

# Universidad de Alcalá Escuela Politécnica Superior

## Grado en Ingeniería en Electrónica y Automática Industrial

### Trabajo Fin de Grado

Análisis de unidades iniciales de medida (IMU) y diseño de controlador de ángulo de ataque aplicado a cuadricóptero

ESCUELA POLITECNICA  
SUPERIOR

**Autor:** Sergio Martín Gómez

**Tutor:** D. Iván García Daza

2015



# UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

**Grado en Ingeniería en Electrónica y Automática Industrial**

## Trabajo Fin de Grado

**Análisis de unidades inerciales de medida (IMU) y diseño de controlador de ángulo de ataque aplicado a cuadricóptero**

Autor: Sergio Martín Gómez

Director: D. Iván García Daza

### Tribunal:

**Presidente:** D. David Fernández Llorca

**Vocal 1º:** D. Ignacio Parra Alonso

**Vocal 2º:** D. Iván García Daza

Calificación: .....

Fecha: .....



# Agradecimientos

Este trabajo es el fruto de muchas horas de estudio y de trabajo que no habría sido posible sin la ayuda de mis compañeros Alejandro y Marcial y de mi director de proyecto Iván García.

Agradezco también a mi padre y a mi hermano su apoyo incondicional a lo largo de toda la carrera, que ha sido fundamental para superarla.



# Índice general

Índice general	vii
Índice de figuras	xi
Índice de tablas	xiii
Índice de listados de código fuente	xv
Resumen	xvii
Abstract	xix
Resumen Extendido	xxi
<b>I Introducción</b>	<b>1</b>
<b>1 Introducción</b>	<b>3</b>
1.1 Presentación . . . . .	3
1.2 Estado del arte . . . . .	5
1.3 Motivación y objetivos del proyecto . . . . .	5
1.3.1 Motivación . . . . .	5
1.3.2 Objetivos . . . . .	6
1.4 Medios y herramientas necesarios . . . . .	6
<b>II Elección de los componentes</b>	<b>7</b>
<b>2 Elección de los componentes</b>	<b>9</b>
2.1 Controlador . . . . .	9
2.1.1 Arduino Uno . . . . .	9
2.1.2 A-Star 32U4 Prime SV . . . . .	10
2.1.3 Comparativa y elección . . . . .	11
2.2 Unidad inercial de medidas (IMU) . . . . .	11

2.2.1	AltIMU-10 . . . . .	11
2.2.2	MPU 6050 . . . . .	12
2.2.3	Comparativa y elección . . . . .	13
2.3	Motores . . . . .	13
2.3.1	Motores DC (Brushed Motors) . . . . .	13
2.3.2	Motores sin escobillas (Brushless Motors) . . . . .	14
2.3.3	Comparativa y elección . . . . .	14
2.4	Controlador electrónico de velocidad . . . . .	15
2.5	Hélices . . . . .	16
2.6	Batería o fuente de alimentación . . . . .	17
2.7	Montaje final . . . . .	18
<b>III</b>	<b>Sensores de posición</b>	<b>19</b>
<b>3</b>	<b>Sensores de posición</b>	<b>21</b>
3.1	Introducción . . . . .	21
3.2	Conexión de la IMU . . . . .	21
3.3	Giróscopo . . . . .	23
3.3.1	Fuentes de error . . . . .	25
3.3.1.1	Offset . . . . .	25
3.3.1.2	Deriva (Drift) . . . . .	26
3.3.1.3	Precisión . . . . .	26
3.3.2	Medidas . . . . .	26
3.4	Acelerómetro . . . . .	28
3.4.1	Fuentes de error . . . . .	30
3.4.1.1	Offset . . . . .	30
3.4.1.2	Ruido . . . . .	31
3.4.2	Medidas . . . . .	31
3.5	Filtros . . . . .	32
3.5.1	Filtro de Kalman . . . . .	32
3.5.2	Filtro complementario . . . . .	33
3.5.3	Medidas . . . . .	33
<b>IV</b>	<b>Motores y ESC</b>	<b>37</b>
<b>4</b>	<b>Motores y ESC</b>	<b>39</b>
4.1	Introducción . . . . .	39
4.2	Conexión de los motores y los ESC . . . . .	39
4.3	Regulación de velocidad de los motores . . . . .	40

<b>V Estabilización del brazo</b>	<b>43</b>
<b>5 Estabilización del brazo</b>	<b>45</b>
5.1 Introducción . . . . .	45
5.2 Controladores . . . . .	45
5.2.1 PID . . . . .	45
5.2.2 PPI . . . . .	50
5.2.3 Definiciones específicas del tipo de documento . . . . .	53
5.2.4 Plantilla de anteproyecto . . . . .	56
5.2.5 Papeleo adicional para la defensa de los TFGs . . . . .	56
5.2.6 Generación del documento con control de cambios (para revisión) . . . . .	56
5.2.7 Problemas conocidos . . . . .	57
5.3 Ejemplos de elementos de utilidad . . . . .	58
5.3.1 Uso de comandos definidos . . . . .	58
5.3.2 Uso de “frases célebres” . . . . .	58
5.3.3 Inclusión de diagramas . . . . .	58
5.3.4 Definición y uso de acrónimos (aquí también se pueden poner acrónimos como ETTS) . . . . .	58
5.3.5 Definición y uso de símbolos (aquí también se pueden símbolos como $x_i(t)$ ) . . . . .	59
5.4 Motivación y objetivos . . . . .	60
5.5 Organización de la memoria . . . . .	60
<b>6 Estudio teórico</b>	<b>61</b>
6.1 Introducción . . . . .	61
6.2 Estado del Arte . . . . .	61
6.3 Técnicas utilizadas . . . . .	62
6.3.1 Subsección . . . . .	62
6.3.1.1 Subsubsección . . . . .	62
6.3.1.1.1 Paragraph . . . . .	62
Subparagraph . . . . .	62
6.4 Conclusiones . . . . .	63
<b>7 Desarrollo</b>	<b>65</b>
7.1 Introducción . . . . .	65
7.2 Desarrollo del sistema de experimentación . . . . .	65
7.3 Planteamiento matemático . . . . .	65
7.4 Conclusiones . . . . .	65

---

<b>8 Resultados</b>	<b>67</b>
8.1 Introducción . . . . .	67
8.2 Entorno experimental . . . . .	67
8.2.1 Bases de datos utilizadas . . . . .	67
8.2.2 Métricas de calidad . . . . .	67
8.2.3 Estrategia y metodología de experimentación . . . . .	67
8.3 Resultados experimentales . . . . .	67
8.4 Conclusiones . . . . .	74
<b>9 Conclusiones y líneas futuras</b>	<b>77</b>
9.1 Conclusiones . . . . .	77
9.2 Líneas futuras . . . . .	77
<b>A Manual de usuario</b>	<b>79</b>
A.1 Introducción . . . . .	79
A.2 Manual . . . . .	79
A.3 Ejemplos de inclusión de fragmentos de código fuente . . . . .	79
A.4 Ejemplos de inclusión de algoritmos . . . . .	80
<b>B Herramientas y recursos</b>	<b>83</b>

# Índice de figuras

1.1 Cuadricóptero de Bothezat . . . . .	3
1.2 Cuadricóptero de Oehmichen . . . . .	3
1.3 Cuadricóptero de Amazon en pruebas . . . . .	4
1.4 Cuadricóptero con cámara acoplada . . . . .	4
1.5 Detalle del vuelo invertido del cuadricóptero . . . . .	5
2.1 Microcontrolador Arduino Uno . . . . .	10
2.2 Microcontrolador A-Star 32U4 Prime SV . . . . .	10
2.3 AltIMU-10 . . . . .	12
2.4 MPU 6050 . . . . .	12
2.5 Motor de corriente continua . . . . .	13
2.6 Motor sin escobillas . . . . .	14
2.7 ESC. . . . .	15
2.8 Señal ESC. . . . .	16
2.9 Batería LiPo . . . . .	17
2.10 Fuente de alimentación utilizada . . . . .	17
2.11 Brazo montado. . . . .	18
2.12 Imágenes del esquema y del circuito montado. . . . .	18
3.1 IMU con los nombres de los pines. . . . .	21
3.2 Esquema del circuito. . . . .	22
3.3 Conexión de la IMU. . . . .	22
3.4 Giróscopo mecánico. . . . .	23
3.5 Resultados de la medida del ángulo con el giróscopo. . . . .	27
3.6 Ángulo medido con el giróscopo variando el ángulo. . . . .	27
3.7 Acelerómetro de estructura vibrante. . . . .	28
3.8 Resultados de la medida del ángulo con el acelerómetro. . . . .	31
3.9 Ruido en la medida del ángulo con el acelerómetro. . . . .	32
3.10 Algoritmo recursivo del filtro de Kalman. . . . .	33

3.11 Resultados de la medida del ángulo con el giróscopo. . . . .	34
3.12 Medida del ángulo moviendo el brazo. . . . .	34
3.13 Resultados de la medida del ángulo con el giróscopo. . . . .	35
4.1 Esquema de conexión motores. . . . .	39
4.2 Fases intercambiadas en uno de los motores. . . . .	40
5.1 Esquema del sistema. . . . .	46
5.2 Respuesta del sistema para controlador PID. . . . .	48
5.3 Detalle del ruido presente en la respuesta en régimen permanente del controlador PID. . . . .	49
5.4 Esquema del controlador PPI. . . . .	50
5.5 Respuesta del sistema para controlador PPI. . . . .	52
5.6 Detalle del ruido presente en la respuesta en régimen permanente del controlador PPI. . . . .	53
5.7 Clasificación de los objetos para la gramática (aquí también se pueden poner acrónimos como ETTS y símbolos como $x_i(t)$ ). . . . .	59
6.1 Departamento de Electrónica. . . . .	62
6.2 Departamento de Electrónica en el lateral. . . . .	62
8.1 Optimal frames number in the training data set. . . . .	69
8.2 Idiap Smart Meeting Room for AV16.3 recordings. (a) Room layout showing the centered table, and the microphones arranged in two circular arrays. (b) Sample of recorded video frame showing the arrays area. . . . .	70
8.3 Geometrical details for the experiments carried out. Only the relevant section of the room is shown, and microphone pairs are connected by solid lines. . . . .	70
8.4 Comparison between the steered power response generated by the model (solid line) and that calculated using simulated waveforms in the AV16.3 environment (stems). Results for the speaker in given angles and the array steered from -90° to +90° are shown. . . . .	71
8.5 Comparison between the SRP-PHAT map predicted by the model (left graphics), the real SRP-PHAT map (middle graphics), and the average (real) SRP-PHAT map (right graphics), for several speaker positions ( $f_0 = 1,5 \text{ KHz}$ ). See figure 8.3.b for geometrical references. . . . .	73

# Índice de tablas

2.1	Comparativa controladores programables. . . . .	11
2.2	Comparativa IMU. . . . .	13
2.3	Comparativa Motores. . . . .	14
2.4	Especificaciones motor. . . . .	15
2.5	Especificaciones ESC. . . . .	16
2.6	Especificaciones fuente de alimentación. . . . .	17
5.1	Parámetros en régimen transitorio PID. . . . .	50
5.2	Parámetros en régimen transitorio PPI. . . . .	53
8.1	Comparativa. . . . .	68
8.2	Resultados de la correlación cruzada. . . . .	68
8.3	Resultados TEST CLEAR 2006. . . . .	75



# Índice de listados de código fuente

3.1	Ejemplo de programa para tomar medidas del giróscopo. . . . .	24
3.2	Ejemplo para obtener una medida de la posición. . . . .	24
3.3	Eliminación del error de offset del giróscopo. . . . .	25
3.4	Ejemplo de programa para tomar medidas del acelerómetro. . . . .	29
3.5	Estimación del ángulo mediante el acelerómetro. . . . .	30
3.6	Eliminación del error de offset del acelerómetro. . . . .	30
3.7	Línea de código para implementar el filtro complementario. . . . .	33
4.1	Ejemplo de programa para arrancar los motores. . . . .	41
5.1	Implementación del PID. . . . .	47
5.2	Cálculo de parámetros en régimen transitorio con Matlab. . . . .	49
5.3	Implementación del PID. . . . .	51



# Resumen

Este Trabajo de Fin de Grado ha consistido en estimar los ángulos de inclinación de una plataforma, sobre la que se han colocado dos motores, con el fin de estabilizarla en un ángulo de referencia indicado por el usuario. Para la estimación de los ángulos se ha utilizado un conjunto de sensores llamados giróscopos y acelerómetros, ambos de tres ejes, que están integrados en un sensor llamado IMU. Los motores se han controlado mediante un ESC (Electronic Speed Controller), que además cumple la función de inversor.

**Palabras clave:** IMU, ESC, Arduino, Giróscopo, Acelerómetro .



# Abstract

This Final Year Project's purpose is to estimate the tilt angle of a platform, which has two engines installed above, in order of stabilize it in the angle that the user desires. To estimate the angles a set of sensors called gyroscopes and accelerometers have been used, both of three axis, integrated into a sensor called IMU. The engines have been controlled using a electronic speed controller (ESC), that also serves as a DC/AC converter.

**Keywords:** IMU, ESC, Arduino, Gyroscope, Accelerometer.



# Resumen Extendido

Este proyecto consiste en controlar los dos motores que se encuentran en un eje de un cuadridóptero para que se ajuste a un ángulo de referencia pasado como entrada al sistema. El proyecto consta de varias etapas que se explicarán brevemente en este resumen.

La primera etapa de este proyecto será estudiar los componentes disponibles de los distintos fabricantes y decidir cuáles son los más adecuados para el proyecto en función de su precisión, de su facilidad de uso y del precio. Los componentes necesarios para llevar a cabo este proyecto son:

- **Sensor de posición**
- **Microcontrolador**
- **Controladores de los Motores (ESC)**
- **Motores**
- **Batería o fuente de alimentación**

Una vez elegidos los componentes que se van a utilizar se debe realizar un estudio del principio de medida de los sensores utilizados. Los sensores que se van a utilizar para estimar el ángulo de la plataforma, ambos integrados en una IMU (Inertial Measurement Unit), son:

- **Giróscopo**
- **Acelerómetro**

El principio de medida de estos dos sensores no es el mismo y en ninguno de los dos casos proporcionan una medida directa del ángulo, pero con unas sencillas operaciones se puede obtener de ambos sensores una medida de la posición. Ambos sensores presentan errores que son intolerables para esta aplicación, sin embargo utilizando los dos en conjunto se obtiene una medida de alta precisión del ángulo. Se especificará en la memoria cómo se obtiene el ángulo a partir de sus medidas.

El controlador que se va utilizar para el proyecto será un Arduino Uno, elegido por su bajo coste y la facilidad en la programación. La plataforma Arduino tiene una gran cantidad de librerías disponibles que hacen que sea muy fácil interactuar con sensores y con actuadores conectados al Arduino. La comunicación entre la IMU y el Arduino se realiza mediante el protocolo de comunicación serie  $I^2C$ , mediante el cual se pueden conectar distintos componentes al Arduino y controlarlos mediante el puerto serie. Por tanto el primer paso para la adquisición de datos será realizar un programa para la recepción de datos

mediante el puerto serie desde la IMU. Teniendo el programa de recepción de datos hecho sólo queda realizar un programa para procesar esas medidas y obtener una medida del ángulo dar finalizada esta etapa..

La siguiente etapa del proyecto consistirá en realizar un estudio del ESC (Electronic Speed Controller) y de los motores. Se debe determinar el funcionamiento de ambos componentes para poder realizar un controlador adecuado. Para el estudio se utilizarán las especificaciones del fabricante además de realizar pruebas experimentales con el fin de comprobar su funcionamiento real.

La etapa final del proyecto es la estabilización del brazo. Llegado a este punto ya se ha implementado el programa para medir el ángulo del brazo con una precisión adecuada y ya se ha estudiado la forma de variar la velocidad de los motores, por lo tanto se realizará el control del brazo mediante distintos reguladores.

CAPITULO I

## **Introducción**



# Capítulo 1

## Introducción

### 1.1 Presentación

Para conocer el primer concepto de cuadricóptero construido hay que remontarse al año 1922. Ese año el norteamericano George de Bothezat fue el primero en hacer volar un aparato con cuatro motores pero sin mucho éxito, pues el aparato no consiguió subir más de 5 metros.

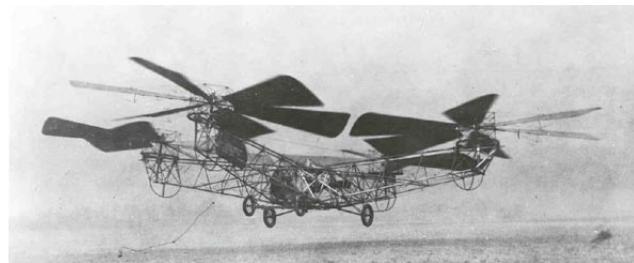


Figura 1.1: Cuadricóptero de Bothezat

Un europeo, el francés Étienne Oehmichen realizó experimentos durante los años 1923 y 1924 en los que consiguió un vuelo estacionario de cinco minutos de duración y un vuelo estacionario de siete minutos de duración elevando el aparato 10 metros sobre el suelo.

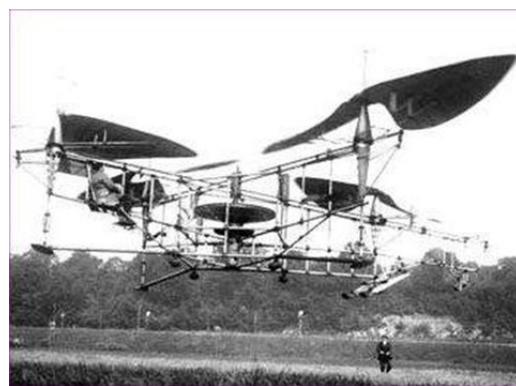


Figura 1.2: Cuadricóptero de Oehmichen

El trabajo de estos dos ingenieros se puede considerar el comienzo de los aparatos aéreos con cuatro motores.

Debido a la complejidad en el control de estos aparatos las investigaciones se centraron más en los helicópteros de una hélice que conocemos hoy en día. Sin embargo, a lo largo de la última década el uso de aparatos aéreos de reducidas dimensiones, como cuadricópteros y drones, se ha incrementado exponencialmente. En la actualidad es muy común su uso en diversos ámbitos, siendo los más destacables su uso militar, uso comercial y uso como método de grabación en la industria cinematográfica. En todos los casos el aparato más usado es un vehículo aéreo no tripulado (UAV) basado en un sistema de aterrizaje y despegue vertical (VTOL), categoría que engloba a los cuadricópteros. Las ventajas comunes a todos los ámbitos son su facilidad de uso, facilidad de transporte y su reducido coste. Los usos que se le dan en cada campo son:

- **Militar:** se utiliza como observador de la zona en la que van a operar los soldados sin poner en riesgo vidas humanas.
- **Comercial:** varias empresas, entre las que se incluye Amazon, están investigando su uso como medio para repartir paquetes en áreas urbanas.



Figura 1.3: Cuadricóptero de Amazon en pruebas

- **Industria cinematográfica:** su uso permite reducir los costes de producción en tomas aéreas. Las tomas que hasta ahora se tenían que realizar mediante un helicóptero se pueden realizar con los cuadricópteros a un coste mucho menor.



Figura 1.4: Cuadricóptero con cámara acoplada

Estas son las aplicaciones más conocidas de estos aparatos. No obstante, tienen otras muchas aplicaciones como revisión del estado de las líneas de alta y media tensión por parte de las compañías

eléctricas, revisión del estado de los aerogeneradores mediante un modelo llamado Aracnocopter, análisis del ambiente en zonas de alto riesgo para los humanos como zonas volcánicas...

## 1.2 Estado del arte

En este apartado se van a enumerar algunos trabajos de otros grupos de investigación en el campo de los cuadricópteros.

- En el trabajo de Mark Johnson \*\*\*[?] se realiza el diseño y el control de un cuadricóptero con vuelo autónomo y con variador del ángulo de inclinación de las hélices. Los aspectos más destacables de este proyecto son los servos añadidos a las hélices para variar el ángulo de éstas; además ha implementado un software que permite el vuelo invertido del aparato.

