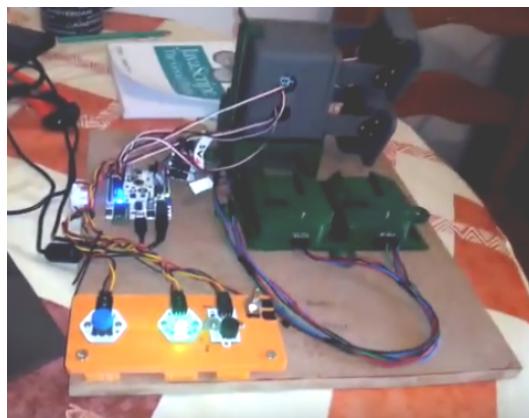


FIGURA 3.12: Banco impreso para Zowi y 2 zapatos

La calibración/inicialización de los sensores muestra resultados no tan buenos cuando el sensor se inclina notablemente. Por ello que el éxito de la calibración de la cadera (0° de set point) sea mayor que el de la calibración del pie (90°). Se considera corregir el set point de los pies en la fase de calibración de los sensores.

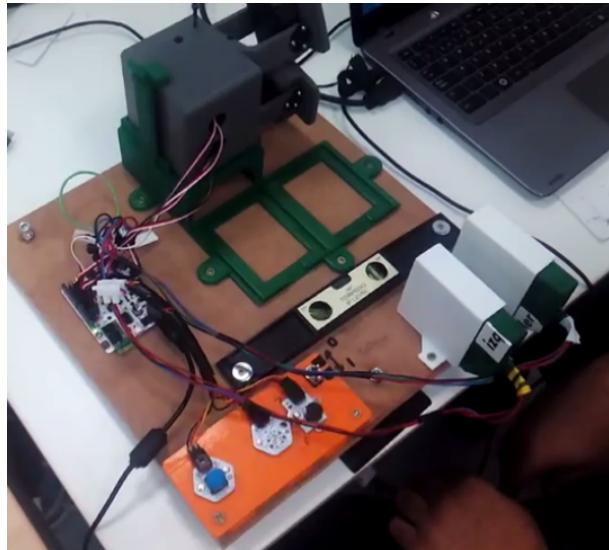


FIGURA 3.13: Banco con inclinación regulable

Se diseñan unos cajetines para el ajuste del sensor a los 90°, de modo que antes de comenzar la calibración de los juguetes, el sensor (zapato) debe ser calibrado en las posiciones horizontal y vertical.

Se impide avanzar en el procedimiento si, tras calibrar los sensores al inicio, las medidas obtenidas difieren demasiado de 0° en ambos ejes. En caso de no avanzar, se vuelve a un estado en el que se espera pulsar el botón para volver a realizar la calibración, es decir, se comienza a implementar una gestión de errores.

Se añaden al conjunto un nivel y unos tornillos para regular la inclinación y mantener todas las partes del sistema lo más cerca posible a una posición completamente en horizontal. Ver Figura 3.13.

3.3.1.2. Tomando forma

En este punto todas las pruebas se habían realizado en la misma controladora de Zowi. Se recibe información sobre la placa controladora que irá instalada en los robots y se conoce que el número de entradas/salidas será reducido y ajustado a los sensores/actuadores del propio juguete, por lo que realizar todas las conexiones que se plantean en nuestras pruebas en la misma placa junto a todos los periféricos de Zowi no es una opción.

Se plantea el uso de otra placa conectada por I²C a la placa controladora de Zowi y la librería Wire.h de Arduino.

La nueva placa empleada sería una Freeduino Mega, versión basada en la Arduino Mega, es elegida por ser compatible con todo lo desarrollado, por su bajo coste y por presentar una cantidad de entradas y salidas mayor que los modelos UNO.

Se vuelven a diseñar unos bancos para calibrar los sensores tanto horizontal como verticalmente, es decir, inicializar ejes y ajustar el eje Y en vertical. Todo en el

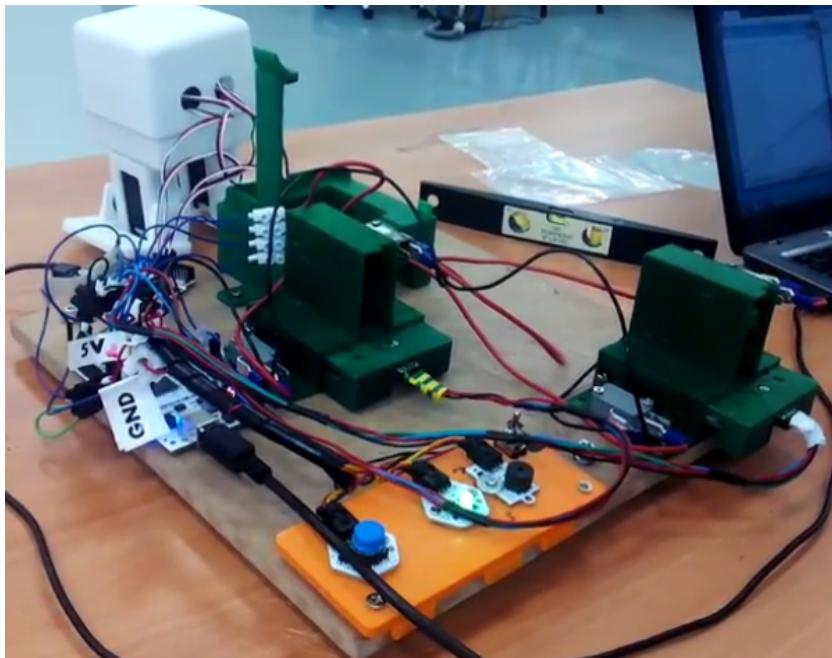


FIGURA 3.14: Banco con conexión MEGA-ZUM

mismo espacio, como se puede ver en la Figura 3.14.

El empleo de una Mega nos proporciona una gran cantidad de pines de entrada salida, se incorporan al sistema unos finales de carrera para detectar que los zapatos están en la posición correcta para su calibración.

Como característica **principal**, se diseña una interfaz de comunicación entre ambas placas mediante I²C, se construyen determinadas “instrucciones” en la placa Mega (placa con la máquina de estados) y un programa que funciona como intérprete que es cargado en la ZUM (placa del robot), se encarga de traducir las instrucciones recibidas de la otra placa por I²C en movimientos de servos, entre otras cosas. Puede verse la arquitectura del sistema en la Figura 3.15.

La máquina de estados se hace más robusta en general, detectando fallos y facilitando la recuperación para un correcto funcionamiento.

Al final de esta etapa se conoce, además de la controladora que llevará el juguete, la versión definitiva mecánica de Zowi, un prototipo sinterizado por láser ya con sus dimensiones y forma finales. Se puede ver en la Figura 3.16.

3.3.1.3. Prototipo final: Banco de Calibración

En este punto se produce el mayor cambio en el proyecto. La solución de comunicación por I²C entre las diferentes placas asumía que la controladora de Zowi tendría accesibles los pines al micro para I²C y SPI y, además, hacía necesario el acceso al interior del robot. El robot podría llegar ya montado de las etapas de producción anteriores. Se sugiere implementar la comunicación a través del puerto USB que tendrá la placa de Zowi.

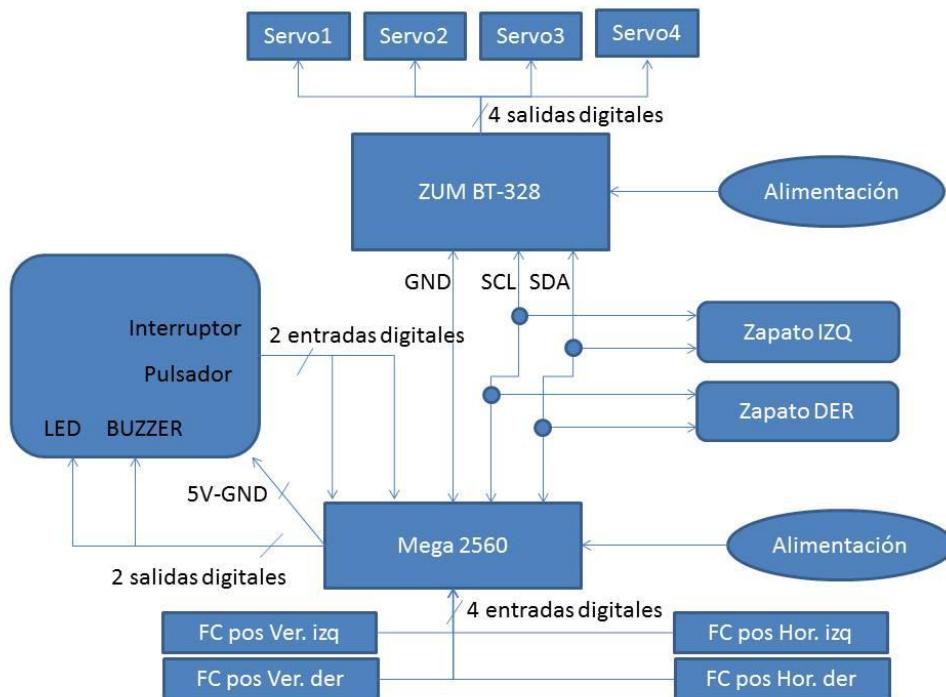
FIGURA 3.15: Arquitectura versión MEGA-ZUM por I²C

FIGURA 3.16: Prototipo final Zowi SLS

Las controladoras Arduino pueden ser “esclavos” serie por USB pero no maestros, realizar una comunicación serie de una a otra no es posible utilizando el cable USB, de ahí que se plantee usar una Raspberry PI (Figura 3.28). Un ordenador, con el driver correcto, podrá establecer una conexión serie por USB con ambas placas gracias al FTDI de cada controladora, que convierte de USB a UART. Se sustituye entonces la interfaz I²C implementada en la versión anterior por una interfaz serie con la Raspberry haciendo de puente entre ambas Arduinos.

Se contempló la posibilidad de quitar la Mega y emplear solamente la Raspberry como núcleo del sistema, sin embargo, eso conllevaría rehacer gran parte del trabajo y encontrar la forma de hacer funcionar los sensores con este dispositivo, lo que requería gran cantidad de tiempo.

Una primera iteración fue la de sustituir la comunicación por I²C, por la comunicación serie, se adaptaron los programas de MEGA y de ZUM. La comunicación sería unidireccional. Mega >Raspberry >Zowi. La Raspberry PI funcionaría como mero driver de comunicaciones:

- Se modificaron los comandos enviados con la librería Wire.h desde la free-duino Mega para ser comandos enviados por serie (Librería Serial.h) con ciertos caracteres de inicio y fin de comunicación a modo de protocolo (desarrollado específicamente para esta aplicación).
- Se implementó en la Raspberry un programa en Python que hacía de intérprete. Haciendo escucha a la comunicación serie abierta con la Freeduino Mega, y enviando comandos a la placa controladora de Zowi a través de la comunicación Serie abierta con ella.

La incorporación al sistema de una Raspberry PI, un ordenador al fin y al cabo, abría gran cantidad de mejoras de fácil implementación, por lo que se aprovechó la ocasión para desarrollar un prototipo “casi definitivo”, no solamente a nivel de software.

Se decide introducir la electrónica en un armario eléctrico y, a su vez, emplear el mismo como estructura del soporte de Zowi y de los zapatos. Ver Figura 3.17. Se integran nivel y patas regulables.

Se introdujo un display LCD conectado a la Mega por I²C que hacía más fácil la interacción del usuario con el banco de calibración y se trabajó en mejorar la máquina de estados para hacer el uso del sistema más amigable; y se mejoró el conexionado añadiendo una PCB custom como shield de la Mega, que ofrecía borneros para conectar los leds, display LCD, IMUs, finales de carrera y botones.

Durante el proceso de calibración se guardarán los offsets calculados en ciertas posiciones de la memoria EEPROM de la placa controladora de Zowi. Se usa para ello la librería EEPROM.h. Esto es necesario para el funcionamiento de los programas por defecto, cuyas librerías (zowi.h y oscillator.h) están configuradas para usar los valores en dichas posiciones de la memoria.

Multitud de mejoras software definitivas fueron implementadas en esta fase para aumentar su funcionalidad, en el subcapítulo siguiente se verán en mayor detalle, a destacar:

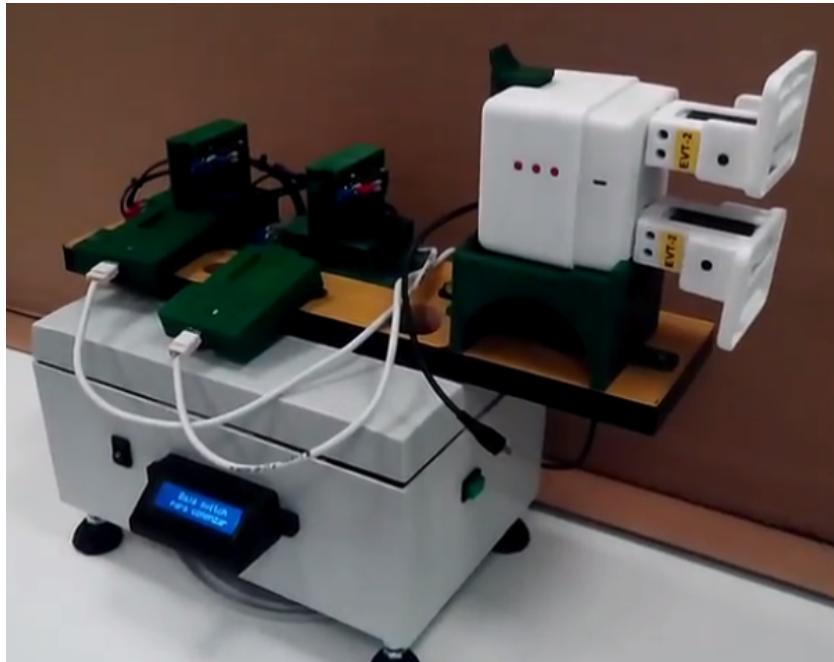


FIGURA 3.17: Banco prototipo final

- Raspberry utilizada como programador, empleando la utilidad AVRDUDE cargará el programa de calibración en cada Zowi.
- Cargará, del mismo modo y al finalizar la calibración, el programa de fábrica con que saldrá a la venta el producto. Sobrescribiendo el de calibración.
- Se le habilita acceso remoto por SSH y por XRDП para poder acceder a ella desde fuera.
- Se instala un servidor de MySQL y se configura para guardar datos tanto en local como en nuestros servidores.
- El entorno de programación de Arduino es instalado, para poder modificar la Mega directamente desde Raspberry.

3.3.2. Pruebas realizadas

Durante todo el desarrollo se han realizado pruebas, sin embargo, se comentan los resultados la prueba previa a MP (massive production), realizada con el prototipo final. La Figura 3.18 muestra el armario de calibración preparado para la prueba.

Los operadores fueron compañeros que no habían tenido contacto alguno con la máquina, haciendo de operarios de forma distraída.

Se logran calibrar satisfactoriamente 62 juguetes, con solamente 1 contratiempo: Al no respetar el protocolo en cuanto a orden de operaciones, concretamente, intento de programar la placa sin energizar. Si se intenta programar la placa conectada pero sin encender, el programador AVRDUDE realiza hasta 10 intentos de programarla sin éxito; al energizarla durante los intentos, según en qué momento se haga, puede ser recuperable, o puede llevar al sistema a un estado de fallo. Se tuvo que reiniciar y



FIGURA 3.18: Prueba pre MP

volver a hacer la calibración de los zapatos. A pesar de ser un error de operador, se intenta corregir para la versión final.

Un total de 4 veces se produce una calibración mala, inducidas para poner a prueba el sistema, el sistema notifica (Display y sonido) la calibración como mala y se necesita que se vuelva a pulsar el botón. De ahí una calibración de apenas unos segundos en las gráficas siguientes, por no tener que conectar y calzar al juguete).

Analizando la información extraída de la base de datos tras las calibraciones, se muestra (Figura 3.19) Segundos vs. nº Zowi, una segunda gráfica (Figura 3.20) muestra 1 para calibraciones válidas, 0 para las declaradas fallidas. Los tiempos del primer y último registro no son representativos.

3.4. Armarios finales

El trabajo asignado al autor del proyecto debió haber concluido con el prototipo presentado en el subcapítulo anterior, dejando por diseñar una versión definitiva de los soportes y zapatos del banco al Dpto. de Mecánica de BQ, y el montaje de tantos armarios y bancos como fuese necesario para abordar la producción de todos los juguetes a Rosti (empresa de extrusión de plásticos y montaje subcontratada en Polonia), encargados del ensamblado de los robots Zowi; sin embargo, se decidió realizar el montaje de 2 armarios con ligeras modificaciones y planos eléctricos para enviar a Polonia ya probados y validados. Ver armarios en Figura 3.21.

Modificaciones destacables respecto al prototipo:

- Armario de componentes eléctricos separado de la parte mecánica, se siguieron utilizando los diseños impresos de versiones anteriores para validación del

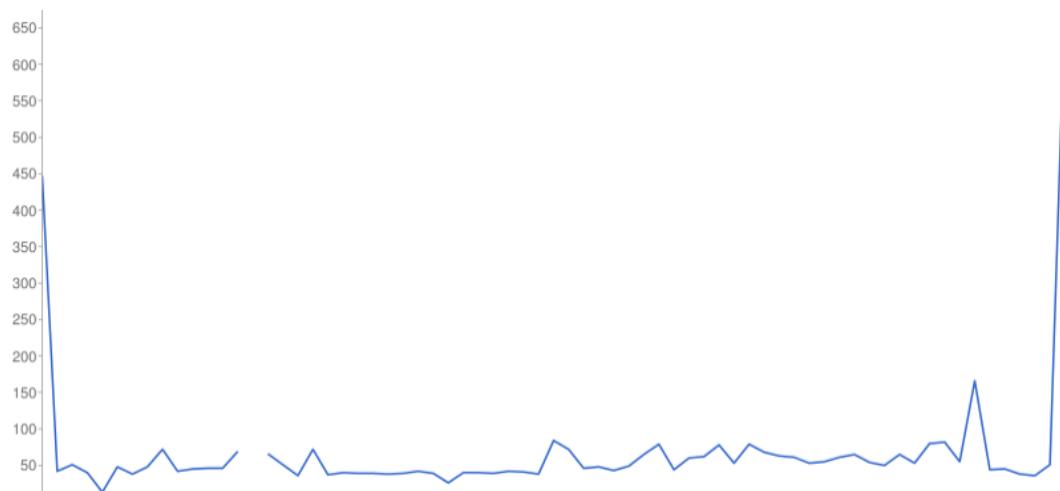


FIGURA 3.19: Segundos por cada calibración de Zowi

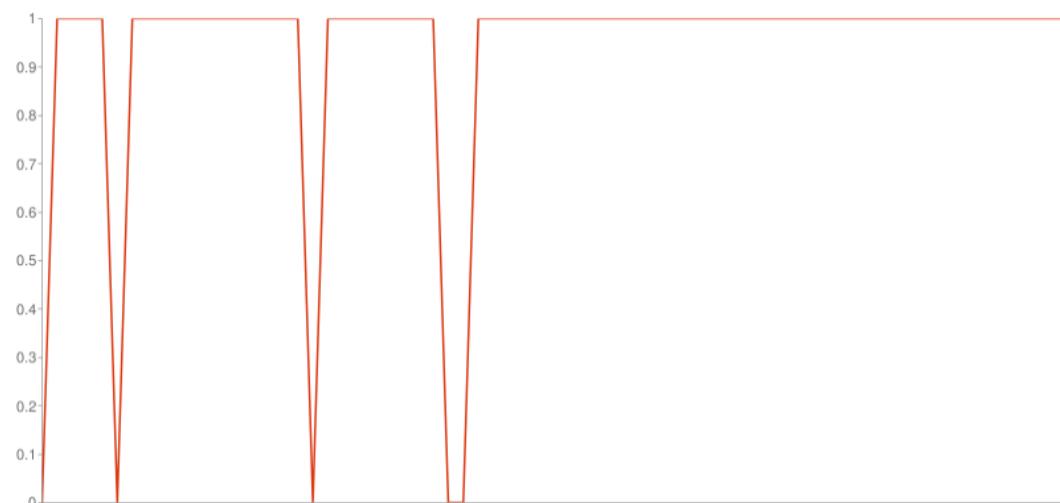


FIGURA 3.20: OK:1 | NOK:0 - para cada calibración de Zowi



FIGURA 3.21: Armarios finales preparados para su envío a fábrica

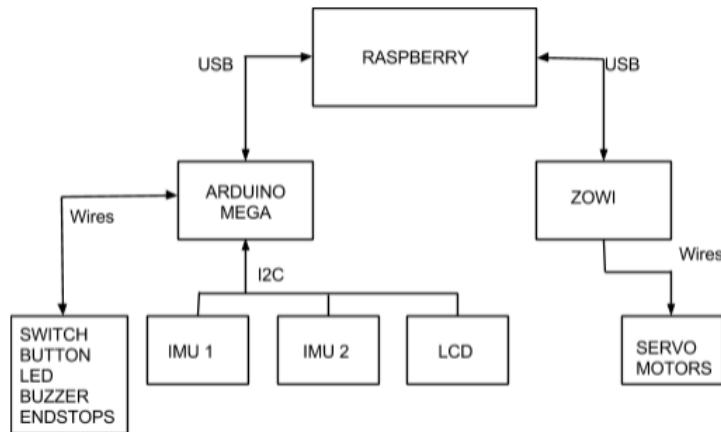


FIGURA 3.22: Esquema de conexiones

funcionamiento, pero las piezas finales quedaban a cargo del Dpto. de Mecánica. Este nuevo armario sería más amplio que el anterior.

- Mangueras de cables más largas (pues armario y banco de calibración irán separados).
- Pulsador exterior con una manguera de cable para ser colocado donde se desee en fábrica, en lugar del pulsador en el lateral del armario de la versión anterior.
- Conexión USB y Ethernet por pasamuros al interior del armario.
- Cambios en el diseño del cajetín del display LCD, pues el armario irá colocado a la altura de los ojos del operador.

3.4.1. Componentes e implementación física

3.4.1.1. Arquitectura y conexiones

El diagrama que se puede ver en la Figura 3.22, muestra, a grandes rasgos, un esquema de las conexiones entre los principales dispositivos que intervienen en el sistema. El sistema completo consta del armario eléctrico, además del juguete Zowi conectado al armario. Por ello se muestran en el esquema tanto las placas internas del armario (Mega y Raspberry), como la placa controladora de Zowi (Zum).

3.4.1.2. Armario eléctrico

Se elige instalar la parte fija del sistema en una placa de montaje dentro de un armario eléctrico de 40x30x18cm. Se emplean canaletas y bornas en un carril DIM para facilitar las conexiones al exterior del armario. Se utilizan pasamuros para las mangueras a pulsador y sensores IMU, así como para conexión USB (Raspberry - Zowi) y RJ45.



FIGURA 3.23: Armario final: Lateral

En el carril DIM se instala una toma de Schuko, al que se conectará la Raspberry PI para alimentar toda la electrónica. El armario es energizado por un conector a red eléctrica JR-101 con interruptor y fusible.

Se pueden ver las Figuras 3.24 y 3.23, interior y lateral del armario, respectivamente. Para detalle de las conexiones al clemero, se pueden consultar los planos eléctricos en el Anexo B.

3.4.1.3. Mega

Esta placa hace de cerebro del sistema a pesar de ser alimentada a través de la Raspberry, y de comunicar a través de ella con Zowi. Se pueden consultar las especificaciones en el Anexo R.

Recibe las lecturas de los sensores IMU, tiene conectados los finales de carrera, interruptor, pulsador, panel LCD, buzzer y leds. Todas las conexiones cableadas de entrada/salida, así como la alimentación pasan por un shield diseñado para esta aplicación.

Define una máquina de estados que mueve al operador por todos los pasos de la calibración y controla a la raspberry mediante un protocolo de instrucciones implementado para este fin que usa como canal el puerto serie por USB, también empleado para hacer llegar instrucciones a la placa de Zowi.

Se puede decir que Mega es la encargada de interactuar con el exterior y, dentro de nuestro sistema, realiza las funciones de un PLC. Para ello se ha elegido cuidadosamente cómo conectar los diferentes componentes a sus entradas/salidas, en el Anexo D se puede consultar el pin-out de esta placa:



FIGURA 3.24: Armario final: Interior

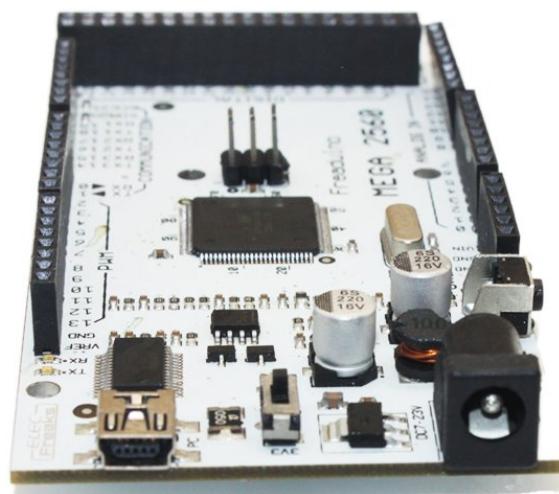


FIGURA 3.25: Freaduino Mega2560

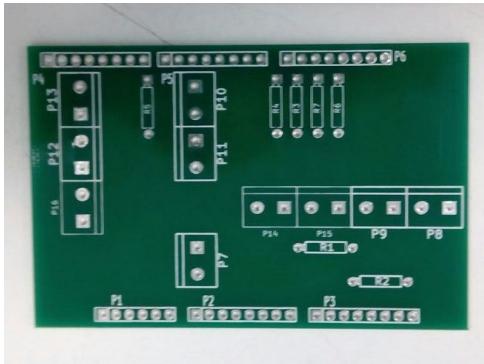


FIGURA 3.26: PCB Shield



FIGURA 3.27: Shield soldada

- Leds y buzzer necesitan PWM para poder configurar los sonidos/colores.
- Los pulsadores y finales de carrera se han conectado a entradas que tienen asociadas resistencias de pull-up, conectadas fácilmente por software.
- Se ha alimentado por Vin en lugar de por USB para poder alimentar a todos sus dispositivos.

Información sobre software en la Subsección 3.4.2. Para conocer con detalle las conexiones se pueden consultar los documentos del Anexo B.

3.4.1.4. Shield Mega

Se emplea el diseño de la shield de la versión prototipo del armario. En este caso se mandan a fabricar las PCBs, ya que nuestra fresadora no dió unos resultados tan profesionales.

Esta shield es colocada sobre los pines de la Mega y nos proporciona unos borneros para facilitar la conexión cableada a los dispositivos de entrada y salida, así como a la alimentación.

Los esquemáticos y gerbers con el layout del diseño de la PCB se pueden consultar en el Anexo E. Para información más clara sobre conexiones, se pueden consultar los planos eléctricos de un armario, Anexo B.

3.4.1.5. Raspberry PI 2

Este ordenador (Figura 3.28) aporta muchísimas mejoras de fácil implementación al sistema, además de su función principal, hacer de enlace entre Mega y la controladora de Zowi. Estas mejoras serán definidas en la Subsección 3.4.2, apartado dedicado al software. Se pueden consultar las especificaciones en el Anexo S.

Comunica con Mega y con las controladoras de los Zowi (Zum), para ello utiliza protocolo serie por cable USB con cada una de las placas. Además, por Ethernet, se puede acceder a ella por SSH (puerto 22) y por XRD (puerto 3389) desde la misma red local o desde internet configurando propiamente el NAT; y una IP y puerto

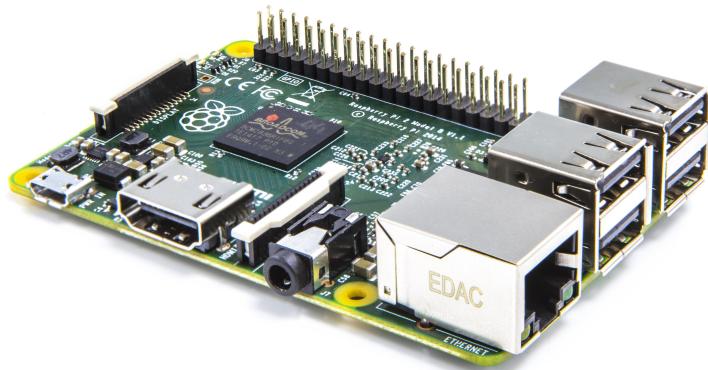


FIGURA 3.28: Raspberry PI 2 - 1GB Ram

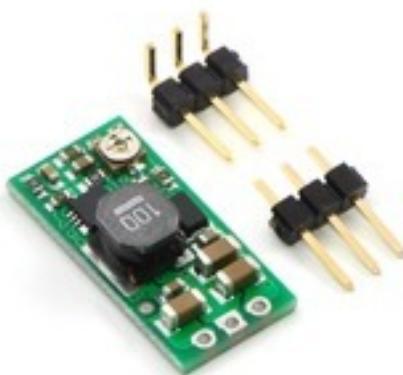


FIGURA 3.29: Power Boost

pueden ser configurados para guardar datos de calibración contra una base de datos remota.

Es alimentada por su toma microUSB, a través de un transformador conectado al Schuko del armario. Se encarga de alimentar y programar por USB a los Zowi, para lo que ha sido necesario desbloquear el límite de corriente suministrada por puerto USB por el sistema operativo (Se necesita suficiente potencia para mover los servos). También alimenta a la Mega por su pin de 5V+, para lo que ha sido necesario elevar dicha tensión. Una vez más, se pueden ver los planos eléctricos para mayor detalle, Anexo B y Anexo C.

3.4.1.6. Power Boost

La Mega debe ser alimentada a más de 7V. Para reducir número de transformadores y el espacio ocupado, se decide alimentarla a través de la raspberry.

Este dispositivo (Figura 3.29) acepta una tensión de entrada de 1.5V a 25V y, mediante el potenciómetro, permite ajustar su salida a valores entre 4V y 25V.

El power boost es ajustado para elevar la tensión suministrada por pines de la

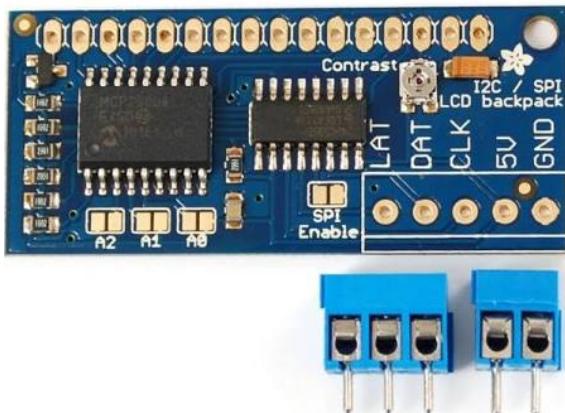


FIGURA 3.30: Backpack para Display LCD

Raspberry de 5V a 9V para alimentar la Mega. Necesario para no tener que dar la corriente por el puerto USB. Aún desbloqueando el límite software de intensidad de la Raspberry, con lo que se consigue hasta 1.2A a 5V, el sistema no funciona correctamente al alimentar por USB ambas placas, Zowi carga su batería y mueve sus servos utilizando dicha corriente. De modo que se alimenta la Mega (y todo lo conectado a ella) desde el pin de 5v de Raspberry, sin usar el puerto USB y sin tener que introducir otro transformador en el armario.