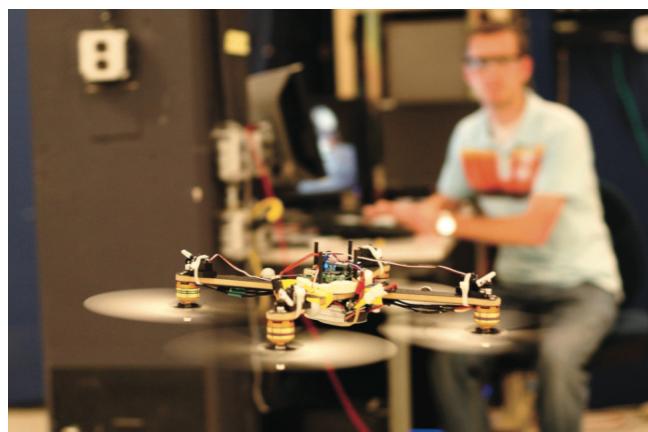


Figura 1.5: Detalle del vuelo invertido del cuadricóptero

\*\*\*Añadir otro trabajo

- **Instrumentación Electrónica** \*\*\* Indexar citas a la bibliografía

## 1.3 Motivación y objetivos del proyecto

### 1.3.1 Motivación

Debido a la mayor presencia de estos aparatos año tras año en múltiples aplicaciones este proyecto se va a centrar en estudiar el método de medida de la posición de un eje de un cuadricóptero y su posterior estabilización mediante distintos reguladores. Ya que la distribución de los motores en el cuadricóptero es simétrica, el algoritmo de control de estabilización del brazo con el que se va a trabajar se puede aplicar al otro brazo para estabilizar el vuelo del cuadricóptero; por lo tanto este proyecto no se va a centrar en aspectos mecánicos ni en la aerodinámica del aparato sino en la teoría de control y en la programación del controlador para procesar los datos de los sensores y usarlos para el algoritmo de estabilización.

Mediante la realización de este proyecto se afianzan conocimientos estudiados durante la carrera en los campos de:

- **Control:** se deben utilizar los conocimientos adquiridos en las distintas asignaturas de control para el control del brazo.

- **Programación:** se requieren conocimientos estudiados en asignaturas de programación y en sistemas digitales (para el manejo de registros de un microcontrolador) para programar el microcontrolador, que generalmente requieren el lenguaje C o C++.
- **Instrumentación Electrónica:** se utilizará el programa LabView aprendido en esta asignatura para realizar una interfaz gráfica que se comunique con el microcontrolador y permita controlar el brazo con más facilidad.

### 1.3.2 Objetivos

El proyecto se dará por concluido cuando se cumplan los siguientes objetivos:

1. Análisis y estimación de los 3 grados de libertad del cuadricóptero obtenidos de la IMU.
2. Diseño del PID y de otros reguladores para el control del ángulo del brazo.
3. Programación de la interfaz gráfica en LabView para manejar el brazo.

## 1.4 Medios y herramientas necesarios

En este apartado sólo se enumerarán los recursos necesarios para la realización del proyecto, más adelante, en otro apartado, se explicarán más en detalle las características de cada uno de los componentes y programas aquí enumerados. Para realizar este proyecto es necesario disponer de:

- |                                    |                                    |
|------------------------------------|------------------------------------|
| • Microcontrolador                 | • Motores Brushless                |
| • IMU                              | • Hélices                          |
| • LabView                          | • Controladores de velocidad (ESC) |
| • PC                               | • Estructura de ensamblaje         |
| • Batería o fuente de alimentación | • Matlab                           |

CAPITULO II

## **Elección de los componentes**



# Capítulo 2

## Elección de los componentes

En este apartado del trabajo se van a estudiar los distintos componentes presentes en el mercado y se van a elegir los más adecuados. De cada componente se comentarán sus especificaciones y sus características detalladamente.

### 2.1 Controlador

Este componente es el cerebro del cuadricóptero, es el encargado de recoger todos los datos de los sensores para procesarlos y generar las señales necesarias que se enviarán a los motores para estabilizar el brazo. Para esta aplicación no se requiere un procesador con mucha capacidad de cómputo, los requisitos necesarios para que el controlador sea válido son:

1. **Puerto serie:** es necesario para la comunicación con el sensor y con el ordenador. Mediante la lectura del puerto serie se leerán los datos de los sensores y mediante la escritura en el puerto serie se enviarán órdenes desde el programa realizado en el ordenador al controlador.
2. **Entradas/Salidas digitales con PWM:** la excitación de los motores se realiza mediante una señal PWM, la cual se envía a través de estos puertos.

Con estas características se puede buscar en Internet la oferta de controladores disponible que cumplan las características especificadas. Los dos controladores más completos y con mejor relación calidad/precio son el Arduino Uno y el A-Star 32U4 Prime SV, cuyas características se enumerarán a continuación.

#### 2.1.1 Arduino Uno

Esta placa está basada en un microcontrolador ATmega328 y dispone de 14 entradas/salidas digitales, de las cuales 6 pueden ser usadas como salidas PWM. Se conecta al ordenador mediante USB, que sustituye al antiguo puerto serie RS232 y dispone de puertos para la comunicación serie. Por lo tanto este microcontrolador cumple con los requisitos necesarios para poder realizar este trabajo.

Una de las ventajas de usar la plataforma Arduino es la enorme cantidad de documentación disponible en Internet para cualquier proyecto relacionado con Arduino. Ya que es una plataforma de código abierto hay una comunidad de desarrolladores que ha generado toda esa información y resulta muy sencillo aprender a programar el micro.

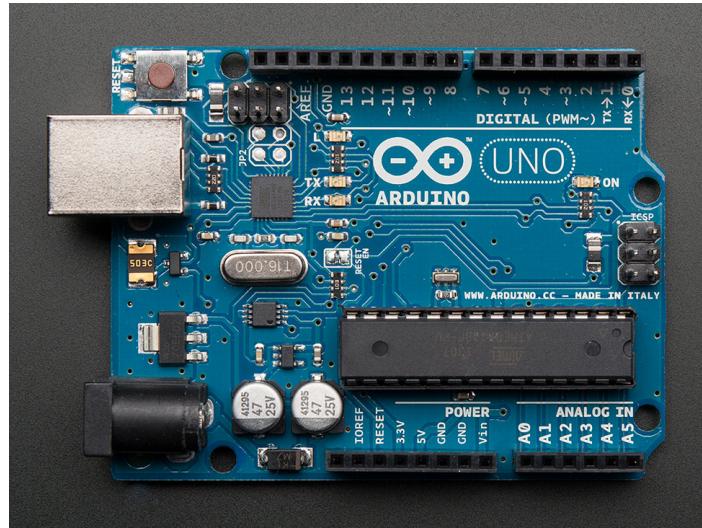


Figura 2.1: Microcontrolador Arduino Uno

### 2.1.2 A-Star 32U4 Prime SV

Esta placa está basada en un microcontrolador ATmega32U4 y dispone de 14 entradas/salidas digitales, de las cuales 7 pueden ser usadas como salidas PWM. Se conecta al ordenador mediante micro-USB, que sustituye al antiguo puerto serie RS232 y dispone de puertos para la comunicación serie. Por lo tanto este microcontrolador cumple con los requisitos necesarios para poder realizar este trabajo.

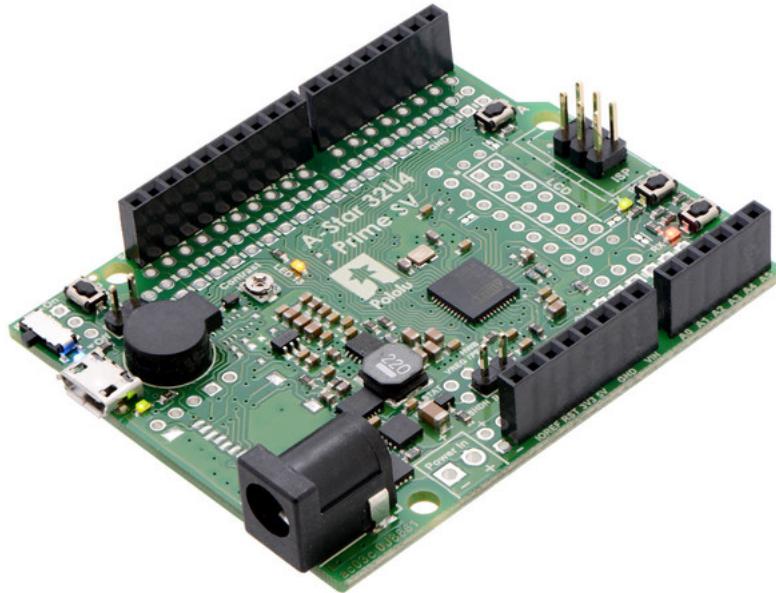


Figura 2.2: Microcontrolador A-Star 32U4 Prime SV

La ventaja de este microcontrolador respecto al Arduino es un mayor rango de voltaje de alimentación y mayor número de entradas/salidas disponibles para el usuario.

### 2.1.3 Comparativa y elección

Tabla 2.1: Comparativa controladores programables.

	<b>Arduino Uno</b>	<b>A-Star 32U4 Prime SV</b>
Microcontrolador	ATmega328P	ATmega32U4
Entradas/Salidas	20	26
Salidas PWM	6	7
Rango de tensión de entrada	7 a 12 V	5 a 36 V
Precio	20 €	28 €

Las mayores diferencias entre ambos controladores están en el número de entradas y en el margen de tensiones de entrada. En este trabajo sólo se van a necesitar 2 salidas PWM, por lo tanto cualquiera de los controladores son válidos. Las tensiones de entrada tampoco son relevantes a la hora de elegir una opción u otra ya que la oferta de baterías y fuentes de alimentación es muy amplia. La mayor disponibilidad de documentación disponible en la plataforma Arduino hace que el aprendizaje y el desarrollo sean más rápidos, por lo tanto es la mejor opción. El controlador usado será el **Arduino Uno**.

## 2.2 Unidad inercial de medidas (IMU)

La unidad inercial de medidas o IMU por sus siglas en inglés es un dispositivo que combina un conjunto de sensores capaces de medir la aceleración, la velocidad angular y el campo magnético. Los sensores capaces de medir esas 3 magnitudes son el acelerómetro, el giróscopo y el magnetómetro. En general, las IMUs disponibles en el mercado son capaces de medir estas magnitudes en las 3 direcciones del espacio pues integran 3 acelerómetros, 3 giróscopos y 3 magnetómetros. En capítulos posteriores se explicará detalladamente el funcionamiento del giróscopo y del acelerómetro, pues el magnetómetro no es usado en este trabajo. Esta apartado se centra en comparar los modelos más apropiados para este trabajo. Las dos IMU que se van a comparar son la Alt-IMU-10 de Pololu y la MPU-6050 de Invensense.

### 2.2.1 AltIMU-10

Esta IMU combina un barómetro digital, un giróscopo de 3 ejes, un acelerómetro de 3 ejes, un magnetómetro de 3 ejes y un altímetro. Por lo tanto es un dispositivo con 10 grados de libertad. La ventaja de elegir esta IMU es que el vendedor proporciona unas librerías mediante las que leer los valores de los sensores sin tener que recurrir a lecturas de registros; sólo hay que hacer llamadas a las funciones definidas en la librería y se leerán los datos de los sensores.

Esta IMU es compatible con el controlador elegido pues rango de tensiones de alimentación es 2.5 V a 5.5 V y la corriente necesaria 6 mA (cada pin del Arduino entrega hasta 40 mA de corriente). Además incluye un regulador de tensión de 3.3 V, cuando al pin de alimentación VIN se le alimenta a 3.3 V o más, el pin VDD actúa como una fuente de 3.3 V que puede suministrar hasta 150 mA a componentes externos. Los rangos de FS<sup>1</sup> se especificarán más adelante en la tabla comparativa.

---

<sup>1</sup>FS hace referencia a fondo de escala

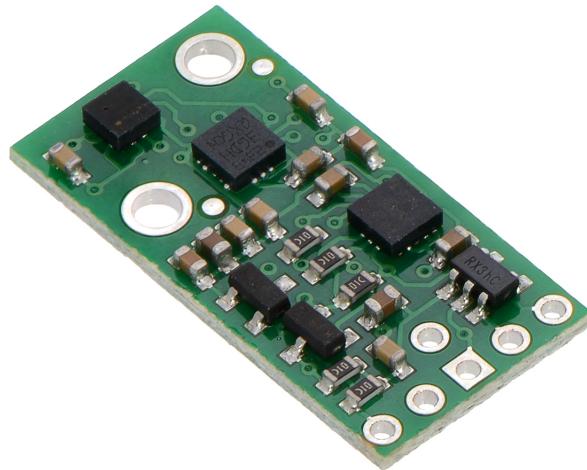


Figura 2.3: AltIMU-10

La lectura se realiza mediante comunicación I<sup>2</sup>C, que es un protocolo de comunicación serie. Tiene un conversor digital-analógico por canal, es decir uno para cada eje de cada sensor, característica que asegura medidas de mucha precisión.

### 2.2.2 MPU 6050

Este dispositivo combina un giroscopio de 3 ejes y un acelerómetro de 3 ejes. Es un dispositivo con 6 grados de libertad. En este caso el fabricante no proporciona librerías y las operaciones para leer datos de los sensores son algo más complejas, necesitando acceder a registros y realizar operaciones de desplazamiento de bits para obtener los datos de los sensores.

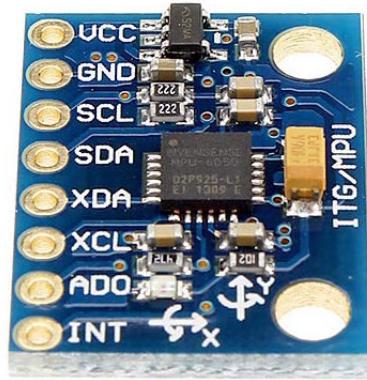


Figura 2.4: MPU 6050

También es compatible con Arduino pues su rango de tensión de alimentación es 2.375 V a 3.46 V. La lectura se realiza mediante comunicación I<sup>2</sup>C. Tiene un conversor digital-analógico por canal, es decir uno para cada eje de cada sensor, característica que asegura medidas de mucha precisión.

### 2.2.3 Comparativa y elección

Tabla 2.2: Comparativa IMU.

	<b>AltIMU-10</b>	<b>MPU 6050</b>
Rangos FS giróscopos	$\pm 245, \pm 500, \pm 2000^{\circ}/s$	$\pm 250, \pm 500, \pm 1000, \pm 2000^{\circ}/s$
Rangos FS acelerómetros	$\pm 2, \pm 4, \pm 6, \pm 8, \pm 16g$	$\pm 2, \pm 4, \pm 6, \pm 8, \pm 16g$
Rango de tensión de entrada	2.5 a 5.5 V	2.375 a 3.46 V
Precio	30 €	6 €

La única diferencia significativa es el precio. No obstante la mayor facilidad a la hora de programar para leer los valores de los sensores va a reducir el error significativamente y merece la pena esa diferencia de costes. La IMU que se usará en este trabajo es la **AltIMU-10**.

## 2.3 Motores

La elección de unos motores adecuados resulta imprescindible, pues de sus características dependerá la estabilización y el consumo de energía. Los dos tipos de motores entre los que hay que elegir son los motores con escobillas, que generalmente son motores de continua, o motores sin escobillas. Para analizar las diferencias entre los motores se ha utilizado información obtenida de \*\*\*[?].

### 2.3.1 Motores DC (Brushed Motors)

Estos motores están compuestos de dos partes, un estator que sirve como soporte mecánico y en el que se encuentran los polos, y un rotor que recibe la corriente de las escobillas. La función de las escobillas es la de cambiar la dirección de la corriente que circula por sus bobinas, esta conmutación produce un par de rotación en el eje. Estos motores no necesitan ningún tipo de controlador para variar su velocidad debido a la conmutación mecánica mediante escobillas, por lo que el control es simple.



Figura 2.5: Motor de corriente continua

El principal inconveniente de este tipo de motores es que debido al rozamiento del rotor con las escobillas se produce un desgaste debido al que hay sustituir las escobillas con mucha frecuencia. En consecuencia el coste de mantenimiento es elevado.

### 2.3.2 Motores sin escobillas (Brushless Motors)

Son similares a los motores de corriente continua en su estructura pues usan bobinas e imanes para mover el eje pero su funcionamiento es muy distinto. Con motores de alterna trifásicos y debido a que no hay ningún elemento mecánico como las escobillas que inviertan el sentido de la corriente es necesario para regular la velocidad de giro un controlador electrónico de velocidad (ESC). El ESC además funciona como inversor, convirtiendo la corriente continua suministrada por las baterías a corriente alterna necesaria para alimentar el motor.



Figura 2.6: Motor sin escobillas

De este tipo de motores existen dos posibles configuraciones, de rotor externo o de rotor interno. Su nombre indica la posición de las bobinas del respecto a los imanes permanentes.

### 2.3.3 Comparativa y elección

Tabla 2.3: Comparativa Motores.

	<b>Motor sin escobillas</b>	<b>Motor continua</b>
Mantenimiento	Bajo	Elevado
Eficiencia	Alta	Media
Comutación	Electrónica	Mecánica
Ruido electrónico	Bajo	Alto
Precio	15 €	7 €

Las ventajas de elegir un motor sin escobillas se observan en la tabla 1.3. Sin embargo esas no son todas las ventajas, al no haber escobillas no hay rozamiento y el rango de velocidad es más elevado al no tener limitación mecánica, también tienen mayor eficiencia pues la pérdida de calor es mucho menor y, como última consecuencia de no tener escobillas, el rendimiento es mayor, con la misma potencia eléctrica suministrada el motor sin escobillas adquiere mayor velocidad. Por todas estas ventajas, a pesar de la diferencia de coste, se va a utilizar el **Motor Brushless**. Por lo tanto el siguiente paso será elegir un controlador electrónico de velocidad.

Para estos tipos de motores el fabricante suele especificar, además de los parámetros comunes a los motores de continua, el factor Kv, que indica el número de revoluciones por minuto por cada voltio de tensión que se le aplica al motor. Ya que el peso que tienen que levantar los motores no es muy elevado,

no se requiere de un gran factor Kv ni de un gran par en los motores para este trabajo. En la tabla 2.4 se indican las especificaciones del motor elegido.

Tabla 2.4: Especificaciones motor.

Motor sin escobillas A2212	
Eficiencia Máxima	80 %
Kv	1000
Diámetro del eje	3.17mm
Tensión alimentación	7.4 a 14.8 V
Corriente sin carga	0.5 A
Corriente Máxima	12 A

## 2.4 Controlador electrónico de velocidad

Los motores brushless, como se ha mencionado en la sección 2.3, necesitan para regular su velocidad de giro mediante la conmutación de la sentido de la corriente un dispositivo llamado controlador electrónico de velocidad. Es un circuito electrónico utilizado para variar la dirección del motor, la velocidad e incluso para actuar como freno.

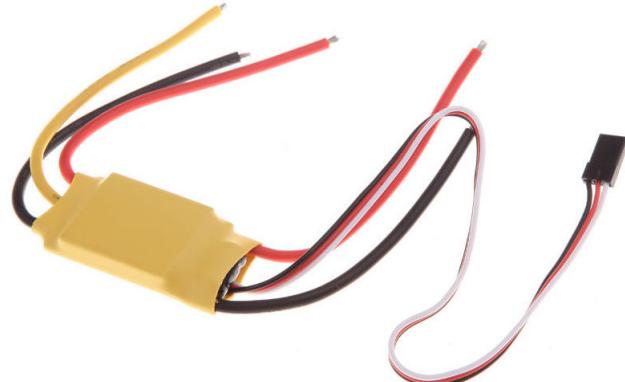


Figura 2.7: ESC.

Los distintos cables que se ven en la figura 2.7 corresponden a:

- **3 cables izquierda (Amarillo, rojo y negro):** cables de conexión al motor, cada uno corresponde a una fase.
- **2 cables derecha (Rojo y negro):** estos cables deben conectarse a la batería, son los cables de alimentación.
- **Conector de 3 cables derecho:** estos 3 cables son los cables de señal, el rojo y negro corresponden a Vcc y a masa y el blanco es el cable de señal.

Los controladores están regulados por una señal PPM (modulación por posición de pulso), que son similares a las señales PWM. En este tipo de señales la tensión aplicada en bornes del motor será el valor medio del tren que estará acotado entre 1 y 2 ms, es decir, con 1 ms el motor gira a la velocidad mínima

y con 2 ms el motor gira a la velocidad máxima. La frecuencia a la que se trabaja en este proyecto es de 33.33 Hz.

Un ejemplo de una señal de excitación para un ESC se muestra en la figura 2.8.