3.4.1.7. LCD Display

Una pantalla es instalada para indicar al operador instrucciones e información sobre el uso del sistema. Se elige un display de 16 filas x 2 columnas y un backpack de Adafruit (Figura 3.30) que nos permite comunicar con el display utilizando I²C en lugar de una buena cantidad de salidas digitales, y que funciona con librerías más que probadas hechas por la comunidad.

3.4.1.8. Otros componentes

Además de los componentes principales comentados anteriormente, y de los sensores IMU tratados en el Marco Teórico (Capítulo 2), se emplean otros componentes menores como son los leds, el buzzer, los botones, finales de carrera, así como sus resistencias, ademas de las clemas, puntas, y más. En los planos eléctricos (Anexo B) y hoja de costes (Anexo I), se puede consultar más información sobre su conexión o modelo.

3.4.1.9. Zowi - Zum

Como se ha comentado anteriormente, Zowi también forma parte del sistema. Su controladora tendrá conectados, entre otras cosas, los 4 servomotores responsables de mover las articulaciones. Comunica solamente con la Raspberry PI utilizando protocolo serie por cable USB. La Raspberry le descargará un software para poder

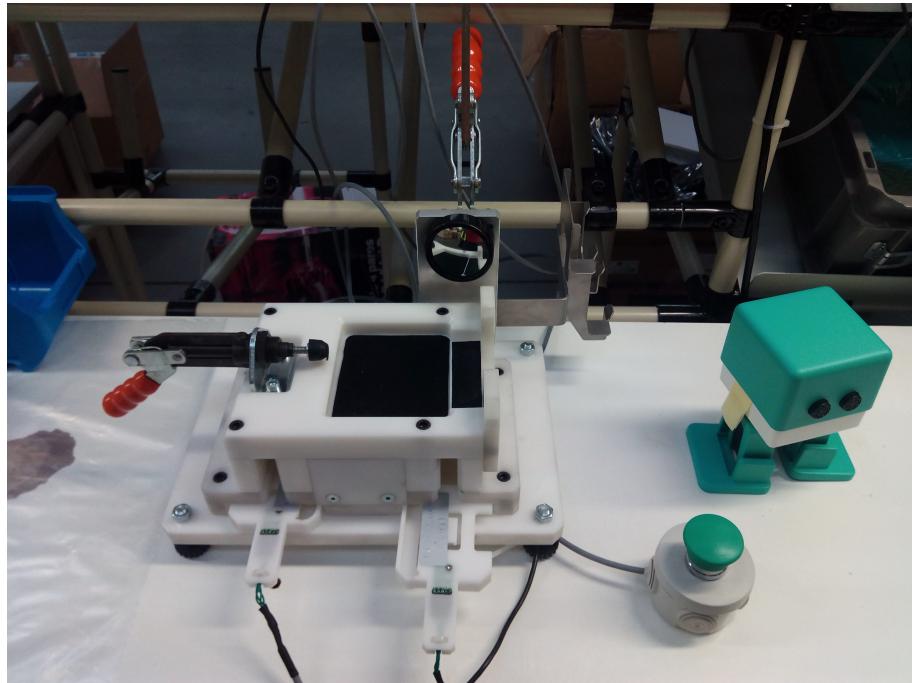


FIGURA 3.31: Banco soporte y zapatos finales

interpretar las instrucciones recibidas, y ser capaz de mover los servos o guardar información en su EEPROM.

3.4.1.10. Banco soporte Final de Zowi y zapatos

Para validaciones y durante todo el desarrollo del proyecto -así como para su uso en Madrid, tras acabar la fase de producción en fábrica- se han empleado, y emplean, los diseños realizados por el autor, cuyas vistas se pueden consultar en el Anexo J. No se especifican cotas de tolerancias por estar pensados para fabricar con impresoras 3D de filamento fundido, donde las dimensiones finales varían por parámetros como la temperatura o la velocidad de extrusión de la máquina.

Sin embargo, el utilaje empleado en cadena de montaje (Figura 3.31) fue rediseñado por el Dpto. de Mecánica y fabricado por sinterizado selectivo por láser, para lograr un acabado más preciso y mayor resistencia. Le añadieron palancas para fijar cada juguete y mejoraron los zapatos para facilitar la colocación.

3.4.2. Software

En esta subsección se explicarán los aspectos relacionados con la programación del sistema y los algoritmos empleados. Se ha dividido en tres partes, una para cada uno de los tres dispositivos principales, Mega, Raspberry y Zowi. Puede ser útil, para entender el funcionamiento del sistema, consultar los anexos A, O, P y Q, donde se muestra el Manual de Usuario y el Código desarrollado.

3.4.2.1. Mega

Este dispositivo se puede programar utilizando el lenguaje de Arduino, que es una adaptación de C++ que proviene de avr-libc y provee de una librería de C de alta calidad para usar con GCC en los microcontroladores AVR de Atmel y muchas funciones específicas para los MCU AVR de Atmel. Su simplicidad y la implicación por parte de la comunidad hacen realmente fácil y rápido su uso y aprendizaje.

El programa principal ha sido creado en el entorno de Arduino, en un fichero (.ino). Se emplean las funciones de Arduino precocinadas y se extiende de librerías de Arduino, programadas en C++, para todo lo posible, a destacar:

- LiquidCrystal para escribir al display LCD.
- Wire para la comunicación I²C.
- Ficheros de la librería MinIMU, que extiende a su vez de las librerías del giroscopio y el acelerómetro, L3G.h y LSM303.h, respectivamente.

El programa de Mega sigue una estructura de máquina de estados que se encarga de llevar al operador por los diferentes pasos del proceso de calibración, es aquí dónde se toman las lecturas de las posiciones de los servos y se decide cuánto se han de mover, dónde se valida el resultado de la calibración y dónde se produce la interacción entre la máquina y el usuario. Se puede decir que es el principal dispositivo del sistema.

Para conocer la lógica de la máquina de estados, se pueden consultar los diagramas del Anexo F, para conocer el proceso de uso, como operador, se puede consultar el manual de usuario del Anexo A.

3.4.2.1.1 Protocolo de comunicación

Para lograr enviar instrucciones a la controladora del robot utiliza como intermediario a la Raspberry, que establece una comunicación serie por USB con ambas controladoras. Se desarrolla un protocolo de instrucciones que tiene las siguientes partes:

- Signo de dólar (\$) como carácter inicial de la trama.
- 4 caracteres que indican el comando función en cuestión.
- Dos puntos (:) para indicar inicio de valores de parámetros.
- Parámetros de la función, si los tuviera, separados por el carácter (*).

Instrucciones desde Mega:

- **IZUM:** Comando que indica a Raspberry que cargue el programa calibración en Zowi.

- **ROFC:** Comando leer offset de las articulaciones de Zowi.
- **WOFC:** Ordena a Zowi escribir los valores de offset en su EEPROM. Se envía con la instrucción un número (1 o 2) para indicar pierna derecha/izquierda, además del separador (*) y las posiciones en grados, 3 dígitos cada una, separadas por el asterisco (*).
- **M90C:** Ordena a Zowi mover todos los servos a la posición 90º, utilizando la librería servo.
- **MHOC:** Ordena a Zowi mover todos los servos a la posición 90º con la corrección guardada en la EEPROM, si la hay.
- **MHOC:** Ordena a Zowi mover todos los servos a la posición indicada, necesita como parámetro un número de tres cifras que indique la posición en grados.
- **MSxC:** Comando a Zowi para mover el servo indicado a la posición indicada. Necesita un número del 1 al 4 que indica el servo a mover, el separador (*), además de 3 dígitos que indiquen el grado.
- **WERC:** Comando que envía los errores medidos tras la calibración a Raspberry. Para cada servo se envía el error con 3 dígitos enteros más 2 decimales, separados por punto. Los datos de cada servo están separados por (*).
- **FZUM:** Ordena a Raspberry que cargue el programa final en Zowi.
- **ROFF:** Comando a Raspberry para apagar el sistema (Shut down).
- **WSQL:** Ordena Raspberry que cierre comunicación con Zowi y registre una entrada de fin de calibración en la base de datos.

Si bien las instrucciones anteriores fueron útiles durante el desarrollo, se acabó utilizando, principalmente, el movimiento de un solo servo por ser más controlado, se observaron algunos funcionamientos no deseados al tener en movimiento los 4 servos simultáneamente, además de la batería en carga.

Como entrada, Mega recibirá "M" o "B" desde Raspberry en los estados que requieren feedback para determinar el siguiente paso en la máquina de estados, indicando estos "Mal" y "Bien", respectivamente. Las funciones de lectura como ROFC tambien resultaron útiles durante el desarrollo, sin embargo no se utilizan en la versión final.

3.4.2.1.2 Configuración

Mega permite hasta 3 canales adicionales de comunicación serie, Serial1 utiliza los pines 19 como RX y 18 como TX para una comunicación serie TTL a 9600 baudios. Gracias a un cable USB-TTL, es posible debugear y desarrollar fácilmente, aún teniendo utilizado el puerto serie habitual para comunicar con Raspberry.

Programación para los componentes acústicos y luminosos utilizando las librerías estándar de Arduino, que permiten emitir un sonido o establecer el color de una luz con una simple línea de código, también se recurre al uso de las resistencias de pull-up integradas, activadas en la declaración del tipo de pin, para reducir la cantidad de electrónica necesaria para los finales de carrera y pulsadores.

3.4.2.1.3 Algoritmo básico de calibración

Este algoritmo se ejecuta para la correcta calibración de cada uno de los juguetes.

Inicialmente se mandan los servos del robot Zowi a las posiciones correspondientes a 90°, se recuerda que el rango del servo es 0-180°, y que el juguete necesita un rango de solamente unos 80°.

La calibración consiste en tomar la lectura de los sensores IMU, de cada sensor interesan 2 orientaciones, una de ellas directamente relacionada con la posición de la cadera del robot, y la otra con el pie. Leyendo ambos sensores se obtienen las 4 posiciones de los servos.

Se calcula el desfase entre la posición leída y la posición actual según el programa de Zowi (inicialmente 90°), y se calcula la nueva posición necesaria a establecer en el servo para obtener los 90° deseados.

El proceso de lectura-corrección se lleva a cabo de forma iterativa hasta obtener una lectura con error inferior a 1° respecto a 90°.

3.4.2.1.4 Calibración de los sensores

Esta calibración se ha de realizar cada vez que el sistema es iniciado o reiniciado.

Mediante las funciones de la librería de MinIMU, se establecen los ángulos cero para cada eje, para ello el operario es guiado a insertar los zapatos (que contienen los sensores) en los cajetines del banco de calibración, en este paso se toman medidas hasta valorar que la calibración ha sido correcta. En esta etapa se corrigen pequeñas inclinaciones que pueda haber en el espacio de trabajo. Es la parte más crítica del proceso, porque la calibración se ve muy afectada por el movimiento, vibraciones o incluso campos magnéticos.

Durante el algoritmo de calibración se ha hablado del ángulo de 90° como ángulo deseado. Realmente, el valor este ángulo para las caderas se corresponde con el 0°, mientras que para los pies se corresponde con el 90°, por lo que tambien es ajustado en esta etapa. Tras valorar que los ángulos 0° han sido bien definidos y su lectura es estable, se ha de girar cada zapato 90° utilizando los cajetines verticales. En esta fase se corregirá el pequeño error que pueda tener el sensor en esos 90° (no suele ser mayor de 1°).

3.4.2.1.5 Calibración del acelerómetro

Configuración necesaria cada vez que se reemplace el sensor. Ésta es la única configuración necesaria para replicar el sistema.

Esta calibración no forma parte del programa del sistema, sino de su configuración e instalación. En los 2 armarios que se han creado, se utilizan en total 4 sensores IMUs, los circuitos integrados que contiene no son exactamente iguales y, por recomendación del fabricante y creador de la librería de MinIMU, se han de ajustar

los valores máximos y mínimos crudos del magnetómetro (mismo encapsulado que el acelerómetro). Para ello se utiliza un programa del fabricante que muestra, por el puerto serie, el valor máximo y mínimo crudos registrados mientras el sensor es movido en todos los ángulos. Estas constantes son introducidas en el programa de cada armario para ser utilizadas por la librería del acelerómetro, LSM303.

3.4.2.2. Zowi

El programa definido para la controladora de Zowi es un intérprete de los comandos definidos. La Raspberry descargará dicho intérprete en el microcontrolador de Zowi para poder comunicarle las posiciones a las que ha de mover los servos, y los valores de calibración que tendrá que guardar en su memoria EEPROM, memoria que no es sobrescrita en el proceso de programación habitual de la placa controladora.

Adicionalmente, la controladora de Zowi cuenta con un módulo de EEPROM por I²C, donde se recogen datos de fabricación de la placa. De esta forma se tienen 2 módulos de EEPROM, dejando el módulo del microcontrolador disponible para ser modificado por los usuarios de la placa.

Es en el módulo habitual donde se registran los valores de calibración del servo, pudiendo ser configurados por los usuarios en caso de que desmonten el juguete o reemplacen algún componente. Por otro lado, desde el módulo EEPROM por I²C, se obtendrá el número de serie, que almacenará la Raspberry en la base de datos con fines de trazabilidad.

Las librerías que se utilizan serán I2C_eeprom.h y las estándar Servo.h para el movimiento de los servos y EEPROM.h para guardar los valores de calibración.

3.4.2.2.1 Protocolo visto desde Zowi

Las siguientes instrucciones son recibidas por Zowi en la última versión del intérprete:

- **MSxC:** Al recibir este comando, Zowi mueve el servo indicado a la posición indicada. El primer parámetro es un número del 1 al 4 que indica el servo a mover, seguido del separador (*) y 3 dígitos que indican el grado.
- **WOFc:** Zowi calcula el valor de offset a partir de las posiciones recibidas y los escribe en su EEPROM. Las instrucciones contienen un número (1 o 2) para indicar pierna derecha/izquierda, además del separador (*) y las posiciones en grados, 3 dígitos cada una, separadas por el asterisco (*).
- **M90C:** Zowi mueve todos los servos a la posición 90° por defecto.
- **MHOC:** Zowi lee el valor del offset de las articulaciones en su EEPROM y mueve todos los servos a la posición 90° con la corrección.

Se implementa una instrucción de salida, utilizando el mismo protocolo que se ha visto en Mega (instrucción delimitada por “\$” y “#”), Zowi lee el número de serie de su controladora (6 dígitos) y se lo envía a la Raspberry con la instrucción **OKNS**:

3.4.2.3. Raspberry

La Raspberry resulta ser un componente tan importante como la Mega dentro del sistema. Inicialmente fue incluída para hacer de nexo en la comunicación entre ambos controladores (Mega y ZUM de Zowi), pero inmediatamente se implementaron algunas funcionalidades adicionales.

Este ordenador tiene instalado un sistema operativo Raspbian, versión adaptada de Debian a Raspberry, concretamente se usa la última versión estable en el momento de desarrollo, Wheezy.

El lenguaje elegido para hacer el software es python, por lo rápido que resulta desarrollar con este lenguaje y por ser el más familiar para el autor. Se instalan por tanto los paquetes necesarios (el entorno python-dev, el gestor de paquetes PIP y el gestor de entornos virtuales, virtualenv). Las librerías empleadas son serial para las comunicaciones serie, os para poder ejecutar comandos del sistema operativo y pymysql para poder escribir en bases de datos MySQL.

Para que el controlador de Zowi interprete las instrucciones que recibirá, se ha de descargar el intérprete creado. Esto requiere que Raspberry programe el micro ATMega328p de Zowi, para ello se utiliza la herramienta AVRdude como instrucción del sistema operativo, directamente desde el programa corriendo en python. El programa en Arduino del intérprete está convertido en un fichero hexadecimal ya compilado con el entorno de Arduino, AVRdude escribirá este programa en el microcontrolador de Zowi.

Al arrancar el sistema operativo y la script, se inicia comunicación con Mega usando el puerto ttyUSB0 y envía un “1” para inicializar el programa. La Raspberry permanecerá a la escucha, los dispositivos conectados (Mega -y Zowis tras la descarga del intérprete-) envían instrucciones utilizando el protocolo implementado ya comentado en los apartados anteriores, cualquier instrucción a Raspberry irá contenida entre “\$... #”. Las instrucciones de Mega que han llegar a Zowi, se envían ya sin los delimitadores (\$ y #), es decir, solamente el contenido de la instrucción.

Además de delimitar fácilmente qué datos son instrucciones, nos permite poder hacer eco del resto de datos recibidos por serie, para debug o información de los programas de las Arduino en Raspberry.

3.4.2.3.1 Máquina de estados en Raspberry

Se muestra la respuesta de la script de python a las diferentes instrucciones recibidas:

IZUM: Cuando se recibe este código, Raspberry programará la ZUM de Zowi con el software de calibración (el intérprete), ubicado en el siguiente directorio:

/home/pi/zowi/python/zowi_offset_i2c.cpp.hex

El programa es subido usando la instrucción de AVRdude:

```
avrdude -patmega328p -carduino -P/dev/ttyUSB1 -b 115200 -D  
-Uflash:w:/home/pi/zowi/python/zowi_offset_i2c.cpp.hex:i
```

Como se ha dicho anteriormente, este programa es necesario para interpretar los códigos e instrucciones enviados desde Mega. Se envía un código de confirmación con el texto “M” o “B” indicando a Mega indicando el resultado del proceso.

FZUM: Cuando se recibe este código, se programa ZUM de Zowi con el programa final del juguete. El fichero está en el siguiente directorio:

/home/pi/zowi/python/ZOWI_BASE_v0.cpp.hex

Se envía un código de confirmación con el texto “M” o “B” indicando a Mega indicando el resultado del proceso.

ROFF: Indica que el sistema se debe apagar, cuando se recibe este comando desde Mega, se apaga el sistema operativo.

WERC: Este código indica los errores de calibración de cada articulación, es decir, cuántos grados de diferencia se han conseguido entre los valores teóricos y los medidos tras finalizar una calibración, sea exitosa o fallida. Se almacenan dichos valores para ser guardados, posteriormente, en la base de datos.

MSxC: El resto de instrucciones de movimiento de servos ya comentadas en los apartados de software de Mega y Zowi, se envían directamente de Mega a Zowi. Para el caso de movimiento de un servo específico, **MSxC**, se almacenan los últimos valores enviados, para poder ser volcados a la base de datos cuando se reciba la instrucción adecuada.

WOFC: Este código es enviado indicando las nuevas posiciones de los servos de Zowi, las posiciones que hacen que sus articulaciones estén alineadas y calibradas. Raspberry envía estos valores a Zowi para que éste calcule los desfases y los escriba en su memoria EEPROM.

OKNS: Es la única instrucción que proviene de Zowi, e indica el número de serie de la placa controladora del ejemplar, Raspberry almacena dicho número para su escritura en base de datos.

3.4.2.3.2 Base de datos

Se instala en el sistema una base de datos MySQL local, contra la que se registran los eventos que suceden durante el uso del sistema. Además, la script del programa principal de python está también preparada para escribir a una base de datos remota. Se escribe en la base de datos tras cada calibración, sea buena o mala. Para lo que se utiliza un campo de “Estado” que valdrá 1 o 0, respectivamente. Se registran los

errores de calibración leídos en Arduino Mega para cada uno de los 4 servos. Se registran además las 4 posiciones “trim” de los servos, las mismas que se hacen llegar a la placa de Zowi para que guarde en su EEPROM para salir a venta. Se registra la hora, con fines estadísticos de tiempos y también el número de serie del Zowi conectado con fin de trazabilidad en caso de errores/reclamaciones.

Cuando se inicia o apaga el sistema, se escriben todos los campos a 1 salvo la hora para el caso de inicio. Y todos los valores a 0 salvo la hora para el fin.

Durante el uso de los primeros días en producción surgieron algunos problemas, para los que se definieron unos códigos de error y se implementaron ciertas correcciones de forma remota, de la misma forma que para inicio y final se usan valores a 1 y a 0, para estos errores se utilizan:

- **Valores a 2:** Excepción del cable, producido por pérdida de comunicación con Mega.
- **Valores a 3:** Otros errores no conocidos, por excepción en el programa principal.
- **Valores a 4:** Error conexión fallida al intentar descargar intérprete a Zowi.
- **Valores a 5:** Error conexión fallida al intentar descargar programa final a Zowi.

Para ver la estructura de la tabla, se puede consultar el Anexo H.

3.4.2.3.3 Configuración adicional

El servicio de escritorio remoto XRDП es instalado, lo que nos permite utilizar la interfaz gráfica de forma remota.

El entorno de Arduino es también instalado, permitiendo modificar el programa de Mega desde Raspberry. Se instalan también las librerías empleadas en los programas creados. El entorno de Arduino instalará AVRdude para realizar la programación de los microcontroladores.

En el fichero de configuración de Raspberry **/boot/config.txt** se configura *max-usb-current=1* lo que nos permite suministrar hasta 1.2A por USB para alimentar a Zowi (por defecto, limitado a 600mA).

Se configura el Login automático mediante el fichero **/etc/inittab**. De este modo el sistema funciona como servidor en lugar de esperar que un usuario introduzca sus credenciales. Procedimiento en el Anexo M.

Se crea una script, **running.sh** que funciona como monitor, hará que la script principal sea arrancada siempre que no esté corriendo. Esta medida rearma el sistema ante algunos errores, como por ejemplo, intentar programar una placa que deja de estar energizada o es apagada o desconectada durante la programación. Anexo K.

Se configura el arranque automático de la script anterior al encender el sistema, dicha script levantará el programa principal nada más ser iniciado el sistema. El

arranque automático se logra mediante el fichero **ZowiInit**, que se puede consultar en el Anexo L.

Durante el desarrollo se creó un documento de notas para facilitar la reproducción de la instalación desde cero, además de la creación de unas imágenes completas de la tarjeta SD de la Raspberry, utilizando Win32DiskImager. Se anexan las mencionadas notas por si pudiesen resultar de interés para el lector, Anexo N.

Tras observar el funcionamiento del sistema en producción durante algunos días, se detectan algunos errores al conectar y desconectar repetidas veces los robots. El sistema podría no funcionar si se asigna un puerto diferente a Mega, o si se inicia el sistema con un Zowi ya conectado. Para mejorar esto, se crean unas reglas (fichero **50-usbportsbq.rules**) para definir el puerto según el ID del fabricante de la placa, asignando a cualquier puerto USB conectado a Raspberry el alias de "mega" o "zowi" en lugar de ttyUSBX, nombre por defecto). Dichas reglas son instaladas dentro de **/etc/udev/rules.d/**. El programa de python es modificado para asumir estos cambios. Las notas sobre el fichero de las reglas se puede ver en el Anexo N. Además, se implementa una nueva función en el código que desconecta y conecta eléctricamente la bahía del puerto USB de Zowi por software (utilizando **bind** y **unbind**) antes de cada programación. Los errores y reinicios se reducen inmediatamente.

Capítulo 4

Conclusiones y trabajos futuros

4.1. Conclusiones

El sistema desarrollado fue instalado en fábrica, con localización en Polonia, cumpliendo satisfactoriamente con su función y logrando una producción de 25000 unidades. Se puede ver el puesto en la Figura 4.1. Otras imágenes de la fábrica en las Figuras 4.2 y 4.3.



FIGURA 4.1: Puesto de calibración en la línea de montaje

El envío de dos unidades del sistema fue un acierto, en varias ocasiones fue necesario reemplazar componentes o cableado, además de las mejoras implementadas tras los primeros días de operación. Estos contratiempos habrían sido de mayor gravedad si no se contara con el segundo armario.

Los errores más habituales fueron los producidos en el intento de programar



FIGURA 4.2: Operadoras en línea de montaje



FIGURA 4.3: Línea de montaje de Zowi en Rosti

la controladora de Zowi: Puerto USB no liberado entre Zowi y Zowi, alta del dispositivo por el SO con otro nombre de puerto, programación de Zowi sin haberlo energizado, desconexión del cable durante la programación... Los dos primeros se solucionaron de forma preventiva, mientras que para los otros dos se aplicaron medidas correctivas.

Los errores podían dejar el sistema totalmente congelado y el reinicio supone una re-calibración de los zapatos, lo que consume alrededor de 2-3 minutos adicionales.

Se observa la base de datos para dos muestras de 4000 unidades -1 semana aproximadamente-. Una muestra al inicio (antes de las mejoras) y otra al final de la producción (tras mejoras). Se atiende a la forma de apagar el sistema: controlado vs. "botonazo", confiando en el buen uso de los operadores, como indicador del funcionamiento. Se muestra también el tiempo ciclo entre calibraciones satisfactorias. También el porcentaje de calibraciones "buenas". También se muestran "recuperaciones del sistema" vs. "total de veces iniciado". Estos elementos son indicativos, hay otros factores externos que podrían afectar a los resultados, como un mal montaje de las partes mecánicas del juguete en las etapas anteriores de la línea de montaje, o un mal uso de los operadores, entre otros. Los datos se resumen en la Tabla 4.1.

	Pre-mejoras	Post-mejoras
Tiempo calibración medio	81s	50s
Calibraciones a la primera	98.61 %	99.13 %
Calibraciones OK	99.76 %	99.89 %
Apagados "forzados"	63 %	43 %
Recuperaciones vs. restarts	-	35 %

CUADRO 4.1: Conclusiones y resultados

4.2. Trabajos futuros

Tras la finalización de la producción inicial, los armarios de calibración y los bancos finales fueron enviados a otra fábrica en Navarra, dónde continúan funcionando.

El último prototipo creado fue actualizado con todos los cambios desarrollados para la versión final. Actualmente se encuentra funcionando en el servicio técnico, utilizando el utilaje diseñado en la fase de desarrollo, Anexo J.

Hablar de futuras mejoras sobre este mismo sistema no tiene sentido ya que el proyecto quedó cerrado y hay otras etapas más lentas en la línea de montaje. Sin embargo, si que se podría haber logrado un mejor diseño, como se comenta a continuación.

Una posible y elegante mejora en la velocidad sería el empleo de un PID en la lectura-actuación de las posición de los servos en lugar del método por iteraciones empleado, o la utilización de sensores y dispositivos electrónicos directamente desde Raspberry.

De cara a otros proyectos utilizando arquitecturas similares (Raspberry y Arduino), donde los plazos y tiempos no jueguen un papel tan importante entre los requisitos, habría algunos puntos a tener en cuenta tras lo aprendido en la realización de este proyecto:

- Trabajar en el software: Implementar un logger en las mismas scripts de python y un mayor uso de try-except y comprobaciones que permitan conocer con mayor detalle la traza de operación.
- En la misma línea que el punto anterior, utilizar un campo en la estructura de la base de datos para representar qué se está guardando en dicha entrada, esto permitiría identificar fácilmente los diferentes errores, warnings, mensajes informativos...
- Desarrollar un pequeño protocolo o herramienta para la comunicación entre Python y Arduino por serie, con una gestión de errores para validación en la interpretación. Dicho protocolo haría más fácil definir en ambos dispositivos las instrucciones que se desean implementar para la comunicación.
- El empleo de un reloj RTC (Real time clock) en la Raspberry evitaría la desconfiguración de la hora cuando ésta no tiene acceso a internet y no se puede utilizar NTP para mantener su hora sincronizada.

Apéndice A

Manual de usuario

Manual de usuario entregado a la empresa contratada para la fabricación, destinado a los operadores de la fábrica.

User manual

1. Workbench preparation

First of all, it is necessary to adjust the workbench calibration using the four adjustable legs installed at the bottom side and the level incorporated. It is important to set the work area as parallel as possible to the ground plane, and fix the bank to ensure that the position does not change during the process.

2. Switching on the system

When switched on the main switch (at the back side), it is necessary to wait some seconds (around 30-50s) until the system has completely started. During this time, the blue led will be blinking slowly and the LCD will be showing “*Zowi workbench. Starting...*”. After this, the workbench will emit a sound and the LED will stop blinking.

At this point, it depends on the switch position (at the front side) to start the process.

If the switch is set at “1” the LCD will show “*Change switch to start process*”, whilst whether the position is set at “0”, the LCD will show “*Place shoes in Horizontal box*”, and it will be ready to start the calibration process.

Notes:

It is really important not to have any Zowi plugged when the system is switching on, to avoid communication problems (usb port assignment done by raspberry).

It helps to the horizontal offset calibration having the shoes placed in the horizontal box when the system is switching on, as the IMU sensors will be getting right measures all the time.

3. IMUs calibration

The operator must follow the display instructions to calibrate the offset of the sensors correctly. Firstly, the horizontal position is calibrated. When this calibration is right, the LCD will show “*Horizontal done. Insert Vertical*” to continue with the vertical calibration.

It is really important not to disturb the system, trying to avoid vibrations, movements etc that could interfere in the sensors calibration. If after trying to calibrate the sensor five or six times, the LCD shows “*Horizontal calib failed. Retry*”, switch off the system as shown at step 6 (Shutting down the system), and switch on it again, being sure that the shoes are correctly placed in the horizontal box.

When the sensors calibration is finished, the LCD will show “*Plug, Switch on, Put on and Push*” and the blue LED will start blinking slowly.

Note: If the workbench is moved at any point of the process, it must be repeated this step to ensure the correct calibration relative to the ground.

4. Zowi calibration

After calibrating correctly the offset of the sensors, it will proceed to calibrate as many Zowis as desired.

To calibrate each Zowi, place it at the workbench with the face looking to the operator (or with the connector looking to the operator depending on the program loaded in the system initially, by default the option with the connector looking to the operator is loaded) and put the shoes at each foot, paying attention to use the correct shoe (top, bottom). After this, switch on Zowi, plug it and push the button. The LCD will show “*initiating communication*”.

The Raspberry Pi will try to upload the calibration program to the Zowi board. There are three possible situations:

- The calibration program is uploaded correctly. The system will continue with the calibration of each joint.
- The Zowi board does not respond after 40 seconds. The LCD will show “*Timeout. Check Zowi and push*”. Check Zowi is switched on and push the button to retry.
- The Zowi board responds but it is impossible to upload the calibration program. The LCD will show “*Connection fault. Check Zowi*”. Check Zowi is plugged and push button to retry.

When the calibration program is uploaded correctly, the LCD will show “*Communication OK. Calibrating...*” and the LED will blink quickly. Zowi will move its hips and feet to calibrate them. After various iterations trying to calibrate the four joints, there are three possible situations:

- The calibration has failed. It has been impossible to reach a calibrated position. The LCD will show “*CALIBRATION FAILED*” and the red LED will turn on. The LCD will show “*Push for a new calibration*” after a second.
- The calibration is right, so the Raspberry Pi will try to upload the Final Test program. The LCD will show “*Calibration OK. Loading Test Prg*”. If it is not possible to upload this program the LCD will show “*CALIBRATION FAILED*” and the red LED will turn on. The LCD will show “*Push for a new calibration*” after a second.
- The calibration is right, so the Raspberry Pi will try to upload the Final Test program. The LCD will show “*Calibration OK. Loading Test Prg*”. If the program is uploaded correctly the green LED will turn on and the LCD will show “*Push for a new calibration*”.

Once the process is done, switch off Zowi and unplug it. It is possible to connect other Zowi to repeat the Zowi calibration pushing the button.

5. Reset the system

At any point of the process, excepting for the step 2(Switching on the system) and 6 (Shutting down), it is possible to switch to “1” the front switch, and the system will change to step 3 (IMUs calibration).

6. Shutting down the system

At step 3 (IMUs calibration), it is possible to switch off the whole system. To do it, it is necessary to push the button for at least two seconds. The LCD will show “*Shutting down...*” and afterwards, the LCD will show nothing. When this happens, switch off the power switch (at the back side).

Apéndice B

Planos Eléctricos: Armario 1

Planos eléctricos del armario 1.

A B C D E F

9
8
7
6
5
4
3
2
1
0



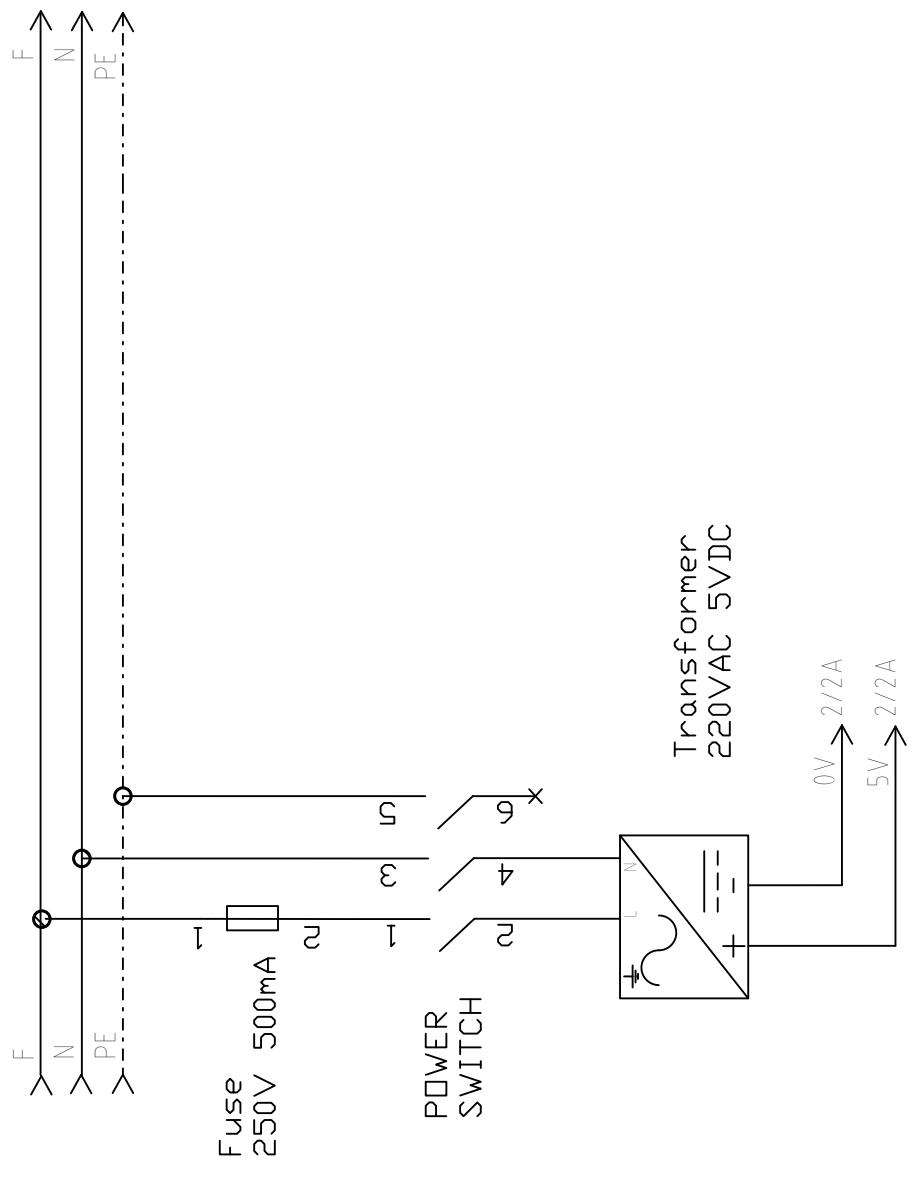
COMPANY: BQ

PROJECT: ZOWI CALIBRATION CABINET

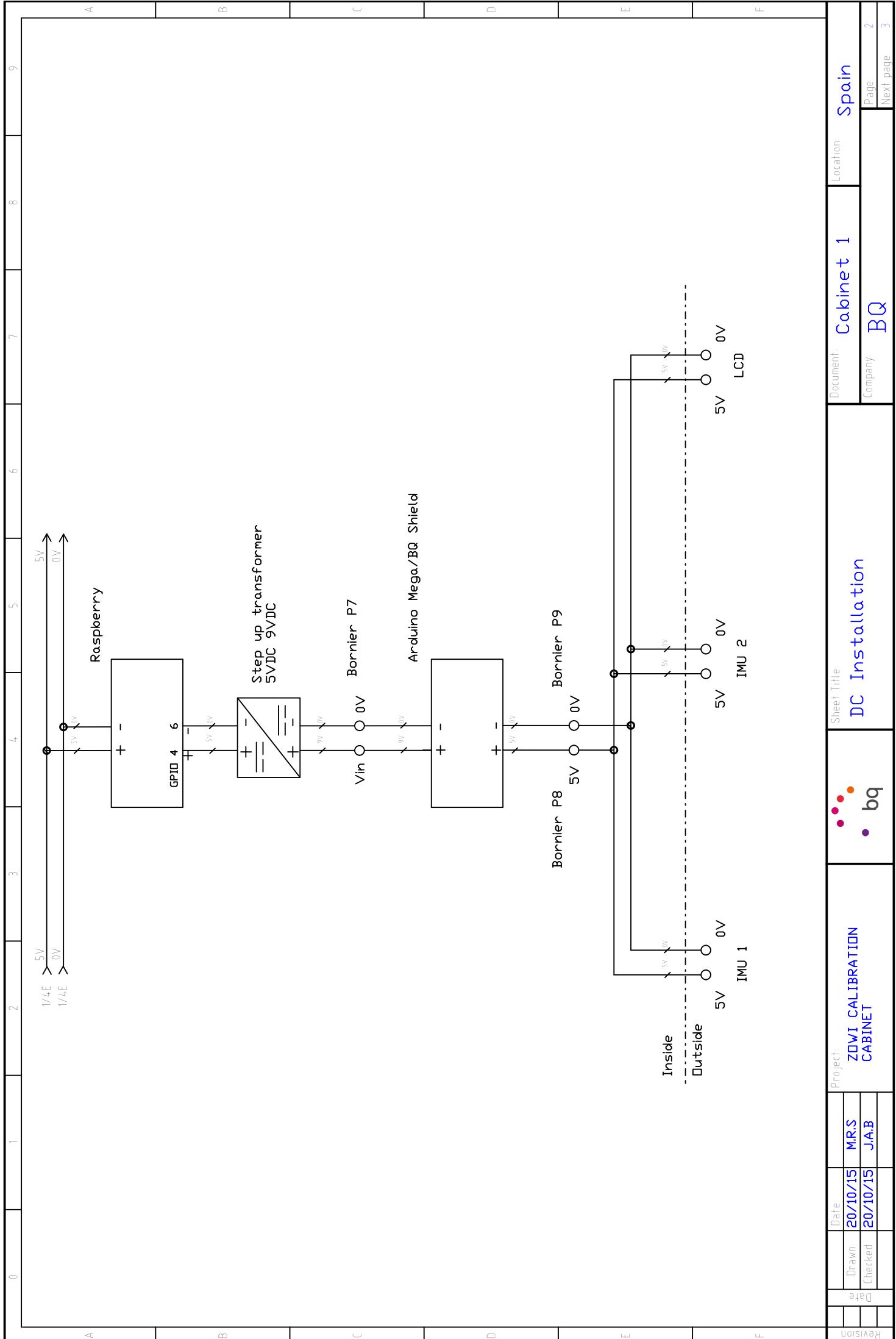
Project:	ZOWI CALIBRATION CABINET	Sheet Title:	FRONT PAGE
Date Drawn:	20/10/15	Date Checked:	J.A.B R.D.S
Revised:		Page:	Spain

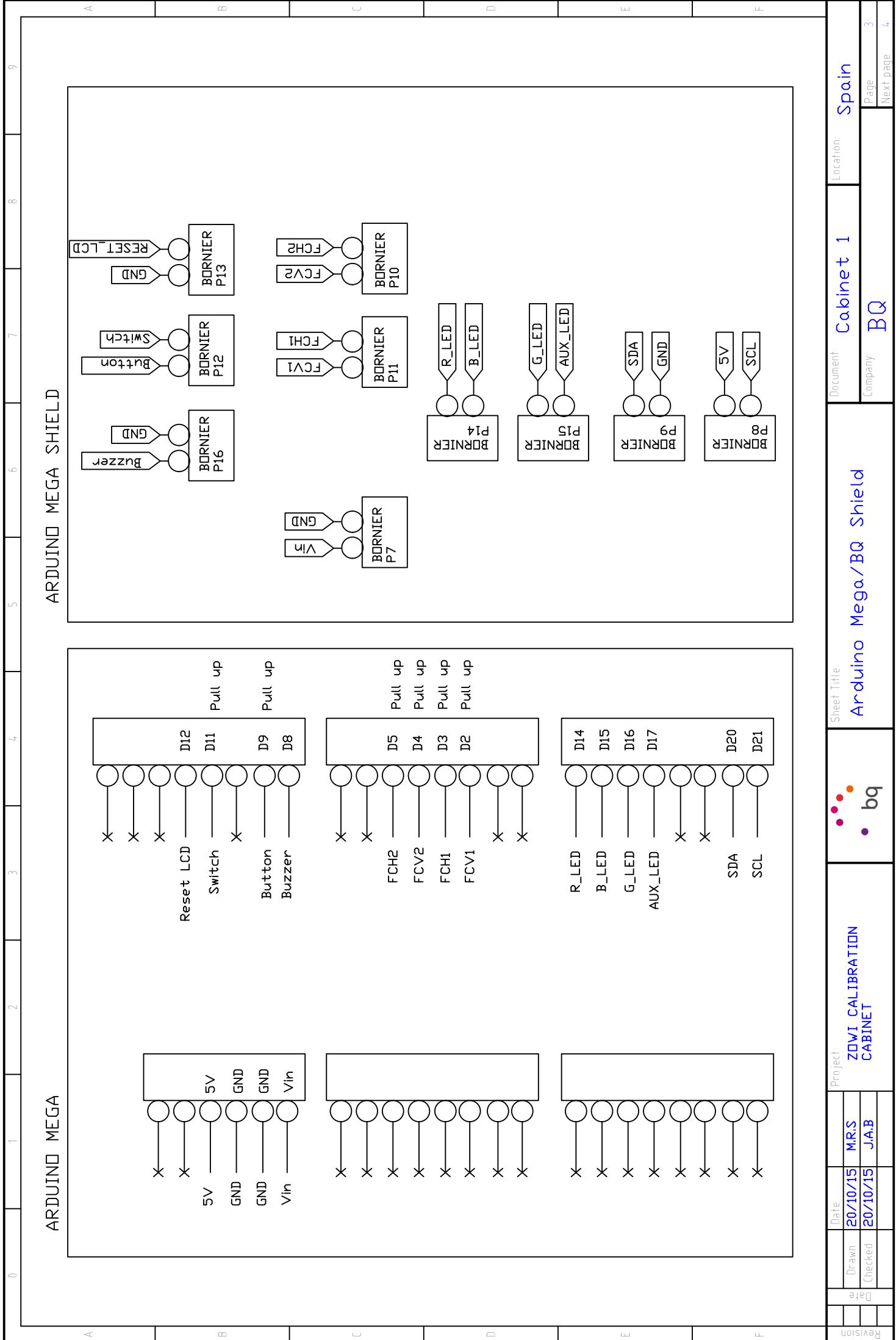
Document:	Cabinet 1	Location:	Spain
Company:	BQ	Page:	0

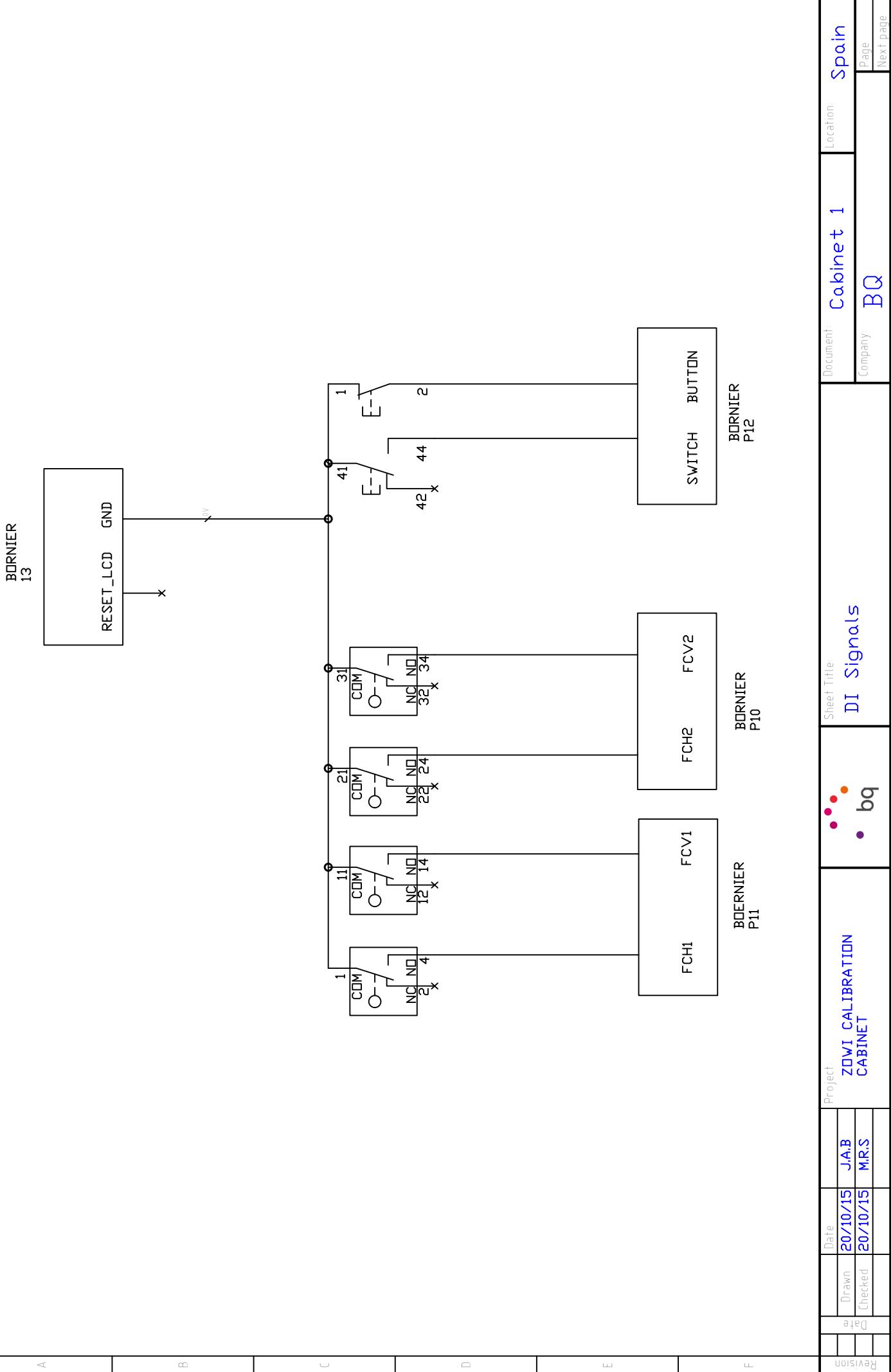
Next Page 1

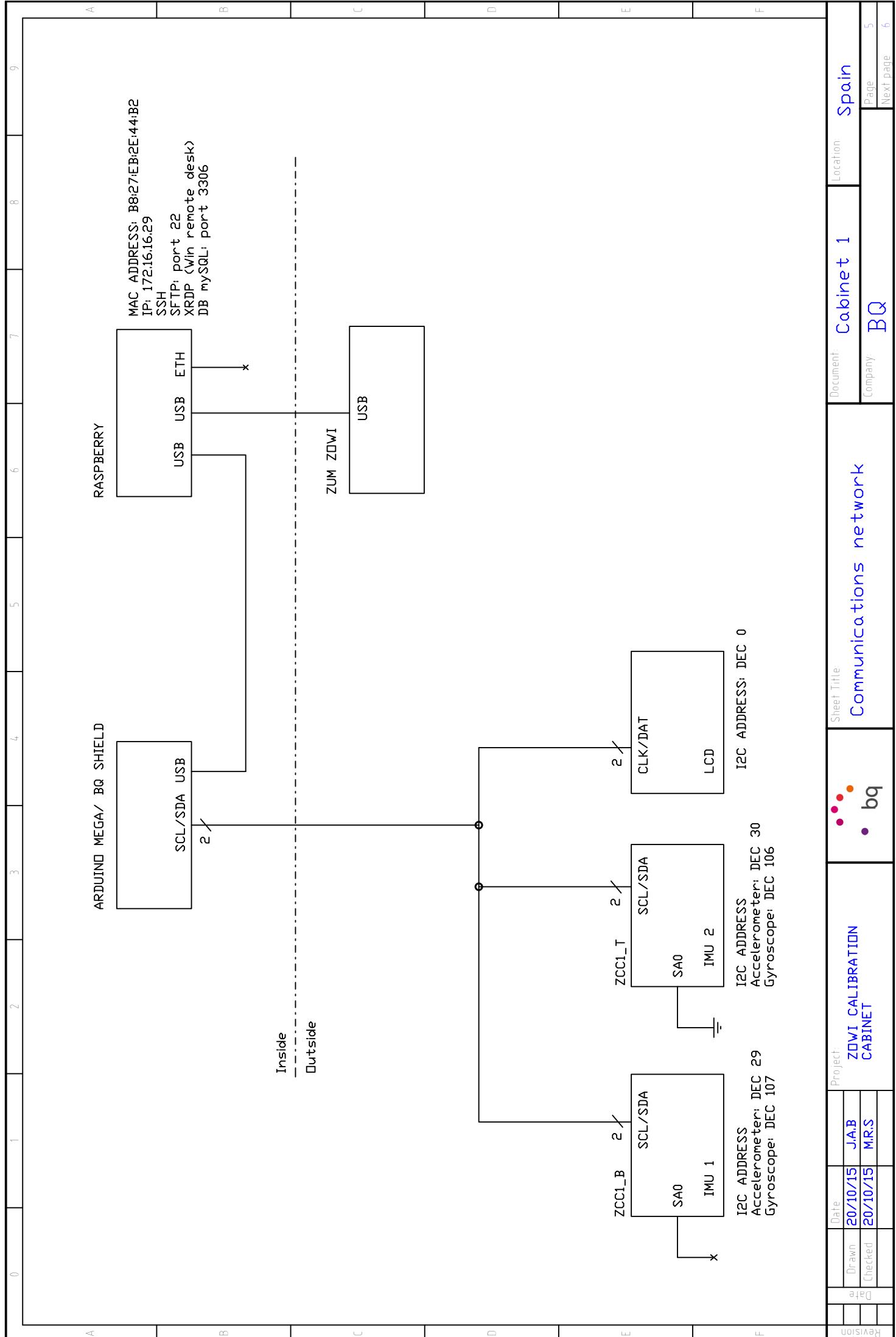


Revision	Date Drawn	Date Checked	Project:	Sheet Title	Document	Cabinet 1		Location:
						Company	Page	
	20/10/15	20/10/15	ZOWI CALIBRATION CABINET	AC Installation	BQ			Spain
	M.R.S	J.A.B	bq					









A	B	C	D	E	F																																														
9																																																			
8																																																			
7																																																			
6																																																			
5																																																			
4																																																			
3																																																			
2																																																			
1																																																			
0																																																			
A	B	C	D	E	F																																														
																																																			
																																																			
<p>Sheet Title: Bornier connections</p> <p>Project: ZWI CALIBRATION CABINET</p> <table border="1"> <tr><td>Document:</td><td>Cabinet 1</td><td>Location:</td><td>Spain</td></tr> <tr><td>Company:</td><td>BQ</td><td>Page:</td><td>6</td></tr> </table> <p>bq logo</p>						Document:	Cabinet 1	Location:	Spain	Company:	BQ	Page:	6																																						
Document:	Cabinet 1	Location:	Spain																																																
Company:	BQ	Page:	6																																																
<table border="1"> <tr><td>Vin Boost</td><td></td></tr> <tr><td>Vout Boost</td><td></td></tr> <tr><td>VCC Mega 1</td><td>Red</td></tr> <tr><td>VCC Mega 2</td><td>Blue</td></tr> <tr><td>0V</td><td>Black</td></tr> <tr><td>0V</td><td>Black</td></tr> <tr><td>SDA</td><td>LCD: Green / IMU: White</td></tr> <tr><td>SCL</td><td>Orange</td></tr> <tr><td>Switch</td><td>Grey</td></tr> <tr><td>Button</td><td>Red</td></tr> <tr><td>FCH2</td><td>Red</td></tr> <tr><td>FCV2</td><td>Green</td></tr> <tr><td>FCH1</td><td>White</td></tr> <tr><td>FCV1</td><td>Orange</td></tr> <tr><td>Green LED</td><td>Grey</td></tr> <tr><td>Blue LED</td><td>Grey</td></tr> <tr><td>Red LED</td><td>Grey</td></tr> <tr><td>Buzzer</td><td>Grey</td></tr> <tr><td>Black USB</td><td>Black</td></tr> <tr><td>Red USB</td><td>Red</td></tr> <tr><td>Green USB</td><td>Green</td></tr> <tr><td>White USB</td><td>White</td></tr> <tr><td>Shielded USB</td><td>Grey</td></tr> </table>						Vin Boost		Vout Boost		VCC Mega 1	Red	VCC Mega 2	Blue	0V	Black	0V	Black	SDA	LCD: Green / IMU: White	SCL	Orange	Switch	Grey	Button	Red	FCH2	Red	FCV2	Green	FCH1	White	FCV1	Orange	Green LED	Grey	Blue LED	Grey	Red LED	Grey	Buzzer	Grey	Black USB	Black	Red USB	Red	Green USB	Green	White USB	White	Shielded USB	Grey
Vin Boost																																																			
Vout Boost																																																			
VCC Mega 1	Red																																																		
VCC Mega 2	Blue																																																		
0V	Black																																																		
0V	Black																																																		
SDA	LCD: Green / IMU: White																																																		
SCL	Orange																																																		
Switch	Grey																																																		
Button	Red																																																		
FCH2	Red																																																		
FCV2	Green																																																		
FCH1	White																																																		
FCV1	Orange																																																		
Green LED	Grey																																																		
Blue LED	Grey																																																		
Red LED	Grey																																																		
Buzzer	Grey																																																		
Black USB	Black																																																		
Red USB	Red																																																		
Green USB	Green																																																		
White USB	White																																																		
Shielded USB	Grey																																																		

Apéndice C

Planos Eléctricos: Armario 2

Planos eléctricos del armario 2.

A B C D E F

9
8
7
6
5
4
3
2
1
0

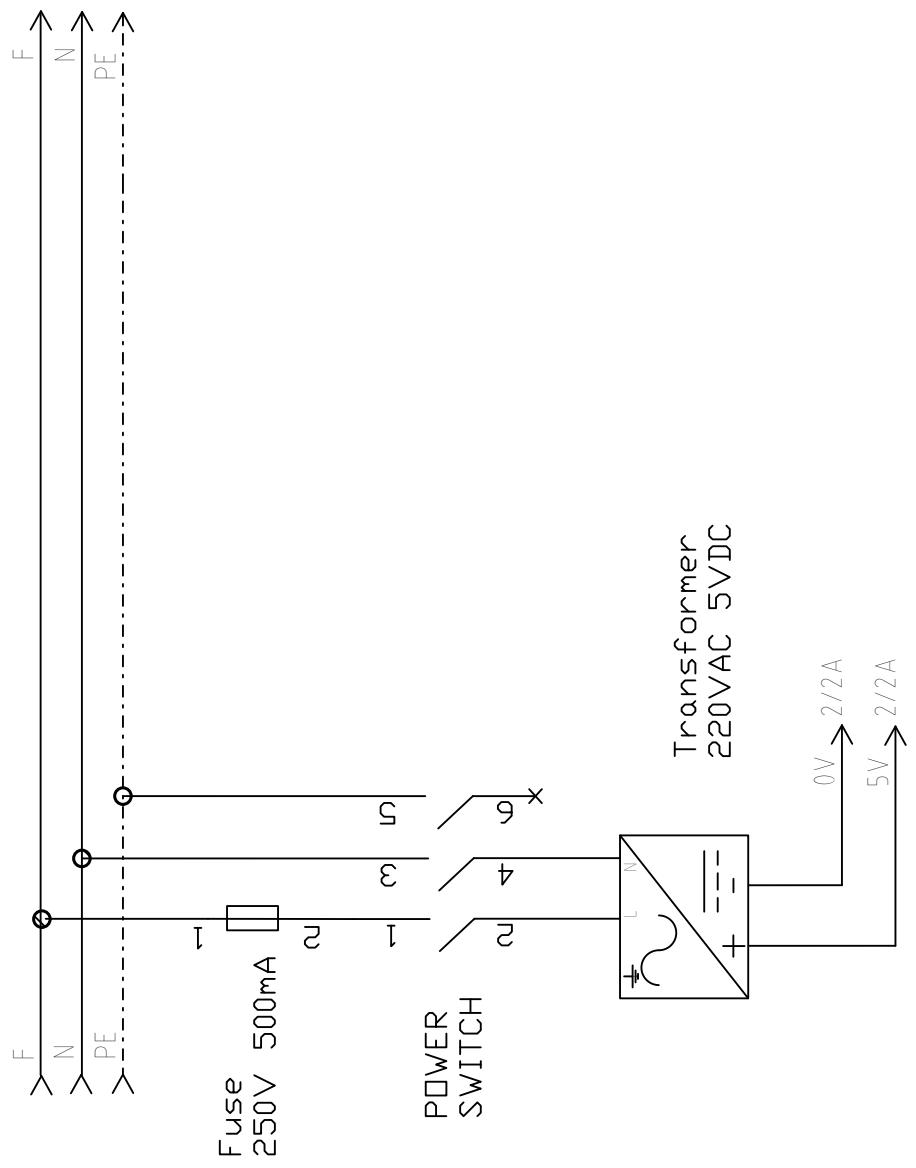


COMPANY: BQ

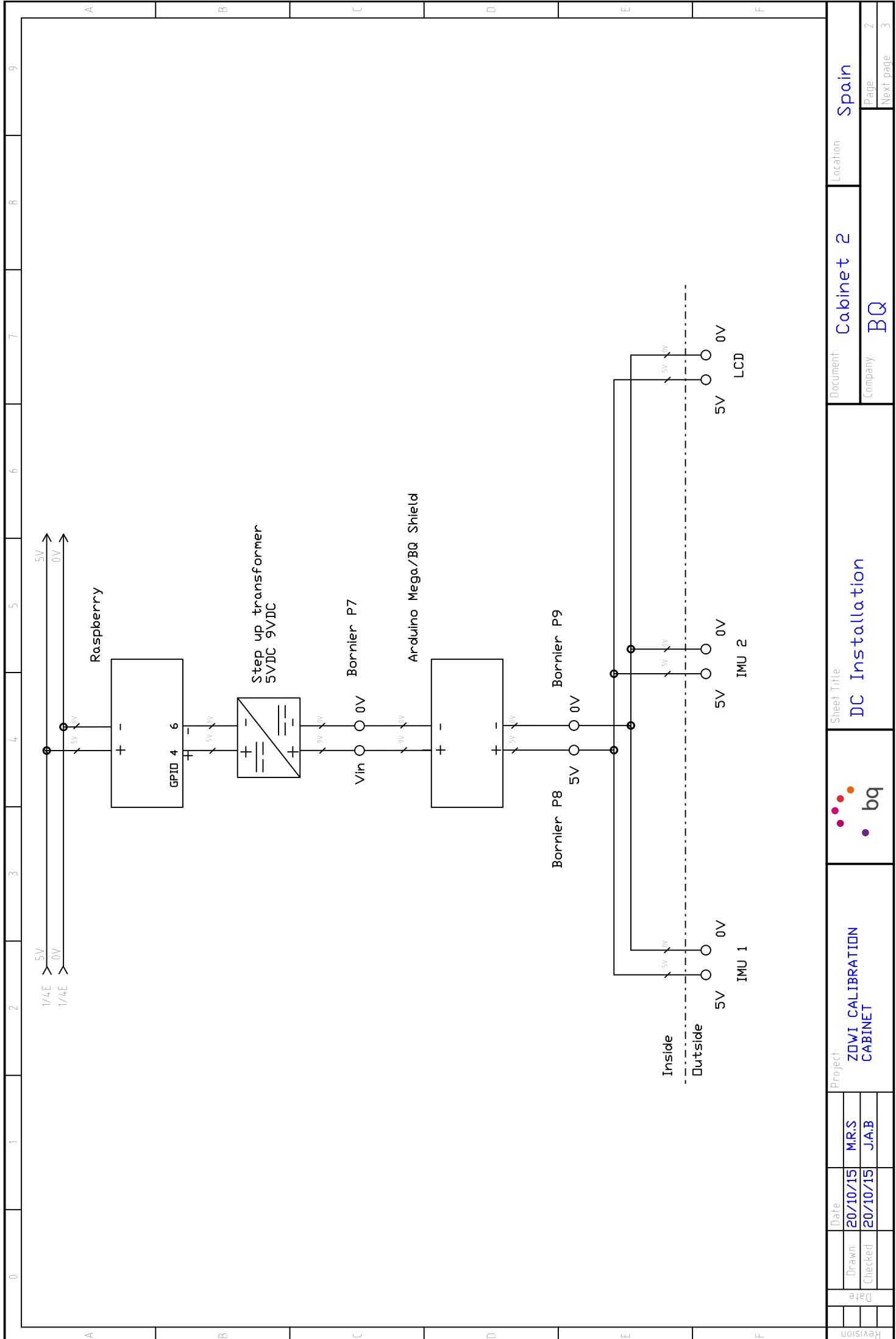
PROJECT: ZOWI CALIBRATION CABINET

Project:	ZOWI CALIBRATION CABINET	Sheet Title:	FRONT PAGE
Date Drawn:	20/10/15	Date Checked:	J.A.B R.D.S
Revised:		Page:	0

Document:	Cabinet 2	Location:	Spain
Company:	BQ	Page:	1



Revision	Date	Project	Sheet title	Location	Spain
Drawn	20/10/15	M.R.S	AC Installation	Cabinet 2	Page 1
Checked	20/10/15	R.D.S		Company	Next page
			bq		