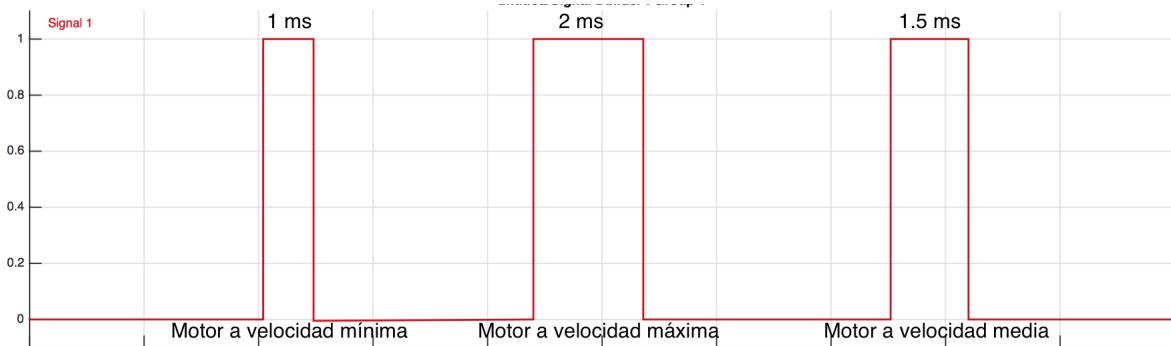


Figura 2.8: Señal ESC.

En esta señal se puede observar el rango de velocidades y el funcionamiento del motor en cada caso. Si se excita el motor a una velocidad menor de la mínima no arrancará y si se excita a una velocidad mayor de la máxima existe el riesgo de romper el motor. La forma de mandar estas señales al ESC se explicará en detalle en el capítulo IV.

Las características de las distintas opciones disponibles en el mercado son muy similares, por lo tanto se ha elegido un ESC que tiene una buena relación calidad/precio. Sus especificaciones se muestran en la tabla 2.5.

Tabla 2.5: Especificaciones ESC.

	<b>ESC</b>
Salida	30 A continuo, 40A hasta 10 secs
Voltaje de entrada	2-4 baterías LiPo

## 2.5 Hélices



Como el brazo sólo va a realizar un movimiento de rotación en un solo eje no se necesitan unas hélices muy grandes pues no es necesaria mucha fuerza de empuje. Por ello se han elegido unas hélices de unas dimensiones estándar que darán la fuerza de empuje necesaria para mover el brazo. Las dimensiones de las hélices son 200 mm de un extremo a otro y 6 mm el diámetro del eje.

## 2.6 Batería o fuente de alimentación

Las corrientes que necesitan los motores son bastante elevadas debido a la oposición de las bobinas a cambios en la corriente que circulan a través de ellas. Las baterías que se suelen utilizar en los cuadricópteros se denominan baterías LiPo, que son baterías de polímero de Litio. Cada una de estas baterías está compuesta por celdas en paralelo, cada celda suele tener un voltaje de 3.7 V y las ventajas que tienen es que son muy ligeras (conveniente para el cuadricóptero) y proporcionan grandes descargas de corriente.



Figura 2.9: Batería LiPo

Uno de los problemas de estas baterías es que si se descargan demasiado, es decir, la tensión es inferior a un valor facilitado por el fabricante, la batería deja de funcionar. Además si se les suministra demasiada corriente en la recarga pueden llegar a explotar. Ya que este trabajo está centrado en estabilizar un brazo que va a tener una base de apoyo se va a utilizar una fuente de alimentación, que son más seguras y la tensión proporcionada es más estable.



Figura 2.10: Fuente de alimentación utilizada

La fuente utilizada tiene dos fuentes independientes, una de 5 V y una de 12 V. En este trabajo se utilizará la fuente de 12 V. Las características de la fuente de alimentación se muestran en la tabla 2.6.

Tabla 2.6: Especificaciones fuente de alimentación.

	<b>5 V</b>	<b>12 V</b>
Salida	13 A (Pico)	6 A (Pico)
Entrada	180-240 V	180-240 V

## 2.7 Montaje final

Para montar el brazo se necesita un material que sea ligero y que sea fácil de trabajar, para que el montaje de la estructura no resulte excesivamente largo y complejo. El material usado en este trabajo es el aglomerado, que se obtiene a partir de virutas o serrín, pues su manipulación sólo requiere de una sierra de vaivén y se puede taladrar fácilmente para acoplar las distintas partes. El montaje del brazo debe tener suficiente peso y suficiente superficie en la base para que ante movimientos bruscos en la fase de pruebas del PID la estructura no se balancee, sólo debe moverse el brazo a estabilizar.



Figura 2.11: Brazo montado.

La estructura de la figura 2.11 tiene un eje central sobre el que puede girar libremente el brazo. Para que las hélices no pasen de un ángulo de aproximadamente  $75^\circ$  por cada lado, caso en el que chocarían contra las barras de soporte del brazo y sería peligroso, se ha colocado un tope de madera entre las barras para limitar el movimiento; además se limitará también por software para mayor seguridad. Para que el centro de gravedad del montaje descienda se ha colocado una pesa en la base y se disminuye el riesgo de volcar ante acelerones bruscos de los motores.

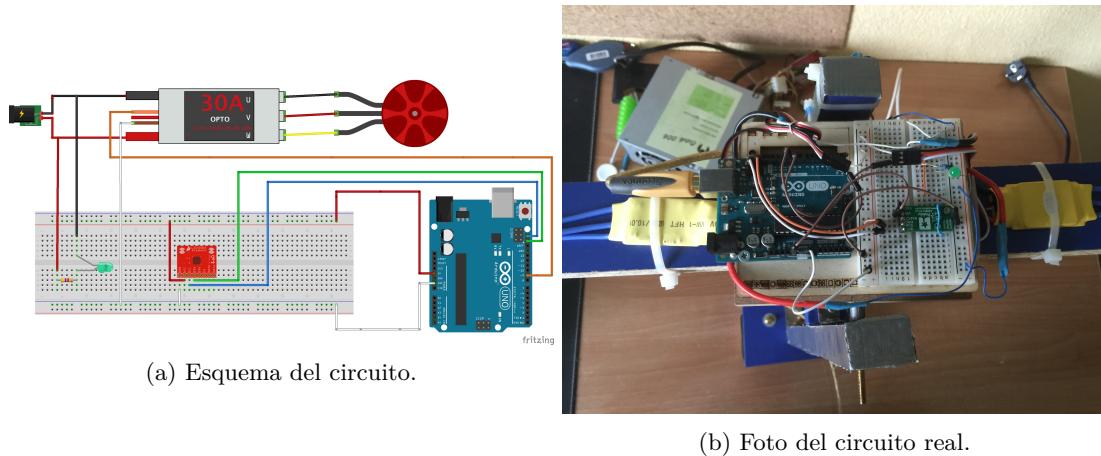


Figura 2.12: Imágenes del esquema y del circuito montado.

En la figura 2.12 se muestran el esquema del circuito montado y una imagen real del circuito. En el esquema sólo se ha representado uno de los motores porque la conexión del otro es idéntica.

CAPITULO III

## Sensores de posición



# Capítulo 3

## Sensores de posición

### 3.1 Introducción

Como ya se ha comentado previamente, en este trabajo se van a utilizar dos sensores para estimar el ángulo del brazo, el giróscopo y el acelerómetro. Ninguno de los dos sensores da una medida directa del ángulo, por lo tanto hay que realizar algunas cálculos para estimar el ángulo. En este capítulo se explicarán todos los pasos necesarios para llegar a obtener medidas precisas del ángulo del brazo, desde la conexión de los componentes hasta la realización de los programas.

Ambos sensores están integrados integrados en un chip denominado IMU. La IMU utilizada es la AltIMU-10 de Pololu e integra en el mismo chip un giróscopo de 3 ejes, un acelerómetro de 3 ejes, un magnetómetro de 3 ejes y un altímetro dando un total de 10 medidas independientes, lo que forma un sensor de 10 DOF<sup>1</sup>.

### 3.2 Conexión de la IMU

Para poder tomar medidas de la IMU hay que conectar 4 pines, los de alimentación y los de la comunicación serie. Los pines de la IMU se muestran en la figura 3.1. Los pines de alimentación son el pin Vin para el positivo y GND para masa. Se puede conectar a un rango de tensiones de entre 2.6 y 5.5 V. La corriente de suministro que necesita es 6 mA, cada pin de Arduino suministra 40 mA, por lo que ese requisito lo cumple holgadamente. Los pines de comunicación son SCL y SDA, que corresponden a la señal de reloj y a la línea de datos, respectivamente. El valor alto corresponde a una tensión Vin y el valor bajo corresponde a 0 V.

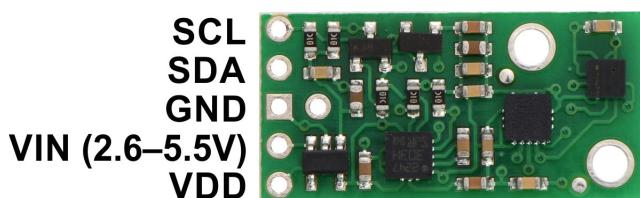


Figura 3.1: IMU con los nombres de los pines.

---

<sup>1</sup>Grados de libertad

El pin Vdd corresponde a una fuente de tensión de 3.3 V para otros componentes conectados al circuito, por lo que en este trabajo se dejarán al aire. El pin Vdd corresponde a una fuente de tensión de 3.3 V para otros componentes conectados al circuito, por lo que en este trabajo se dejarán al aire. En la figura 3.2 se muestra el esquema del circuito conectado.

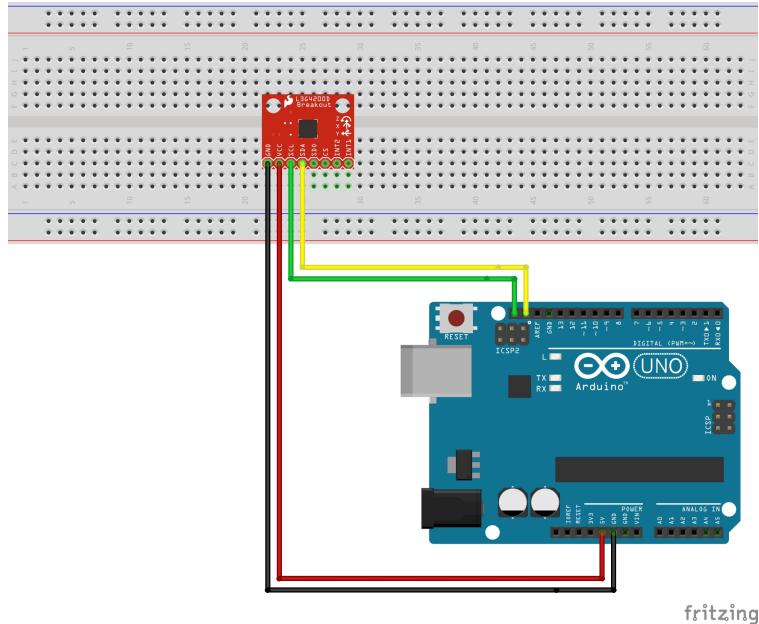


Figura 3.2: Esquema del circuito.

Uno de los aspectos a tener en cuenta para identificar correctamente el ángulo medido es el sentido de los ejes en la IMU y la dirección en que incrementan y decrementan cuando la IMU gira. Se explicará mediante la figura 3.3, que representa la conexión realizada en el brazo.

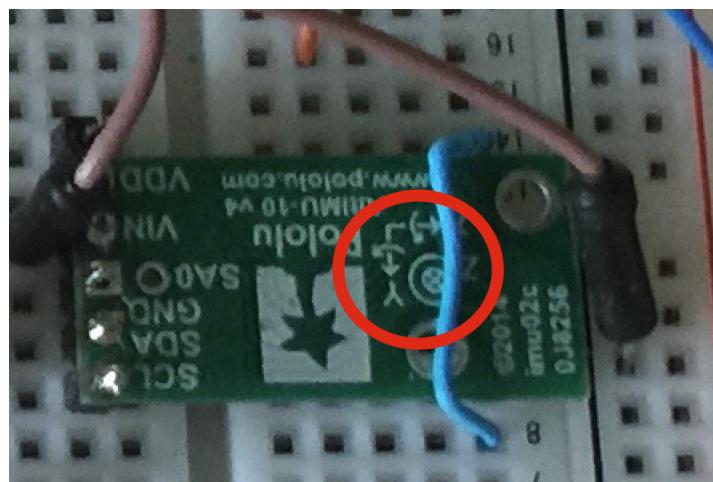


Figura 3.3: Conexión de la IMU.

Teniendo una vista global de la conexión completa en la figura 2.12, en esta vista en detalle se sabe que el eje que se va a medir es el eje Y. La flecha significa que en la dirección hacia la que apunta la flecha los valores del sensor crecen y en el sentido contrario decrecen.

### 3.3 Giróscopo

Es un sensor que sirve para medir la velocidad angular de un algún aparato o vehículo. En la figura 3.4 se observa su estructura. Para medir la velocidad de giro utiliza los principios del momento angular. Como se indica en \*\*\*[?], cuando el giróscopo se somete a un momento de fuerza que tiende a cambiar la orientación de su eje de rotación, éste cambia de orientación girando respecto a un tercer eje, perpendicular tanto a aquel respecto del cual se lo ha empujado a girar, como a su eje de rotación inicial. Si está montado sobre un soporte de Cardano<sup>2</sup> que minimiza cualquier momento angular externo, o simplemente gira libre en el espacio, el giroscópo conserva la orientación de su eje de rotación ante fuerzas externas que tiendan a desviarlo.

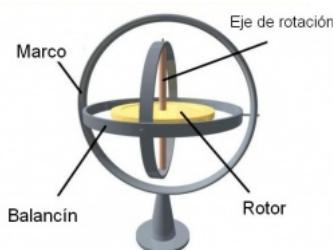


Figura 3.4: Giróscopo mecánico.

Debido a los tamaños con los que se trabaja hoy en día en el campo de la electrónica resulta imposible construir un giróscopo electrónico con la estructura de la figura 3.4, por lo que se suelen utilizar giróscopos denominados de estructura vibrante o de vibración de Coriolis. Como se indica en [?] funcionan como resultado de fuerzas de Coriolis desarrolladas que ocurren cuando una partícula experimenta movimiento lineal en un bastidor rotatorio de referencia.

Conociendo los tipos de giróscopos y su funcionamiento ya se puede pasar a realizar el programa para tomar medidas con el giróscopo utilizado en este trabajo. El giróscopo integrado en la IMU utilizada es el L3GD20H de la marca STMicroelectronics, un giróscopo de 3 ejes con salida digital. Se comunica mediante el bus de comunicación serie  $I^2C$ . Como ya se ha comentado previamente el fabricante del giróscopo adquirido provee una librería para que la lectura de datos de los sensores de la IMU resulte más sencilla. A continuación se muestran los pasos a seguir para poder empezar a leer datos del sensor:

1. **Incluir la librería:** para poder utilizar las funciones y clases de datos definidas en la librería hay que incluir en el proyecto mediante un include.
2. **Declarar una variable:** en la librería hay definida una clase de datos llamada L3G que será la que usaremos para leer datos del giróscopo, por lo tanto hay que declarar una variable de este tipo para llamar a las funciones de la librería.
3. **Inicializar:** el giróscopo por defecto está configurado en modo "sleep"; hay que inicializarlo para poder empezar a leer datos. Además para poder comunicarse con Arduino hay que inicializar la comunicación serie.
4. **Lectura datos:** con los pasos previos ya realizados se pueden empezar a leer datos con la función `read` definida en la librería.

<sup>2</sup>Es un mecanismo de suspensión consistente en dos aros concéntricos cuyos ejes forman un ángulo recto.

Un ejemplo de un programa mediante el que se podrían leer datos del giróscopo se muestra en el código 3.1. En él se han programado todos los pasos arriba mencionados para tomar medidas del giróscopo.

Código 3.1: Ejemplo de programa para tomar medidas del giróscopo.

```

1 #include <Wire.h>           //Librería para la comunicación serie
2 #include <L3G.h>            //Librería del giróscopo
3
4 L3G gyro;                  //Declaración variable del tipo L3G
5
6 void setup()
7 {
8     Serial.begin(115200);      //Inicia el puerto serie
9     Wire.begin();             //Comunicación I2C entre IMU y Arduino
10
11    if (!gyro.init())         //Inicialización del giróscopo
12    {
13        Serial.println("Failed to autodetect gyro type!");
14        while (1);
15    }
16
17    gyro.enableDefault(); //Habilita varios registros a su valor por defecto
18
19 }
20
21 void loop()
22 {
23     gyro.read();
24 }
```

Estos valores leídos del sensor y almacenados en la variable gyro son valores “raw”, es decir, es un entero de 16 bits, no la velocidad angular. Para obtener la velocidad angular hay que realizar una serie de operaciones especificadas por el fabricante en la hoja de características del sensor. Para obtener valores de velocidad angular se debe multiplicar el valor “raw” obtenido por un factor de conversión llamado sensibilidad, cuyas unidades son mdps<sup>3</sup>/digit. Los valores de estas constantes para cada FS se encuentran en \*\*\*[?]. El objetivo final no es obtener la velocidad angular sino la posición, que es el ángulo. Como la velocidad es la derivada de la posición respecto del tiempo la solución es aplicar la ecuación 3.1 para obtener la posición.

$$\text{angulo} = \int \text{dps\_y} dt \quad (3.1)$$

Para poder programar esa integral hay que controlar el tiempo que tarda en ejecutarse el bucle principal, que debe ser siempre el mismo. Se debe medir el tiempo al inicio del bucle y al final del bucle para asegurarse de que en ninguna iteración se sobrepasa el tiempo de ejecución y si está por debajo introducir un delay. Para medir el tiempo transcurrido se puede utilizar la función “millis()“.

Código 3.2: Ejemplo para obtener una medida de la posición.

```

1 #include <Wire.h>
2 #include <L3G.h>
3
4 #define FACTOR_CONVERSION_245DPS 0.00875
5
```

<sup>3</sup>mdps hace referencia a milidegrees per second (miligrados por segundo)

```

6 #define DT 0.03           //Tiempo en segundos
7
8 int16_t dps_y;
9 unsigned long time_loop1, time_loop2; //Variables para medir el tiempo de ejecución
10 double angulo_giroscopio_y;
11
12 ...
13
14 void loop()
15 {
16
17     time_loop1 = millis();
18     gyro.read();
19
20     dps_y = (gyro.g.y)*FACTOR_CONVERSION_245PS;
21     angulo_giroscopio_y += (double) dps_y*DT;      //La integral se realiza mediante una
22     multiplicación
23
24     time_loop2 = millis();
25
26     while(time_loop2-time_loop1 < DT)
27     {
28         delay(1);
29     }
}

```

Con el código 3.2 ya se obtiene una estimación del ángulo del brazo, sin embargo esa medida tiene fuentes de error que hacen que la precisión de la medida sea muy baja. Hay métodos para minimizar estos errores pero no se pueden eliminar completamente. A continuación se comentan estos errores y cómo reducirlos.

### 3.3.1 Fuentes de error

#### 3.3.1.1 Offset

Este error se refiere al valor que dan los sensores si el brazo está parado, es decir, la velocidad angular es 0. Para corregirlo basta con estimarlo y restárselo a cada lectura.

Código 3.3: Eliminación del error de offset del giróscopo.

```

1 #include <Wire.h>
2 #include <L3G.h>
3
4 #define FACTOR_CONVERSION_245DPS 0.00875
5
6 #define DT 0.03           //Tiempo en segundos
7
8 int16_t dps_y;
9 int sampleNum = 500, offsetg_y = 0;;
10
11
12 void setup()
13 {
14
15     for(int n=0;n<sampleNum;n++)
16

```

```

17     gyro.read();
18     offsetg_y+=(int)gyro.g.y;
19 }
20 offsetg_y = offsetg_y/sampleNum;
21 }
22 }
23
24
25 void loop()
26 {
27     gyro.read();
28     dps_y = (gyro.g.y-offsetg_y)*FACTOR_CONVERSION_245PS;
29     ...
30 }
```

Mediante el código 3.3 se obtiene la media del offset en sampleNum muestras y se consigue hacer despreciable el error de offset.

### 3.3.1.2 Deriva (Drift)

Para obtener la estimación del ángulo hay que realizar una integral, esto significa que cada iteración se suma un nuevo valor a la variable en la que se almacena el ángulo. Esta característica de la medida del ángulo provoca que si en cada medida hay un error pequeño se va a ir acumulando y tras varios segundos de ejecución de programa las medidas no van a ser válidas. En el caso del programa 3.2 se suma ese error cada 30 ms, lo que hace que tras 10 segundos de ejecución se haya sumado ese error 333 veces.

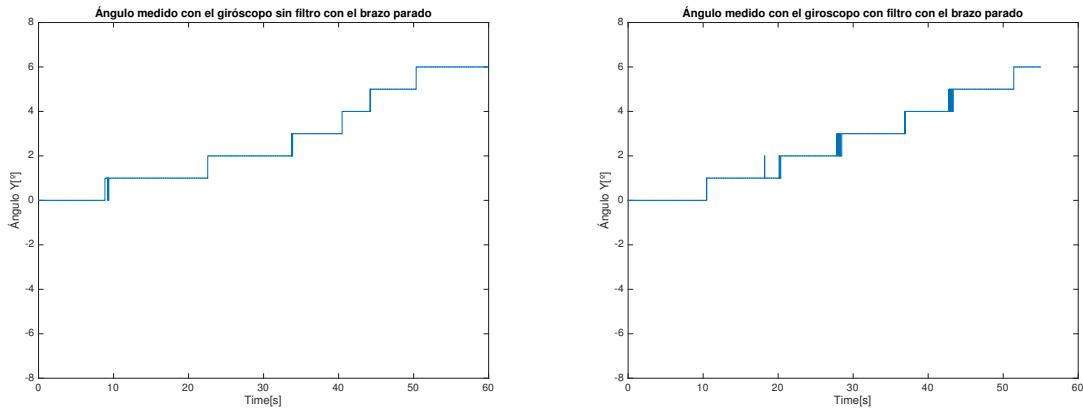
La única manera de reducir este error usando solo el giróscopo son los filtros paso-alto que incluye el modelo de giróscopo utilizado. Mediante este filtro se eliminan de la medida las muestras que tengan un valor muy pequeño como es el error por deriva. A pesar de utilizar este filtro no se consigue una medida válida de la posición del brazo; es necesario utilizar el acelerómetro para que la medida del ángulo sea precisa. La manera en la que se combinan los valores de los dos sensores se explicará en la sección 3.5.

### 3.3.1.3 Precisión

Además de los dos errores ya comentados hay otra característica del giróscopo, que no se puede considerar error pero sí limita la estimación del ángulo, y es que como este sensor mide la velocidad angular y ésta se integra la precisión de la medida es muy mala para movimientos lentos del brazo. Si la velocidad angular es muy baja el término que se va a sumar al ángulo es muy pequeño comparado con la suma total y no se va a percibir.

## 3.3.2 Medidas

En este apartado se van a mostrar ejemplos de medidas tomadas con los programas escritos en la sección 3.3. Las pruebas realizadas consisten en medir el ángulo con estos programas para poder observar las fuentes de error descritas en la sección 3.3.1. La primera prueba consiste en ejecutar el programa y dejar el brazo en reposo para comprobar si se produce error por deriva. La segunda prueba consiste en mover el brazo un ángulo conocido para comprobar la precisión de la medida.



(a) Angulo medido con el giróscopo sin filtro.

(b) Angulo medido con el giróscopo con filtro.

Figura 3.5: Resultados de la medida del ángulo con el giróscopo.

Se puede observar perfectamente el error de deriva, en menos de un minuto el error en la medida es de  $7^\circ$ , que es un error intolerable si se pretende estabilizar el brazo. Además en el caso del filtro esos picos que se producen como el del segundo 20 son correcciones del filtro ante el pequeño incremento, pero la utilización del filtro no es suficiente para obtener una medida adecuada.

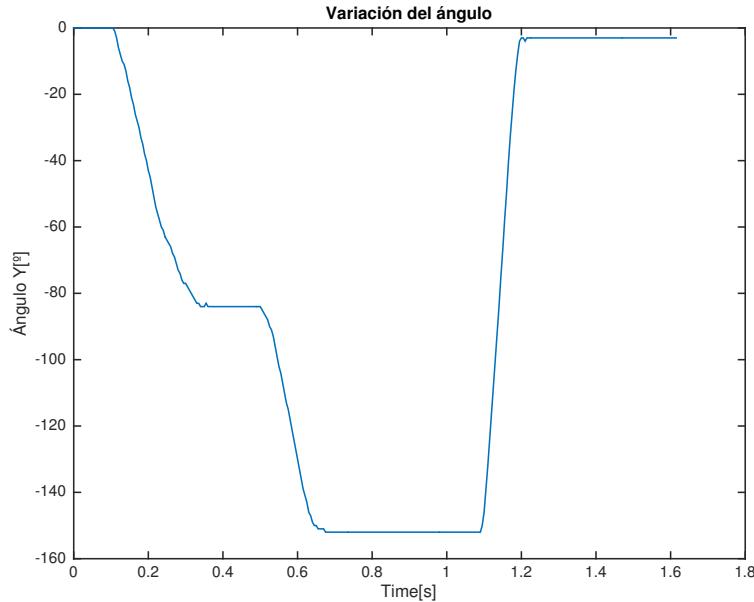


Figura 3.6: Ángulo medido con el giróscopo variando el ángulo.

La variación máxima real del ángulo es de  $110^\circ$  y después se ha vuelto a la posición inicial. El ángulo medido por el giróscopo es de  $140^\circ$ . Este error se debe a la baja precisión del giróscopo y a su principio de medida, que no le permite medir ángulos absolutos sino variaciones del ángulo.

### 3.4 Acelerómetro

Es cualquier instrumento destinado a medir las fuerzas de aceleración. Como se indica en \*\*\*[?], hay muchas maneras diferentes de fabricar un acelerómetro. Algunos acelerómetros usan el efecto piezo-resistivo (contienen cristales microscópicos que se estiran debido a la aceleración y generan una tensión), otro tipo de tecnología es medir los cambios en la capacidad. Este último es el caso del acelerómetro utilizado, que utiliza la tecnología de fabricación MEMS<sup>4</sup>. Estos acelerómetros están compuestos de una masa móvil con unas placas que están unidas mediante un sistema de suspensión mecánica a un marco de referencia, como se muestra en la figura 3.7. Las placas móviles y las placas exteriores fijas forman condensadores. La desviación de la masa se mide usando la diferencia de capacidades.

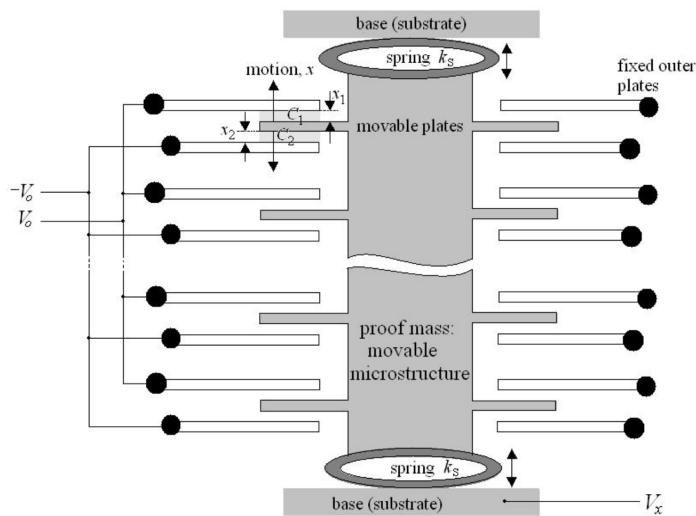


Figura 3.7: Acelerómetro de estructura vibrante.

Conociendo los tipos de acelerómetros y su funcionamiento ya se puede pasar a realizar el programa para tomar medidas con el modelo del acelerómetro utilizado en este trabajo. El modelo integrado en la IMU utilizada es el LSM303D de la marca STMicroelectronics, un acelerómetro de 3 ejes con salida digital. Se comunica mediante el bus de comunicación serie I<sup>2</sup>C. Como ya se ha comentado previamente el fabricante del acelerómetro adquirido provee una librería para que la lectura de datos de los sensores de la IMU resulte más sencilla. A continuación se muestran los pasos a seguir para poder empezar a leer datos del sensor:

1. **Incluir la librería:** para poder utilizar las funciones y clases de datos definidas en la librería hay que incluir en el proyecto mediante un include.
2. **Declarar una variable:** en la librería hay definida una clase de datos llamada LSM303 que será la que usaremos para leer datos del acelerómetro, por lo tanto hay que declarar una variable de este tipo para llamar a las funciones de la librería.
3. **Inicializar:** el acelerómetro por defecto está configurado en modo "sleep"; hay que inicializarlo para poder empezar a leer datos. Además para poder comunicarse con Arduino hay que inicializar la comunicación serie.
4. **Lectura datos:** con los pasos previos ya realizados se pueden empezar a leer datos con la función *read* definida en la librería.

<sup>4</sup>Sistemas Microelectromecánicos

Un ejemplo de un programa mediante el que se podrían leer datos del acelerómetro se muestra en el código 3.4.

Código 3.4: Ejemplo de programa para tomar medidas del acelerómetro.

```

1 #include <Wire.h>           //Librería para la comunicación serie
2 #include <LSM303.h>         //Librería del acelerómetro
3
4 LSM303 compass;            //Declaración variable del tipo LSM303
5
6 void setup()
7 {
8     Serial.begin(115200);    //Inicia el puerto serie
9     Wire.begin();            //Comunicación I2C entre IMU y Arduino
10
11    compass.init();          //Inicialización del giróscopo
12    compass.enableDefault(); //Habilita varios registros a su valor por defecto
13 }
14
15
16 void loop()
17 {
18     compass.read();
19 }
```

Estos valores leídos del sensor y almacenados en la variable *compass* son valores “raw”, es decir, es un entero de 16 bits, no las fuerzas debidas a la aceleración. Para obtener la fuerza debida a la aceleración hay que realizar una serie de operaciones especificadas por el fabricante en la hoja de características del sensor. Para obtener valores de fuerza, medidos en g, se debe multiplicar el valor “raw” obtenido por un factor de conversión llamado sensibilidad de aceleración lineal, cuyas unidades son mg<sup>5</sup>/LSB. Los valores de estas constantes para cada FS se encuentran en \*\*\*[?]. El procedimiento para obtener las fuerzas de aceleración es idéntico al que se ha llevado a cabo para obtener la velocidad angular con el giróscopo. Sin embargo, el objetivo final no es obtener las fuerzas de aceleración sino la posición, que es el ángulo. El ángulo se obtiene mediante los ángulos de Euler, que consiste en resolver unas matrices de rotación cuya solución son los ángulos de inclinación de un sólido rígido. La resolución de estas matrices se ha consultado en el documento del fabricante Freescale Semiconductor \*\*\*[?]. La ecuación usada para estimar el ángulo es la ecuación 3.2, que en el documento equivale a la ecuación 37:

\*\*\*\*APARECE UNA INTERROGACION EN LA ECUACIÓN

$$\tan(\text{angulo_acelerometro}_y) = \frac{-fuerza_x}{\sqrt{fuerza_y^2 + fuerza_z^2}} ?? \quad (3.2)$$

El resultado está en radianes, así que hay que pasarlo a grados con una constante que se llamará *RAD\_TO\_DEG* y cuyo valor es  $\frac{180}{2\pi} = 57,2958$ . Mediante este método sólo se obtienen ángulos entre -90° y +90°. Dado que el brazo sólo puede girar en un rango de entre -75° y +75°, el rango de medidas mediante este método es válido para este trabajo.

El programa para implementar esa ecuación es más sencillo que el programa necesario para estimar los ángulos mediante el giróscopo, pues en el caso del acelerómetro sólo hay que implementar una multiplicación. El código 3.5 implementa todos los pasos necesarios para estimar el ángulo mediante el acelerómetro.

<sup>5</sup>mg hace referencia a miligauss

Código 3.5: Estimación del ángulo mediante el acelerómetro.

```

1 #include <Wire.h>           //Librería para la comunicación serie
2 #include <LSM303.h>          //Librería del acelerómetro
3
4 #define RAD_TO_DEG 57.2958
5 #define FACTOR_CONVERSION_2G 0.000061
6
7
8 LSM303 compass;           //Declaración variable del tipo LSM303
9 double fuerza_x, fuerza_y, fuerza_z;
10 double angulo_acelerometro_y;
11 ...
12
13
14 void loop()
15 {
16     compass.read();
17
18     fuerza_x = -((compass.a.x)*FACTOR_CONVERSION_2G); //El signo - es debido a que la IMU está
19     invertida
20     fuerza_y = -((compass.a.y)*FACTOR_CONVERSION_2G);
21     fuerza_z = -((compass.a.z)*FACTOR_CONVERSION_2G);
22
23     raiz_y = sqrt((fuerza_y*fuerza_y)+(fuerza_z*fuerza_z));
24
25     angulo_acelerometro_y = (atan2(-fuerza_x,raiz_y))*RAD_TO_DEG;
}

```

Con el código 3.5 ya se obtiene una estimación del ángulo del brazo, sin embargo esa medida tiene fuentes de error que hacen que la precisión de la medida sea muy baja. Hay métodos para minimizar estos errores pero no se pueden eliminar completamente. A continuación se comentan estos errores y cómo reducirlos.

### 3.4.1 Fuentes de error

#### 3.4.1.1 Offset

Este error se refiere al valor que dan los sensores si la fuerza en un eje es 0. Para corregirlo basta con estimarlo y restárselo a cada lectura.

Código 3.6: Eliminación del error de offset del acelerómetro.

```

1 #include <Wire.h>
2 #include <LSM303.h>
3
4 #define FACTOR_CONVERSION_2G 0.000061
5
6 double fuerza_y;
7 int sampleNum = 500, offset_y = 0;;
8
9
10 void setup()
11 {
12
13     for(int n=0;n<sampleNum;n++) {

```

```

15     compass.read();
16     offsetg_y+=(int)compass.a.y;
17 }
18 offset_y = offset_y/sampleNum;
19
20 }
21
22
23 void loop()
24 {
25     compass.read();
26     fuerza_y = -((compass.a.y-offseta_y)*FACTOR_CONVERSION_2G);
27     ...
28 }
```

Mediante el código 3.6 se obtiene la media del offset en sampleNum muestras y se consigue hacer despreciable el error de offset restando el valor de offset al valor de cada medida.

#### 3.4.1.2 Ruido

En el caso del giróscopo se necesitaba movimiento del aparato en el que está acoplado el sensor para poder medir el ángulo, sin embargo con el acelerómetro se puede medir el ángulo en cualquier momento. El problema es que las medidas del acelerómetro introducen un ruido de baja frecuencia que provoca que no se pueda medir el ángulo con precisión en un corto período de tiempo. Las medidas del acelerómetro son útiles sólo para medir ángulos en un largo período de tiempo y de variaciones lentas. No hay manera de eliminar este error si sólo se usa para medir el acelerómetro. Para ello se combinan las medidas del acelerómetro y del giróscopo utilizando filtros.

#### 3.4.2 Medidas

En este apartado se van a mostrar ejemplos de medidas tomadas con los programas escritos en la sección 3.4. Las prueba realizada consiste en medir el ángulo con estos programas para poder observar las fuentes de error descritas en la sección 3.4.1.

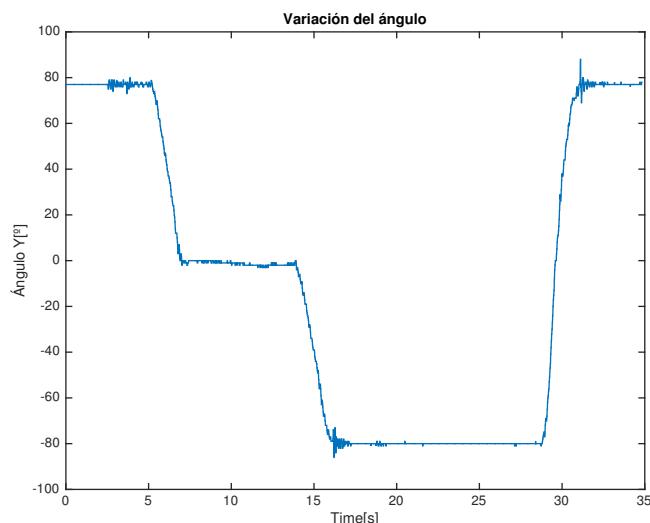


Figura 3.8: Resultados de la medida del ángulo con el acelerómetro.

Esta prueba realizada es similar a la realizada al medir con el giróscopo y variar el ángulo. Como se ve en la gráfica 3.8, al contrario de lo que sucedía con el giróscopo (cuyo punto de referencia estaba donde empezaba el brazo), los resultados ahora son de ángulos absolutos entre  $.90^\circ$  y  $+90^\circ$ , siendo  $0^\circ$  el ángulo en el que el brazo está paralelo al suelo. También se aprecia en la gráfica el error debido al ruido, cuando se produce un cambio brusco del ángulo de inclinación el ruido que aparece en las medidas es considerable. Ante cambios lentos del ángulo la precisión es elevada, pero es debido a las posibles variaciones rápidas del ángulo que se debe usar el giróscopo en conjunto. Para cuantificar mejor si ese ruido que aparece es importante se muestra en la figura 3.13 una ampliación de la anterior imagen.

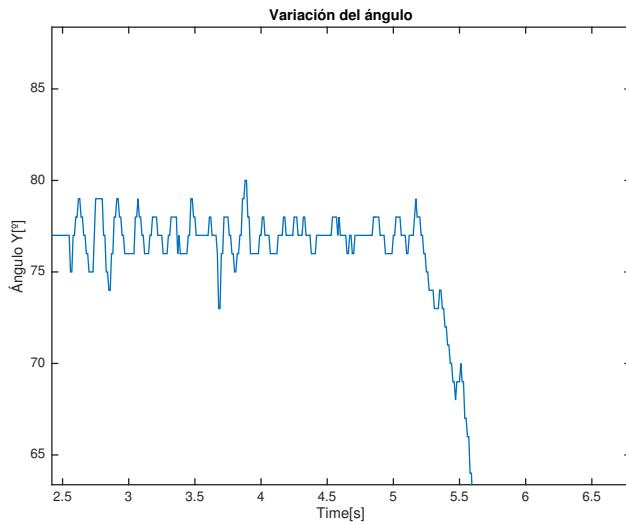


Figura 3.9: Ruido en la medida del ángulo con el acelerómetro.

Las variaciones del ángulo son de hasta  $5^\circ$  para un ángulo medido de  $75^\circ$ , si el ángulo fuese más pequeño, por ejemplo  $10^\circ$ , el error sería mucho mayor y correspondería a un 50 % de su valor. Con este error resultaría muy complejo estabilizar el brazo, es necesario el uso de filtros que cobinen adecuadamente las medidas del acelerómetro y del giróscopo. Los distintos tipos de filtros se explicarán en la sección 3.5.

## 3.5 Filtros

Para que la medida del ángulo tenga una precisión suficiente para poder estabilizar el brazo es necesario combinar las medidas de los dos sensores explicados hasta ahora, el acelerómetro y el giróscopo. Esa es la utilidad de los filtros, mediante un algoritmo combina las medidas de ambos sensores para reducir o eliminar los errores típicos de cada sensor y dar una medida de alta precisión del ángulo. Los dos filtros más comunes y que mejor funcionan son el filtro de Kalman y el filtro complementario.

### 3.5.1 Filtro de Kalman

Este es el filtro más eficaz pero también el más complejo de implementar. Es un filtro recursivo que predice el error y lo corrige teniendo únicamente como datos las mediciones de entrada actuales, el estado calculado previamente y su matriz de incertidumbre. Para implementarlo se utiliza el control en el espacio de estados.

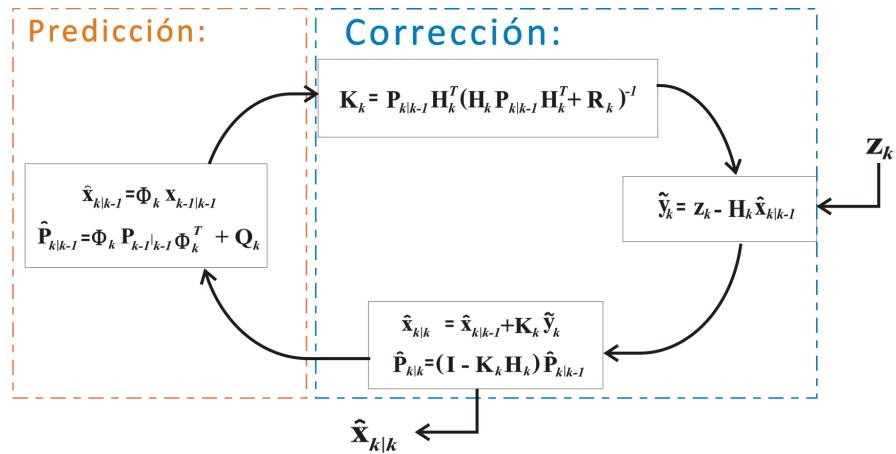


Figura 3.10: Algoritmo recursivo del filtro de Kalman.