A 8

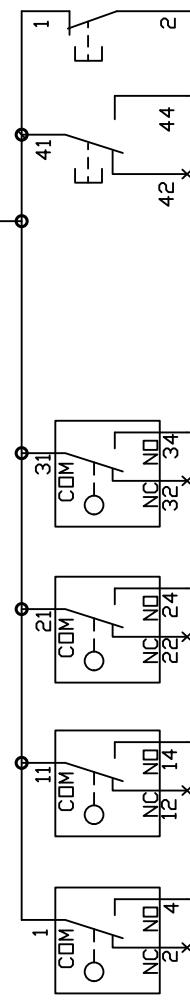
B 7

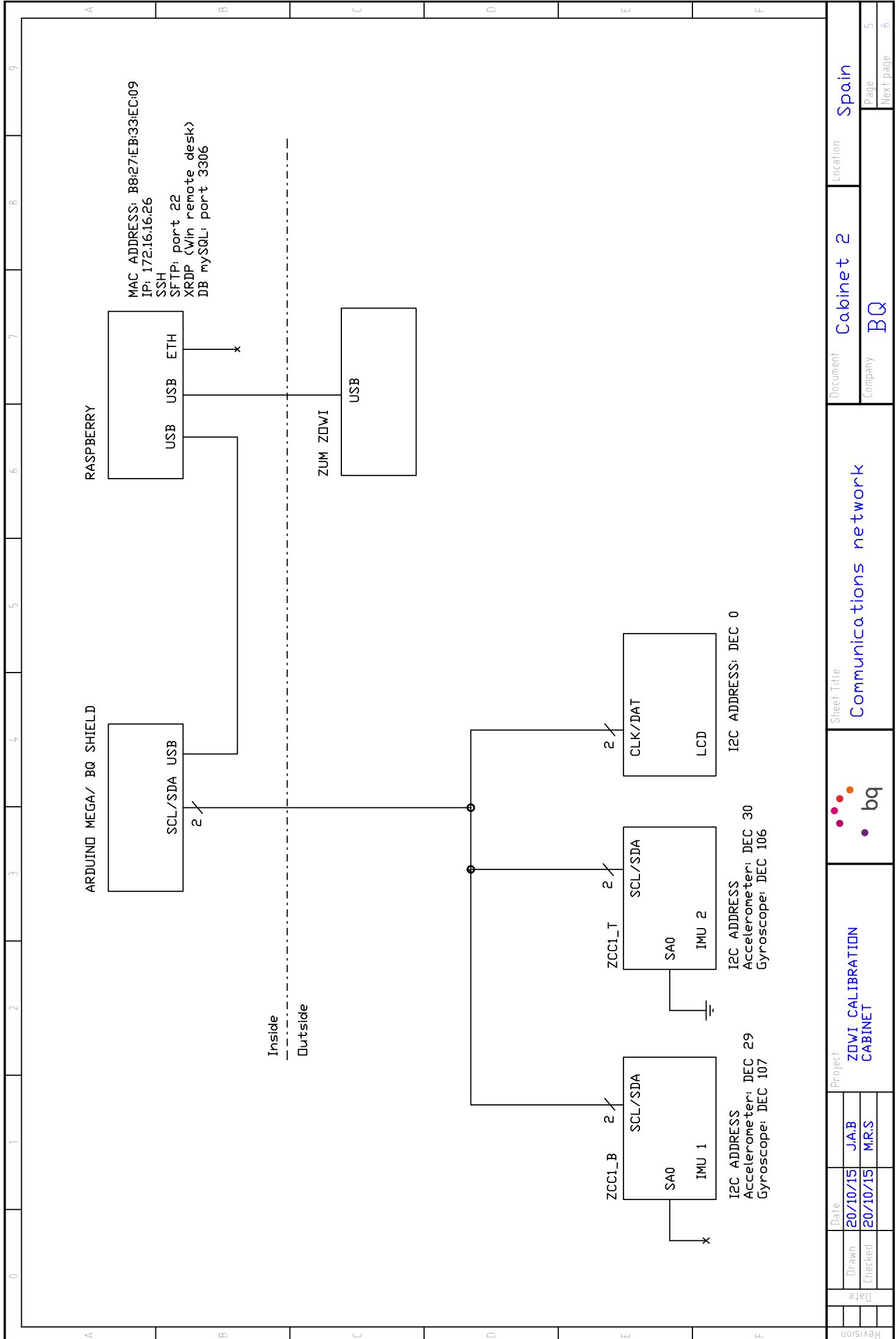
C 6

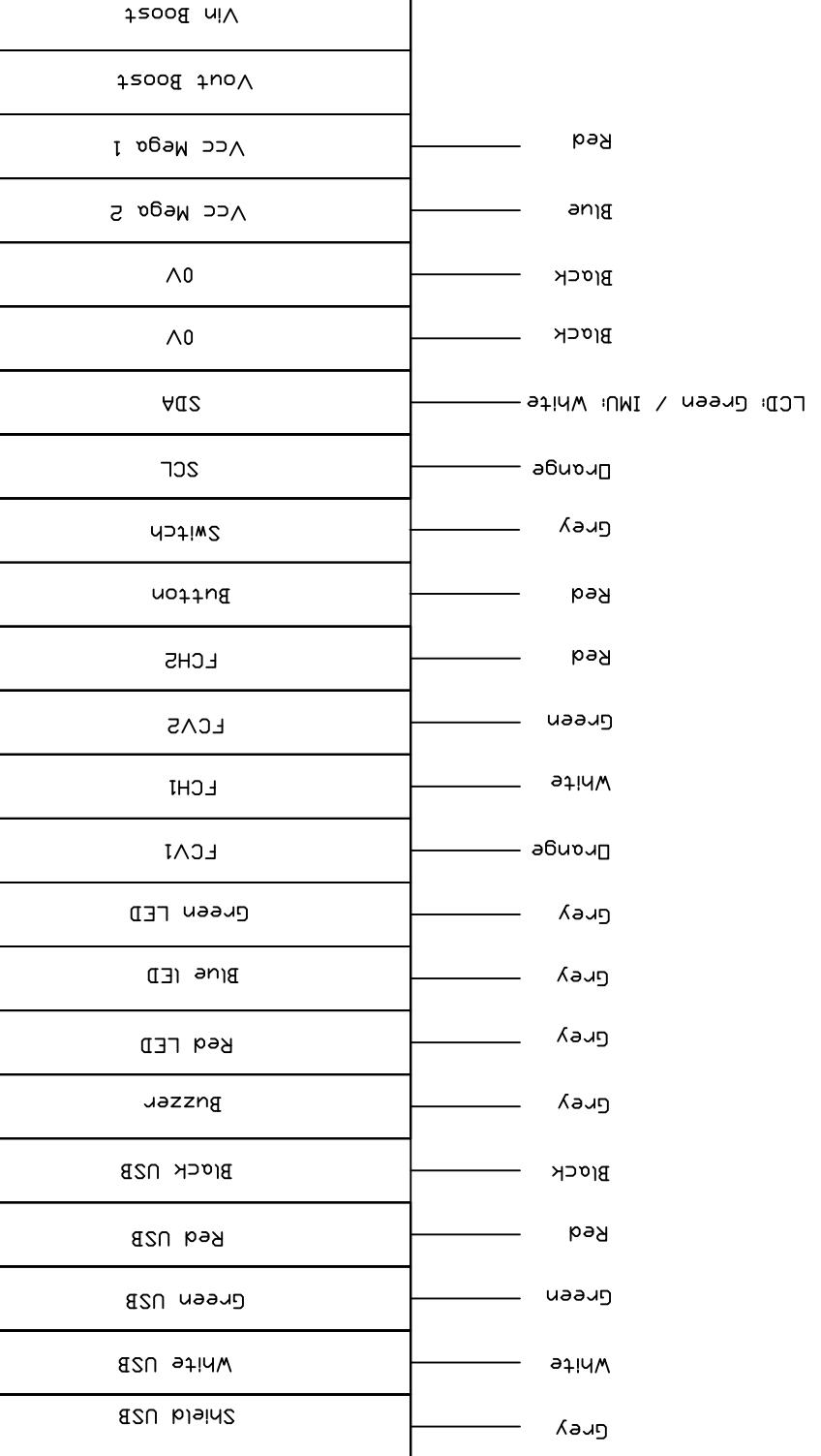
D 5

E 4

F 3

BØRNIER
13**RESET_LCD****GND****SWITCH BUTTON****BØRNIER**
P12**FCH2 FCV2****BØRNIER**
P10**FCH1****BØRNIER**
P11Document: **Cabinet 2** | Location: **Spain**Company: **BQ**Page: **4** | Next Page: **5**Sheet Title: **DI Signals**Project: **ZWI CALIBRATION**Revision: **A**Date: **20/10/15**Drawn: **J.A.B**Checked: **M.R.S**



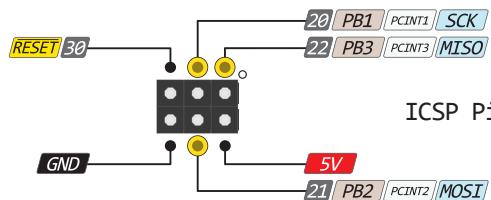
A	B	C	D	E	F								
9													
8													
7													
6													
5													
4													
3													
2													
1													
0													
A	B	C	D	E	F								
													
													
<p>Sheet Title: Bornier connections</p> <p>bq</p> <table border="1"> <tr><td>Project</td><td>ZWI CALIBRATION CABINET</td></tr> <tr><td>Date Drawn</td><td>20/10/15</td></tr> <tr><td>Date Checked</td><td>20/10/15</td></tr> <tr><td>J.A.B</td><td>M.R.S</td></tr> </table> <p>Document: Cabinet 2</p> <p>Location: Spain</p> <p>Company: BQ</p> <p>Page: 6</p> <p>Next page: -</p>						Project	ZWI CALIBRATION CABINET	Date Drawn	20/10/15	Date Checked	20/10/15	J.A.B	M.R.S
Project	ZWI CALIBRATION CABINET												
Date Drawn	20/10/15												
Date Checked	20/10/15												
J.A.B	M.R.S												
													

Apéndice D

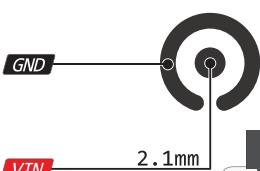
Mega PINOUT

Pinout de Freaduino Mega2560.

MEGA PINOUT

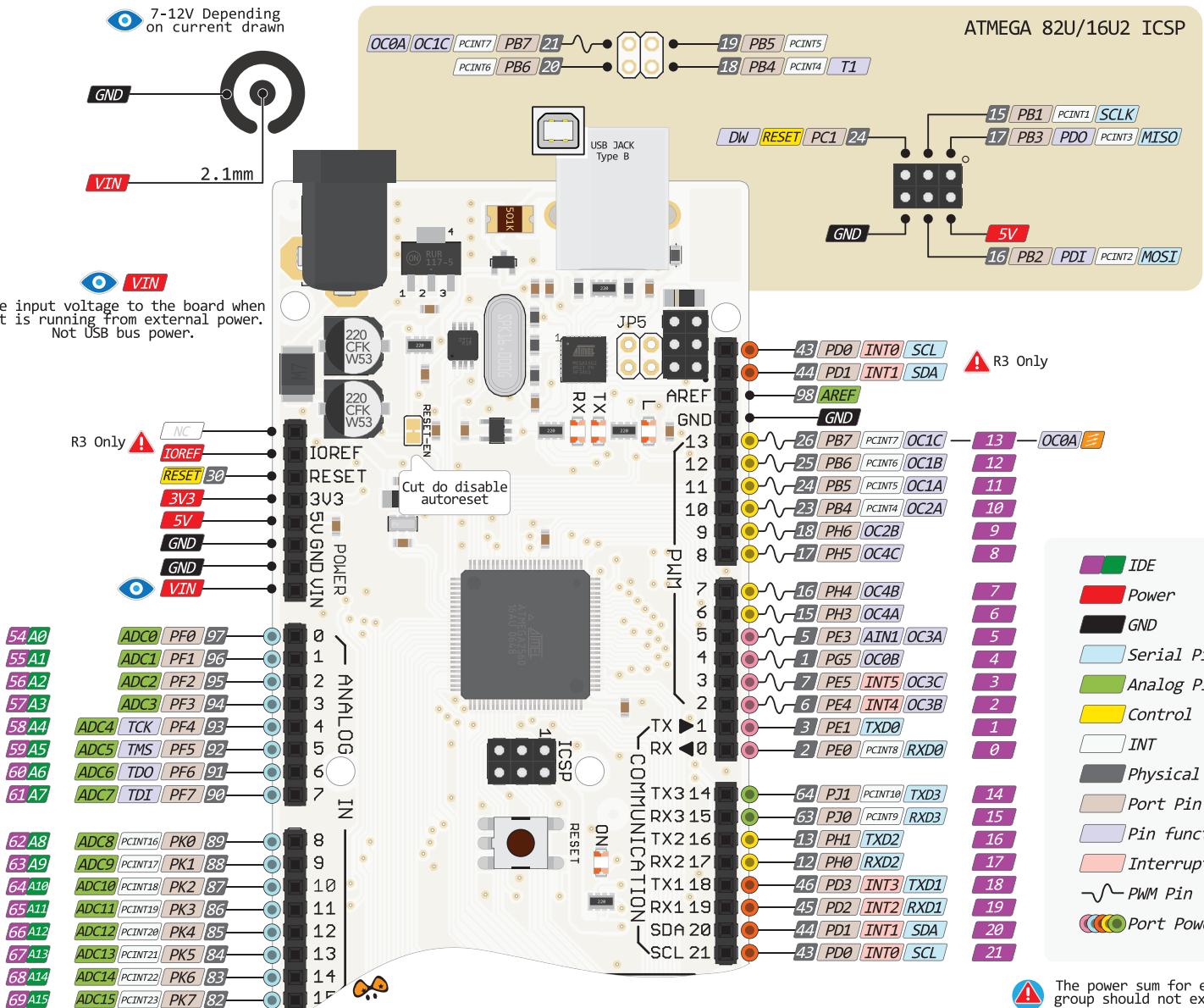


7-12V Depending on current drawn

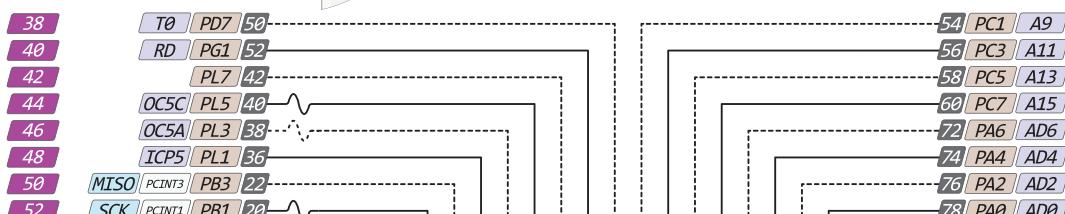
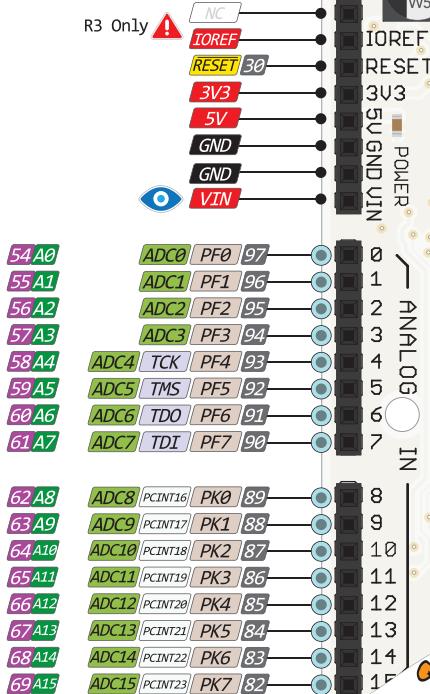


VIN

The input voltage to the board when it is running from external power. Not USB bus power.

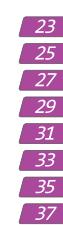
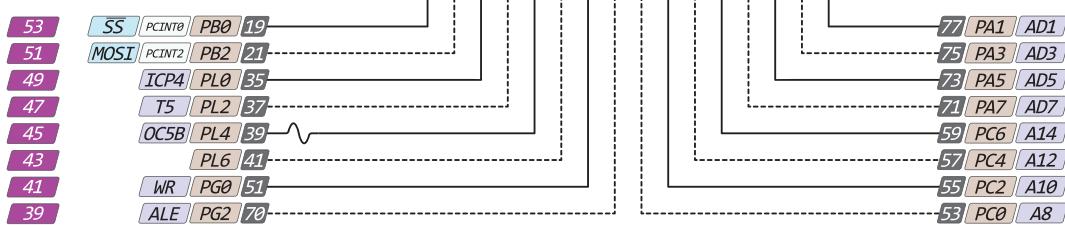


! R3 Only



! Absolute MAX per pin 20mA recommended 10mA

! Absolute MAX 200mA for entire package



www.bq.com



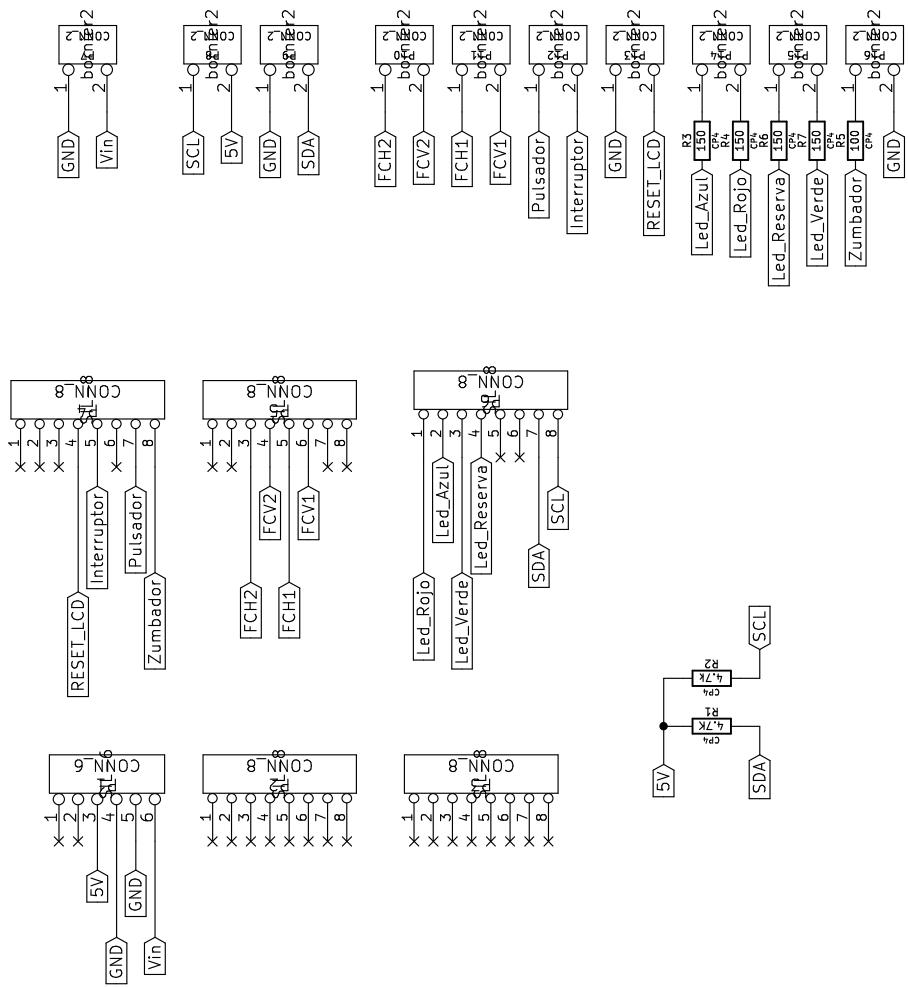
04 AUG 2014

ver 3 rev 1

Apéndice E

Mega Custom Shield

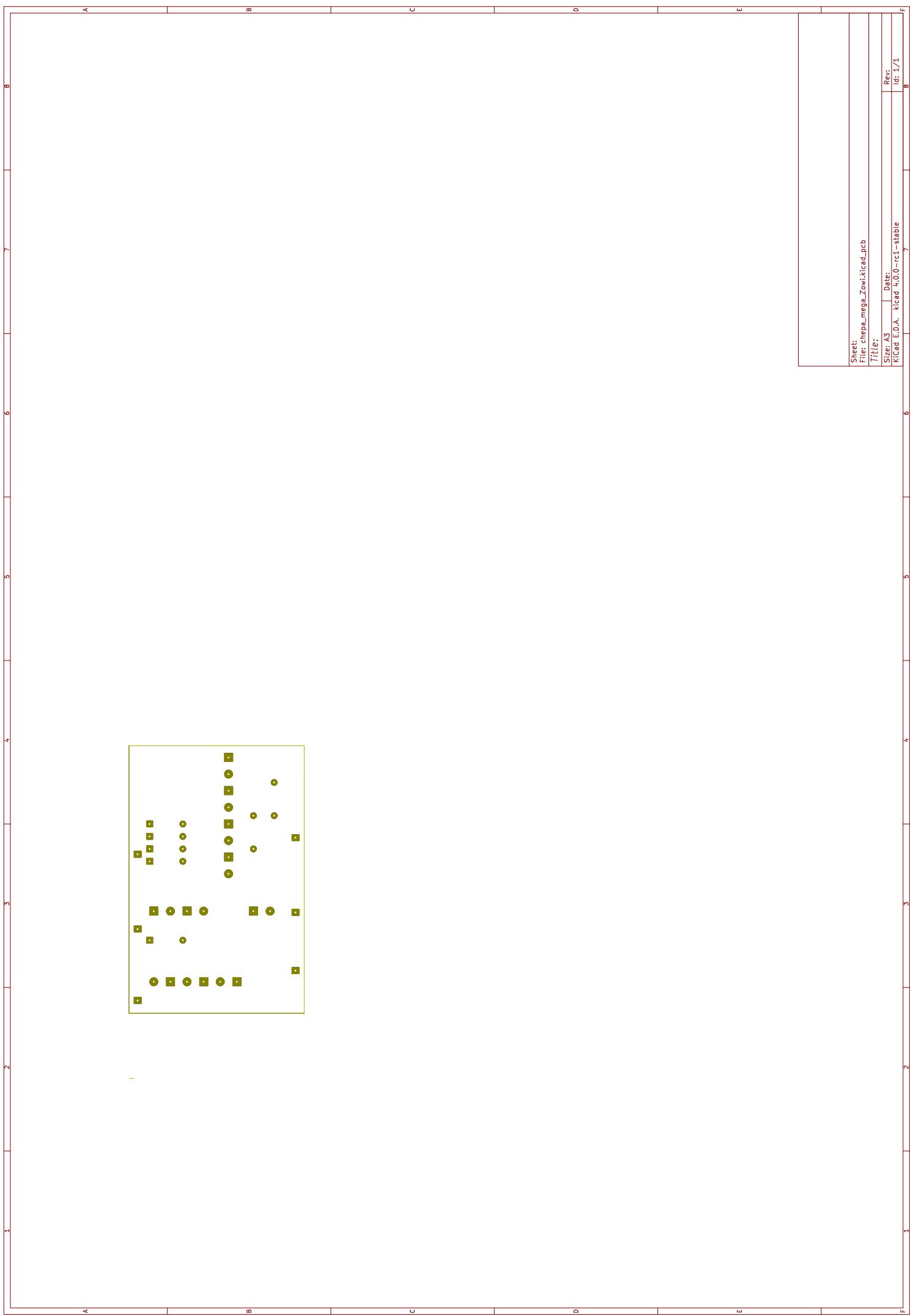
Esquemático y gerbers del shield diseñado para Mega.



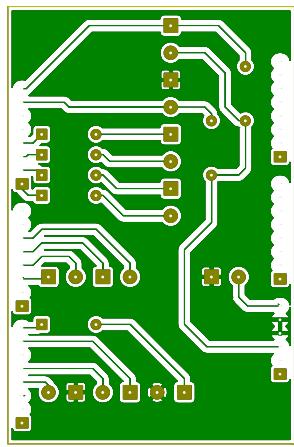
Sheet: /
File: chepa_mega_Zowi.sch

title: Size: A4 Date: 19 aug 2015
KiCad E.D.A. kicad 4.0.0 - rc1 - stable

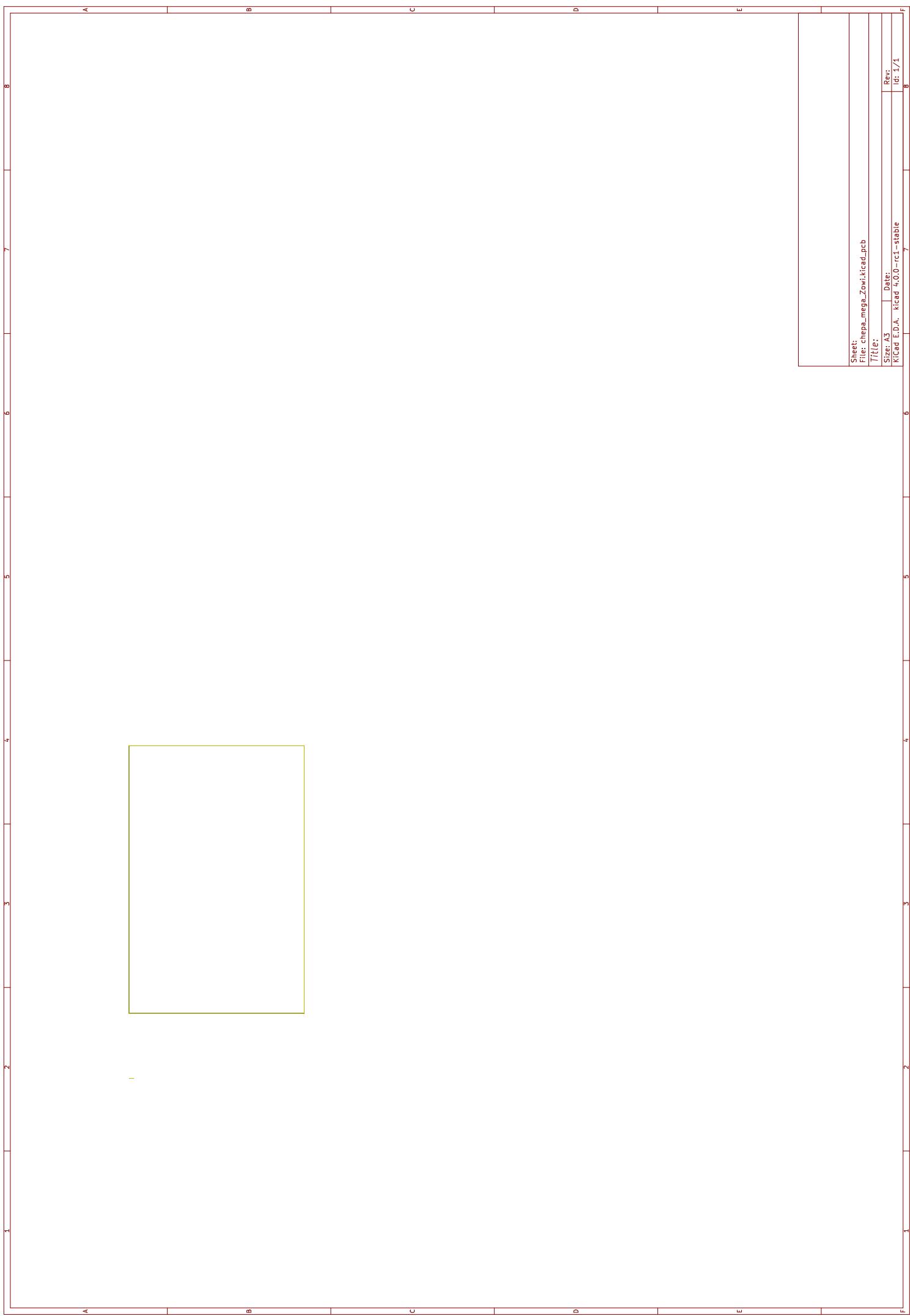
Title:



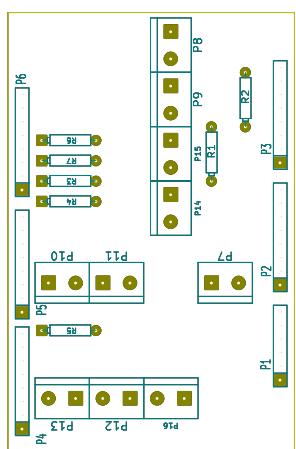
Sheet:
File: chpa_mega_Zow.kicad_pcb
Title:
Size: A3 Date: --rc1--stable
KiCad E.D.A. Rev: Id: 1 / 1