Este filtro, debido a que trabaja con matrices, requiere mucha potencia de cálculo y el Arduino Uno no tiene un procesador muy potente.

### 3.5.2 Filtro complementario

El concepto del filtro complementario consiste en tomar las variaciones lentas del acelerómetro y las variaciones bruscas del giróscopo y combinarlas. El acelerómetro da buenas medidas de la posición en condiciones estáticas y el giróscopo las da en condiciones dinámicas, es decir, cuando el brazo se mueve rápido. La implementación de este filtro se realiza pasando las medidas del acelerómetro por un filtro paso-bajo y las medidas del giróscopo por un filtro paso-bajo y combinar el resultado para obtener la posición final. La implementación se realiza con una línea de código en el bucle principal, mostrada en el código 3.7.

Código 3.7: Línea de código para implementar el filtro complementario.

```
1 angulo_y = 0.95*(angulo_y + dps_y*DT) + 0.05*angulo_acelerometro_y;
```

Teniendo los valores de los ángulos medidos con el giróscopo y el acelerómetro se introduce esa línea de código en el bucle principal y se obtiene una medida precisa del ángulo. Los valores de los filtros se pueden ajustar experimentalmente para obtener los mejores resultados.

Dado que la implementación de este filtro es muy simple y los resultados son satisfactorias para la obtención del ángulo del brazo se va a utilizar el **filtro complementario**.

### 3.5.3 Medidas

En este apartado se va a mostrar cómo los errores mostrados en los apartados del acelerómetro y del giróscopo se eliminan mediante el uso del filtro complementario. Los distintos experimentos realizados mostrarán la efectividad del filtro y la precisión en las medidas del ángulo del brazo.

El primer experimento consiste en dejar el brazo en reposo y registrar los ángulos del giróscopo y del acelerómetro y después obtener el del filtro complementario. El propósito de esta prueba es observar si el filtro corrige la deriva del giróscopo y si la medición del ángulo es correcta.

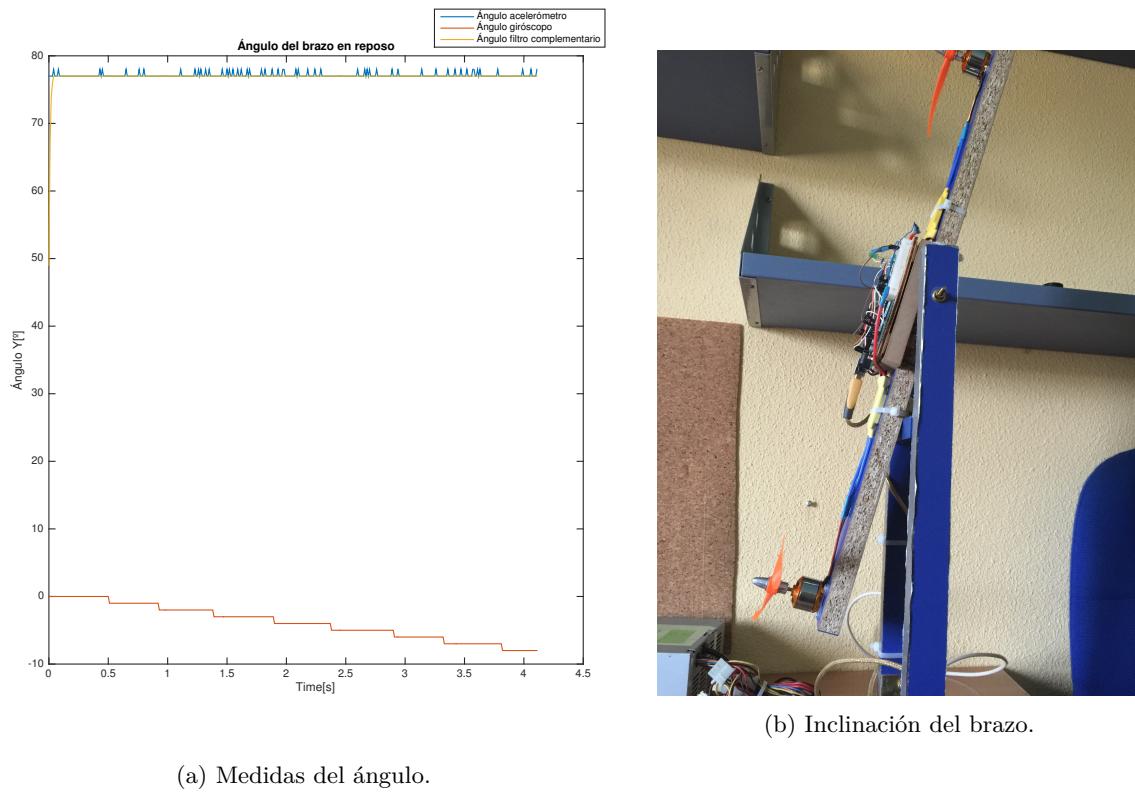


Figura 3.11: Resultados de la medida del ángulo con el giróscopo.

La figura 3.11 muestra cómo en la medida del filtro no hay deriva y además da una medida absoluta y precisa del ángulo. Cuando la variación del ángulo es pequeña el filtro paso-alto aplicado al giróscopo desprecia la medida del giróscopo , teniendo en cuenta únicamente la medida del acelerómetro.

El segundo experimento consiste en mover el brazo manualmente para comprobar los resultados de la medida del giróscopo, el acelerómetro y la combinación de ambos con el filtro complementario.

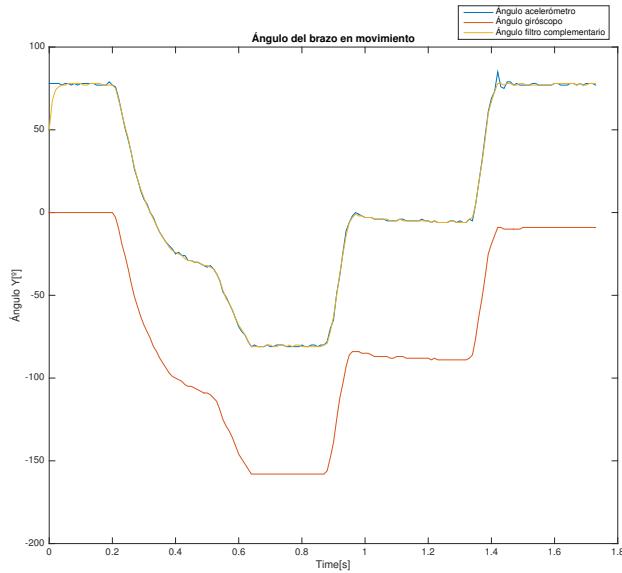
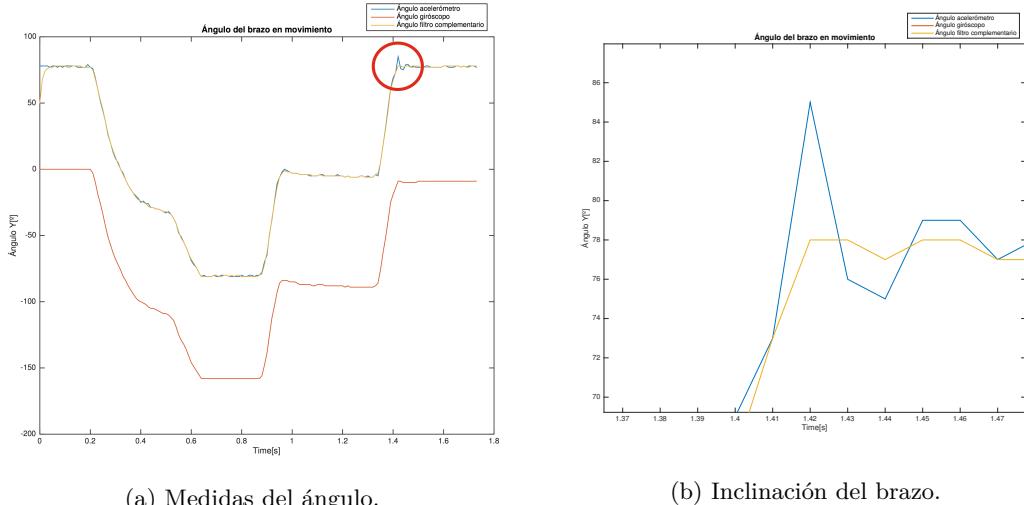


Figura 3.12: Medida del ángulo moviendo el brazo.

En la figura 3.12 se observa cómo el resultado obtenido con el filtro complementario sigue a las medidas dadas por el acelerómetro pero filtrando el ruido. Además también se comprueba que el giróscopo mide correctamente cuando el brazo se está moviendo, aunque no sea en valores absolutos las variaciones las mide correctamente y no aparece deriva al no estar el brazo en reposo durante varios segundos.



(a) Medidas del ángulo.

(b) Inclinación del brazo.

Figura 3.13: Resultados de la medida del ángulo con el giróscopo.

Para observar el ruido se ha aumentado la imagen 3.12. Se comprueba cómo para un ángulo real del brazo de  $78^\circ$  el ruido que introduce el acelerómetro provoca que éste mida  $86^\circ$ , siendo un error considerable.

En este apartado se ha mostrado el principio de funcionamiento de cada sensor y se han enumerado los errores que tiene cada uno al estimar el ángulo del brazo. Se ha mostrado experimentalmente la eficacia del filtro complementario, que corrige los errores típicos de cada filtro con una sola línea de código y requiriendo poco tiempo de cómputo, lo que hace que sea una buena opción utilizar este filtro en aplicaciones en las que no se disponga de un procesador muy potente.



CAPITULO IV

## Motores y ESC



# Capítulo 4

## Motores y ESC

### 4.1 Introducción

Los motores acoplados a los extremos del brazo van a ser los actuadores mediante los que se va a estabilizar el ángulo. Ya que el brazo sólo puede moverse en dos direcciones de un solo eje se necesitan dos motores. Ya se ha explicado en el apartado 2.6 la razón de utilizar los controladores electrónicos de velocidad (ESC), en este apartado se explicará cuál debe ser la conexión entre los motores y el ESC y entre éstos y el Arduino.

### 4.2 Conexión de los motores y los ESC

El esquema de la conexión entre batería, ESC, Arduino y los motores se muestran en la figura 4.1.

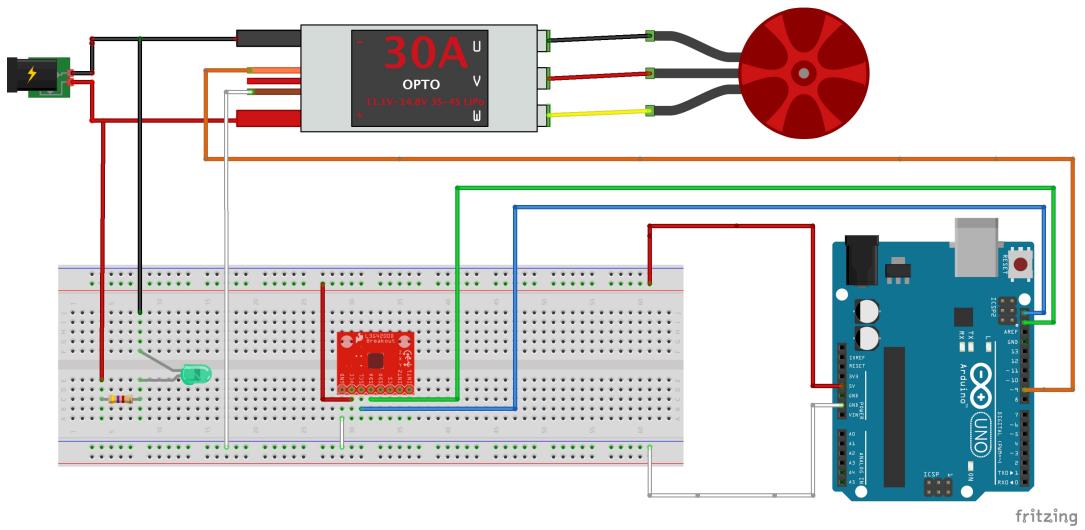


Figura 4.1: Esquema de conexión motores.

Sólo se muestra un motor porque la conexión del otro es idéntica. Como los motores giran a muchas revoluciones y las hélices tienen una sección muy fina son peligrosas cuando están en movimiento. Para asegurarse de que no les está llegando corriente a los motores y no van a empezar a girar se ha conectado un LED verde que se encenderá cuando se conecte el interruptor de la batería. La señal de control del ESC debe llevarse a una salida digital y la masa de los pines de control del ESC debe conectarse al

pin GND del Arduino. **Nunca se debe conectar el pin rojo de la señal de control al pin de 5 V del Arduino.** Como se indica en [?] no hay protección frente en ese pin del Arduino. Este pin está conectado directamente al microcontrolador ATmega328, la interfaz de USB del microcontrolador y al regulador de 5 V, siendo la tensión máxima soportada por todos ellos 6 V.

Antes de conectar los motores hay que asegurarse de que ambos giran en el mismo sentido. Si cada motor gira en un sentido ambos motores van a empujar al brazo en la misma dirección, siendo imposible estabilizarlo una vez pasado el punto de equilibrio. Si por ejemplo el motor izquierdo empuja el brazo hacia arriba y el motor derecho empuja el brazo hacia abajo sólo se puede actuar sobre el brazo girándolo hacia la derecha. Si la conexión realizada hace girar a cada motor en un sentido sólo se debe **intercambiar la conexión de dos fases** en un motor y el problema está solucionado.

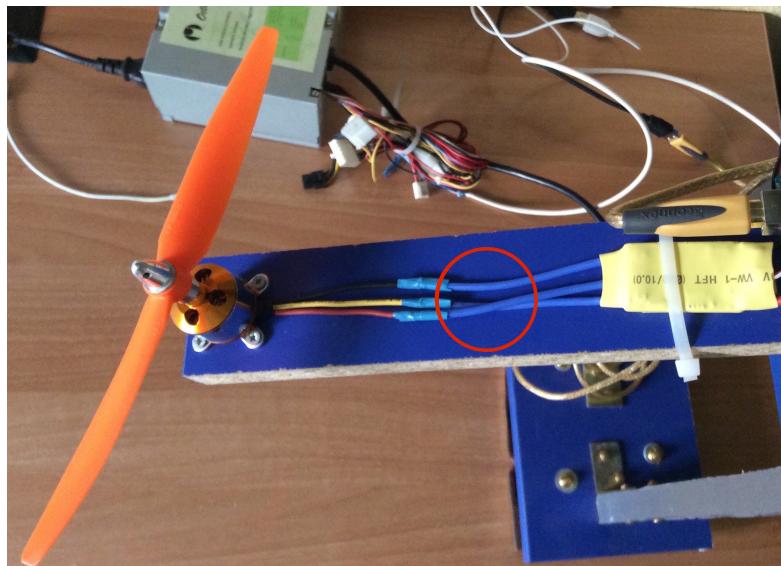


Figura 4.2: Fases intercambiadas en uno de los motores.

Con las conexiones de los motores ya realizadas se pueden empezar a realizar pruebas con los motores para conocer su funcionamiento con el objetivo de que los reguladores que se implementen tengan un funcionamiento óptimo.

### 4.3 Regulación de velocidad de los motores

La señal mediante la que se regulan los motores es una señal PPM (pulse position modulation), como se ha explicado en la sección 2.6. Este apartado se va a centrar en explicar cómo se realiza el programa necesario para generar esas señales y enviarlas al ESC. Tiene varias modos de funcionamiento, entre los que se incluye el modo acelerador o *throttle* y el modo freno. Para que los motores se muevan se debe programar el ESC en modo *throttle*. Una peculiaridad de estos controladores es que la elección del modo de funcionamiento se realiza mediante sonidos, justo después de escuchar el sonido asociado al modo de funcionamiento que se quiere elegir se debe enviar la señal mínima al controlador y el ESC quedará programado en ese modo. El modo *throttle* corresponde al segundo sonido emitido por el ESC, el primero indica que está conectado. Para enviar señales al ESC se debe hacer como si fuese un servo, utilizando la librería servo de Arduino. Los pasos necesarios para llegar a arrancar un motor son los siguientes:

1. **Incluir la librería:** para poder utilizar las funciones y clases de datos definidas en la librería servo hay que incluirla en el proyecto.
2. **Declarar una variable:** en la librería servo hay una clase de datos llamada *servo*, hay que declarar una variable por cada motor.
3. **”Adjuntar” el motor:** mediante la función *attach* declarada en la librería servo se especifica en qué pin está conectado cada motor.
4. **Programación del ESC:** para iniciar el modo programación del ESC hay que escribir la señal máxima de los motores, que son 2 ms.
5. **Elección modo *throttle*:** el modo de funcionamiento se elige mediante sonidos, cuando se oiga el sonido asociado al modo en el que se quiere trabajar se quiere enviar la señal mínima, que son 0.7 ms.
6. **Escribir un valor de velocidad en los motores:** el último paso es escribir el valor de velocidad a la que se quiere que giren los motores, ese valor deberá estar comprendido entre 1 y 2 ms.

Un ejemplo de un programa mediante el que se arrancan los motores se muestra en el código 4.1. En él se han programado todos los pasos arriba mencionados para poder la velocidad de los motores.

Código 4.1: Ejemplo de programa para arrancar los motores.

```

1 #include <Servo.h>                                //Librería para controlar un servo
2
3
4 #define MAX_SIGNAL      2000
5 #define MIN_SIGNAL      700
6
7 #define MOTOR_IZQUIERDO_PIN    9
8
9 Servo motorizquierdo;        //Definición de la variable para poder adjuntar el motor
10 int velocidad_motor_izquierdo;
11
12 void setup()
13 {
14     //Adjuntar el motor, se asocia la variable motorizquierdo con el motor conectado al pin 9
15     motorizquierdo.attach(MOTOR_IZQUIERDO_PIN);
16
17     //Ahora se debe entrar en modo programación, para lo que se escribe la señal máxima en el
18     //ESC
19     motorizquierdo.writeMicroseconds(MAX_SIGNAL);
20
21     // Esperando a que se pulse una tecla
22     while (!Serial.available());
23     Serial.read();
24
25     // Enviar valor mínimo
26     motorizquierdo.writeMicroseconds(MIN_SIGNAL)
27 }
28
29 void loop()
30 {
31     velocidad_motor_izquierdo = 1300;
32     motorizquierdo.writeMicroseconds(velocidad_motor_izquierdo);
33 }
```

El bucle while de la línea 18 está ejecutándose hasta que se pulse una tecla cualquiera. Se coloca para que cuando se escuche el sonido correspondiente al modo de funcionamiento del controlador con el que se quiere trabajar se pulse cualquier tecla y el programa siga ejecutándose. Para mandar un valor de velocidad al controlador se utiliza la función *writeMicroseconds*. Como el valor pasado son microsegundos hay que multiplicar por 1000 el valor de milisegundos correspondiente.

CAPITULO V

## **Estabilización del brazo**



# Capítulo 5

## Estabilización del brazo

\*\*\*Hacer estudio teorico(METERLO EN UN "PART.ºANTERIOR A ESTE) con la identificacionn del sistema y despues comparar con los resultaods expermientales

### 5.1 Introducción

Hasta este punto el trabajo se ha centrado en estudiar los componentes y cómo utilizarlos para tomar medidas en 1 caso de los sensores y variar su velocidad en el caso de los motores, pero no se ha aplicado ningún control sobre ningún componente. En este apartado se va a aplicar la teoría de control para estabilizar el brazo en un ángulo que será indicado al sistema como referencia.

### 5.2 Controladores

Un controlador tiene la función de mantener constante una característica determinada del sistema. En este caso se mantendrá constante el ángulo brazo mediante la actuación de los motores. Los controladores más comunes son los denominados como PID, sin embargo en este trabajo se realizarán pruebas con controladores PPI, PIP, PIDPID. A continuación se explicará cómo implementarlos y se mostrarán los resultados de cada uno.