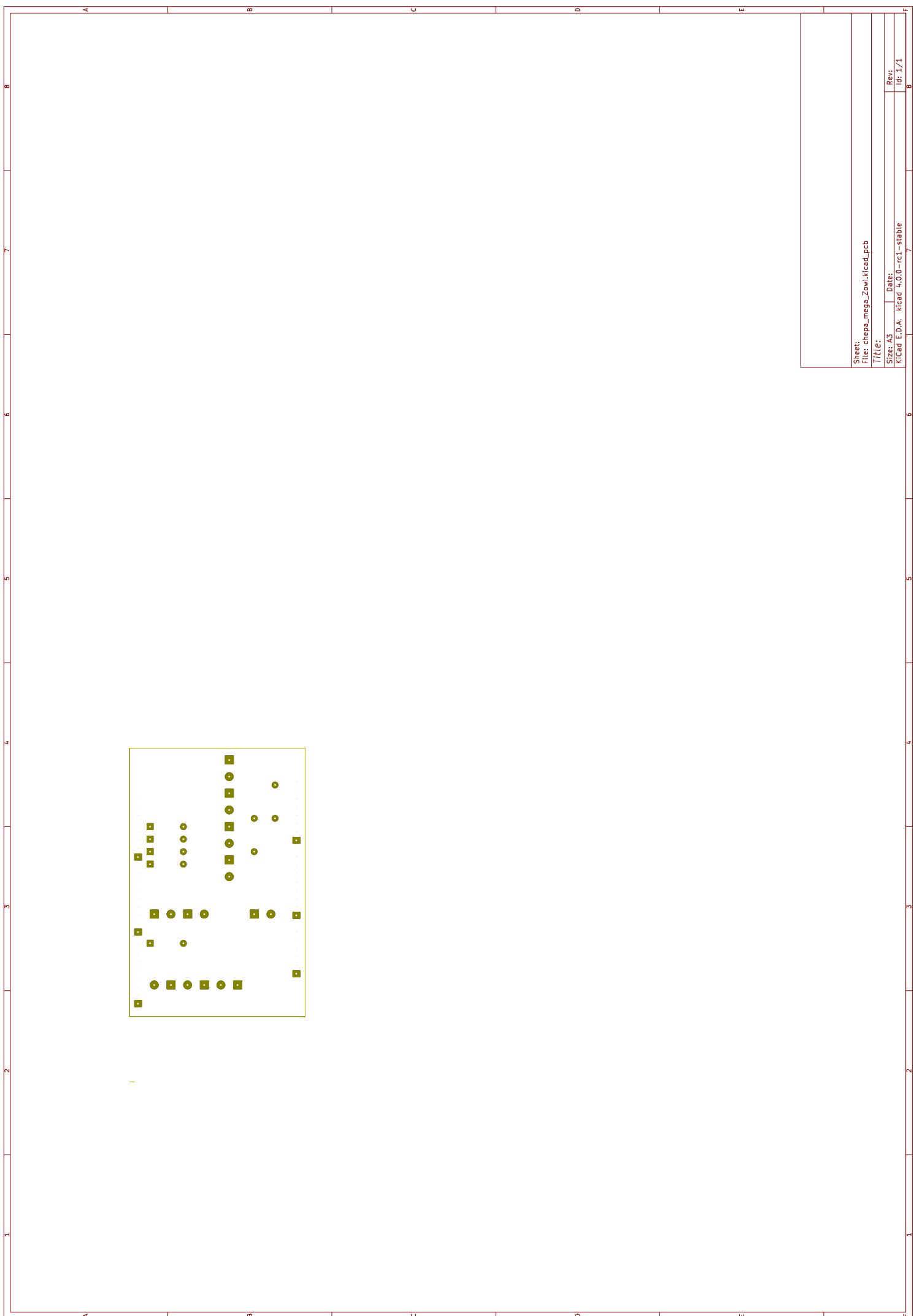


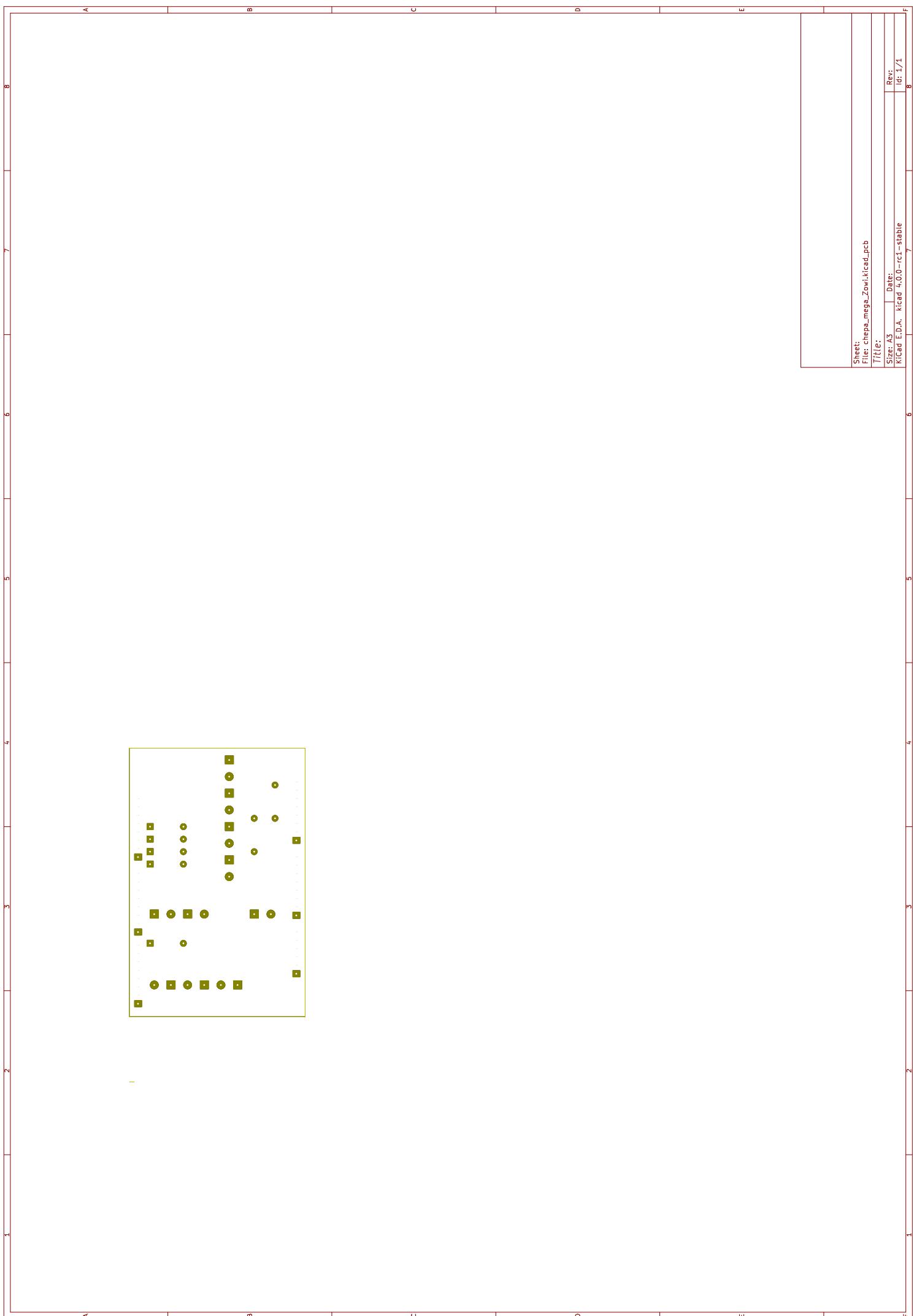
Sheet:
File: chpa_mega_20.kicad_pcb
Title:
Size: A3 Date:
KiCad EDA, kicad 4.0.0 -rc1-stable Rev:
Id: 1 / 1

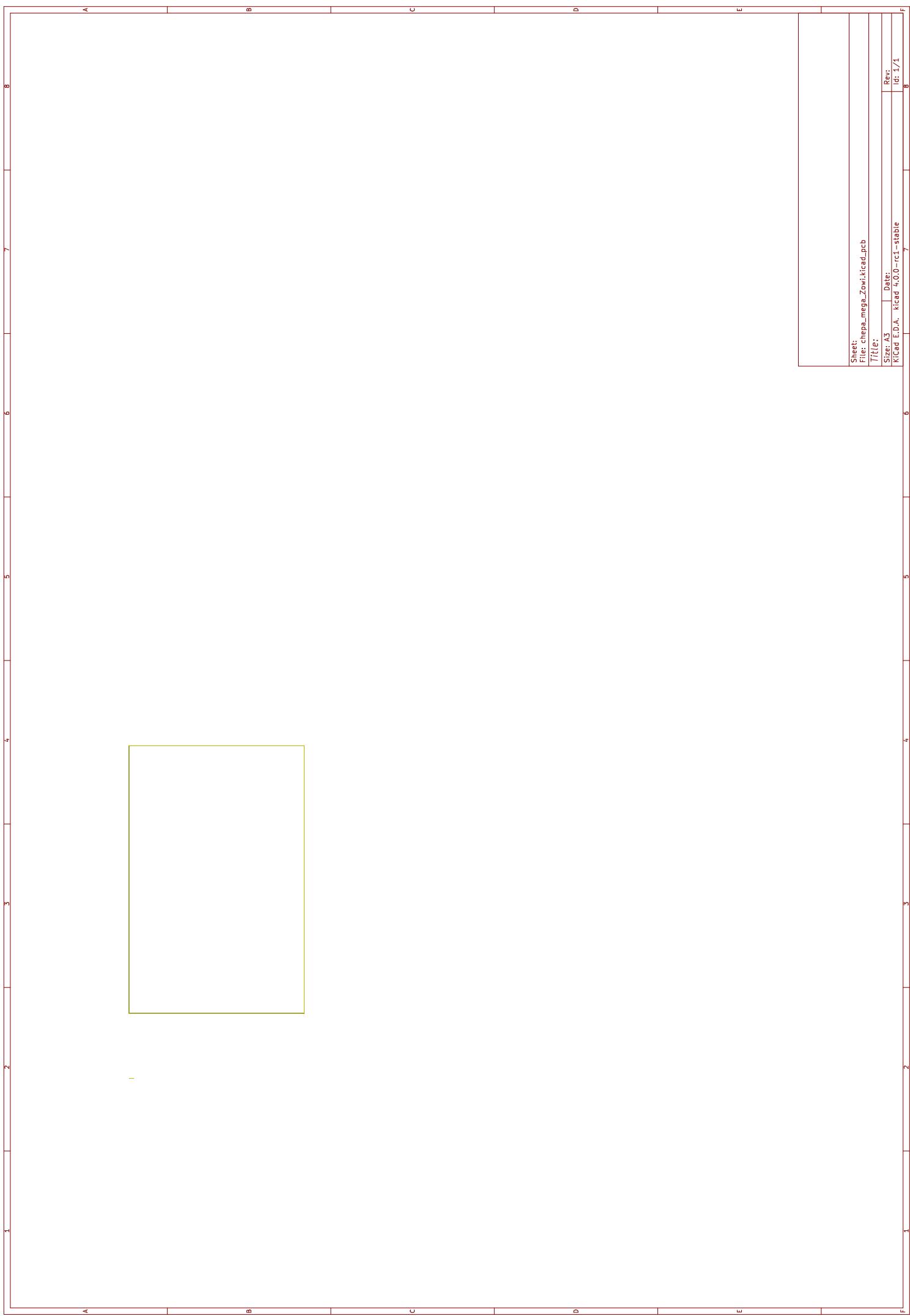


Sheet:
File: chpa_mega_Zowi.kicad_pcb
Title:
Size: A3 Date:
KiCad E.D.A. kicad 4.0.0 -rc1-stable Rev:
Id: 1 / 1





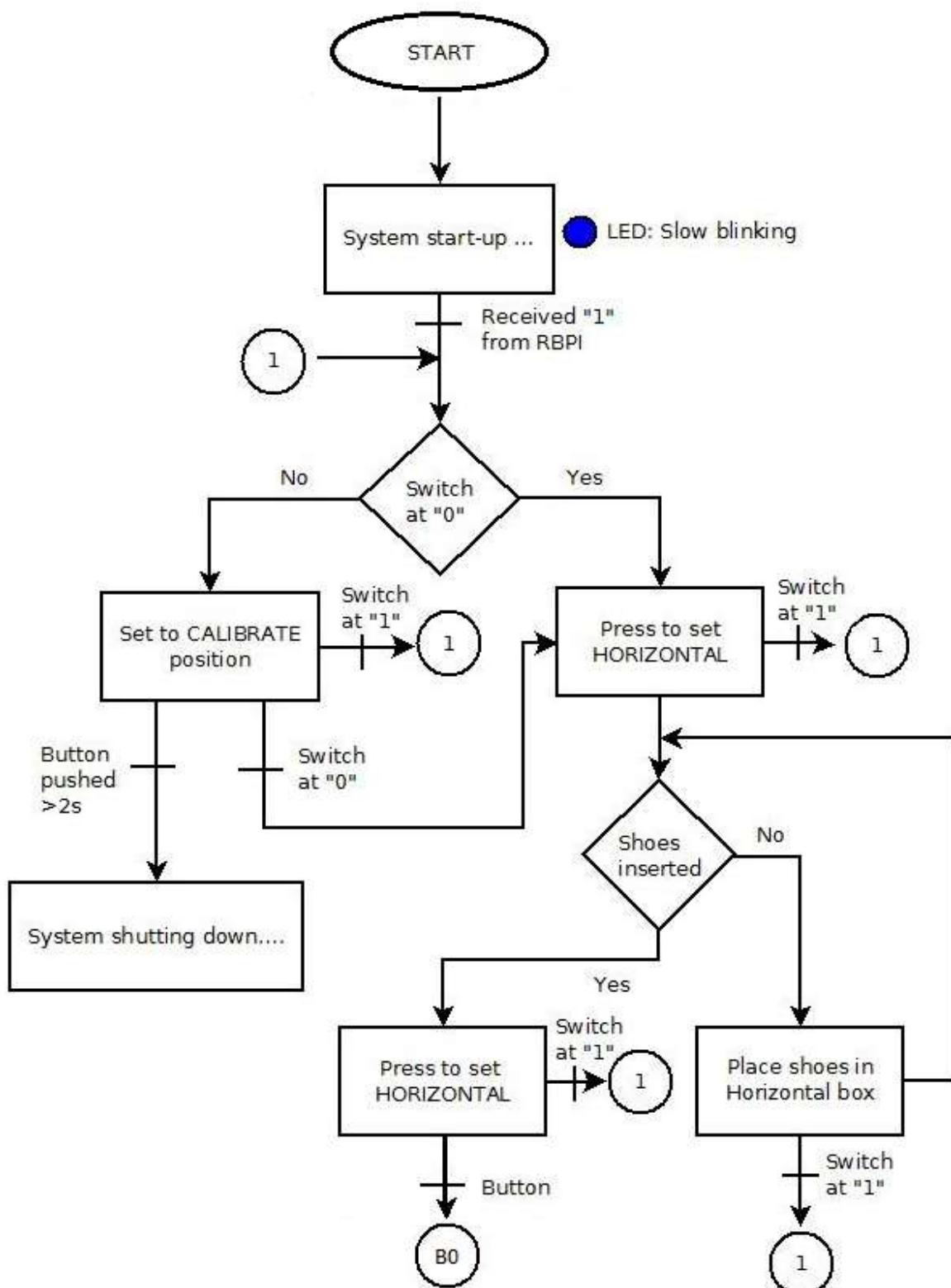


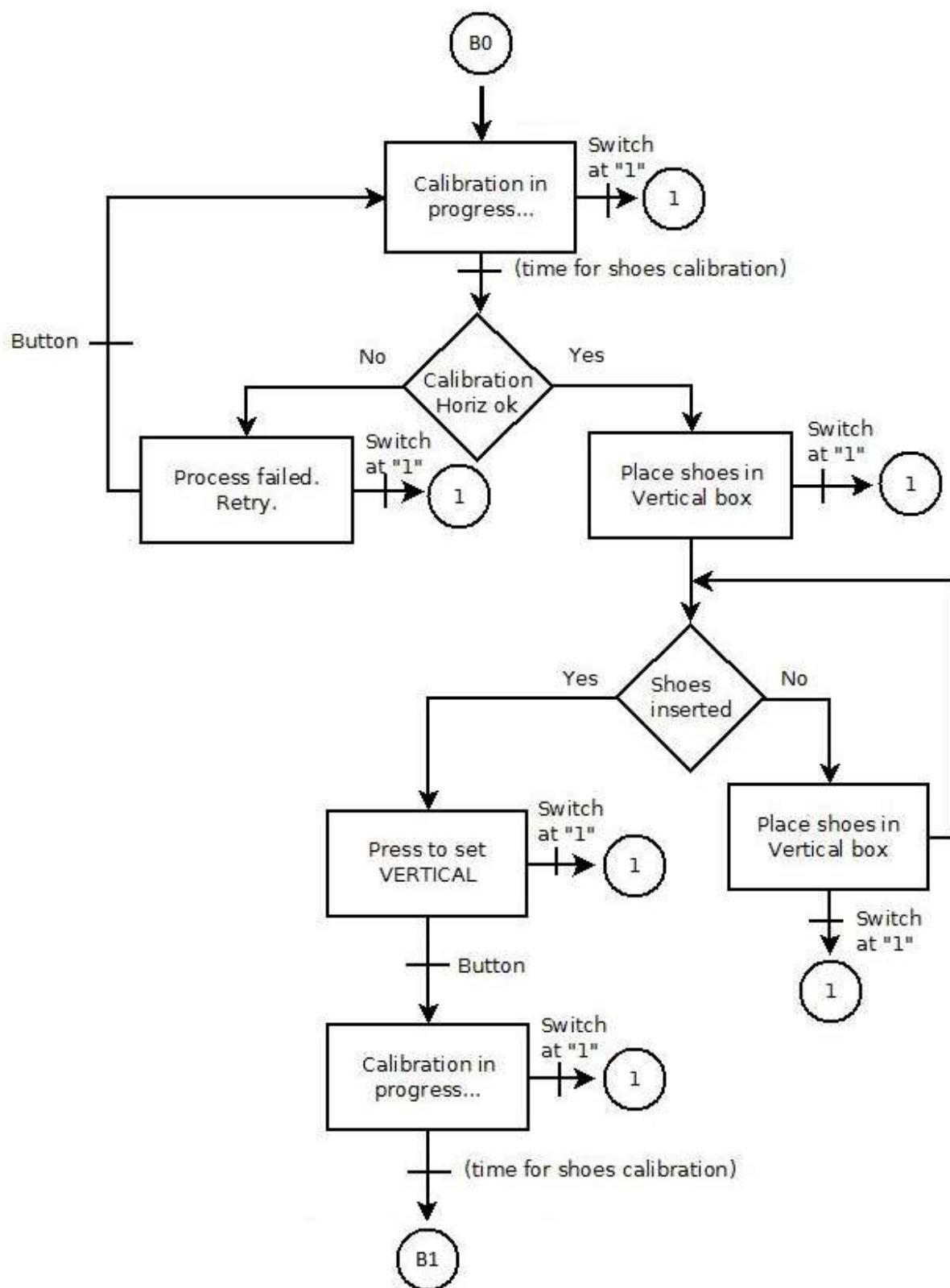


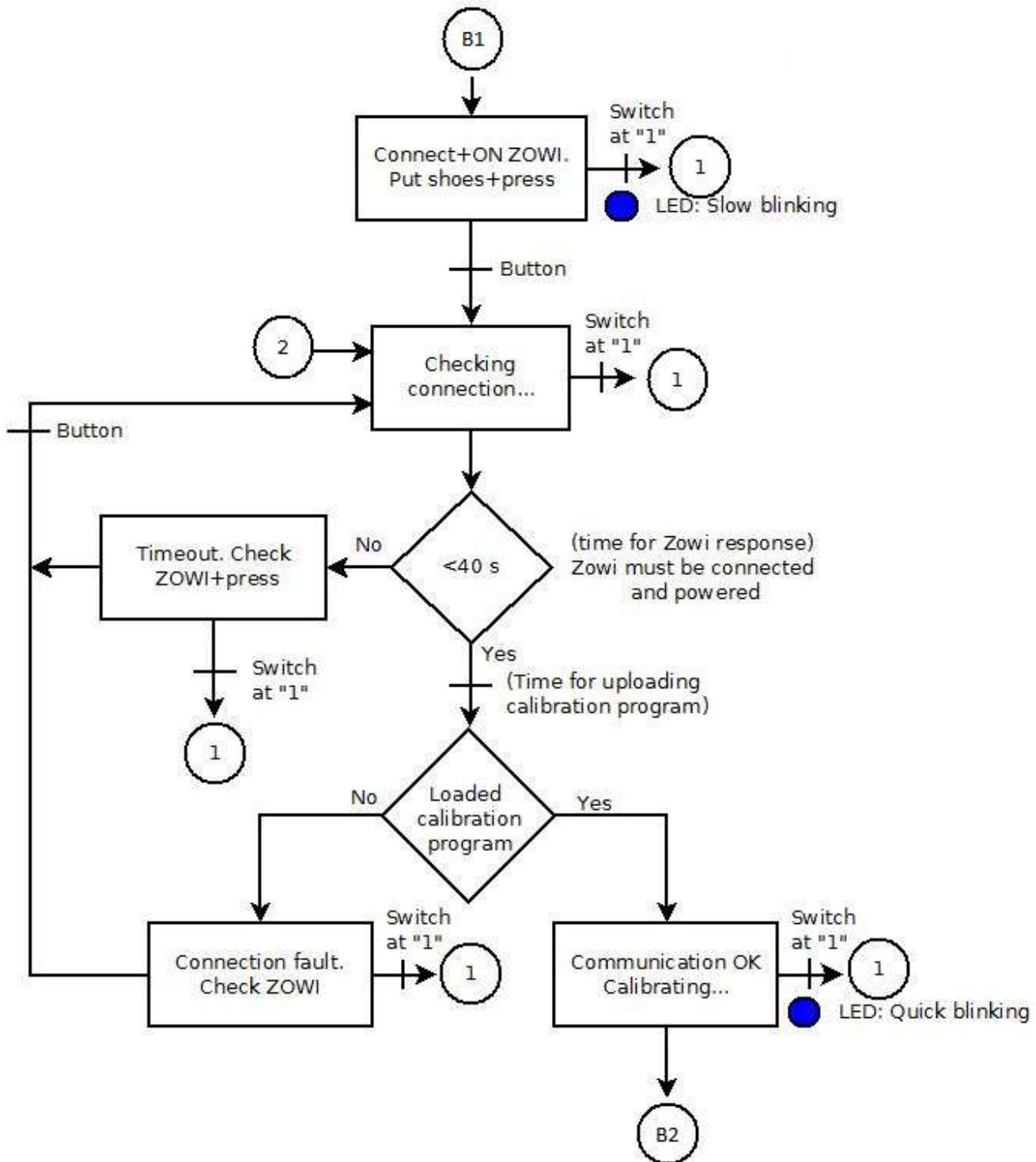
Apéndice F

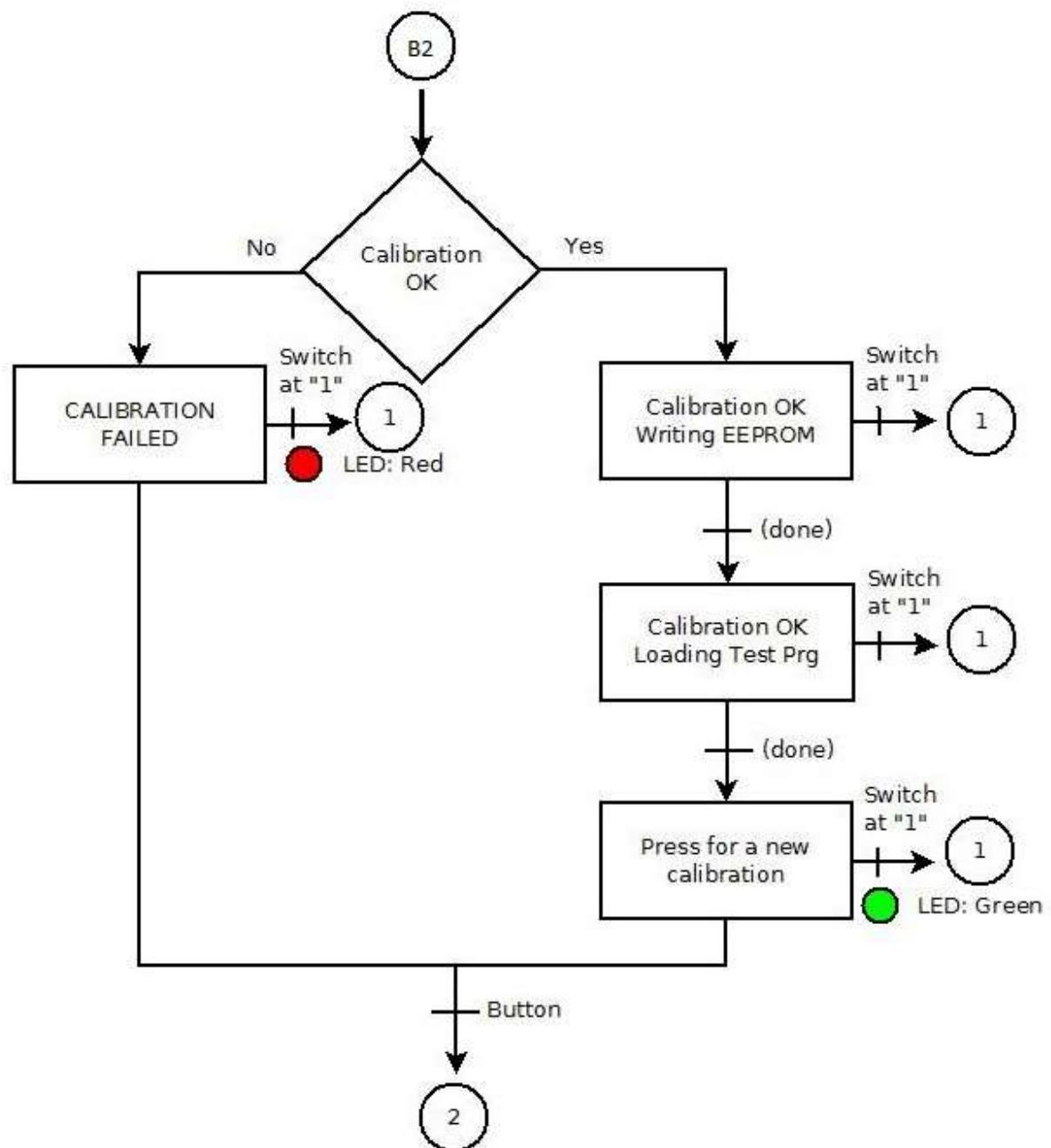
Máquina de estados Mega

Diagrama de la máquina de estados implementada en Mega.









Apéndice G

Configuración IMUS

Valores máximos y mínimos de la calibración de las IMUs instaladas en los dos armarios. Además de información de sus pines SA0, selector de direcciones de esclavo I²C.

IMUS Zowi Calibration Cabinet

ZCC1_T.

min: { -3212, -3273, -2653} max: { +3505, +3260, +4601}

sa0 - gnd bench: 1 foot: Top

ZCC2_T.

min: { -2566, -2723, -2993} max: { +5121, +3067, +3483}

sa0 - gnd bench: 2 foot: Top

ZCC1_B.

min: { -2666, -3446, -4782} max: { +4625, +3695, +4383}

sa0 - no gnd bench: 1 foot: Bottom

ZCC2_B.

min: { -3325, -3341, -4089} max: { +2887, +3480, +2803}

sa0 - no gnd bench: 2 foot: Bottom

Apéndice H

Estructura base de datos

En este documento se muestra la estructura de la base de datos, además de las direcciones de la EEPROM de Zowi donde se guardan las posiciones “offset” de los servos.

Zum EEPROM (Zowi):

[0]	[1]	[2]	[3]	[4]	[5]	[6]	
new home for left hip	new home for right hip	new home for left foot	new home for right foot	empty	1st letter for custom robot name (default value #)	2nd letter for custom robot name (default empty)	...

Database table columns (local and remote have same structure):

id	timestamp	serial_number	e_left_hip	e_right_hip	e_left_foot	e_right_foot	state	h_left_hip	h_right_hip	h_left_foot	h_right_foot
id	time stamp	Zum SN	calibration joint error	calibration joint error	calibration joint error	calibration joint error	OK/ NOK	new home position	new home position	new home position	new home position

Apéndice I

BOM de los armarios finales

Desglose de precios de los componentes del sistema final, para ambos armarios.

www.farnell.es			[€] Unitary cost	[€] Cost
Quantity	Ref	Description		
2	1123635	Fibox 398x298x182	88.16	176.32
2	1123670	Fibox EKIV43 placa montaje	10.16	20.32
45	149295	Entelec Uk 010500220 Terminal block, 2 pos, 10 awg	0.772	34.74
5	147494	Entelec uk 011836816, end plate	0.484	2.42
4	1750512	Entelec UK 1SNK50515R0000 ground block 4mm	3.25	13.00
2	852156	Canalización pvc 25x40, 2m	9.96	19.92
2	3094704	Legrand 0428 power socket, 10A/16A,250V,2P+E, din	16.37	32.74
2	2461029	Raspberry-pi 2, 1GB ram	32.87	65.74
2	2427499	Stontronics T5582DV, power adaptor, euoro,uk, 90V,264V, 10W,5V,2A	7.11	14.22
2	1516058	Multicomp JR-101-1-FRSG-02 INLET, IEC, with fuse holder	6.94	13.88
2	9667725	Bulgin PX0833socket (ethernet mount panel)	26.22	52.44
2	9652850	USB montaje panel receptaculo	17.21	34.42
2	2468268	Multicomp MC000950 cable, usb 2.0 a-micro b-male, 2m, black	2.85	5.70
1	1712543	Manguera 5 hilos, 30.5m, 22awg, 5.11mm diametro cable	48.49	48.49
10	1462823	Molex 50-57-9405 5 vias, 2.54mm	0.117	1.17
60	1777073	Conectores crimpars,26-22awg	0.112	6.72
10	2429543	Conector de Cable a Placa, Agujero Pasante, Header, 5, 2.54 mm	1.1	11.00
12	1178944	Lapp Kabel 53015130 Glandula de cable, 9mm gris	1.04	12.48
12	1178903	Contratuercas gris	0.448	5.38
2	3729497	Actuador del int, pulsador, 40mm, verde	10.27	20.54
2	3052989	Bloque de contacto 1NC, tornillo	5.93	11.86
2	1201896	Cable USB A to B 0,83m	3.1	6.20
2	1355986	Caja de empalmes IP55, gris, 60mmx40mm	1.35	2.70
4	4252019	Racor tubo a tubo 4 a 6 mm	4.23	16.92
2	4251969	Racor en T tubo,tubo,tubo 8mm	4.94	9.88
2	1671744	Cable red 2.5m, IEC a enchufe red (enchufe polonia C,E)	12.67	25.34
24	2008019	Bloque terminal PCB 3 vias, 26-12awg, 5,08mm	0.628	15.07
10	1593422	Conector placa a placa vertical, 2,54mm	0.109	1.09
50	9341315	Resistencia150 Ohm	0.0459	2.30
50	9341099	Resistencia 100Ohm	0.0459	2.30
50	9341951	Resistencia 4.7k	0.0449	2.25
10	1735362	Microinterruptor, SPDT, 10A, Long roller	2.06	20.60
			Total:	708.13

http://es.rs-online.com/			[€] Unitary cost	[€] Cost
Quantity	Ref	Description		
1	528-132	Tuerca hexagonal nylon M2.5 (Bolsa 50 unidades)	3.92	3.92
1	291-329	Tornillon nylon avellanado M2.5x6 (Bolsa 100 unidades)	7	7.00
			Total:	10.92

www.bricogeek.com			[€] Unitary cost	[€] Cost
Quantity	Ref	Description		
2	LCD-0003	LCD 16x2	14.5	29.00
2	PRO-0114	Adafruit LCD backpack I2C/SPI	8.95	17.90
2	PRO-0026	Conversor DC ajustable 4-25V 2A	10.4	20.80
			Total:	67.70

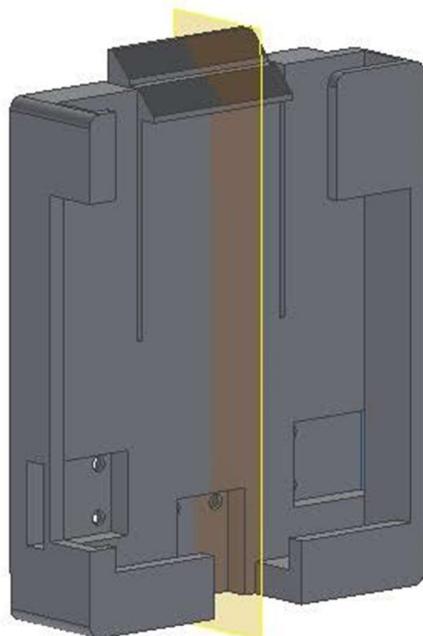
http://www.electronicaembajadores.com/Productos/Detalle/-1/SSACIM2/modulo-minimu-9			[€]	[€]
Quantity	Ref	Description	Unitary cost	Cost
4	SSACIM2	Modulo MinIMU-9 v3	33.84	135.36
		Total:		135.36

ELECROW			[€]	[€]
Quantity	Ref	Description	Unitary cost	Cost
10 -		PCBs shield Arduino Mega para armario calibración Zowi	1.29	12.90 \$
		Shipping		25.56 \$
		Aduanas		25.39 €
		Total:		59.23 €
			TOTAL;	981.35

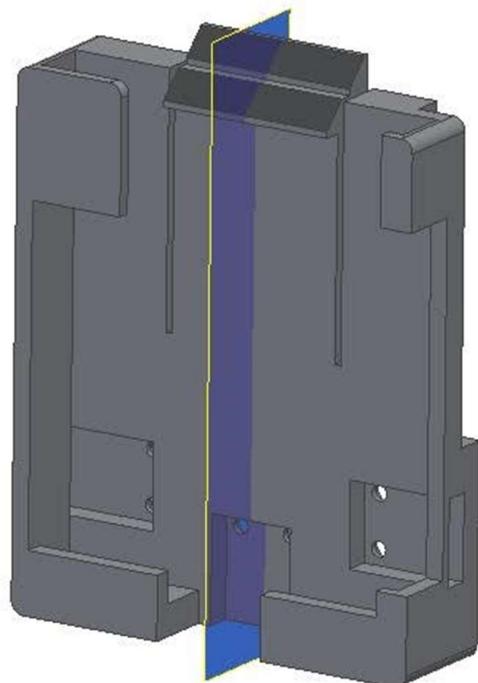
Apéndice J

Piezas Impresas

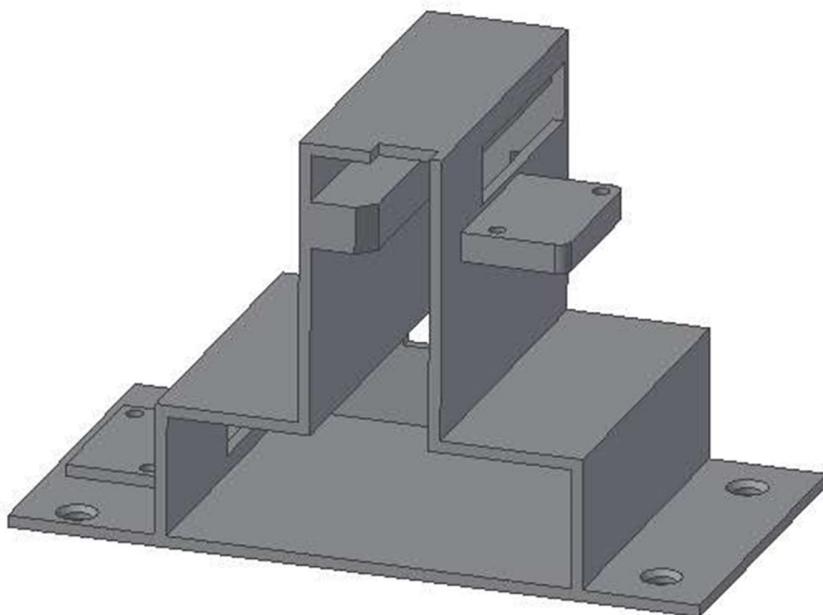
Piezas diseñadas e impresas para el desarrollo. El prototipo del sistema previo a la versión final (funcionando en el SAT de Rivas) utiliza estas piezas.