#### 5.2.1 PID

Es un mecanismo de control sobre un mecanismo de control por realimentación. Estos 3 controladores en conjunto procesan el error calculado a partir de la señal de referencia menos la señal de salida real. El algoritmo del control PID consiste de tres controladores distintos:

- **Proporcional:** su respuesta es proporcional al error. Si sólo se usa este controlador hay mucho ruido en el sistema y el sistema oscilaría mucho hasta estabilizarse.
- **Integral:** su respuesta es proporcional a la integral del error, es decir, a la variación del error. Se puede considerar una memoria que estima lo que va a pasar en función de los errores pasados. Elimina el error en régimen estacionario a costa de aumentar el tiempo de establecimiento y hacer la respuesta más lenta.

- **Derivativo:** su respuesta es proporcional a la derivada del error, es decir, a la velocidad de variación del error. Con este controlador se aumenta la velocidad de respuesta del sistema a costa de introducir ruido a alta frecuencia.

Conjuntando estos tres reguladores se consigue un buen sistema de control. Su comportamiento se varía ajustando las constantes de cada controlador, consiguiendo así la mejor respuesta en función de la aplicación. El esquema del sistema incluyendo el PID se muestra en la figura 5.1.

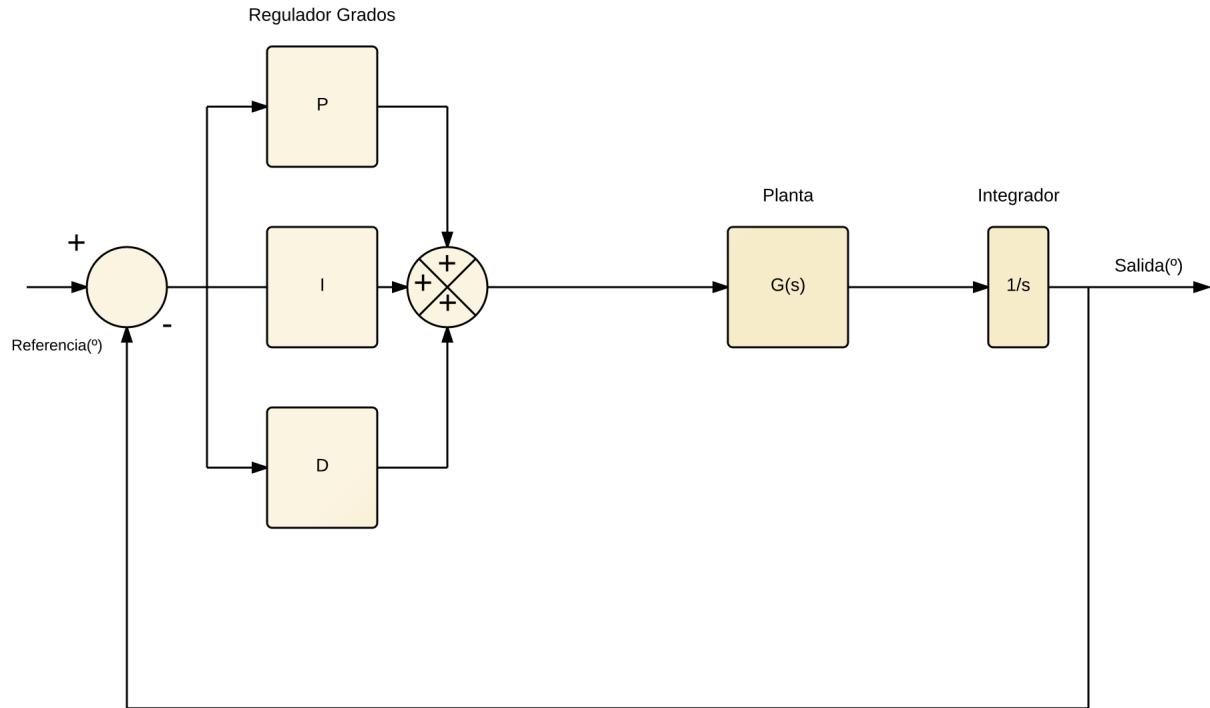


Figura 5.1: Esquema del sistema.

La entrada de referencia es el ángulo en grados al que se quiere estabilizar el brazo. Para estabilizar el brazo se va a establecer una velocidad base de 1,3 ms para cada motor, que será la velocidad a la que giren los motores si el error es nulo. Si hay error el PID actuará acelerando un motor y desacelerando el otro para corregirlo. Los pasos a seguir para implementar este controlador son:

1. **Definir las constantes:** hay que definir los valores de las constantes que se multiplicarán a cada parámetro del controlador, después se ajustarán experimentalmente.
2. **Calcular el error:** como ya se ha calculado el ángulo del brazo en el capítulo 3 ahora se necesita un dato de entrada, el ángulo de referencia. Para empezar se escribirá en el programa como un `# define`, más adelante se modificará desde LabView. Para calcular el error sólo hay que restar el ángulo de referencia al ángulo medido del brazo.
3. **Calcular el integral:** como el parámetro es una integral es imprescindible multiplicar por DT<sup>1</sup> la suma del error.
4. **Calcular el derivativo:** ya que es una derivada hay que dividir por DT la resta del error actual y el error de la iteración anterior.
5. **Cálculo del PID:** teniendo los 3 parámetros hay que combinarlos todos en una única variable.

<sup>1</sup>Delta Time: es la duración del bucle principal en segundos

6. **Aplicar el control a los motores:** como se indica en la sección 4.2 cada motor gira en un sentido, por lo tanto a un motor hay que sumar el valor del controlador y al otro restarlo.

Codigo 5.1: Implementación del PID.

```

1 #include <Servo.h>
2 #include <Wire.h>
3 #include <LSM303.h>
4 #include <L3G.h>
5
6 #define DT 0.03
7 #define ANGULO_REFERENCIA = 0; //Ángulo al que se quiere estabilizar el brazo
8
9 #define MAX_SIGNAL 2000
10 #define MIN_SIGNAL 700
11 #define MAX_VEL 1500
12 #define MIN_VEL 1000
13
14 #define MOTOR_IZQUIERDO_PIN 9
15 #define MOTOR_DERECHO_PIN 10
16
17 #define Throttle_Motor_Izquierdo 1300 //Velocidad de los motores si el error es 0
18 #define Throttle_Motor_Derecho 1300
19
20 double angulo_y
21
22 //VARIABLES MOTORES
23
24 Servo motorizquierdo, motorderecho;
25 int velocidad_motor_izquierdo, velocidad_motor_derecho;
26
27 //VARIABLES PID
28
29 float Kp = 6, Ki = 3, Kd = 2;
30 float error, error_pasado, derivativo, integral;
31 float PID_angulo;
32
33 void setup()
34 {
35     Serial.begin(115200);
36     Wire.begin();
37     //Adjuntar motores, programar ESC's y habilitar sensores
38
39     ...
40 }
41 void loop()
42 {
43     //Medida ángulo
44     ...
45     angulo_y = 0.95*(angulo_y + dps_y*DT) + 0.05*angulo_acelerometro_y;
46
47     //PID
48
49     error = ANGULO_REFERENCIA - angulo_y;
50     integral+=(error*DT);
51     derivativo = (error-error_pasado)/DT;
52     PID_angulo = Kp*error + Ki*integral + Kd*derivativo;
53 }
```

```

54 //Control motores
55
56     velocidad_motor_izquierdo = Throttle_Motor_Izquierdo + PID_angulo;
57     velocidad_motor_derecho = Throttle_Motor_Derecho - PID_angulo;
58
59     velocidad_motor_izquierdo = (velocidad_motor_izquierdo > MAX_VEL) ? MAX_VEL:
60         velocidad_motor_izquierdo;
61     velocidad_motor_izquierdo = (velocidad_motor_izquierdo < MIN_VEL) ? MIN_VEL:
62         velocidad_motor_izquierdo; //Velocidad minima para que no se pare el motor izquierdo
63
64     velocidad_motor_derecho = (velocidad_motor_derecho > MAX_VEL) ? MAX_VEL:
65         velocidad_motor_derecho;
66     velocidad_motor_derecho = (velocidad_motor_derecho < MIN_VEL) ? MIN_VEL:
67         velocidad_motor_derecho;//Velocidad minima para que no se pare el motor derecho
68
68     motorizquierdo.writeMicroseconds(velocidad_motor_izquierdo);
69     motorderecho.writeMicroseconds(velocidad_motor_derecho);
69
69     error_pasado = error; //Para calcular el derivativo
}

```

El código 5.1 muestra cómo se implementa el controlador PID. Las constantes MAX\_VEL y MIN\_VEL se utilizan para limitar la velocidad de los motores desde la línea 64 del código hasta la línea 69. Es recomendable para que la velocidad no sobrepase el límite, si descende de 1 ms el motor se puede parar y si aumenta de 2 ms el motor puede llegar a romperse. Se ha establecido en 1,5 ms la máxima porque la fuente de alimentación no suministra suficiente corriente para llevar a un motor a la velocidad máxima mientras el otro sigue girando. El resultado de la estabilización con este controlador se muestra en la figura 5.2. Los valores de las constantes para estos resultados son  $K_p = 6$ ,  $K_i = 3$  y  $K_d = 2$ .

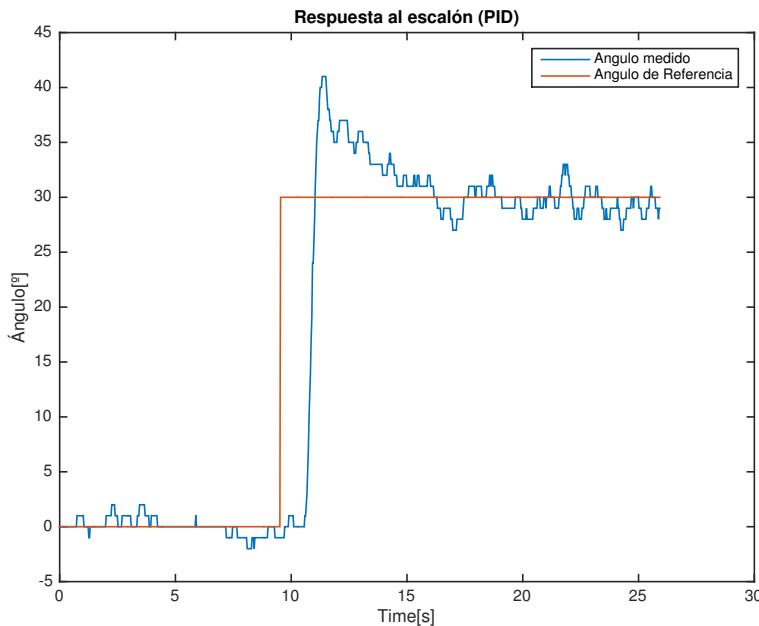


Figura 5.2: Respuesta del sistema para controlador PID.

El ruido presente se debe a las vibraciones de los motores, cuando el brazo llega al ángulo de referencia la vibración de los motores provoca un pequeño error que el controlador intenta corregir constantemente,

y a la constante  $K_d$ , aumentándola se consigue un mayor tiempo de respuesta pero también se introduce más ruido en el sistema. En la figura 5.3 se muestra la magnitud del ruido después del escalón.

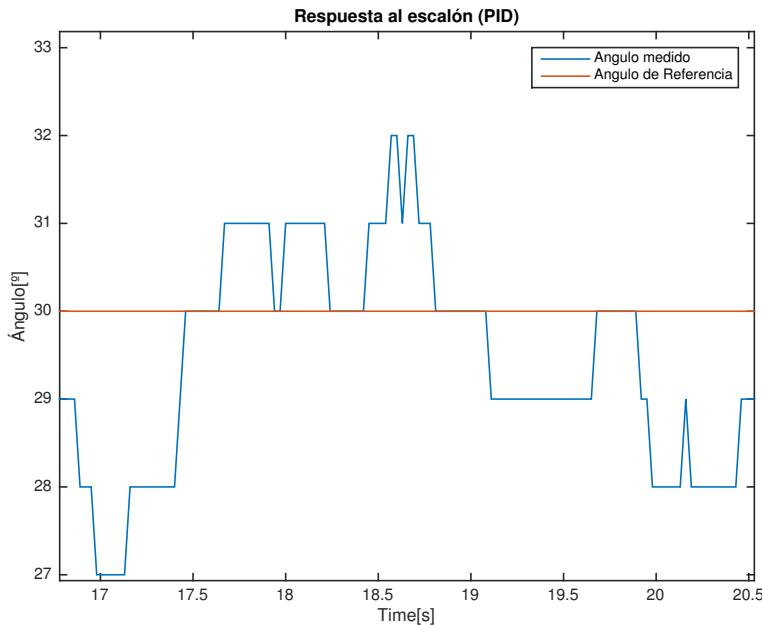


Figura 5.3: Detalle del ruido presente en la respuesta en régimen permanente del controlador PID.

El error máximo es de  $3^\circ$ , que es un error aceptable para la estabilización del brazo. Variando los valores de las constantes del controlador, lo que se denomina *PID tuning*, se puede cambiar el comportamiento del sistema. Si bien se puede mejorar también se puede empeorar y siendo este un buen resultado se van a dejar estos valores de las constantes.

Para poder comparar los resultados con el resto de controladores se van a calcular en la respuesta de cada controlador los parámetros en régimen transitorio. Para poder obtener el valor exacto de algunos de ellos es recomendable escribir un script de Matlab. El código 5.2 permite obtener el tiempo de establecimiento, el tiempo de pico y el sobreimpulso máximo con exactitud.

Código 5.2: Cálculo de parámetros en régimen transitorio con Matlab.

```

figure (1);
plot(dt, angulomedido, dt, anguloreferencia);
title('Respuesta_al_escalón');
xlabel('Time[s]');
ylabel('Ángulo[°]');
legend('Ángulo_medido', 'Ángulo_de_referencia');

m = 1031;
while angulomedido(m) > 27 & angulomedido(m) < 33
    m = m-1;
end
Tiempo_de_establecimiento = dt(m)-9.5;
[Mp, k] = max(angulomedido);
Tiempo_de_pico = dt(k)-9.5;

```

El valor de la variable  $m$  tiene que ser el número de muestras de *angulomedido*, se recorre el array *angulomedido* desde la última muestra y cuando su valor deja de estar en el 10 % del ángulo de referencia. La constante 9.5 restada a los tiempos de establecimiento y de pico es el instante en que comienza el escalón, que es cuando hay que medir los parámetros. El tiempo de retardo y el tiempo de subida se pueden obtener con el puntero de Matlab, situándose sobre el punto adecuado, el momento en que se llega al 50 % del valor del valor máximo en el caso del tiempo de retardo y el momento en el que se llega por primera vez al valor de escalón en el caso del tiempo de subida. En el caso del controlador PID los valores de estos parámetros son:

Tabla 5.1: Parámetros en régimen transitorio PID.

Parámetro	Valor teórico	Valor experimental
Tiempo de retardo ( $t_d$ )	***	2 seg
Tiempo de subida ( $t_r$ )	***	3 seg
Tiempo de pico ( $t_p$ )	***	1.84 seg
Tiempo de establecimiento ( $t_{s10\%}$ )	***	14.83 seg
Máximo sobreimpulso ( $M_p$ )	***	41° (1.36)

### 5.2.2 PPI

En este controlador se incluye un nuevo bucle de realimentación. Ahora se realimenta también la velocidad angular, que es la salida de la planta, para aplicar un regulador. El esquema de este regulador se muestra en la figura 5.4

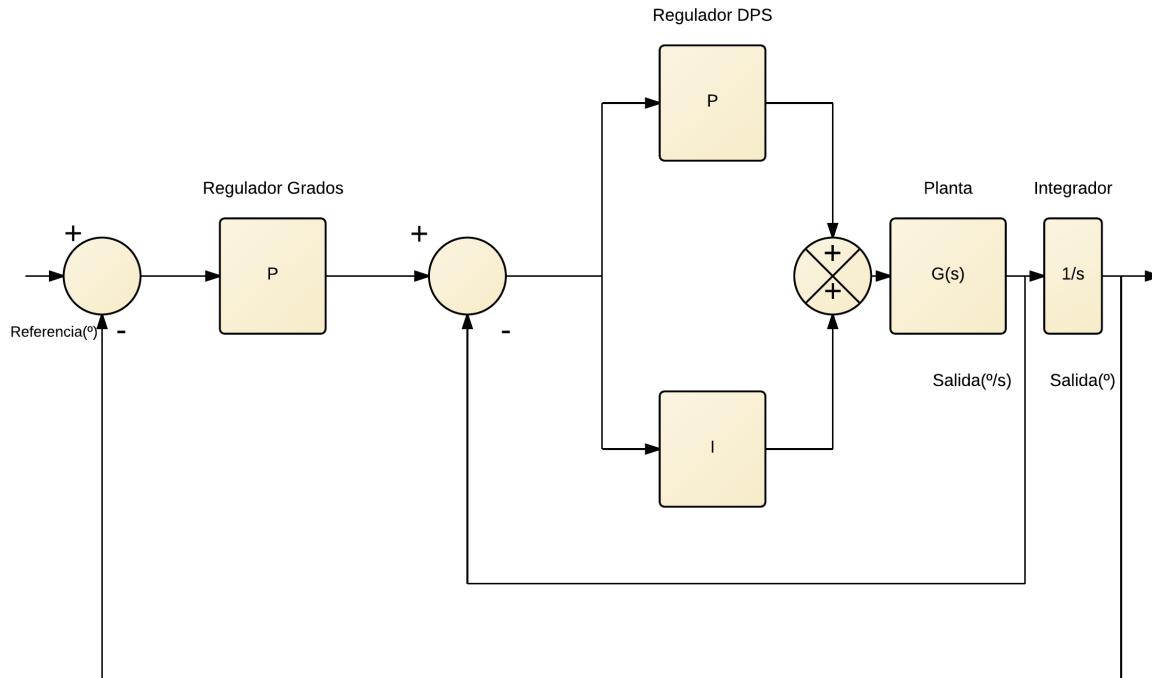


Figura 5.4: Esquema del controlador PPI.

Este controlador consiste en realimentar la salida en grados aplicándole al error un controlador proporcional y realimentar la salida de la planta en grados por segundo y al error aplicarle un controlador

proporcional integral. La implementación de este controlador se realiza mediante el código 5.3.

Código 5.3: Implementación del PID.

```
1 #include <Servo.h>
2 #include <Wire.h>
3 #include <LSM303.h>
4 #include <L3G.h>
5
6 #define DT 0.03
7 #define ANGULO_REFERENCIA = 0; //Ángulo al que se quiere estabilizar el brazo
8
9 #define MAX_SIGNAL 2000
10 #define MIN_SIGNAL 700
11
12 #define MAX_VEL 1500
13 #define MIN_VEL 1000
14
15 #define MOTOR_IZQUIERDO_PIN 9
16 #define MOTOR_DERECHO_PIN 10
17
18 #define Throttle_Motor_Izquierdo 1300 //Velocidad de los motores si el error es 0
19 #define Throttle_Motor_Derecho 1300
20
21 double angulo_y
22
23 //VARIABLES MOTORES
24
25 Servo motorizquierdo, motorderecho;
26 int velocidad_motor_izquierdo, velocidad_motor_derecho;
27
28 //VARIABLES PID
29
30     float Kp = 6, Kpi_p = 3, Kpi_i = 2;
31     float error_p, error_pi, integral;
32     float PPI_angulo;
33
34 void setup()
35 {
36     Serial.begin(115200);
37     Wire.begin();
38     //Adjuntar motores, programar ESC's y habilitar sensores
39
40     ...
41 }
42 void loop()
43 {
44     //Medida ángulo
45     ...
46     angulo_y = 0.95*(angulo_y + dps_y*DT) + 0.05*angulo_acelerometro_y;
47
48     //PPI
49
50     error_p = ANGULO_REFERENCIA - angulo_y;
51     error_pi = Kp*error_p - dps_y;
52     integral+=(error_pi*DT);
53
54     PPI_angulo = Kpi_p*error_pi + Kpi_i*integral;
```

```

56 //Control motores
57
58     velocidad_motor_izquierdo = Throttle_Motor_Izquierdo + PPI_angulo;
59     velocidad_motor_derecho   = Throttle_Motor_Derecho - PPI_angulo;
60
61     //El resto igual que en el PID
62
63     ....
64
65
66 }

```

A partir de la línea 50 está la implementación del controlador, primero se calcula el error de la salida en grados y después a ese error multiplicado por la constante  $K_p$  se le resta la salida en grados por segundo. El resultado de la estabilización con este controlador se muestra en la figura 5.5. Los valores de las constantes para estos resultados son  $K_p = 4$ ,  $K_{pi\_p} = 5$  y  $K_{pi\_i} = 2$ .