Piezas Impresas Prototipo

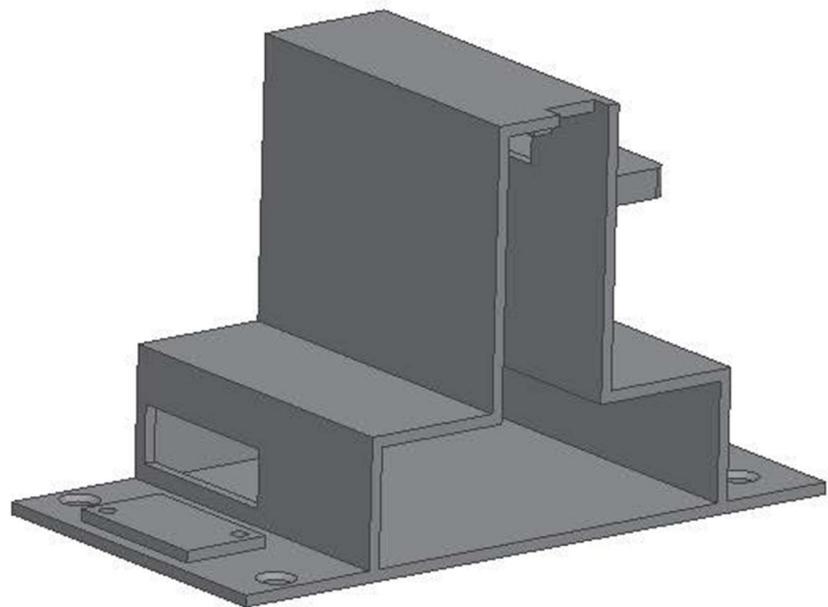
Zapato Top



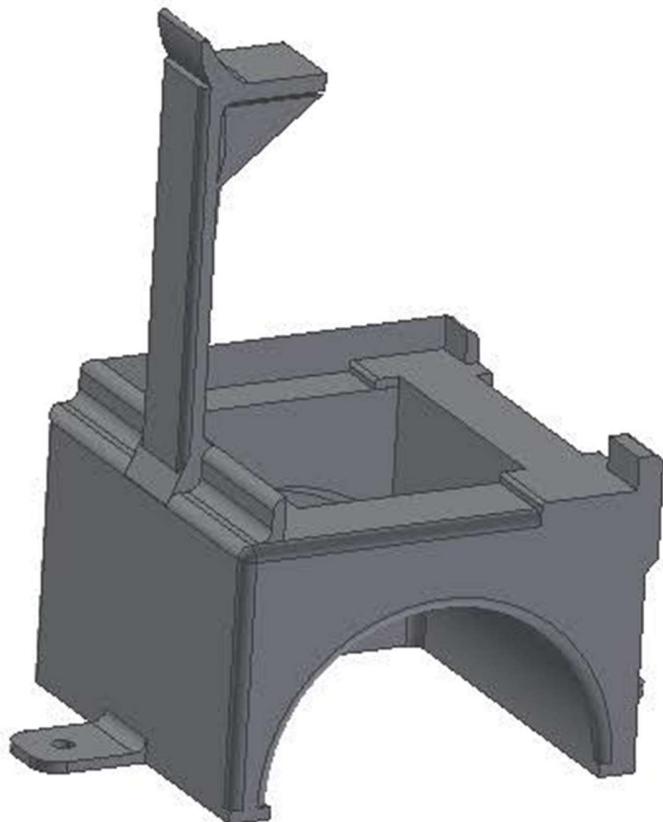
Zapato Bottom



Zapatero Bottom



Zapatero Top



Soporte Zowi

Apéndice K

Fichero running.sh

Fichero running.sh, encargado de lanzar la script principal de python y relanzarla si esta es interrumpida. Debe estar en /etc/init.d/running.sh.

```

#!/bin/bash

#variable bash para el fichero de lock
lockfile=/home/pi/zowi/python/tmp/zowi.lock
EJECUCION=$((0))
PID_OLD=$((0))

#borramos el directorio tmp por seguridad
rm -rf /home/pi/zowi/python/tmp/
#Creamos el directorio tmp nuevo (vacio)
mkdir /home/pi/zowi/python/tmp/

while true
do
    #Si no hay un fichero en el directorio tmp (zowi.lock) ->
    #lanzamos el script python
    LITEMS=`ls /home/pi/zowi/python/tmp/ | wc -l`
    if [[ "$LITEMS" == "0" ]]
    then
        #Comprueba si se está ejecutando script Python.
        #(si no se ejecuta lo lanza)
        if (set -o noclobber; echo $$ > "$lockfile") 2> /dev/null;
        then
            trap 'rm -f "$lockfile"; exit $?' INT TERM EXIT
            echo "Ejecutamos script"
            python /home/pi/zowi/python/main.mysql.py
            #Borramos el fichero lock por cierre del script de python
            rm -f "$lockfile"
            trap - INT TERM EXIT
        else
            echo "Ya se esta ejecutando el script"
            echo "PID de running.sh: $(cat $lockfile)"
        fi
    fi

    #Modificacion matar avrdude
    PID=`ps -A | grep avrdude | awk '{print $1}'`
    if [ "$PID" != "$PID_OLD" ];then
        EJECUCION=$((0))
    fi
    if [ "$PID" != "" ];then
        EJECUCION=$((EJECUCION+1))
    fi
    if [ $EJECUCION -gt 4 ];then
        sudo kill -9 $PID
        EJECUCION=$((0))
    fi
    PID_OLD=$((PID))

    #Tiempo para reiniciar el script
    sleep 4
done

```

Apéndice L

Fichero ZowiInit

ZowiInit lanzará la script running.sh al arranque del sistema. Se muestran también las instrucciones para hacerlo funcionar una vez creado y alojado en */etc/init.d/*.

Fichero:

```
#!/bin/sh
# /etc/init.d/ZowiInit

### BEGIN INIT INFO
# Provides:          ZowiInit
# Required-Start:    $all
# Required-Stop:     $remote_fs $syslog
# Default-Start:    2 3 4 5
# Default-Stop:     0 1 6
# Short-Description: Script arranque calibracion Zowi
# Description:       Script para arranque programa python calibracion Zowi
### END INIT INFO

case "$1" in
  start)
    echo "Starting ZowiInit"
    #!/usr/bin/python /home/pi/zowi/python/main.mysql.py
    /etc/init.d/running.sh
    ;;
  stop)
    echo "Stopping ZowiInit"
    ;;
  *)
    echo "Using mode: /etc/init.d/ZowiInit {start|stop}"
    exit 1
    ;;
esac

exit 0
```

Ahora hacemos el fichero ejecutable:

```
sudo chmod 755 /etc/init.d/ZowiInit
```

Comprobamos que todo se ejecuta correctamente:

```
sudo /etc/init.d/ZowiInit start
```

Y por último activamos el arranque automático:

```
sudo update-rc.d ZowiInit defaults
```

Apéndice M

Autologin al inicio

Logeo de usuario automático al iniciar el sistema, sin pedir usuario ni contraseña.

Auto Login:

Edit /etc/inittab

```
sudo nano /etc/inittab
```

Scroll down to:

```
1:2345:respawn:/sbin/getty 115200 tty1
```

and change to:

```
#1:2345:respawn:/sbin/getty 115200 tty1
```

Under that line add:

```
1:2345:respawn:/bin/login -f pi tty1 </dev/tty1 >/dev/tty1 2>&1
```

save the file and exit, (ctrl + o to save and ctrl + x to exit)

```
sudo nano /etc/rc.local
```

Auto StartX (Run LXDE)

Edit /etc/rc.local

```
sudo nano /etc/rc.local
```

Scroll to the bottom and add the following above exit 0:

```
su -l pi -c startx
```

(where pi is the username you want to run X as).

Reboot the machine.

Apéndice N

Reglas puertos USB

Instrucciones del fichero rules para asignar alias a los puertos USB.

Se sigue el manual:

<http://polaridad.es/nombre-fijo-puerto-serie-usb-arduino/>

El fichero de reglas lo llamamos:

50-usbportsbq.rules

y añadimos las siguientes 2 líneas:

```
KERNEL == "ttyUSB[0-9]*", SUBSYSTEMS=="usb", ATTRS{idVendor}=="0403",  
ATTRS{idProduct}=="6001", SYMLINK="mega", MODE="0660", GROUP="dialout"
```

```
KERNEL == "ttyUSB[0-9]*", SUBSYSTEMS=="usb", ATTRS{idVendor}=="10c4",  
ATTRS{idProduct}=="ea60", SYMLINK="zowi", MODE="0660", GROUP="dialout"
```

Apéndice N

Notas de instalación Raspberry

Documento de anotaciones sobre la configuración realizada en Raspberry.

Imagen raspian en sd:

- Con win32 DiskImager en windows
- Con [<https://computadorasdeplacareducida.wordpress.com/2014/04/08/instalar-raspbian-en-sd-utilizando-linux-ubuntu/>]

Al encender la raspberry la primera vez nos dará la opción de expandir la partición.

Configuración inicial:

- sudo raspi-config
 - 4 Internationalisation Options -I2 Change Timezone - I3 Change Keyboard Layout
 - ssh - Enable

Actualizar:

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

Instalar mySQL [<https://geekytheory.com/tutorial-raspberry-pi-15-instalacion-de-apache-mysql-php/>]:

- sudo apt-get install mysql-server mysql-client

Instalar acceso remoto desde windows (conexión a escritorio remoto)

[<http://comiendorasps.blogspot.com.es/2013/04/accediendo-remotamente-la-raspberry-con.html>]:

- sudo apt-get install xrdp

Instalar python:

- Instalaremos los paquetes necesarios para trabajar con python [1]:
 - python-dev python-pip python-serial
 - pip
 - virtualenv
- \$ sudo apt-get install python-dev python-pip python-serial
- \$ sudo pip install virtualenv pymysql

SFTP (filecilla puerto 22):

- Copiar programa python y hex necesarios en /home/pi/zowi/python/

Ejecutar script al inicio [<http://nideaderedes.urlansoft.com/2013/12/20/como-ejecutar-un-programa-automaticamente-al-arrancar-la-raspberry-pi/>]

Permitir acceso desde host exterior a base de datos:

```
$ mysql -u root -p

GRANT ALL ON *.* to root@'%' IDENTIFIED BY 'toor';
FLUSH PRIVILEGES;
exit;
```

```
editar fichero /etc/mysql/my.cnf  
comentar con # línea (~47)  
# bind-address      = 127.0.0.1
```

```
guardar fichero y reiniciar servidor:  
$ sudo /etc/init.d/mysql restart
```

Instalar Arduino 1.0.6

```
sudo apt-get install arduino
```

```
cd Downloads/  
wget http://arduino.cc/download.php?f=/arduino-1.0.6-linux32.tgz  
tar zxvf arduino-1.0.6-linux32.tgz
```

```
cd ~/Downloads/arduino-1.0.6  
rm -rf hardware/tools
```

Instalación de librerías:

```
cd ~/Downloads/arduino-1.0.6  
sudo cp -ru lib /usr/share/arduino  
sudo cp -ru libraries /usr/share/arduino  
sudo cp -ru tools /usr/share/arduinodosudo  
sudo cp -ru hardware /usr/share/arduino  
sudo cp -ru examples /usr/share/doc/arduino-core  
sudo cp -ru reference /usr/share/doc/arduino-core
```

Cargar programas compilados (HEX) en:

Arduino MEGA:

```
avrdude -patmega2560 -cwiring -P/dev/ttyUSB0 -b115200 -D -Uflash:w:/home/pi/fichero.hex:i
```

Arduino ZUM:

```
avrdude -patmega328p -carduino -P/dev/ttyUSB0 -b 115200 -D -  
Uflash:w:/home/pi/zowi/python/zowi_offset_i2c.cpp.hex:i
```

*NOTA: ttyUSB0 será cambiado posteriormente al dar un alias al puerto en cuestión.

Arranque automático de programa en el inicio:

<http://nideaderedes.urlansoft.com/2013/12/20/como-ejecutar-un-programa-automaticamente-al-arrancar-la-raspberry-pi/>

Recuperación ejecución script python (main.mysql.py) si cierre:

/etc/init.d ->running.sh

Hacer ejecutable los ficheros:

\$ chmod +x /.../...

Ampliación máxima corriente por la bahía usb:

Dentro de: /boot/config.txt

max_usb_current=1
safe_mode_gpio=4

Apéndice O

Código Raspberry

Programa principal de la Raspberry, escrito en python: Fichero con nombre **main.mysql.py**.

```
# -*- coding: utf-8 -*-
#####
#####
#####
#####
#####

# Importamos librerias necesarias para el programa
import serial
import subprocess
from time import sleep
import os
import pymysql

# Variables globales
leer = True
comando= ""          # Iremos almacenando el comando que leeamos
connected=False       # Flag de detección de mega conectada
start_char = "$"      # Caracter que indica el inicio del comando
end_char = "#"        # Caracter que indica final del comando
char_lim = '*'        # Caracter delimitador de comandos

# Lista con los errores de la calibracion
errores=[]

# Lista con la posicion de los home nuevos
homes=[]

# numero de serie del arduino
num_serie = ""

# Variables relacionadas con las BBDD
# Datos del servidor
config = {
    'user': 'root',
    'passwd': 'toor',
    'host': '172.16.16.15',
    'db': 'zowi',
}

# Datos del localhost
config_localhost = {
    'user': 'root',
    'passwd': 'toor',
    'host': '127.0.0.1',
    'db': 'zowi',
```

```
}

# Tabla usada
table = 'calibracion'

# Secuencia SQL
sql = "INSERT INTO " + table + " (serial_number,e_left_hip, e_right_hip, e_left_foot, e_right_foot,
    state, h_left_hip, h_right_hip, h_left_foot, h_right_foot) VALUES (%s, %s, %s, %s, %s,%s,
    %s, %s, %s, %s)"

def funcionPuertos():
    status = 0
    try:
        subprocess.check_call("echo '1-1.5' | sudo tee /sys/bus/usb/drivers/usb/unbind", shell=True)
    except:
        status+=1
        print("Error unbind")
    else:
        print("OK unbind")
    #sleep(1)
    try:
        subprocess.check_call("echo '1-1.5' | sudo tee /sys/bus/usb/drivers/usb/bind", shell=True)
    except:
        status+=2
        print("Error bind")
    else:
        print("OK bind")
    #sleep(1)
    return status

#####
##### FUNCION MAIN()
try:
    #####
    # Nos conectamos a la base de datos remota
    #print("Nos conectamos a la BBDD")
    #try:
    #    pass
    #    # Quitar la conexion a la base de datos remota
    #    #db = pymysql.connect(**config)
    #    # Si hay algun error en la conexion devolvemos un error
```

```
#except pymysql.DatabaseError as e:
#    print("Error %d: %s" % (e.args[0], e.args[1]))
# Si no se produce ningun error continuamos la ejecucion
#else:
#    # Automaticamente hace un commit de los queries que reciba
#    db.autocommit(1)
#    print("Conectado a la BBDD remota")
#    cursor = db.cursor()

#####
# Escribimos en las bbdd remota un mensaje de inicio
# try:
#     # mandamos directamente el diccionario donde están almacenados los errores.
#     # Los valores de errores, se mandan tal cual son recibidos de la Mega
#     cursor.execute( sql, (1, 1,1,1, 1,1,1, 1,1,1))
#     # Si hay algun error en la conexion devolvemos un error
# except pymysql.DatabaseError as e:
#     print("Error %d: %s" % (e.args[0], e.args[1]))
#     # Si no hemos podido conectar a la BBDD al inicio:
# except NameError:
#     print("No se pudo conectar al base de datos remota en el inicio")
# except:
#     print("Se ha producido un error al insertar la
#           sentencia en la base de datos remota")
#     # Si la sentencia se ha introducido correctamente
# else:
#     print("Sentencia introducida correctamente en remoto")
# #Cerramos el curso
#         #cursor.close()

#####
# Codigo de inserccion de la base de datos local
try:
    db_localhost = pymysql.connect(**config_localhost)
# Si hay algun error en la conexion devolvemos un error
except pymysql.DatabaseError as e:
    print("Error %d: %s" % (e.args[0], e.args[1]))
# Si no se produce ningun error continuamos la ejecucion
else:
    # Automaticamente hace un commit de los queries que reciba
    db_localhost.autocommit(1)
    print("Conectado a la BBDD local")
```

```
cursor_localhost = db_localhost.cursor()

#####
# Escribimos en la bbdd local mensaje de inicio
try:
    #mandamos directamente el diccionario donde estan almacenados los errores.
    #Los valores de errores, se mandan tal cual son recibidos de la Mega
    cursor_localhost.execute( sql, (1, 1,1,1, 1,1,1, 1,1,1 ))
    #Si hay algun error en la conexion devolvemos un error
except pymysql.DatabaseError as e:
    print("Error %d: %s" % (e.args[0], e.args[1]))
# Si no hemos podido conectar a la BBDD al inicio:
except NameError:
    print("No se pudo conectar al base de datos local en el inicio")
except:
    print("Se ha producido un error al insertar la
          sentencia en la base de datos local")
# Si la sentencia se ha introducido correctamente
else:
    print("Sentencia introducida correctamente en local")
#Cerramos el curso r
    cursor_localhost.close()
#####
#####
# Abrimos comunicacion con la placa de la MEGA
# en USB0 siempre va a ir la mega
print("Abriendo comunicacion con MEGA")
while(connect==False):
    try:
        mega = serial.Serial("/dev/mega",115200,timeout=0.2,dsrdtr=True )
    except serial.SerialException:
        print("Mega no conectada")
        sleep(5)
    else:
        # Limpiamos la informacion que haya en el serial

        # Espera obligatoria para reiniciar la mega
        sleep(5)
        print("Mandamos uno a la mega")
        mega.flushInput()
        mega.flushOutput()
        mega.write('1')
```

```
sleep(0.5)
connected=True

#####
# LOOP()
#####

print("Esperamos datos de MEGA")
while(1):
    data = mega.read()
    # La trama del mensaje va entre $ y # es decir:
    # $xxxxxxxxxxxxxx#.
    # Por tanto leemos datos desde que recibimos un $ y leemos hasta el #
    # Estos caracteres se pueden cambiar ya que se almacenan en una variable
    while(data != start_char):
        data = mega.read()
    data = mega.read()
    while (data != end_char):
        comando = comando + data
        data = mega.read()
    print(comando)
    print("Mandamos datos a la zum")
    if (comando[:4] == "IZUM"): # Comando de inicializacion de la zum
        comando = ""
    print("Vamos a programar la zum")
    # Programamos la ZUM
    status = funcionPuertos()
    if status == 1:
        try:
            cursor_localhost.execute(sql,(61,61,61,61,61,61,61,61,61))
        except:
            print("Error guardar 61")
    elif status == 2:
        try:
            cursor_localhost.execute(sql,(71,71,71,71,71,71,71,71,71))
        except:
            print("Error guardar 71")
    elif status == 3:
        try:
            cursor_localhost.execute(sql,(81,81,81,81,81,81,81,81,81))
        except:
            print("Error guardar 81")
    try:
```

```
subprocess.check_call("avrdude -patmega328p -carduino -P/dev/zowi -b 115200 -D
                      -Uflash:w:/home/pi/zowi/python/zowi_offset_i2c.cpp.hex:i",
                      shell=True)

# Si se produce algún error damos un mensaje de advertencia
except subprocess.CalledProcessError:
    try:
        cursor_localhost.execute(sql,(4,4,4,4,4,4,4,4,4,4))
    except:
        print("Error guardar 4")
        print ("Programacion fallida")
        mega.write("M")

# Si no se produce ningún error nos conectamos a la ZUM
else:
    print("Conectando a zum")
    try:
        zum = serial.Serial("/dev/zowi",115200,timeout=0.2)
    except serial.SerialException:
        print("ZUM no conectada")
        mega.write("M")
    else:
        # Espera obligatoria para reiniciar la ZUM
        sleep(2)
        print("ZUM conectada")
        print("Esperando OK de ZUM")
        data = zum.read()
        while(data != start_char):
            data = zum.read()
        data = zum.read()
        while (data != end_char):
            comando = comando + data
            data = zum.read()
        if (comando[:4]=="OKNS"):
            num_serie = comando[5:]
            comando=""
            mega.write("B")
            print("B mandado")
        else:
            comando = ""
            mega.write("M")
            print("M mandada")

    elif (comando[:4] == "MSxC"): # Comando con los datos de calibracion
```

```
#No hacemos nada con estos comandos, simplemente los mandamos a la ZUM
try:
    zum.write(comando)
except NameError:
    print("conexion no establecida")
    #sleep(2)
else:
    print("dato mandado")

elif (comando[:4] == "WERC"): #Comando con los datos de los errores en la calibracion
    #Parseamos la trama con los codigos, y los almacenamos en la lista de errores.
    #El orden de los errores es el siguiente:
    #error[0]=Cadera_izquierda
    #error[1]=Cadera_derecha
    #error[2]=Pie_izquierdo
    #error[3]=Pie derecho
    errores = comando[5:1].split(char_lim)
elif (comando[:4] == "WOFC"): #Comando con datos de las posiciones para guardar en EEPROM
    print("COMANDO WOFC")
    #Tratamos la trama reibida para separar los valores de las posiciones home recibidas
    #Filtramos los 6 primeros caracteres que son el propio comando 'WOFC:*
homes=[]
cadena = comando[6:]
#Esperamos cuatro posiciones de home
for i in range (4):
    #almacenamos las posiciones de home respecto de 90 en la lista home
    #El orden de los home es el siguiente:
    #homes[0]=Cadera_derecha
    #homes[1]=Pie_derecho
    #homes[2]=Cadera_izquierda
    #homes[3]=Pie izquierdo
    #Cambiado a nuevo 90-90
    # homes.append(90 - int( cadena[:3] ))
    homes.append(int(cadena[:3])-90)
#vamos borrando los datos tratados
cadena = cadena[3:]
#Una vez almacenadas las posiciones, mandamos el codigo a la zum
try:
    zum.write(comando)
except NameError:
    print("conexion no establecida")
    #Si se produce un error borramos las posiciones guardadas
    homes=[]
```

```
#sleep(2)
else:
    print("dato mandado")
    #añadir codigo para MYSQL
elif (comando[:4] == "WSQL"):
    # Cerramos la comunicacion con la zum. La calibracion ha terminado
    zum.close()
    #Al recibir este comando escribimos en la base de datos
    try:
        #mandamos directamente el diccionario donde estan almacenados los errores.
        #Los valores de errores, se mandan tal cual son recibidos de la Mega
        cursor.execute( sql, (num_serie, errores[0], errores[1], errores[2], errores[3],
                               errores[4], homes[2], homes[0], homes[3], homes[1]) )
        #Si hay algun error en la conexion devolvemos un error
    except pymysql.DatabaseError as e:
        print("Error %d: %s" % (e.args[0], e.args[1]))
        #Si la sentencia se ha introducido correctamente
    except NameError:
        print("No se pudo conectar a la base de datos remota en el inicio")
    except:
        print("Se ha producido un error al insertar la
              sentencia en la base de datos remota")
    else:
        print("Sentencia introducida correctamente en remoto")
    #Al recibir este comando escribimos en la base de datos local para tener una copia
    try:
        #mandamos directamente el diccionario donde estan almacenados los errores.
        cursor_localhost.execute( sql, (num_serie, errores[0], errores[1], errores[2],
                                         errores[3], errores[4], homes[2],
                                         homes[0], homes[3], homes[1]) )
        #Si hay algun error en la conexion devolvemos un error
    except pymysql.DatabaseError as e:
        print("Error %d: %s" % (e.args[0], e.args[1]))
        #Si la sentencia se ha introducido correctamente
    except NameError:
        print("No se pudo conectar a la base de datos local en el inicio")
    except:
        print("Se ha producido un error al insertar la
              sentencia en la base de datos local")
    else:
        print("Sentencia introducida correctamente en local")
        #Reseteamos las variables
```

```
errores=[]
homes=[]
num_serie = ""
#Cerramos el curso r
#cursor_localhost.close()
#Cerramos la conexión
#db_localhost.close()

elif (comando[:4] == "ROFF"):
    #COMANDO CON EL APAGADO SEGURO de la RASpBerry
    #subprocess.check_call("sudo halt",shell=True)
    print("Recibido comando de apagado controlado")

#####
# Escribimos en las bases de datos un mensaje de fin
try:
    # mandamos directamente el diccionario donde están almacenados los errores.
    # Los valores de errores, se mandan tal cual son recibidos de la Mega
    cursor.execute( sql, (0, 0,0,0, 0,0,0, 0,0,0))
    # Si hay algun error en la conexion devolvemos un error
except pymysql.DatabaseError as e:
    print("Error %d: %s" % (e.args[0], e.args[1]))
    # Si no hemos podido conectar a la BBDD al inicio:
except NameError:
    print("No se pudo conectar al base de datos remota en el inicio")
except:
    print("Se ha producido un error al insertar
          la sentencia en la base de datos remota")
    # Si la sentencia se ha introducido correctamente
else:
    print("Sentencia introducida correctamente en remoto")
#Cerramos el curso
#cursor.close()

# Al recibir este comando escribimos en la base de datos local para tener una copia
try:
    #mandamos directamente el diccionario donde estan almacenados los errores.
    #Los valores de errores, se mandan tal cual son recibidos de la Mega
    cursor_localhost.execute( sql, (0, 0,0,0, 0,0,0, 0,0,0 ))
    #Si hay algun error en la conexion devolvemos un error
except pymysql.DatabaseError as e:
    print("Error %d: %s" % (e.args[0], e.args[1]))
    # Si no hemos podido conectar a la BBDD al inicio:
except NameError:
```

```
print("No se pudo conectar al base de datos local en el inicio")
except:
    print("Se ha producido un error al insertar la
          sentencia en la base de datos local")
# Si la sentencia se ha introducido correctamente
else:
    print("Sentencia introducida correctamente en local")
#Cerramos el curso r
#cursor_localhost.close()
#####
try:
    db_localhost.close()
except:
    print("[X] Error al cerrar base de datos local")
else:
    print("[OK] Desconectado de la base de datos local")
try:
    db.close()
except:
    print("[X] Error al cerrar base de datos remota")
else:
    print("[OK] Desconectado de la base de datos remota")
try:
    mega.close()
except:
    print("[X] Error al desconectar de la placa MEGA")
else:
    print("[OK] Desconexión correcta de la placa MEGA")
print("[OK] Mandado comando de apagado")
subprocess.check_call("sudo halt",shell=True)
exit()

elif (comando[:4] == "FZUM"):
    comando=""
    zum.close()

status = funcionPuertos()
if status == 1:
    try:
        cursor_localhost.execute(sql,(62,62,62,62,62,62,62,62))
    except:
        print("Error guardar 62")
elif status == 2:
```

```
try:
    cursor_localhost.execute(sql,(72,72,72,72,72,72,72,72,72,72))
except:
    print("Error guardar 72")
elif status == 3:
    try:
        cursor_localhost.execute(sql,(82,82,82,82,82,82,82,82,82,82))
    except:
        print("Error guardar 82")

try:
    subprocess.check_call("avrdude -patmega328p -carduino -P/dev/zowi -b 115200 -D
                          -Uflash:w:/home/pi/zowi/python/ZOWI_BASE_v0.hex:i",
                          shell=True)
except subprocess.CalledProcessError:
    try:
        cursor_localhost.execute(sql,(5,5,5,5,5,5,5,5,5,5))
    except:
        print("Error guardar 5")
        print("Programacion demo no correcta")
        mega.flushInput()
        mega.flushOutput()
        mega.write("M")
    else:
        print("Programacion demo correcta")
        mega.write("B")
else:
    try:
        zum.write(comando)
    except NameError:
        print("conexion no establecida")
        #sleep(2)
    else:
        print("dato mandado")
        comando = ""
# Se pulsa ctrl+c
except KeyboardInterrupt:
    print("Interrupción detectada de usuario")
###
#####
# Escribimos en las bases de datos un mensaje de fin
```

```
try:
    # mandamos directamente el diccionario donde están almacenados los errores.
    # Los valores de errores, se mandan tal cual son recibidos de la Mega
    cursor.execute( sql, (0, 0,0,0, 0,0,0, 0,0,0) )
    # Si hay algun error en la conexion devolvemos un error
except pymysql.DatabaseError as e:
    print("Error %d: %s" % (e.args[0], e.args[1]))
    # Si no hemos podido conectar a la BBDD al inicio:
except NameError:
    print("No se pudo conectar al base de datos remota en el inicio")
except:
    print("Se ha producido un error al insertar la sentencia en la base de datos remota")
# Si la sentencia se ha introducido correctamente
else:
    print("Sentencia introducida correctamente en remoto")

#Cerramos el curso
    #cursor.close()

# Al recibir este comando escribimos en la base de datos local para tener una copia
try:
    #mandamos directamente el diccionario donde estan almacenados los errores.
    #Los valores de errores, se mandan tal cual son recibidos de la Mega
    cursor_localhost.execute( sql, (0, 0,0,0, 0,0,0, 0,0,0 ) )
    #Si hay algun error en la conexion devolvemos un error
except pymysql.DatabaseError as e:
    print("Error %d: %s" % (e.args[0], e.args[1]))
    # Si no hemos podido conectar a la BBDD al inicio:
except NameError:
    print("No se pudo conectar al base de datos local en el inicio")
except:
    print("Se ha producido un error al insertar la sentencia en la base de datos local")
    # Si la sentencia se ha introducido correctamente
else:
    print("Sentencia introducida correctamente en local")

#Cerramos el curso r
    #cursor_localhost.close()
#####
###
try:
    db_localhost.close()
except:
    print("[X] Error al cerrar base de datos local")
else:
```

```
print("[OK] Desconectado de la base de datos local")  
try:  
    db.close()  
except:  
    print("[X] Error al cerrar base de datos remota")  
else:  
    print("[OK] Desconectado de la base de datos remota")  
try:  
    mega.close()  
except:  
    print("[X] Error al desconectar de la placa MEGA")  
else:  
    print("[OK] Desconexión correcta de la placa MEGA")  
exit()  
  
except serial.serialutil.SerialException:  
    print("Mega desconectada de forma incorrecta")  
    try:  
        cursor_localhost.execute(sql,(2,2,2,2,2,2,2,2,2))  
    except:  
        print("Error escritura 2 en base de datos")  
    try:  
        db_localhost.close()  
    except:  
        print("[X] Error al cerrar base de datos local")  
    else:  
        print("[OK] Desconectado de la base de datos local")  
    try:  
        db.close()  
    except:  
        print("[X] Error al cerrar base de datos remota")  
    else:  
        print("[OK] Desconectado de la base de datos remota")  
    try:  
        mega.close()  
    except:  
        print("[X] Error al desconectar de la placa MEGA")  
    else:  
        print("[OK] Desconexión correcta de la placa MEGA")  
exit()  
  
except:  
    print("Otros errores")  
try:
```

```
cursor_localhost.execute(sql,(3,3,3,3,3,3,3,3,3))
except:
    print("Error escritura 3 en base de datos")
exit()
```