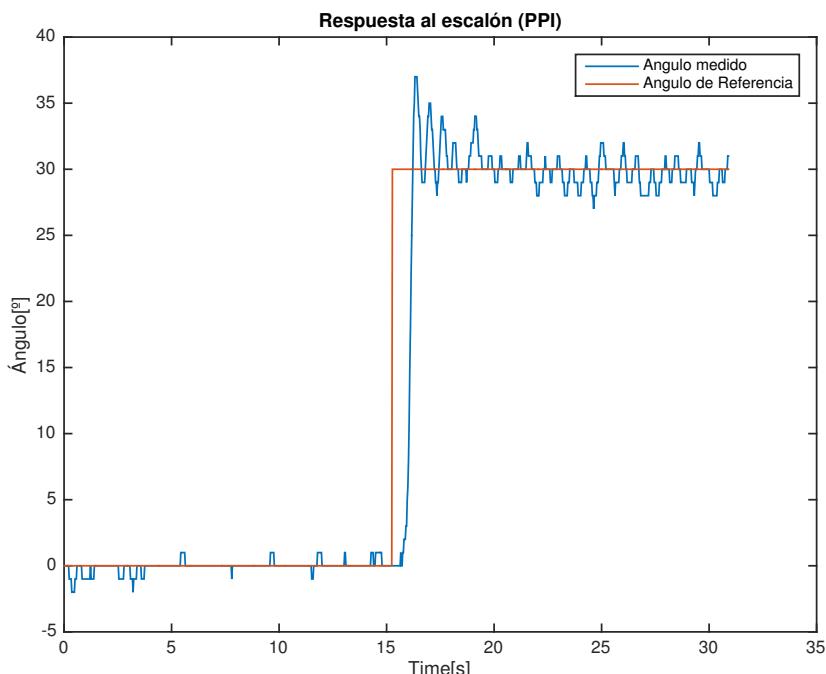


Figura 5.5: Respuesta del sistema para controlador PPI.

A simple vista con este controlador se ha conseguido reducir el máximo sobreimpulso y el ruido en régimen permanente es menor. Cuando está estabilizado a  $0^\circ$  antes de introducir el escalón la respuesta del sistema sigue a la referencia con un error de  $1^\circ$ . El ruido en régimen permanente se muestra en detalle en la figura 5.6.

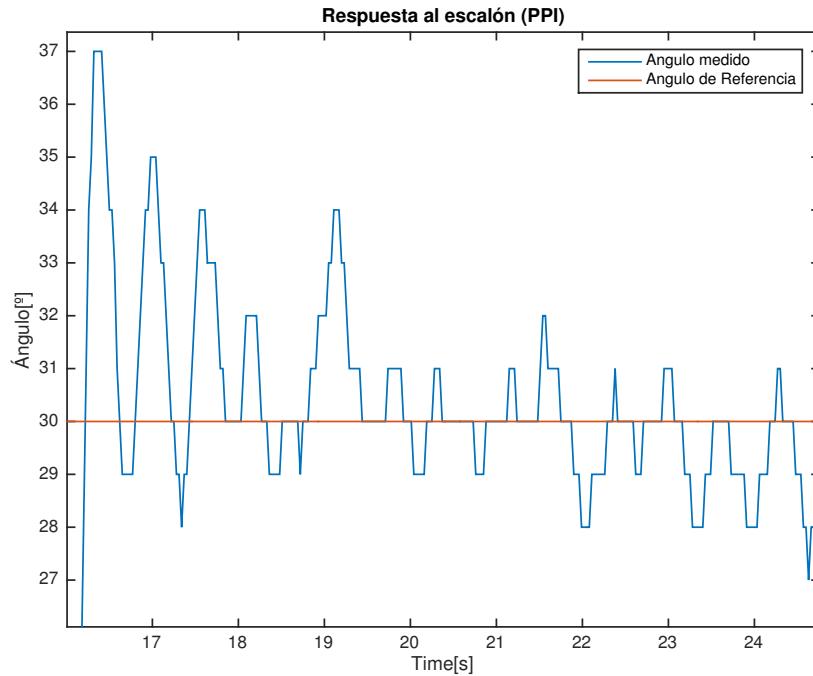


Figura 5.6: Detalle del ruido presente en la respuesta en régimen permanente del controlador PPI.

Los parámetros en régimen transitorio de este controlador son:

Tabla 5.2: Parámetros en régimen transitorio PPI.

Parámetro	Valor teórico	Valor experimental
Tiempo de retardo ( $t_d$ )	***	1.1 seg
Tiempo de subida ( $t_r$ )	***	1.2 seg
Tiempo de pico ( $t_p$ )	***	1.32 seg
Tiempo de establecimiento ( $t_{s10\%}$ )	***	9.63 seg
Máximo sobreimpulso ( $M_p$ )	***	37° (1.23)

\*\*\*Hacer tabal comparativa al final de este apartado con los resultados de cada controlador de tiempo de establecimiento, maximo swobreimpulso, tiempo de subida...

### 5.2.3 Definiciones específicas del tipo de documento

Para comenzar a usar la plantilla es fundamental revisar el fichero `book.tex` en el que se incluyen todos los detalles genéricos de la estructura usada en el documento, con comentarios que esperamos que os ayuden a entenderlo. Si sois de los impacientes, basta con que comencéis por la parte en la que se incluyen los distintos capítulos (buscad la parte de los `\input{chapters/*.tex}`).

Uno de los ficheros de configuración más importantes es el `config/myconfig.tex` en el que se incluyen elementos para determinar la configuración específica de tu documento. El primero de ellos (para facilitar la generación de documentos en español o inglés), es el que define el idioma que vas a utilizar. Para seleccionarlo, basta con asignar `spanish` o `english` a la variable `\mybooklanguage`. A partir de ella, el sistema generará las cabeceras y títulos adecuados a cada una.

Igualmente, tendréis que definir las siguientes variables sobre el tipo de trabajo y el autor:

- Acrónimo de la titulación correspondiente al trabajo (variable `\mydegree`): Que se seleccionará entre los definidos (por ahora<sup>2</sup> son IT, IE, ITTSE, ITTST, ITI, GIEC, GIEAI, GIST, GIT, GIC, GII, GS1, MSEA, PHDUAH y PHDUPM) y que automáticamente configura portadas<sup>3</sup>, entre otras cosas.
- Título del documento (variable `\mybooktitle`).
- Nombre del autor del trabajo (variable `\mybookauthor`).
- DNI del autor del trabajo, usado en el papeleo de los TFGs (variable `\mybookDNI`).
- Departamento en el que se realiza el trabajo (variable `\mybookdepartment`, en español, y `\mybookdepartmentEnglish`, en inglés).
- Programa de Doctorado (en su caso) en el que se realiza el trabajo (variable `\mybookphdprogram`, en español, y `\mybookphdprogramEnglish`, en inglés).
- Grupo de investigación en el que se realiza el trabajo (variable `\mybookresearchgroup`).
- Centro en el que se realiza el trabajo (variable `\mybookschool`, que debería ser la Escuela Politécnica Superior, pero se incluye por generalidad).
- Universidad en el que se realiza el trabajo (variable `\mybooksuniversidad`, que debería ser la de Alcalá, pero se incluye por generalidad).
- Titulación del autor (usada en las tesis de UPM, (variable `\mybookauthordegree`)).
- Email del autor (variable `\mybookemail`).
- Nombre del primer (o único, en su caso) director del trabajo (variable `\mybookNameFirstAdvisor`).
- Nombre del segundo (en su caso) director del trabajo (variable `\mybookNameSecondAdvisor`).
- Nombre del presidente del tribunal (variable `\mybookpresident`).
- Nombre del primer vocal del tribunal (variable `\mybookfirstvocal`).
- Nombre del segundo vocal del tribunal (variable `\mybooksecondvocal`).
- Nombre del secretario del tribunal, en su caso (variable `\mybooksecretary`).
- Año del trabajo (variable `\mybookyear`).
- Fecha del anteproyecto, en su caso (variable `\mybookanteproyectodate`), en el caso de que se usen la plantilla del anteproyecto.
- Fecha de la defensa del trabajo, en su caso (variable `\mydefensedate`, en español, o `\mydefensedateEnglish`).
- Palabras clave en inglés (variable `\mybookkeywords`).

---

<sup>2</sup>Este documento se generó a finales de 2013.

<sup>3</sup>Un comentario sobre el color de las bandas en la portada de los TFGs: de acuerdo con la normativa, dicho color debe ser PANTONE 160c. Yo he intentado utilizar dicho color, pero el aspecto con el que finalmente aparece no es ni de lejos similar al del modelo que proporciona la EPS, con lo que he optado por utilizar el que se ve realmente en dicho modelo. Si quieres cambiarlo, busca “PANTONE” en `config/preamble.tex`.

- Palabras clave en español (variable \mybookpalabrasclave).

Y también variables para el caso de los trabajos que necesitan papeleo adicional (publicación en abierto, autorizaciones, etc.).

- Nombre del Secretario del Departamento (que firmará parte del papeleo) (variable \mybookdepartmentsecretary).
- Fecha que aparecerá en la firma del papeleo (variable \mybookdateforpaperwork).
- DNI del alumno (variable \mybookDNIOpenPublishing).
- Figura del autor del trabajo, que es normalmente “alumno” (variable \mybookProfFigureOpenPublishing).
- DNIs del/de los tutores, para los permisos de publicación en abierto (variables \mybookDNIFirstAdvisor y \mybookDNISecondAdvisor, en su caso)

Se ha comenzado a trabajar en la versión alfa del soporte para *research reports*, y en ese caso se usa la variable \mybookresearchreportID.

Parte de esa información se utilizará para llenar la metainformación incluida en el fichero pdf que se genera.

También se definen los colores que se usarán en los hiperenlaces del documento. En concreto<sup>4</sup>:

- Color de los enlaces en el índice de contenidos (variable \mytoclinkcolor).
- Color de los enlaces en el índice de figuras (variable \myloflinkcolor).
- Color de los enlaces en el índice de tablas (variable \mylotlinkcolor).
- Color de los enlaces en otros índices (variable \myothertoclinkcolor), de los que ahora se incluye un ejemplo en el fichero cover/extralistings.tex.
- Color de los enlaces (\ref) en el documento (variable \mylinkcolor).
- Color de los enlaces a URLs (variable \myurlcolor).
- Color de los enlaces a referencias bibliográficas (variable \mycitecolor).

Basta con que defináis las variables correspondientes y la plantilla generará automáticamente las portadas adecuadas a la normativa y usará las definiciones específicas que hayas seleccionado.

Por si os hace falta, en config/postamble.tex se definen algunas variables relacionadas con el tipo de trabajo. Por ejemplo las variables \mydegreefull (igual a “Grado en Ingeniería en Electrónica y Automática Industrial” en esta compilación), \mybookworktype (igual a “TFG” en esta compilación) y \mybookworktypefull (igual a “Trabajo Fin de Grado” en esta compilación). Otro ejemplo sería el autor de contacto: Sergio Martín Gómez <sergio.martingomez@edu.uah.es>.

Importante para las tesis de UAH: si necesitáis incluir ficheros pdf (los de autorización e informes de los tutores, por ejemplo), esta plantilla lo permite: mirad los \includepdf en el book.tex.

---

<sup>4</sup>Os rogamos encarecidamente que cambiéis los colores definidos actualmente, que se han usado para verificar que todo funciona correctamente.

### 5.2.4 Plantilla de anteproyecto

Para el caso de los TFM/TFGs/TFCs, se incluye una plantilla para realizar el anteproyecto.

La plantilla se encuentra en el directorio `anteproyecto`, y en fichero `anteproyecto.tex` tenéis un ejemplo. El `Makefile` genera automáticamente el `anteproyecto.pdf`, haciendo un `make` en ese directorio, y tiene targets similares a los del `Makefile` del documento principal, incluyendo `flatten` y `latexdiff`, con lo que también puede generarse el fichero con control de cambios, tal y como se describe en la sección 5.2.6 (cambiando las referencias a `book...` por `anteproyecto...`).

### 5.2.5 Papeleo adicional para la defensa de los TFGs

Para el caso de los TFGs y TFM (al menos los del MUSEA), se incluyen en el directorio `papeleo` algunos de los documentos que hay que generar en el momento de la defensa. En concreto:

- La autorización del tutor para la publicación en abierto, en el fichero `autorizacionTutorPublicarRepositorio.tex` para TFGs.
- La autorización del autor para la publicación en abierto, en el fichero `autorizacionAutorPublicarRepositorio.tex`
- El visto bueno del tutor para la defensa del TFG `vistoBuenoTutorTFG.tex`
- La autorización del autor y tutor/tutores para la publicación en abierto, en el fichero `autorizacionPublicarAbiertoTFM-MUSEA.tex`.
- El visto bueno del tutor para la defensa del TFM `vistoBuenoTutorTFM-MUSEA.tex`

En el directorio se incluye un `Makefile` que genera los pdfs correspondientes a esos documentos. El sistema automáticamente adapta los singulares/plurales necesarios para el caso de que haya uno o varios directores/tutores.

### 5.2.6 Generación del documento con control de cambios (para revisión)

Uno de los problemas de L<sup>A</sup>T<sub>E</sub>X frente a otros sistemas de edición de textos viene en el momento de la revisión de cambios realizados a un documento. En el caso que nos ocupa serían los que nos sugiere nuestro tutor de TFC/TFG/TFG o director de tesis doctoral y que luego querrá verificar cómo se han aplicado. En un procesador al estilo de libreoffice (vaaaaaaaale, o Microsoft Word también), tenemos la opción de comparar documentos o llevar el control de cambios y que el sistema automáticamente nos marque los añadidos o borrados.

En L<sup>A</sup>T<sub>E</sub>X también es posible si usamos un sistema de control de versiones (y si no lo estáis usando, deberíais plantearos seriamente el hacerlo), con la ayuda de herramientas adicionales.

Hay varias soluciones disponibles, la mayoría basada en el uso de la `latexdiff` [?] (o derivados). Lo recomendable sería el uso de un sistema automatizado como el disponible en `git-latexdiff` [?], pero necesita el soporte de `git`, y por el momento estoy manteniendo esto en `cvs` (flames to `/dev/null`, please).

`latexdiff` permite comparar dos versiones de un documento y generar un nuevo fichero fuente en L<sup>A</sup>T<sub>E</sub>X que, al compilarlo, muestra un pdf “bonito”, con las marcas de las diferencias (resaltando lo añadido

y lo quitado). El resultado es perfecto para hacer una buena revisión de los cambios, y por supuesto no tiene punto de comparación con ver un `diff` a palo seco de las dos versiones del documento.

Lo que finalmente he implementado es muy ad-hoc, pero funciona. El procedimiento para hacer uso de ello sería (asumiendo que usáis `cvs` como sistema de control de versiones<sup>5</sup>:

- Primero hay que preparar la versión base sobre la que luego se harán los cambios. Esa preparación se hace normalmente cuando se tiene una versión razonablemente estable, y con la que luego se quiere comparar (por ejemplo cuando le entregas a tu tutor tu primer borrador del documento completo). La preparación es sencilla: basta hacer un `make flatten`. Eso generará un fichero `book-flatten.tex` que contiene el estado actual del documento, expandido. Luego hay que registrarlo en el repositorio basta hacer un `cvs commit -m "New flattened version"` `book-flatten.tex`.
- A partir de ahí ya se puede trabajar en los cambios al documento, los que sean necesarios.
- Cuando se quiera obtener el fichero `pdf` con el control de cambios, bastará hacer un `make latexdiff`, que acabará generando `book-flatten-diff.pdf`, que será lo que estábamos buscando.

### 5.2.7 Problemas conocidos

Resumimos a continuación los problemas con los que nos hemos ido encontrando tras el uso más generalizado de esta plantilla, y, en su caso, la solución propuesta/adoptada:

- Al menos en la versión 12.04 de ubuntu, se producía un fallo de compilación por un problema del *option clash* del paquete `xcolor`. La solución fue incluir las opciones `[RGB,rgb]` de dicho paquete en el `documentclass`, y eliminar la inclusión de `xcolor` (que ya lo incluye, al menos, `listings`). No es muy bonito como solución, pero funcionaba. Eso dio lugar a otro problema, que hacía que todas las páginas aparecieran como en color cuando se llevaba a la imprenta (con el consiguiente incremento de precio). Para intentar solucionarlo, he vuelto a eliminar las opciones `[RGB,rgb]` del `documentclass`, y se las paso con un `\PassOptionsToPackage`, pero está pendiente de verificación. Please, confirmadme que funciona (tanto lo del color como la compilación en un ubuntu 12.04 (o posterior)).
- También hemos observado en la versión 12.04 de ubuntu que `evince` no es capaz de visualizar la primera página de los TFGs (generada con `tikz`), y que `xpdf` genera un core cuando intenta abrir el fichero compilado. La solución es usar `qpdfview` o `acroread`.

Si das con más problemas, escribid por favor a Sergio Martín Gómez <[sergio.martingomez@edu.uah.es](mailto:sergio.martingomez@edu.uah.es)> contándonoslos y trataremos de solucionarlo (y si tenéis la solución, contádnosla también).

---

<sup>5</sup>Si usáis `git` (lo que también os recomiendo que hagáis), el proceso es más sencillo porque `git-latexdiff` lo hace todo automático, aunque os tendrás que trabajar la parte correspondiente del `Makefile`

## 5.3 Ejemplos de elementos de utilidad

### 5.3.1 Uso de comandos definidos

A modo de ejemplo, hemos definido el comando `\texten{}` en `config/myconfig.tex` para usarlo, por ejemplo, para marcar palabras escritas en inglés (aka *English*). Sigue el ejemplo para definir aquellos que utilices con frecuencia.

Si quieres escribir el símbolo backslash puedes usar el comando `\backslash`: `\backslash`.

Lo mismo aplica para el símbolo tilde, para lo que puedes usar el comando `\textasciitilde{~}`.

### 5.3.2 Uso de “frases célebres”

Respecto a la frase célebre del inicio de los capítulos: todas las que hemos usado y las definiciones que las generen están sacadas del excelente trabajo de Marco Antonio Gomez-Martín y Pedro Pablo Gomez-Martín en el proyecto `TEXIS`, una plantilla para la creación de tesis y otros documentos y disponible en [?].

### 5.3.3 Inclusión de diagramas

Para incluir gráficos, la compilación que utilizamos permite usar ficheros `png`, `jpg` y `pdf`, en el comando `\includegraphics`. Si queréis ahorraros incluir el path a cada fichero, podéis definir todos aquellos en los que haya ficheros gráficos en el `\graphicspath` del `book.tex`.

En la figura 5.7 se muestra un ejemplo de gráfico generado automáticamente a partir de un fichero `.dia`<sup>6</sup>: `diagrams/Esquema_objetos.dia` (podéis generalizar su generación en el `Makefile`).

### 5.3.4 Definición y uso de acrónimos (aquí también se pueden poner acrónimos como ETTS)

El uso del paquete `glossaries` permite definir los acrónimos y el sistema automáticamente gestiona su inclusión completa la primera vez que se usa. Los acrónimos de ejemplo están en el fichero `acronyms/defacronymsgl.tex` (con opciones adicionales de configuración en `acronyms/acronymsgl.tex`).

Así, si nos referimos a Emotional Text To Speech (ETTS) o bien a Berlin Database of Emotional Speech (EMODB), veremos como aparecen expandidas la primera vez. A partir de ahí, sólo se usará el acrónimo como puede verse al volver a hablar de ETTS y EMODB.

Tiene también soporte para resetear todos los acrónimos como si no estuvieran usados. Vuelvo a incluir el párrafo anterior tras un `reset` (que se hace con un `\glsresetall[acronym]`):

El uso del paquete `acronym` permite definir los acrónimos y el sistema automáticamente gestiona su inclusión completa la primera vez que se usa. Así, si nos referimos a Emotional Text To Speech (ETTS) o bien a Berlin Database of Emotional Speech (EMODB), veremos como aparecen expandidas de nuevo (como si fuera la primera vez que se usan). A partir de ahí, sólo se usará el acrónimo como puede verse al volver a hablar de ETTS y EMODB.