Apéndice P

Código Mega

Programa principal de la Mega, de Arduino: Se muestra el programa principal para uno de los armarios ligeramente retocado (indentos y saltos de línea principalmente). Las diferencias entre distintos armarios serán solamente los valores de calibración del acelerómetro.

```

/*
Programa creado sobre MinIMU-9-Arduino-AHRS.
MRS.

MinIMU-9-Arduino-AHRS
Pololu MinIMU-9 + Arduino AHRS (Attitude and Heading Reference System)

Copyright (c) 2011 Pololu Corporation.
http://www.pololu.com/

MinIMU-9-Arduino-AHRS is based on sf9domahrs by Doug Weibel and Jose Julio:
http://code.google.com/p/sf9domahrs/

sf9domahrs is based on ArduIMU v1.5 by Jordi Munoz and William Premerlani, Jose
Julio and Doug Weibel:
http://code.google.com/p/ardu-imu/

*/
// Uncomment the below line to use this axis definition:
// X axis pointing forward
// Y axis pointing to the right
// and Z axis pointing down.
// Positive pitch : nose up
// Positive roll : right wing down
// Positive yaw : clockwise
int SENSOR_SIGN[9] = { 1, 1, 1, -1, -1, -1, 1, 1, 1 }; //Correct directions x,y,z -
gyro, accelerometer, magnetometer
int SENSOR_SIGNd[9] = { 1, 1, 1, -1, -1, -1, 1, 1, 1 }; //Correct directions x,y,z -
gyro, accelerometer, magnetometer
// Uncomment the below line to use this axis definition:
// X axis pointing forward
// Y axis pointing to the left
// and Z axis pointing up.
// Positive pitch : nose down
// Positive roll : right wing down
// Positive yaw : counterclockwise
//int SENSOR_SIGN[9] = {1,-1,-1,-1,1,1,1,-1,-1}; //Correct directions x,y,z - gyro,
//accelerometer, magnetometer
//int SENSOR_SIGN[9] = {1,-1,-1,-1,1,1,1,-1,-1}; //Correct directions x,y,z - gyro,
//accelerometer, magnetometer
//int SENSOR_SIGNd[9] = {1,-1,-1,-1,1,1,1,-1,-1}; //Correct directions x,y,z - gyro,
//accelerometer, magnetometer

// tested with Arduino Uno with ATmega328 and Arduino Duemilanove with ATMega168

#include <Wire.h>

#include <LiquidCrystal.h> // F Malpartida's NewLiquidCrystal library

#define caraZowi 0 // =1: Operador ve CARA de Zowi. =0: Operador percibe CULO.

// LSM303 accelerometer: 8 g sensitivity
// 3.9 mg/digit; 1 g = 256

```

```
#define GRAVITY 256 //this equivalent to 1G in the raw data coming from the
accelerometer

#define ToRad(x) ((x)*0.01745329252) // *pi/180
#define ToDeg(x) ((x)*57.2957795131) // *180/pi

// L3G4200D gyro: 2000 dps full scale
// 70 mdps/digit; 1 dps = 0.07
#define Gyro_Gain_X 0.07 //X axis Gyro gain
#define Gyro_Gain_Y 0.07 //Y axis Gyro gain
#define Gyro_Gain_Z 0.07 //Z axis Gyro gain

#define Gyro_Scaled_X(x) ((x)*ToRad(Gyro_Gain_X))
//Return the scaled ADC raw data of the gyro in radians for second
#define Gyro_Scaled_Y(x) ((x)*ToRad(Gyro_Gain_Y))
//Return the scaled ADC raw data of the gyro in radians for second
#define Gyro_Scaled_Z(x) ((x)*ToRad(Gyro_Gain_Z))
//Return the scaled ADC raw data of the gyro in radians for second

#define Gyro_Scaled_Xd(x) ((x)*ToRad(Gyro_Gain_X))
//Return the scaled ADC raw data of the gyro in radians for second
#define Gyro_Scaled_Yd(x) ((x)*ToRad(Gyro_Gain_Y))
//Return the scaled ADC raw data of the gyro in radians for second
#define Gyro_Scaled_Zd(x) ((x)*ToRad(Gyro_Gain_Z))
//Return the scaled ADC raw data of the gyro in radians for second

// LSM303 magnetometer calibration constants; use the Calibrate example from
// the Pololu LSM303 library to find the right values for your board
#define M_X_MIN -2666
#define M_Y_MIN -3446
#define M_Z_MIN -4782
#define M_X_MAX 4625
#define M_Y_MAX 3695
#define M_Z_MAX 4383

#define M_X_MINd -3212
#define M_Y_MINd -3273
#define M_Z_MINd -2653
#define M_X_MAXd 3505
#define M_Y_MAXd 3260
#define M_Z_MAXd 4601

#define Kp_ROLLPITCH 0.02
#define Ki_ROLLPITCH 0.00002
#define Kp_YAW 1.2
#define Ki_YAW 0.00002

// Mesa botonera: *** ELIMINAR LED Y BUZZER
#define pin_pulsador1 9
#define pin_interruptor 11
//#define pin_led 7 //OLD: 12. 7 para debug en cambio a versión raspberry.
//#define pin_buzz 8

#define umbralError 0.8
#define ErrorArtMAX 1

/*For debugging purposes*/
```

```
//OUTPUTMODE=1 will print the corrected data,
//OUTPUTMODE=0 will print uncorrected data of the gyros (with drift)
#define OUTPUTMODE 1

//#define PRINT_DCM 0           //Will print the whole direction cosine matrix
#define PRINT_ANALOGS 0 //Will print the analog raw data
#define PRINT_EULER 1 //Will print the Euler angles Roll, Pitch and Yaw

#define STATUS_LED 13

//Pines pinales de carrera zapateros
#define FCV1 2
#define FCH1 3
#define FCV2 4
#define FCH2 5

//pines leds
#define Led_rojo 14
#define Led_verde 16
#define Led_azul 15
#define Led_reserva 17
#define Pin_buzz 8

#define round(a) (int)(a + 0.5) //Para redondear

float G_Dt = 0.02; // Integration time (DCM algorithm)
                    // We will run the integration loop at 50Hz if possible

long timer = 0; //general purpose timer
long timer_old;
long timer_restart = 0;
long timer24 = 0; //Second timer used to print values
int AN[6]; //array that stores the gyro and accelerometer data
int AN_OFFSET[6] = { 0, 0, 0, 0, 0, 0 };
//Array that stores the Offset of the sensors

int AND[6]; //array that stores the gyro and accelerometer data
int AN_OFFSETd[6] = { 0, 0, 0, 0, 0, 0 };
//Array that stores the Offset of the sensors

int gyro_x;
int gyro_y;
int gyro_z;
int accel_x;
int accel_y;
int accel_z;
int magnetom_x;
int magnetom_y;
int magnetom_z;
float c_magnetom_x;
float c_magnetom_y;
float c_magnetom_z;
float MAG_Heading;

int gyro_xd;
int gyro_yd;
int gyro_zd;
```

```

int accel_xd;
int accel_yd;
int accel_zd;
int magnetom_xd;
int magnetom_yd;
int magnetom_zd;
float c_magnetom_xd;
float c_magnetom_yd;
float c_magnetom_zd;
float MAG_Headingd;

float Accel_Vector[3] = { 0, 0, 0 }; //Store the acceleration in a vector
float Gyro_Vector[3] = { 0, 0, 0 }; //Store the gyros turn rate in a vector
float Omega_Vector[3] = { 0, 0, 0 }; //Corrected Gyro_Vector data
float Omega_P[3] = { 0, 0, 0 }; //Omega Proportional correction
float Omega_I[3] = { 0, 0, 0 }; //Omega Integrator
float Omega[3] = { 0, 0, 0 };

float Accel_Vectord[3] = { 0, 0, 0 }; //Store the acceleration in a vector
float Gyro_Vectord[3] = { 0, 0, 0 }; //Store the gyros turn rate in a vector
float Omega_Vectord[3] = { 0, 0, 0 }; //Corrected Gyro_Vector data
float Omega_Pd[3] = { 0, 0, 0 }; //Omega Proportional correction
float Omega_Id[3] = { 0, 0, 0 }; //Omega Integrator
float Omegad[3] = { 0, 0, 0 };

// Euler angles pierna izquierda
float roll;
float pitch;
float yaw;

// Euler angles pierna derecha
float rolld;
float pitchd;
float yawd;

float errorRollPitch[3] = { 0, 0, 0 };
float errorYaw[3] = { 0, 0, 0 };

float errorRollPitchd[3] = { 0, 0, 0 };
float errorYawd[3] = { 0, 0, 0 };

unsigned int counter = 0;
byte gyro_sat = 0;
byte gyro_satd = 0;

float DCM_Matrix[3][3] = {
    { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 }
};

float DCM_Matrixd[3][3] = {
    { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 }
};

float Update_Matrix[3][3] = { { 0, 1, 2 }, { 3, 4, 5 }, { 6, 7, 8 } }; //Gyros here
float Update_Matrixd[3][3] = { { 0, 1, 2 }, { 3, 4, 5 }, { 6, 7, 8 } }; //Gyros here

float Temporary_Matrix[3][3] = {

```

```
{ 0, 0, 0 }, { 0, 0, 0 }, { 0, 0, 0 }
};

float Temporary_Matrixd[3][3] = {
    { 0, 0, 0 }, { 0, 0, 0 }, { 0, 0, 0 }
};

// !! botonera
int estado = 0;
long timerLed = 0;
int freqLed = 1;
int pulsador_old = 0;
int pulsador_apagar = 1;
int num_vueltas = 0;

double comprPitch_izq = 0;
double comprRoll_izq = 0;
//double comprYaw_izq=0;
double comprPitch_der = 0;
double comprRoll_der = 0;
//double comprYaw_der=0;
double contadorEstabilizacion = 0;
int flagBoton = 0;

double nuevo90_der = 0;
double nuevo90_izq = 0;
double nuevo0_der = 0;
double nuevo0_izq = 0;

double err_pie_izq = 0;
double err_cadera_izq = 0;
double err_pie_der = 0;
double err_cadera_der = 0;

double pos_old_pie_izq = 0;
double pos_old_cadera_izq = 0;
double pos_old_pie_der = 0;
double pos_old_cadera_der = 0;

int cadera_izq_OK = 0;
int cadera_der_OK = 0;
int pie_izq_OK = 0;
int pie_der_OK = 0;

int primeraVez = 0;
int numeroData = 0;
int flag_espera_zum = 0;
double timer_Zum = 0;
double timer_WD;

char lectura_RB = 0;

int FCH_1 = 0;
int FCV_1 = 0;
int FCH_2 = 0;
int FCV_2 = 0;
```



```
pinMode(STATUS_LED, OUTPUT); // Status LED

I2C_Init();

Serial1.println("Banco calibracion Zowi");

lcd.clear();
lcd.setCursor(0, 0);
lcd.print("      System      ");
lcd.setCursor(0, 1);
lcd.print(" start-up...   ");
digitalWrite(Led_rojo, LOW);
digitalWrite(Led_azul, LOW);
digitalWrite(Led_verde, LOW);
digitalWrite(Led_reserva, LOW);
digitalWrite(Pin_buzz, LOW);
digitalWrite(STATUS_LED, LOW);
delay(1500);

Accel_Init();
Serial1.println("acel init done");
Compass_Init();
Serial1.println("compass initdone");
Gyro_Init();
Serial1.println("gyro init done");

delay(20);

for (int i = 0; i < 32; i++) // We take some readings...
{
    Read_Gyro();
    Read_Accel();
    for (int y = 0; y < 6; y++) // Cumulate values
    {
        AN_OFFSET[y] += AN[y];
        AN_OFFSETd[y] += ANd[y];
    }
    delay(20);
}

for (int y = 0; y < 6; y++) {
    AN_OFFSET[y] = AN_OFFSET[y] / 32;
    AN_OFFSETd[y] = AN_OFFSETd[y] / 32;
}

AN_OFFSET[5] -= GRAVITY * SENSOR_SIGN[5];
AN_OFFSETd[5] -= GRAVITY * SENSOR_SIGNd[5];

delay(2000);
digitalWrite(STATUS_LED, HIGH);

timer = millis();
delay(20);
counter = 0;

estado = 97;
primeraVez = 1;
```

```
}

void loop() //Main Loop (Maquina estados)
{
    if (digitalRead(pin_interruptor) == LOW && estado != 97 && estado != 98
        && estado != 96) {
        estado = 0;
        delay(10);
    }

    switch (estado) {
        case 96: //Apagado de RPI
            WriteComm(9);
            lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print("      System      ");
            lcd.setCursor(0, 1);
            lcd.print("Shutting down... ");
            Serial1.println("Apagando");
            delay(15000);
            lcd.clear();
            lcd.noDisplay();
            digitalWrite(Led_rojo, LOW);
            parpadeoLed(0); //desactivamos parpadeo led azul
            digitalWrite(Led_azul, LOW);
            digitalWrite(Led_verde, LOW);
            digitalWrite(Led_reserva, LOW);
            digitalWrite(Pin_buzz, LOW);
            digitalWrite(STATUS_LED, LOW);
            break;

        case 97: //Arranque de RPI //esperamos que la RPI envíe un 1 a la Mega
            parpadeoLed(1);

            if (Serial.available() >= 1) {
                lectura_RB = Serial.read();
                Serial1.println("lectura serial RB:");
                Serial1.println(lectura_RB);
            }

            //Dependiendo del estado del switch vamos a un estado u otro:
            if ((lectura_RB == '1') && (digitalRead(pin_interruptor) == HIGH)) {
                estado = 14;
                sonidostart();
                parpadeoLed(0);
            }
            else if ((lectura_RB == '1') && (digitalRead(pin_interruptor)) == LOW) {
                estado = 0;
                sonidostart();
                parpadeoLed(0);
            }

            break;

        //Llegamos a estado 0 al subir el switch desde cualquier estado
        case 0:
            digitalWrite(Led_rojo, LOW);
```

```
parpadeoLed(0); //desactivamos parpadeo led azul
digitalWrite(Led_azul, LOW);
digitalWrite(Led_verde, LOW);
digitalWrite(Led_reserva, LOW);
digitalWrite(Pin_buzz, LOW);
digitalWrite STATUS_LED, LOW;

if (digitalRead(pin_pulsador1) == 1 && pulsador_apagar == 1) {
//Apagado del sistema:
    delay(5);
    timer_restart = millis();
    pulsador_apagar = 0;
}
else if (digitalRead(pin_pulsador1) == 0) {
    timer_restart = millis();
    pulsador_apagar = 1;
}

if ((millis() - timer_restart) >= 2000 && digitalRead(pin_pulsador1) == 1) {
    //si mantenemos el pulsador 2 segundos apagamos sistema
    estado = 96; //estado para apagar
    break;
}

lcd.setCursor(0, 0);
lcd.print("      Set to      ");
lcd.setCursor(0, 1);
lcd.print(" CALIBRATE pos. ");
//Serial1.println("Estado 0");
/* interruptor en posición 0. Desde cualquier estado podremos volver
   al estado 0 poniendolo a 0. Los servos energizados reciben el valor 90,
   el operario deberá montar y atornillar los pies del ZOWI en la posición
   más correcta posible.
Se deben colocar los zapatos en la plataforma antes de bajar el interruptor*/
//al cambiar el interruptor pasamos al funcionamiento del sistema
if (digitalRead(pin_interruptor) == HIGH) {
    estado = 14;
    delay(10);
    primeraVez = 1;
};

break;

case 14: //calibracion horizontal tras pulsar el boton
Serial1.println("Estado 14: Espera colocar zapatos y pulsar boton");
//deben estar los zapatos insertados para calibrar las imus
FCH_1 = digitalRead(FCH1);
FCH_2 = digitalRead(FCH2);

if (FCH_1 == HIGH || FCH_2 == HIGH) {

    lcd.setCursor(0, 0);
    lcd.print(" Place shoes in ");
    lcd.setCursor(0, 1);
    lcd.print(" Horizontal box ");
}
}
```

```
if (FCH_1 == LOW && FCH_2 == LOW) {  
  
    lcd.setCursor(0, 0);  
    lcd.print(" Press to set ");  
    lcd.setCursor(0, 1);  
    lcd.print(" HORIZONTAL ");  
  
    if (digitalRead(pin_pulsador1) == 1) {  
        estado = 10;  
        delay(10);  
        primeraVez = 1;  
    };  
}  
  
break;  
  
case 10: //inicio de calibración IMUS en horizontal  
Serial1.println("Estado 10: Iniciando Calib Horizontal");  
//lcd.clear();  
lcd.setCursor(0, 0);  
lcd.print(" Calibration ");  
lcd.setCursor(0, 1);  
lcd.print(" in progress... ");  
estado = 1;  
break;  
  
case 1:  
Serial1.println("Estado 1");  
  
init_imus();  
  
//Preparamos para lecturas comprobación imu  
timer = millis();  
comprPitch_izq = 0;  
comprRoll_izq = 0;  
comprPitch_der = 0;  
comprRoll_der = 0;  
contadorEstabilizacion = 0;  
estado = 11;  
break;  
  
case 11: //Comprobacion de offset  
////Serial1.println("Estado 11");  
FCH_1 = digitalRead(FCH1);  
FCH_2 = digitalRead(FCH2);  
//se realiza la calibración con lecturas a 50Hz ya que es  
//la frecuencia a la que trabajan las IMUS  
if ((millis() - timer) >= 20) // Main loop runs at 50Hz  
{  
    if (FCH_1 == LOW && FCH_2 == LOW) {  
        //aseguramos que los zapatos están en zapatero horizontal  
        calculosIMU();  
  
        contadorEstabilizacion++;  
  
        if (contadorEstabilizacion > 100) {  
            Serial1.println("Calibración IMUS finalizada");  
            Serial1.println("Estado 12: Calibración IMUS finalizada");  
            estado = 12;  
        }  
    }  
}
```

```

        if (caraZowi) {
            comprPitch_der += pitch;
            comprPitch_izq += pitchd;
            comprRoll_der += roll;
            comprRoll_izq += rolld;
        }

        else {
            comprPitch_der += pitchd;
            comprPitch_izq += pitch;
            comprRoll_der += rolld;
            comprRoll_izq += roll;
        }
    }

    if (contadorEstabilizacion > 150) {
        comprPitch_der = abs(ToDeg(comprPitch_der) / 50);
        comprPitch_izq = abs(ToDeg(comprPitch_izq) / 50);
        comprRoll_izq = abs(ToDeg(comprRoll_izq) / 50);
        comprRoll_der = abs(ToDeg(comprRoll_der) / 50);
        if ((comprPitch_der > umbralError) || (comprPitch_izq >
            umbralError) || (comprRoll_der > umbralError) || 
            (comprRoll_izq > umbralError)) //Media
        { //Error en offset imus
            estado = 12;
            Serial1.println("ERROR. > a ESTADO 12 ");
            //delay(1000);
        }
        else { //Offset en 0° IMUs OK
            estado = 20;
            nuevo0_der = comprPitch_der;
            nuevo0_izq = comprPitch_izq;

            Serial1.println("Nuevo 0:");
            Serial1.println(nuevo0_der);
            Serial1.println(nuevo0_izq);
        }
    }
}

else { //Zapato no está en zapatero horizontal
    Serial1.println("Zapatos mal puestos en posicion Horizontal");

    //lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(" Place shoes in ");
    lcd.setCursor(0, 1);
    lcd.print(" Horizontal box ");

    delay(1000);

    estado = 14;
}
}

break;

case 12: //Estado error en offset 0° de imus No se han calibrado correctamente

```

```
sonidoNOK();
Serial1.println("Repite Posicion Horizontal");

lcd.clear();
lcd.setCursor(0, 0);
lcd.print(" Process failed ");
lcd.setCursor(0, 1);
lcd.print("      Retry      ");

delay(1000);

estado = 14;
break;

case 20: //Estado calibración 0°ok, pasamos a calibración vertical
sonidoOK();
Serial1.println("Calibracion IMU Horizontal ok");

//lcd.clear();
lcd.setCursor(0, 0);
lcd.print(" Place shoes in ");
lcd.setCursor(0, 1);
lcd.print(" Vertical box  ");

contadorEstabilizacion = 0;
estado = 21; //zapatos a 90 "calibracion"
timer = millis();
timerLed = millis();
delay(20);
counter = 0;
flagBoton = 0;
break;

case 21: //Estado para offset de 90°(calibración zapatos en vertical)

FCV_1 = digitalRead(FCV1);
FCV_2 = digitalRead(FCV2);

if (FCV_1 == LOW && FCV_2 == LOW && flagBoton == 0) {
    //los zapatos deben estar insertados

    //lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("  Press to set  ");
    lcd.setCursor(0, 1);
    lcd.print("      VERTICAL      ");
};

if ((FCV_1 == HIGH || FCV_2 == HIGH) && flagBoton == 0) {

    //lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(" Place shoes in ");
    lcd.setCursor(0, 1);
    lcd.print("  Vertical box  ");
};
```

```

if ((millis() - timer) >= 20) // Main loop runs at 50Hz
    //calibración zapatos en vertical
{
    calculosIMU();

    if (digitalRead(pin_pulsador1) == 1) //esperamos pulsación para calibrar
    {
        flagBoton = 1;
    }

    if (flagBoton == 1) {
        if (FCV_1 == LOW && FCV_2 == LOW) { //Zapatos en zapatero vertical
            contadorEstabilizacion++;

            // lcd.clear();
            //lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print(" Calibration ");
            lcd.setCursor(0, 1);
            lcd.print(" in progress... ");

            if (contadorEstabilizacion > 50) //esperamos a que se estabilice
            {
                if (caraZowi) {
                    comprRoll_izq += rolld;
                    comprRoll_der += roll;
                }
                else {
                    comprRoll_izq += roll;
                    comprRoll_der += rolld;
                }
            }
        }

        if (contadorEstabilizacion > 100)
        //se han obtenido 50 medidas tras estabilizar
        {

            comprRoll_izq = ToDeg(comprRoll_izq / 50);
            comprRoll_der = ToDeg(comprRoll_der / 50);

            nuevo90_izq = comprRoll_izq;
            nuevo90_der = comprRoll_der;

            Serial1.println("Calibracion IMU Vertical Ok");
            //calibración vertical hecha
            //pasamos a colocar zapatos
            //lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print("Connect+ON ZOWI ");
            lcd.setCursor(0, 1);
            lcd.print("Put shoes+press ");

            Serial1.println("Nuevo 90:");
            Serial1.println(nuevo90_der);
            Serial1.println(nuevo90_izq);

            flagBoton = 0;
        }
    }
}

```

```
        estado = 2;
        Serial1.println("Coloca zap, enchufa y pulsa
                        boton para ajustar ");
        sonidoOK();
        timer = millis();
        timerLed = millis();
        delay(20);
        counter = 0;
        //Serial1.println("Aqui1");
    }
}
else { //Zapatos no están en zapatero vertical
    Serial1.println("Zapatos mal colocados en posicion vertical");

    lcd.setCursor(0, 0);
    lcd.print(" Place shoes in ");
    lcd.setCursor(0, 1);
    lcd.print(" Vertical box ");

    delay(1000);

    lcd.clear();

    estado = 21;
}
}
}

break;

case 2:
Serial1.println("case2");
// el led parpadea, se espera que se coloquen los zapatos
// en los pies de ZOWI.
//Salimos del estado actuando sobre el pulsador.
if ((millis() - timer) >= 20) // Main loop runs at 50Hz
{
    calculosIMU();
}
parpadeoLed(1); //sucede cada 1*500ms

if (digitalRead(pin_pulsador1) == 1) //AQUIIIIIII!!!!!
{ //Arrancamos calibracion con el boton
    parpadeoLed(0);
    estado = 22;
    cadera_izq_OK = 0;
    cadera_der_OK = 0;
    pie_izq_OK = 0;
    pie_der_OK = 0;
    num_vueltas = 0;

    digitalWrite(Led_rojo, LOW);
    digitalWrite(Led_verde, LOW);

    Serial1.println("Iniciando comunicacion con Zum Zowi...");

    //lcd.clear();
}
```

```
lcd.setCursor(0, 0);
lcd.print("    Checking      ");
lcd.setCursor(0, 1);
lcd.print(" connection...  ");

};

break;

case 22:
Serial1.println("case22");
if ((millis() - timer) >= 20) // Main loop runs at 50Hz
{
    calculosIMU();
}

if (flag_espera_zum == 0) {
    timer_Zum = millis();
    flag_espera_zum = 1;

    while (Serial.available() >= 1) {
        Serial1.println(Serial.read());
    }

    WriteComm(0); //Inicio comunicación Raspberry - ZUM mandamos IZUM
}

if (Serial.available() >= 1) {

    lectura_RB = Serial.read();
    //Serial1.println(lectura_RB);
    if (lectura_RB == 'B') {

        //lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Communication OK");
        lcd.setCursor(0, 1);
        lcd.print("Calibrating...  ");

        Serial1.println("Comunicacion establecida.");
        Serial1.println("Calibrando...");
        estado = 3;
        flag_espera_zum = 0;

        while (Serial.available() >= 1) {
            Serial1.println(Serial.read());
        } //vaciamos serial

        break;
    }

    else if (lectura_RB == 'M') {
        //si la RPI envía una M indica que no se
        //ha cargado el programa de calibración en Zowi

        //lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Connection fault");
        lcd.setCursor(0, 1);
    }
}
```

```
lcd.print("    Check ZOWI    ");

estado = 2;
flag_espera_zum = 0;

Serial1.println("MAL. No cargada ZUM");
Serial1.println("Leyendo serial (3 veces):");
//para vaciar últimos datos del serial
Serial1.println(Serial.read());
Serial1.println(Serial.read());
Serial1.println(Serial.read());
}
}

else if (((millis() - timer_Zum) >= 40000)) {
//si no responde en x secs: volver a intentar
Serial1.println("TIMEOUT");
Serial1.println("Pulsa para volver a intentar...");

//lcd.clear();
lcd.setCursor(0, 0);
lcd.print(" Timeout. Check ");
lcd.setCursor(0, 1);
lcd.print(" Zowi and push ");
estado = 2;
flag_espera_zum = 0;
}

break;

case 3: //Estado calibración articulaciones:
// Los zapatos ya están colocados en los pies y se comienza
// a ajustar offset de los servos.
parpadeoLed(4);
if ((millis() - timer) >= 20) // Main loop runs at 50Hz
{
    calculosIMU();
    //Calculo errores de posicion respecto a los 0° y los 90°
    //ajustados para el banco

    if (caraZowi) { //dependiendo si está puesto el zowi de cara o de culo
        err_pie_izq = nuevo90_izq - ToDeg(rolld);
        err_cadera_izq = nuevo0_izq - ToDeg(pitchd);
        err_pie_der = nuevo90_der - ToDeg(roll);
        err_cadera_der = nuevo0_der - ToDeg(pitch);
    }

    else {
        err_pie_izq = nuevo90_izq - ToDeg(roll);
        err_cadera_izq = nuevo0_izq - ToDeg(pitch);
        err_pie_der = nuevo90_der - ToDeg(rolld);
        err_cadera_der = nuevo0_der - ToDeg(pitchd);
    }

    //Evaluamos error
    if (abs(err_cadera_izq) > ErrorArtMAX) {
```

```

        cadera_izq_OK = 0;
    }
    else {
        cadera_izq_OK = 1;
    }
    if (abs(err_pie_izq) > ErrorArtMAX) {
        pie_izq_OK = 0;
    }
    else {
        pie_izq_OK = 1;
    }
    if (abs(err_cadera_der) > ErrorArtMAX) {
        cadera_der_OK = 0;
    }
    else {
        cadera_der_OK = 1;
    }
    if (abs(err_pie_der) > ErrorArtMAX) {
        pie_der_OK = 0;
    }
    else {
        pie_der_OK = 1;
    }

    num_vueltas++; //para case de calibra servos

    if (!(cadera_der_OK == 1 && cadera_izq_OK == 1 &&
          pie_der_OK == 1 && pie_izq_OK == 1)) {
        //Si articulaciones con error de posicion->calibraremos
        if (num_vueltas <= 330)
            //Para 5 iteraciones de calibración en servos se llega a 330
        {
            calibra_servos();
        }
        else {
            estado = 40;
        };
    }
    else {
        estado = 40;
    };
}

break;

case 40: //Estado chequeo de la calibracion
num_vueltas = 0;

Serial.read();
Serial.read();
Serial.read();
Serial.read();

if (cadera_der_OK==1 && cadera_izq_OK==1 && pie_der_OK==1 && pie_izq_OK==1) {
    //Calibracion articulaciones OK
    resultado = '1';
    result = 1;
}

```

```
//          parpadeoLed(0);
//          digitalWrite(Led_rojo,LOW);
//          digitalWrite(Led_verde, HIGH);
//          sonidoOK();

Serial1.println("Calibración OK");

//lcd.clear();
lcd.setCursor(0, 0);
lcd.print(" Calibration OK ");
lcd.setCursor(0, 1);
lcd.print(" Writing EEPROM ");

cadera_der_OK = 0;
cadera_izq_OK = 0;
pie_der_OK = 0;
pie_izq_OK = 0;

dataError[0] = err_cadera_izq;
dataError[1] = err_cadera_der;
dataError[2] = err_pie_izq;
dataError[3] = err_pie_der;
WriteComm(7); //Comando WERC: errores de calibracion para RB)

data[0] = pos_old_cadera_der;
data[1] = pos_old_pie_der;
data[2] = pos_old_cadera_izq;
data[3] = pos_old_pie_izq;
WriteComm(22);
    //Comando WOFF Pasamos a Zowi las posiciones HOME
delay(300);

data[0] = pos_old_cadera_der;
data[1] = pos_old_pie_der;
data[2] = pos_old_cadera_izq;
data[3] = pos_old_pie_izq;
WriteComm(2);
delay(300);

WriteComm(10); //Comando WSQL Salva base datos

Serial1.print("Cadera Izq :");
Serial1.println(pos_old_cadera_izq);
Serial1.print("Pie Izq :");
Serial1.println(pos_old_pie_izq);
Serial1.print("Cadera Der :");
Serial1.println(pos_old_cadera_der);
Serial1.print("Pie Der :");
Serial1.println(pos_old_pie_der);

Serial1.print("Error cadera izq :");
Serial1.println(err_cadera_izq);
Serial1.print("Error pie izq :");
Serial1.println(err_pie_izq);
Serial1.print("Error cadera der :");
Serial1.println(err_cadera_der);
Serial1.print("Error pie der :");
```

```
Serial1.println(err_pie_der);

delay(500);

lcd.setCursor(0, 1);
lcd.print("Loading Test Prg");

WriteComm(8); //Comando FZUM Cargar programa Test

lectura_RB = 0;
timer_WD = millis();
do {
    if (Serial.available() >= 1) {
        lectura_RB = Serial.read();
        Serial.println(lectura_RB);
    }
    if ((millis() - timer_WD) >= 20000) {
        sonidoNOK();
        parpadeoLed(0);
        digitalWrite(Led_rojo, HIGH);
        digitalWrite(Led_verde, LOW);
        break;
    }
} while ((lectura_RB != 'M') && (lectura_RB != 'B'));

if (lectura_RB == 'B') {
    sonidoOK();
    parpadeoLed(0);
    digitalWrite(Led_rojo, LOW);
    digitalWrite(Led_verde, HIGH);
}

else if (lectura_RB == 'M') {
    sonidoNOK();
    parpadeoLed(0);
    digitalWrite(Led_rojo, HIGH);
    digitalWrite(Led_verde, LOW);
}

lectura_RB = 0;

Serial.read();
Serial.read();
Serial.read();
Serial.read();

delay(1000);
estado = 4; //Realizar otra calibración si pulsador
}
else { //La calibracion de alguna articulacion ha fallado
    sonidoNOK();
    resultado = '0';
    result = 0;
    parpadeoLed(0);
}
```

```
digitalWrite(Led_rojo, HIGH);
digitalWrite(Led_verde, LOW);

dataError[0] = err_cadera_izq;
dataError[1] = err_cadera_der;
dataError[2] = err_pie_izq;
dataError[3] = err_pie_der;
WriteComm(7); //Comando WERC: errores de calibracion para RB)

data[0] = 90;
data[1] = 90;
data[2] = 90;
data[3] = 90; //si sale mal, mandamos 90 En BBDD = 0 (90-90)
WriteComm(2); //Comando WOFF Pasamos a Zowi las posiciones HOME
WriteComm(10); //Salva base datos

//lcd.clear();
lcd.setCursor(0, 0);
lcd.print(" CALIBRATION ");
lcd.setCursor(0, 1);
lcd.print(" FAILED ");

Serial1.println("Calibracion MAL");
Serial1.println("Error calibracion articulaciones");
Serial1.print("Error cadera izq :");
Serial1.println(err_cadera_izq);
Serial1.print("Error pie izq :");
Serial1.println(err_pie_izq);
Serial1.print("Error cadera der :");
Serial1.println(err_cadera_der);
Serial1.print("Error pie der :");
Serial1.println(err_pie_der);
delay(1000);
estado = 4; //Realizar otra calibración si pulsador
}

break;

case 4: //Estado otra calibracion
num_vueltas = 0;

lcd.setCursor(0, 0);
lcd.print("Press for a new ");
lcd.setCursor(0, 1);
lcd.print(" calibration ");

if (digitalRead(pin_pulsador1) == 1)
    //pasamos a realizar otra calibración al pulsar
{
    Serial1.println("Iniciamos otra calibracion 4 a 2");
    estado = 2;
    timer = millis();
    timerLed = millis();
    delay(20);
    counter = 0;
}

break;
```

```
    }

}

void calibra_servos()
{
    switch (num_vueltas) {
    case 50:
        data[0] = 3; //Indicador cadera izquierda
        data[1] = 90;
        data[2] = 0;
        WriteComm(6); //MsXC Comando mover servo especifico a posicion especifica

        pos_old_cadera_izq = 90;
        break;

    case 60:
        data[0] = 4; //Indicador pie izquierdo
        data[1] = 90;
        data[2] = 0;
        WriteComm(6); //MsXC Comando mover servo especifico a posicion especifica

        pos_old_pie_izq = 90;
        break;

    case 70:
        data[0] = 1; //Indicador cadera derecha
        data[1] = 90;
        data[2] = 0;
        WriteComm(6); //MsXC Comando mover servo especifico a posicion especifica
        pos_old_cadera_der = 90;
        break;

    case 80:
        data[0] = 2; //Indicador pie derecho
        data[1] = 90;
        data[2] = 0;
        WriteComm(6); //MsXC Comando mover servo especifico a posicion especifica
        pos_old_pie_der = 90;
        break;

    case 100:
    case 150:
    case 200:
    case 250:
    case 300:

        data[0] = 3; //Indicador cadera izquierda
        data[1] = round(pos_old_cadera_izq + err_cadera_izq);
        data[2] = 0;
        WriteComm(6); //MsXC Comando mover servo especifico a posicion especifica

        pos_old_cadera_izq = round(pos_old_cadera_izq + err_cadera_izq);
        //Redondeamos en lugar de cast para mejorar el error [servo.write(int)]
        break;

    case 110:
    case 160:
```

```
case 210:
case 260:
case 310:

    data[0] = 4; //Indicador pie izquierdo
    if (caraZowi) {
        data[1] = round(pos_old_pie_izq + err_pie_izq);
    }
    else {
        data[1] = round(pos_old_pie_izq - err_pie_izq);
    }
    data[2] = 0;
    WriteComm(6); //Comando mover servo especifico a posicion especifica
    if (caraZowi) {
        pos_old_pie_izq = round(pos_old_pie_izq + err_pie_izq);
    }
    else {
        pos_old_pie_izq = round(pos_old_pie_izq - err_pie_izq);
    }
    break;

case 120:
case 170:
case 220:
case 270:
case 320:
    data[0] = 1; //Indicador pie izquierdo
    data[1] = round(pos_old_cadera_der + err_cadera_der);
    data[2] = 0;
    WriteComm(6); //MsXCComando mover servo especifico a posicion especifica
    pos_old_cadera_der = round(pos_old_cadera_der + err_cadera_der);
    break;

case 130:
case 180:
case 230:
case 280:
case 330:

    data[0] = 2; //Indicador pie derecho
    if (caraZowi) {
        data[1] = round(pos_old_pie_der + err_pie_der);
    }
    else {
        data[1] = round(pos_old_pie_der - err_pie_der);
    }

    data[2] = 0;
    WriteComm(6); //MsXC Comando mover servo especifico a posicion especifica

    if (caraZowi) {
        pos_old_pie_der = round(pos_old_pie_der + err_pie_der);
    }
    else {
        pos_old_pie_der = round(pos_old_pie_der - err_pie_der);
    }
```

```

        break;

    default:
        break;
    }
}

void parpadeoLed(int freqLed)
{
    if (freqLed == 0) {
        digitalWrite(Led_azul, LOW);
    }
    else if ((millis() - timerLed) >= (500 / freqLed))
        // Led parpadea N vez por segundo, N = freqLed
    {
        timerLed = millis();
        digitalWrite(Led_azul, !digitalRead(Led_azul));
    }
}

void calculosIMU()
{
    counter++;
    timer_old = timer;
    timer = millis();
    if (timer > timer_old)
        G_Dt = (timer - timer_old) / 1000.0;
        // Real time of loop run. We use this on the DCM algorithm
        // (gyro integration time)
    else
        G_Dt = 0;

    // *** DCM algorithm
    // Data adquisition
    Read_Gyro(); // This read gyro data
    Read_Accel(); // Read I2C accelerometer

    if (counter > 5) // Read compass data at 10Hz... (5 loop runs)
    {
        counter = 0;
        Read_Compass(); // Read I2C magnetometer
        Compass_Heading(); // Calculate magnetic heading
    }

    // Calculations...
    Matrix_update();
    Normalize();
    Drift_correction();
    Euler_angles();
    // **

    //printdata(); //Muestra datos leidos IMU
}

int init_imus()

```

```
{  
    Serial1.println("Calibrando imus...");  
  
    digitalWrite(STATUS_LED, LOW);  
    delay(1500);  
  
    Accel_Init();  
    Compass_Init();  
    Gyro_Init();  
  
    delay(20);  
  
    for (int i = 0; i < 32; i++) // We take some readings...  
    {  
        Read_Gyro();  
        Read_Accel();  
        for (int y = 0; y < 6; y++) // Cumulate values  
        {  
            AN_OFFSET[y] += AN[y];  
            AN_OFFSETd[y] += ANd[y];  
        }  
        delay(20);  
    }  
  
    for (int y = 0; y < 6; y++) {  
        AN_OFFSET[y] = AN_OFFSET[y] / 32;  
        AN_OFFSETd[y] = AN_OFFSETd[y] / 32;  
    }  
  
    AN_OFFSET[5] -= GRAVITY * SENSOR_SIGN[5];  
    AN_OFFSETd[5] -= GRAVITY * SENSOR_SIGNd[5];  
  
    delay(4000);  
    digitalWrite(STATUS_LED, HIGH);  
  
    timer = millis();  
    delay(20);  
    counter = 0;  
}  
  
int WriteComm(int command)  
{  
  
    int ind = 0;  
  
    switch (command) {  
  
        case 0: // Iniciar comunicación con ZUM.  
            for (int i = 0; i < 30; i++) {  
                trama[i] = 0;  
            }  
            for (int i = 0; i < 6; i++) {  
                trama[i] = C0[i];  
            }  
            trama[6] = '#';  
  
            Serial.write(trama);  
    }  
}
```

```
Serial.flush();
Serial1.println(trama);

break;

case 8: //cargar programa test y guardar en base de datos.
    for (int i = 0; i < 30; i++) {
        trama[i] = 0;
    }
    for (int i = 0; i < 6; i++) {
        trama[i] = C8[i];
    }
    trama[6] = '#';
    Serial.write(trama);
    ;
    Serial1.println(trama);

break;

case 1: //Leer offset

delay(100);
ind = 0;
//Quedamos a espera de la lectura
while (Wire.available()) {
    data[ind] = Wire.read();
    ind++;
}
dataINT[0] = data[0];
dataINT[1] = data[1];
dataINT[2] = data[2];
dataINT[3] = data[3];
Serial1.println("Posicion HOME leida de ZOWI:");
Serial1.print("Home cadera izq: ");
Serial1.println(dataINT[0]);
Serial1.print("Home pie izq: ");
Serial1.println(dataINT[1]);
Serial1.print("Home cadera der: ");
Serial1.println(dataINT[2]);
Serial1.print("Home pie der: ");
Serial1.println(dataINT[3]);
break;

case 22: //Escribir offset WOFC Izquierda
    for (int i = 0; i < 29; i++) {
        trama[i] = 0;
    }
    ///////////////////////////////////////////////////
/* INFORMACION
    data[0]=pos_old_cadera_der; //COLUMNA 1 para BASE DATOS
    data[1]=pos_old_pie_der; //COLUMNA 3 oara BASE DATOS
    data[2]=pos_old_cadera_izq; //COLUMNA 0 oara BASE DATOS
    data[3]=pos_old_pie_izq; //COLUMNA 2 para base DATOS
    */ ///////////////////////////////////////////////////
for (int i = 0; i < 6; i++) {
    trama[i] = C2[i];
```

```
}

trama[6] = '2';

numeroData = 2;
for (int j = 7; j < 19;) {
    str = String(int(data[numeroData] / 100));
    str.toCharArray(temp, 2);
    trama[j] = temp[0];
    j++;
    str = String(int((data[numeroData] % 100) / 10));
    str.toCharArray(temp, 2);
    trama[j] = temp[0];
    j++;
    str = String(int(data[numeroData] % 10));
    str.toCharArray(temp, 2);
    trama[j] = temp[0];
    numeroData++;
    if (numeroData == 4) {
        numeroData = 0;
    };
    j++;
    //Serial1.println("aqui");
}
trama[19] = '#';

Serial1.println(trama);
//Mandamos los valores
Serial.write(trama);
break;

case 2: //Escribir offset WOFC
for (int i = 0; i < 29; i++) {
    trama[i] = 0;
}
/////////////////////////////
/* INFORMACION
data[0]=pos_old_cadera_der; //COLUMNA 1 para BASE DATOS
data[1]=pos_old_pie_der; //COLUMNA 3 oara BASE DATOS
data[2]=pos_old_cadera_izq; //COLUMNA 0 oara BASE DATOS
data[3]=pos_old_pie_izq; //COLUMNA 2 para base DATOS
*/
/////////////////////////////

for (int i = 0; i < 6; i++) {
    trama[i] = C2[i];
}

trama[6] = '1';

numeroData = 0;
for (int z = 7; z < 19;) {
    str = String(int(data[numeroData] / 100));
    str.toCharArray(temp, 2);
    trama[z] = temp[0];
    z++;
    str = String(int((data[numeroData] % 100) / 10));
    str.toCharArray(temp, 2);
```

```
        trama[z] = temp[0];
        z++;
        str = String(int(data[numeroData] % 10));
        str.toCharArray(temp, 2);
        trama[z] = temp[0];
        numeroData++;
        z++;
        //Serial1.println("aqui");
    }
    trama[19] = '#';

    Serial1.println(trama);
    //Mandamos los valores
    Serial.write(trama);
    break;

case 3: //Mover servos a 90°
    for (int i = 0; i < 30; i++) {
        trama[i] = 0;
    }
    for (int i = 0; i < 6; i++) {
        trama[i] = C3[i];
    }
    trama[6] = '#';

    Serial.write(trama);

    Serial1.println("Enviado M90C:");
    break;
case 4: //Mover servos a HOME (90 calibrado)
    for (int i = 0; i < 30; i++) {
        trama[i] = 0;
    }
    for (int i = 0; i < 6; i++) {
        trama[i] = C4[i];
    }
    trama[6] = '#';

    Serial.write(trama);

    break;
case 5: //Mover 4 servos a posicion especifica (para calibracion con IMU)
    Serial1.println("Enviado MSSC:");
    //Mandamos los valores
    //Wire.write(data);
    break;
case 6: //Mover un servo a posicion especifica (para calibracion con IMU)MSxC

    for (int i = 0; i < 6; i++) {
        trama[i] = C6[i];
    }

    //Serial1.println(int(data[1]));

    str = String(int(data[0]));
    str.toCharArray(temp, 2);
    trama[6] = temp[0];
```

```
str = String(int(data[1] / 100));
str.toCharArray(temp, 2);
trama[7] = temp[0];
str = String(int((data[1] % 100) / 10));
str.toCharArray(temp, 2);
trama[8] = temp[0];
str = String(int(data[1] % 10));
str.toCharArray(temp, 2);
trama[9] = temp[0];

trama[10] = '#';
Serial.write(trama);
//Serial1.write(trama);
Serial1.println(trama);
break;

case 7: //WERC
//    for (int i =0; i<=40;i++){
//        tramaError[i]=0;
//    }

for (int i = 0; i < 6; i++) {
    tramaError[i] = C7[i];
}

for (int i = 0; i < 4; i++) {
    if (dataError[i] < 0) {
        tramaError[(6 + i * 8)] = '-';
    }
    else {
        tramaError[(6 + i * 8)] = '+';
    }
    dataError[i] = abs(int(dataError[i] * 100));

    str = String(int(dataError[i] / 10000));
    str.toCharArray(temp, 2);
    tramaError[(7 + i * 8)] = temp[0];

    str = String((int(dataError[i] / 1000)) % 10);
    str.toCharArray(temp, 2);
    tramaError[(8 + i * 8)] = temp[0];

    str = String((int(dataError[i] / 100) % 10));
    str.toCharArray(temp, 2);
    tramaError[(9 + i * 8)] = temp[0];

    tramaError[(10 + i * 8)] = '.';

    str = String((int(dataError[i]) / 10) % 10);
    str.toCharArray(temp, 2);
    tramaError[(11 + i * 8)] = temp[0];

    str = String((int(dataError[i]) % 10));
    str.toCharArray(temp, 2);
    tramaError[(12 + i * 8)] = temp[0];
```

```
        tramaError[(13 + i * 8)] = '*';
    }

tramaError[38] = resultado;
tramaError[39] = '#';

Serial1.println(tramaError);
Serial.write(tramaError);
//      for (int i =0; i<=40;i++) {
//          tramaError[i]=0;
//      }
break;

case 9: //Apagar RB
    for (int i = 0; i < 30; i++) {
        trama[i] = 0;
    }
    for (int i = 0; i < 6; i++) {
        trama[i] = C9[i];
    }

    trama[6] = '#';
    Serial.write(trama);

    break;

case 10:
    for (int i = 0; i < 30; i++) {
        trama[i] = 0;
    }
    for (int i = 0; i < 6; i++) {
        trama[i] = C10[i];
    }

    trama[6] = '#';
    Serial.write(trama);

default:
    break;
}

data[0] = 0;
data[1] = 0;
data[2] = 0;
data[3] = 0;
data[4] = 0;

return 1;
}

void sonidoNOK()
{
    tone(Pin_buzz, 100);
    delay(800);
    noTone(Pin_buzz);
}
```

```
void sonidoOK()
{
    tone(Pin_buzz, 400);
    delay(300);
    noTone(Pin_buzz);
    delay(50);
}

void sonidostart()
{
    tone(Pin_buzz, 560);
    delay(300);
    noTone(Pin_buzz);
    delay(10);
    tone(Pin_buzz, 545);
    delay(150);
    tone(Pin_buzz, 575);
    delay(400);
    noTone(Pin_buzz);
    delay(50);
}
```


Apéndice Q

Código Zowi

Programa intérprete que se descarga en Zowi para ser calibrado.

```

#include <Wire.h>
#include <Servo.h>
#include <EEPROM.h>
#include <I2C_eeprom.h>

I2C_eeprom ee(0x50, 128);

int slave_add = 1;
int Comando = 0;
int Envio = 0;
int flagRepcion = 0;

int ang_cad_der, ang_pie_der, ang_cad_izq, ang_pie_izq;

int giro;

int trama[30]; //5 primeros son el código de Comando. "XXXX:". trama[5] indica,
// si procede, el número de servo.
// 6,7,8 indican (3 cifras en Ascii) valor numerico para nº grados
// de servo indicado en trama 5.
// para funciones generales a todos los servos con parametros de
// grados (escribir eeprom o mover todos al mismo tiempo):
// 6, 7, 8= cadera derecha // 9, 10, 11= pie derecho // 12, 13, 14 = cadera izq
// // 15, 16, 17 = pie izquierdo
// 18 Fin carro
char data[4] = {
  0, 0, 0, 0
}; //Para enviar las posiciones home de las articulaciones al maestro

Servo cadera_der;
Servo cadera_izq;
Servo pie_der;
Servo pie_izq;

int home_pie_izq = 0;
int home_pie_der = 0;
int home_cad_izq = 0;
int home_cad_der = 0;
int LnNumSerie = 6;

//Mapa memoria EEPROM, offset de fabricacion en los servos
//Cambios a fecha 09/09/2015: reordenado
int add_off_cad_izq = 0;
int add_off_cad_der = 1;
int add_off_pie_izq = 2;
int add_off_pie_der = 3;

int add_nombre = 5;

char NS[13] = { '$', 'O', 'K', 'N', 'S', ':', 0, 0, 0, 0, 0, '#' };

void setup()
{
  //Wire.begin(slave_add); // join i2c bus with address #1
  //Wire.onReceive(receiveEvent); // register event
  //Repcion datos en esclavo (escritura desde el maestro)
  //Wire.onRequest(requestEvent); // register event
}

```



```

// ang_cad_izq = (trama[12]-'0')*100 + (trama[13]-'0')*10 +
//                  (trama[14]-'0');
// ang_pie_izq = (trama[15]-'0')*100 + (trama[16]-'0')*10 +
//                  (trama[17]-'0');

//iteracion 4:
ang_cad_der = ang_cad_der - 90;
ang_pie_der = ang_pie_der - 90;
// ang_cad_izq = ang_cad_izq-90;
// ang_pie_izq = ang_pie_izq-90;

//EEPROM.write(add_off_pie_izq,ang_pie_izq); //3
//delay(30);
//EEPROM.write(add_off_cad_izq,ang_cad_izq); //2
//delay(30);
EEPROM.write(add_off_pie_der, ang_pie_der); //1
delay(30);
EEPROM.write(add_off_cad_der, ang_cad_der); //0
delay(30);
//    Serial.println("HOME escrito en EEPROM:");
//    Serial.println(ang_cad_der);
//    Serial.println(ang_pie_der);
//    Serial.println(ang_cad_izq);
//    Serial.println(ang_pie_izq);
break;

case 22: //Escribir offset

// ang_cad_der = (trama[6]-'0')*100 + (trama[7]-'0')*10 + (trama[8]-'0')+
// ang_pie_der = (trama[9]-'0')*100 + (trama[10]-'0')*10 +
// (trama[11]-'0');
ang_cad_izq = (trama[6] - '0') * 100 + (trama[7] - '0') * 10 +
               (trama[8] - '0');
ang_pie_izq = (trama[9] - '0') * 100 + (trama[10] - '0') * 10 +
               (trama[11] - '0');

//iteracion 4:
// ang_cad_der = ang_cad_der-90;
// ang_pie_der = ang_pie_der-90;
ang_cad_izq = ang_cad_izq - 90;
ang_pie_izq = ang_pie_izq - 90;

EEPROM.write(add_off_pie_izq, ang_pie_izq); //3
delay(30);
EEPROM.write(add_off_cad_izq, ang_cad_izq); //2
delay(30);
//EEPROM.write(add_off_pie_der,ang_pie_der); //1
//delay(30);
//EEPROM.write(add_off_cad_der,ang_cad_der); //0
//delay(30);
//    Serial.println("HOME escrito en EEPROM:");
//    Serial.println(ang_cad_der);
//    Serial.println(ang_pie_der);
//    Serial.println(ang_cad_izq);
//    Serial.println(ang_pie_izq);
break;

```

```

case 3: //Mover a 90°
    cadera_der.write(90);
    cadera_izq.write(90);
    pie_der.write(90);
    pie_izq.write(90);
    Serial.println("A 90");
    break;
case 4: //Mover a HOME
    home_pie_izq = EEPROM.read(add_off_pie_izq);
    home_cad_izq = EEPROM.read(add_off_cad_izq);
    home_pie_der = EEPROM.read(add_off_pie_der);
    home_cad_der = EEPROM.read(add_off_cad_der);
    cadera_der.write(home_cad_der);
    cadera_izq.write(home_cad_izq);
    pie_der.write(home_pie_der);
    pie_izq.write(home_pie_izq);
    Serial.println("A HOME");
    break;
case 5: //Mover servos a posicion especifica
    ang_cad_der = (trama[6] - '0') * 100 + (trama[7] - '0') * 10 +
                  (trama[8] - '0');
    ang_pie_der = (trama[9] - '0') * 100 + (trama[10] - '0') * 10 +
                  (trama[11] - '0');
    ang_cad_izq = (trama[12] - '0') * 100 + (trama[13] - '0') * 10 +
                  (trama[14] - '0');
    ang_pie_izq = (trama[15] - '0') * 100 + (trama[16] - '0') * 10 +
                  (trama[17] - '0');

    cadera_der.write(ang_cad_der);
    pie_der.write(ang_pie_der);
    cadera_izq.write(ang_cad_izq);
    pie_izq.write(ang_pie_izq);

    Serial.println("Todos A POS dada");
    break;
case 6: //Mover servo indicado a posicion especifica
    giro = 0;
    giro = (trama[6] - '0') * 100;
    giro = giro + (trama[7] - '0') * 10;
    giro = giro + (trama[8] - '0');

    switch (trama[5] - '0') {
        case 1: //Cadera derecha
            cadera_der.write(giro);
            Serial.println("Cad_der A POS dada");
            break;
        case 2: //Pie derecho
            pie_der.write(giro);
            Serial.println("pie_der A POS dada");
            break;
        case 3: //Cadera izquierda
            cadera_izq.write(giro);
            Serial.println("Cad_izq A POS dada");
            break;
        case 4: //Pie izquierdo
    }
}

```

```
    pie_izq.write(giro);
    Serial.println("pie_izq A POS dada");
    break;
  default:
    Serial.println("Seleccion servo NOK");
    break;
}
break;
default:
  Serial.println("Comando NOK");
  break;
}

Serial.println("Trama recibida:");
Serial.println(trama[0]);
Serial.println(trama[1]);
Serial.println(trama[2]);
Serial.println(trama[3]);
Serial.println(trama[4]);
Serial.println(trama[5]);
Serial.println(trama[6]);
Serial.println(trama[7]);
Serial.println(trama[8]);
Serial.println(trama[9]);
Serial.println(trama[10]);
Serial.println(trama[11]);
Serial.println(trama[12]);
Serial.println(trama[13]);
Serial.println(trama[14]);
Serial.println(trama[15]);
Serial.println(trama[16]);
Serial.println(trama[17]);
Serial.println(trama[18]);
Serial.println(trama[19]);
Serial.println(trama[20]);

flagRepcion = 0;
trama[0] = 0;
trama[1] = 0;
trama[2] = 0;
trama[3] = 0;
trama[4] = 0;
trama[5] = 0;
trama[6] = 0;
trama[7] = 0;
trama[8] = 0;
trama[9] = 0;
trama[10] = 0;
trama[11] = 0;
trama[12] = 0;
trama[13] = 0;
trama[14] = 0;
trama[15] = 0;
trama[16] = 0;
trama[17] = 0;
trama[18] = 0;
trama[19] = 0;
```

```
trama[20] = 0;
trama[21] = 0;
trama[22] = 0;
trama[23] = 0;
trama[24] = 0;
trama[25] = 0;
trama[26] = 0;
trama[27] = 0;
trama[28] = 0;
trama[29] = 0;
}
delay(100);
}

// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void receiveEvent(int howMany)
{
    int indice = 0;

    while (1 <= Serial.available()) // loop through all but the last
    {
        trama[indice] = Serial.read(); // receive byte as a character
        indice++;
    }

    switch (trama[0]) {
    case 82: // "R":
        //if(trama[1]==="O" && trama[2]==="F" && trama[3]==="C" &&trama[4]==":") {
        //Comando=1;};
        if (trama[1] == 79 && trama[2] == 70 && trama[3] == 67 && trama[4] == 58) {
            Comando = 1;
        };
        break;
    case 87: // "W":
        //if(trama[1]==="O" && trama[2]==="F" && trama[3]==="C" &&trama[4]==":") {
        // Comando=2;};
        if (trama[1] == 79 && trama[2] == 70 && trama[3] == 67 && trama[4] == 58) {
            if (trama[5] == 49) //Derecha(1)
            {
                Comando = 2;
            }
            else if (trama[5] == 50) //Izquierda(2)
            {
                Comando = 22;
            }
        };
        break;
    case 77: // "M":
        //if(trama[1]==="9" && trama[2]==="0" && trama[3]==="C" &&trama[4]==":") {
        // Comando=3;};
        if (trama[1] == 57 && trama[2] == 48 && trama[3] == 67 && trama[4] == 58) {
            Comando = 3;
        };
        //if(trama[1]==="H" && trama[2]==="O" && trama[3]==="C" &&trama[4]==":") {
        // Comando=4;};
        if (trama[1] == 72 && trama[2] == 79 && trama[3] == 67 && trama[4] == 58) {
```

```
        Comando = 4;
    };
    //if(trama[1]== "S" && trama[2]== "S" && trama[3]== "C" &&trama[4]==":") {
    // Comando=5; }
    if (trama[1] == 83 && trama[2] == 83 && trama[3] == 67 && trama[4] == 58) {
        Comando = 5;
    };
    if (trama[1] == 83 && trama[2] == 120 && trama[3] == 67 && trama[4] == 58) {
        Comando = 6;
    };
    break;
default:
    Comando = 0;
    break;
}

flagRepcion = 1;
}

void requestEvent()
{ //Preparamos la trama para enviar al maestro
switch (Envio) {
case 1:
    data[0] = home_cad_izq;
    data[1] = home_pie_izq;
    data[2] = home_cad_der;
    data[3] = home_pie_der;
    data[4] = 0;
    Wire.write(data); // respond with message of 11 bytes
    // as expected by master
    break;
default:
    break;
}
Envio = 0;
data[0] = 0;
data[1] = 0;
data[2] = 0;
data[3] = 0;
data[4] = 0;
}
```

Apéndice R

Especificaciones Mega

Especificaciones de la Freaduino Mega 2560: se muestran las especificaciones de la original Arduino Mega 2560 en primer lugar y los cambios de la versión clónica.

Arduino Mega 2560

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
LED_BUILTIN	13
Length	101.52 mm
Width	53.3 mm
Weight	37 g

Source: <https://www.arduino.cc/en/Main/arduinoBoardMega2560#techspecs>

Freaduino Mega 2560

Features

- Inherits all of Arduino MEGA2560's features.
- Compatible to Arduino MEGA2560's pin layout, screw hole and dimensions.
- 3.3V or 5v Operating Voltage selectable
- More visible location of Indicator LEDs
- Wide range external input from 7~23V DC
- Evolved with SMD components

Changes vs. original Arduino Mega 2560

	Arduino MEGA2560	Freaduino MEGA2560
USB socket	Type B Female USB connector	Mini USB connector
Operating Voltage	5V	3.3V or 5V selectable by switch
3.3V current	50mA	800mA
5V current	500mA	2A
Input voltage	7V~12V	7V~23V
Reset button location	Hard to press when plug in shield	Easy to press whenever
LED location	Invisible when plug in shield	Visible whenever

MiniUSB changed to Type B USB

Source: https://www.elecfreaks.com/wiki/index.php?title=Freaduino_Mega2560

Apéndice S

Especificaciones Raspberry

Especificaciones de la Raspberry PI 2 Model B.

Raspberry PI 2: Model B

Technical Specifications:

- Broadcom BCM2837 Arm7 Quad Core Processor powered Single Board Computer running at 900MHz
- 1GB RAM
- 40pin extended GPIO
- 4 x USB 2 ports
- 4 pole Stereo output and Composite video port
- Full size HDMI
- CSI camera port for connecting the Raspberry Pi camera
- DSI display port for connecting the Raspberry Pi touch screen display
- Micro SD port for loading your operating system and storing data
- Micro USB power source

Raspberry Pi 2 Model B Features:

- Broadcom BCM2837Arm7 Quad Core Processor powered Single Board Computer running at 900MHz
- 1GB RAM so you can now run bigger and more powerful applications
- Identical board layout and footprint as the Model B+, so all cases and 3rd party add-on boards designed for the Model B+ will be fully compatible.
- Fully HAT compatible
- 40pin extended GPIO to enhance your “real world” projects. GPIO is 100% compatible with the Model B+ and A+ boards. First 26 pins are identical to the Model A and Model B boards to provide full backward compatibility across all boards.
- Connect a Raspberry Pi camera and touch screen display (each sold separately)
- Stream and watch Hi-definition video output at 1080P
- Micro SD slot for storing information and loading your operating systems.
- Advanced power management:
- You can now provide up to 1.2 AMP to the USB port – enabling you to connect more power hungry USB devices directly to the Raspberry PI. (This feature requires a 2Amp micro USB Power Supply)
- 10/100 Ethernet Port to quickly connect the Raspberry Pi to the Internet
- Combined 4-pole jack for connecting your stereo audio out and composite video out