---

<sup>6</sup>Tomadas de los proyectos fin de carrera de David Casillas y Manuel Villaverde.

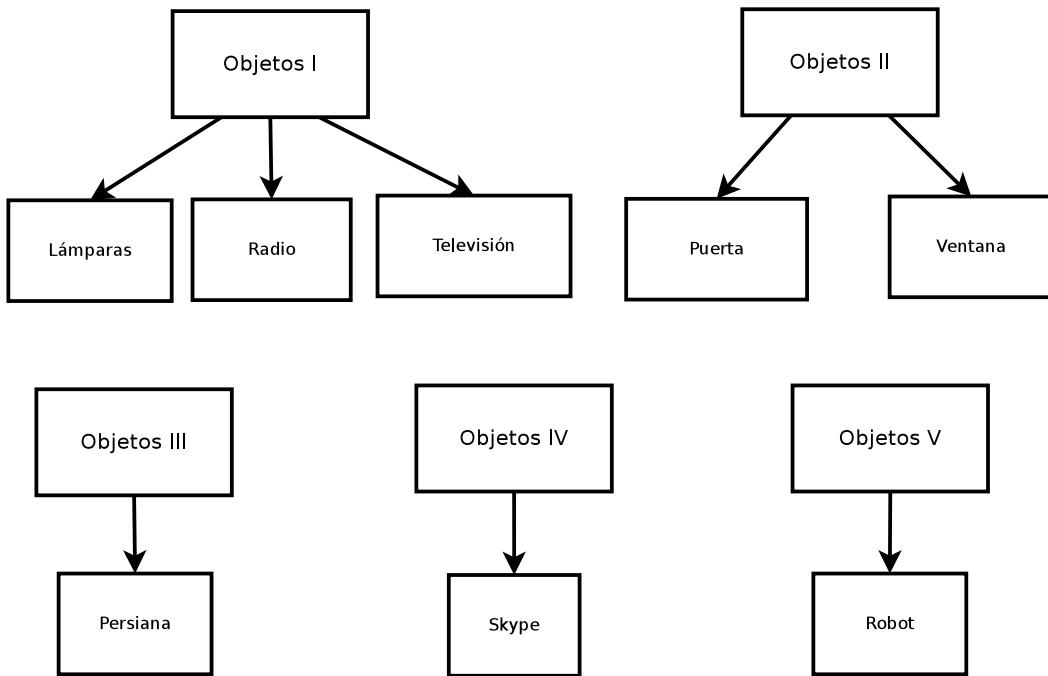


Figura 5.7: Clasificación de los objetos para la gramática (aquí también se pueden poner acrónimos como ETTS y símbolos como  $x_i(t)$ )

Y permite también forzar que se vuelva a citar completo aunque ya se haya utilizado (con el acrónimo entre paréntesis), como puede verse en Emotional Text To Speech (equivalente a Emotional Text To Speech que vale para cualquier glosario), y también a usar forzosamente el acrónimo. Primero reseteamos de nuevo.

Y ahora forzamos el acrónimo: EMODB (equivalente a EMODB que vale para cualquier glosario). También podemos forzar a que lo ponga todo, con Berlin Database of Emotional Speech (EMODB).

Podemos seguir definiendo entradas de acrónimos, referirnos a Dynamic Bayesian Network (DBN) por primera vez, y las siguientes aparecerán como DBN. Pongo ahora el resto de acrónimos Speech Quality (SQ), Emotion Identification Rate (EIR), Speaker Identification Rate (SIR) y Emotional Strength (ES). Finalmente los repito para que se vea el efecto: SQ, EIR, SIR y ES.

Y gestiona bien los plurales, ponemos el plural como la primera vez, y luego la segunda como . Y podemos volver al singular.

### 5.3.5 Definición y uso de símbolos (aquí también se pueden símbolos como $x_i(t)$ )

Los símbolos definidos están incluidos en el fichero `symbols/defsymbolsgl.tex` (con configuración adicional en `symbols/symbolsgl.tex`) y en esta sección mostramos algunos ejemplos.

El Á se usa en biología estructural, mientras que el Ω se usa en electrónica. También podemos poner  $x(t)$ . Y también funciona en entornos math (\$...\$) ( $x(t)$ ).

Es posible incluso deshabilitar los hiperenlaces, usando un “\*”, como en  $x(t)$  o Berlin Database of Emotional Speech (EMODB).

También valen en entornos equation:

$$x(t) \tag{5.1}$$

Y finalmente la que nos falta:  $x_i(t)$ , también dentro de ecuaciones (otra cosa es que sea conveniente o útil):

$$x_i(t) = \sqrt{i} \quad (5.2)$$

Acabamos con un par de acrónimos: Time Domain Pitch Synchronous OverLap Add (TD-PSOLA) y Speech Transformation and Representation using Adaptive Interpolation of weiGHTed spectrum (STRAIGHT).

Recientemente nos han pedido información sobre cómo introducir el comando de independencia incondicional y también funciona (la definición está en el `config/myconfig.tex`, y se usa tal cual en el `symbols/symbolsgl.tex` (revisadlos para ver cómo se implementa): `\|`.

## 5.4 Motivación y objetivos

La motivación de este proyecto...

Los objetivos principales de este trabajo son (ejemplo utilizando "enumerate"):

1. Primer objetivo...
2. Segundo objetivo...
  - (a) Objetivo 2.1...
  - (b) Objetivo 2.2...
3. Tercer objetivo...

## 5.5 Organización de la memoria

Esta memoria se organiza en cinco grandes capítulos. El primero ...

# Capítulo 6

## Estudio teórico

*Y así, del mucho leer y del poco dormir, se le secó el cerebro de manera que vino a perder el juicio<sup>1</sup>.*

Miguel de Cervantes Saavedra

### 6.1 Introducción

En este capítulo se cuenta tal y tal.

El capítulo se estructura en  $n$  apartados...

### 6.2 Estado del Arte

En el estado del arte se enumeran los trabajos más relevantes de otros grupos de investigación. A continuación se muestra un ejemplo del uso de viñetas que nos proporciona `itemize`:

- En el trabajo .....
- En el siguiente trabajo.....

O citas en un párrafo real: Sin embargo, hay entornos acústicos donde las tasas de error conseguidas son todavía demasiado altas. En concreto, las aplicaciones en las que la captura de la señal de habla se hace usando micrófonos alejados del locutor (típicamente para distancias superiores a un metro) muestran una fuerte sensibilidad a los problemas de reverberación, ruido aditivo y baja relación señal a ruido ([?],[?]). En estos entornos, se ha propuesto el uso de arrays de micrófonos como un método para mejorar la calidad del habla capturada [?][?].

Existen múltiples formas de insertar figuras en Latex. A continuación, se muestra un ejemplo del uso de `figure`. Como se puede ver en la Figura 6.1 también se pueden poner referencias a las figuras por medio de `ref` y la etiqueta `label` de la figura en particular.

Y ahora un ejemplo en el que ponemos el `caption` en el lateral:

---

<sup>1</sup>Tomado de ejemplos del proyecto TEXIS.

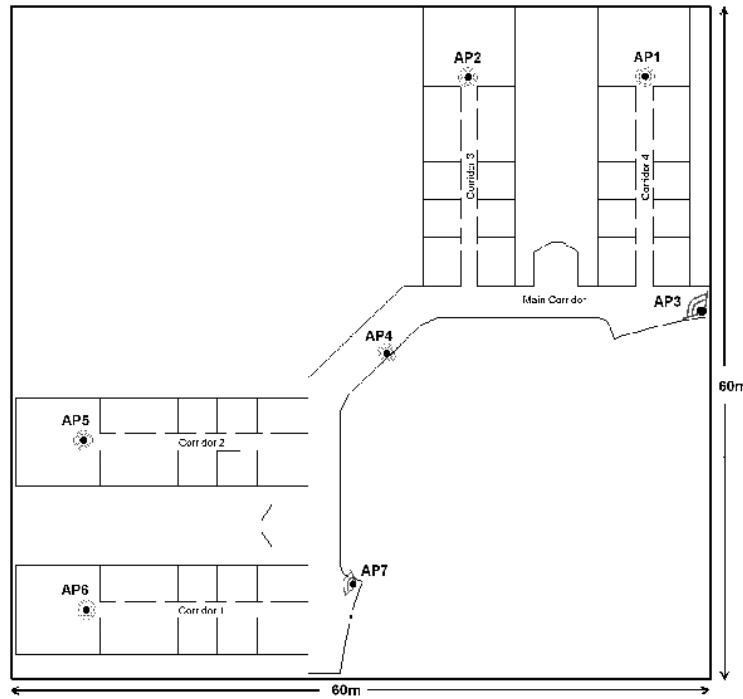


Figura 6.1: Departamento de Electrónica.

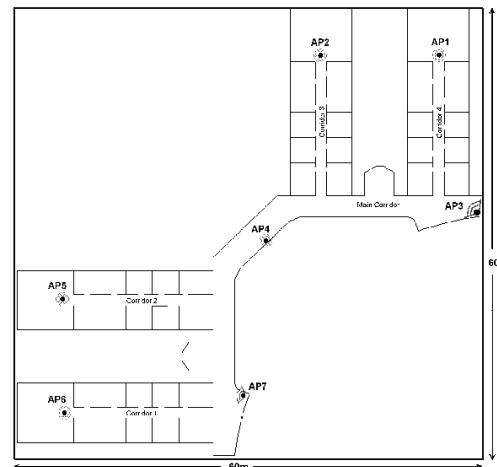


Figura 6.2: Departamento de Electrónica en el lateral.

### 6.3 Técnicas utilizadas

Aquí vamos a probar todos los niveles de sección disponibles, para evaluar la asignación de `tocdepth...`

Blah, blah, blah...

#### 6.3.1 Subsección

##### 6.3.1.1 Subsubsección

###### 6.3.1.1.1 Paragraph

Subparagraph

## 6.4 Conclusiones

Blah, blah, blah...



# Capítulo 7

## Desarrollo

*A fuerza de construir bien, se llega a buen arquitecto<sup>1</sup>.*

Aristóteles

### 7.1 Introducción

En este capítulo se incluirá la descripción del desarrollo del trabajo.

El capítulo se estructura en n apartados:...

### 7.2 Desarrollo del sistema de experimentación

Blah, blah, blah...

### 7.3 Planteamiento matemático

También resulta útil poder introducir ecuaciones que se encuentran tanto en línea con el texto (como por ejemplo  $\sigma = 0,75$ ), como en un párrafo aparte (como en la ecuación 7.1). Al igual que ocurre con las figuras, también se pueden referenciar las ecuaciones.

$$p[q_t = \sigma_t | q_{t-1} = \sigma_{t-1}] \quad (7.1)$$

### 7.4 Conclusiones

Blah, blah, blah...

---

<sup>1</sup>Tomado de ejemplos del proyecto TEXIS.



# Capítulo 8

## Resultados

*Rem tene, verba sequentur (Si dominas el tema, las palabras vendrán solas)<sup>1</sup>.*

Catón el Viejo

### 8.1 Introducción

En este capítulo se introducirán los resultados más relevantes del trabajo.

La estructura del capítulo es...

### 8.2 Entorno experimental

Blah, blah, blah.

#### 8.2.1 Bases de datos utilizadas

Blah, blah, blah.

#### 8.2.2 Métricas de calidad

Blah, blah, blah.

#### 8.2.3 Estrategia y metodología de experimentación

Blah, blah, blah.

### 8.3 Resultados experimentales

A continuación, se muestra un ejemplo de tabla simple (ver tabla 8.1).

---

<sup>1</sup>Tomado de ejemplos del proyecto TEXIS.

Tabla 8.1: Comparativa.

Method	Training Time	Man-Work (%)
Propagation model	< 30 sec	5
Manual	9 h 30 min	24
Automatic	2 h	10 8

Cuando las tablas ocupan más de un página se debe utilizar un tipo especial de tablas denominado `longtable`. A continuación, se muestra un ejemplo del mismo (ver tabla 8.2).

Tabla 8.2: Resultados de la correlación cruzada.

Posición Real	Posición estimada	Coef. Correlación	Acierto/Fallo
2P0	2P0	0,004954	A
2P1	2P4	0,005752	F
2P2	2P2	0,005461	A
2P3	2P0	0,004634	F
2P5	2P4	0,005991	F
2P6	2P16	0,004410	F
2P7	3P9	0,008038	F
2P8	3P9	0,003753	F
2P9	2P7	0,004908	F
2P10	2P10	0,007273	A
2P14	2P16	0,006485	F
2P15	2P15	0,004932	A
2P16	2P16	0,006237	A
2P17	2P15	0,005110	F
2P18	3P18	0,006235	F
2P19	3P18	0,004827	F
2P20	2P20	0,006877	A
2P22	3P18	0,003048	F
2P24	2P24	0,006833	A
2P25	2P25	0,004875	A
2P26	2P31	0,005511	F
2P27	2P28	0,004590	F
2P30	2P31	0,005576	F
2P31	2P31	0,007213	A
2P32	2P35	0,003340	F
2P34	2P34	0,004128	A
2P36	2P35	0,003329	F
2P37	2P37	0,003468	A
2P39	2P38	0,002577	F
2P40	2P43	0,004303	F
2P41	2P41	0,001573	A
Continúa en la página siguiente			

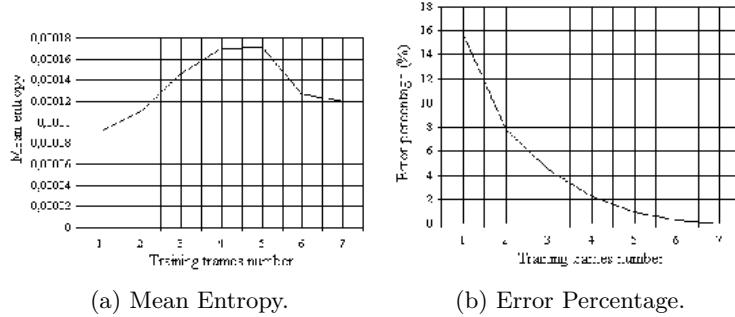


Figura 8.1: Optimal frames number in the training data set.

**Tabla 8.2 – continúa en la página anterior**

Posición Real	Posición estimada	Coef. Correlación	Acierto/Fallo
2P42	2P41	0,000846	F
2P44	2P44	0,002732	A
2P45	23P45	0,001958	F
2P47	2P34	0,002869	F
2P48	2P43	0,004569	F
2P49	3P51	0,001374	F
2P50	2P34	0,002274	F
2P51	2P63	0,003931	F
2P52	2P55	0,003537	F
2P53	3P56	0,003126	F
2P54	2P67	0,005560	F
2P56	2P55	0,002817	F
2P57	2P67	0,006168	F
2P58	2P58	0,005278	A
2P60	3P66	0,004966	F
2P61	3P61	0,004748	A
2P64	2P67	0,005342	F
2P66	2P4	0,004172	F
2P67	2P67	0,005706	A
3P0	3P0	0,003674	A
3P61	2P61	0,003263	F
3P64	2P67	0,003484	F
3P65	2P67	0,002975	F
3P66	2P58	0,005029	F
3P67	3P67	0,003714	A

En algunas ocasiones, también resulta útil emplear el entorno `subfigure` para añadir múltiples imágenes dentro de la misma figura. A continuación, se muestra un ejemplo del uso en la figura 8.1. También se pueden referenciar las sub-figuras de forma individual, por ejemplo la sub-figura 8.1b (usando un método de cita), o bien la sub-figura 8.1.b (usando otro alternativo).

La figura 8.2 muestra otro ejemplo con referencias a las subfigures en el caption principal.

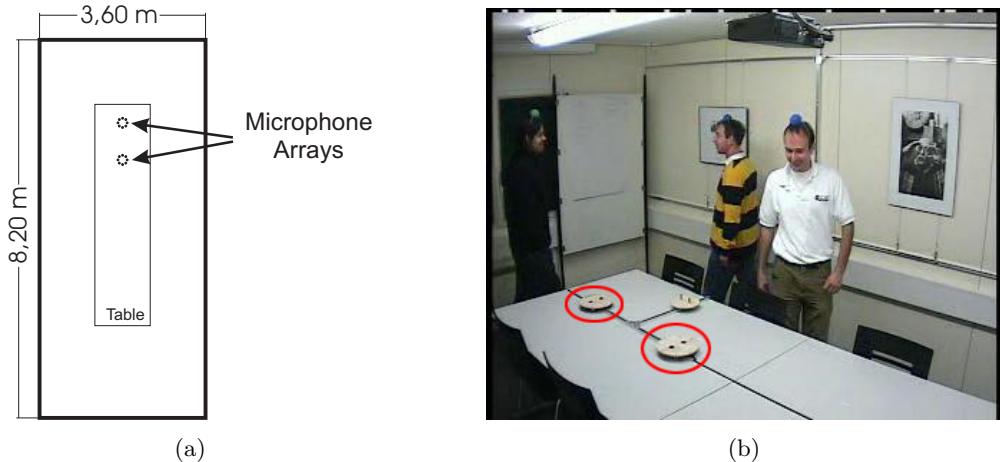


Figura 8.2: Idiap Smart Meeting Room for AV16.3 recordings. (a) Room layout showing the centered table, and the microphones arranged in two circular arrays. (b) Sample of recorded video frame showing the arrays area.

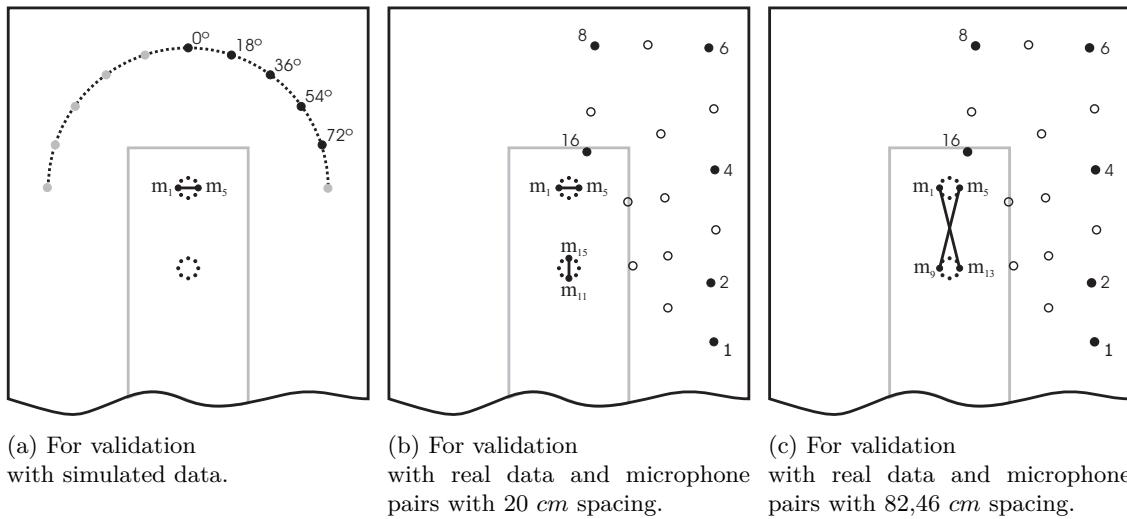


Figura 8.3: Geometrical details for the experiments carried out. Only the relevant section of the room is shown, and microphone pairs are connected by solid lines.

Os incluimos a continuación un párrafo de un artículo en el que hacemos referencia a varias figuras y subfiguras:

*The IDIAP Meeting Room (shown in figure 8.2) is a  $8,2m \times 3,6m \times 2,4m$  rectangular space containing a centrally located  $4,8m \times 1,2m$  rectangular table, on top of which two circular microphone arrays of 10cm radius are located, each of them composed by 8 microphones. The centers of the two arrays are separated by 80cm and the origin of coordinates is located in the middle point between the two arrays. The arrays can be also seen in figures 8.3.a., 8.3.b, and 8.3.c, in which only the relevant section of the room is displayed, each one showing different scenarios that were used in the experiments. A detailed description of the meeting room can be found in [?].*

En la figura 8.4 mostramos un ejemplo de varias figuras organizadas de forma un poco más complejo.

También es posible incluir el código de una figura en un fichero .tex independiente (para hacer más legible el código del documento principal). Un ejemplo lo tenéis a continuación, incluyendo el texto en inglés del documento original:

*Figure 8.5 includes the results of the comparison, for several speaker positions (1, 2, 4, 6, 8 and 16,*

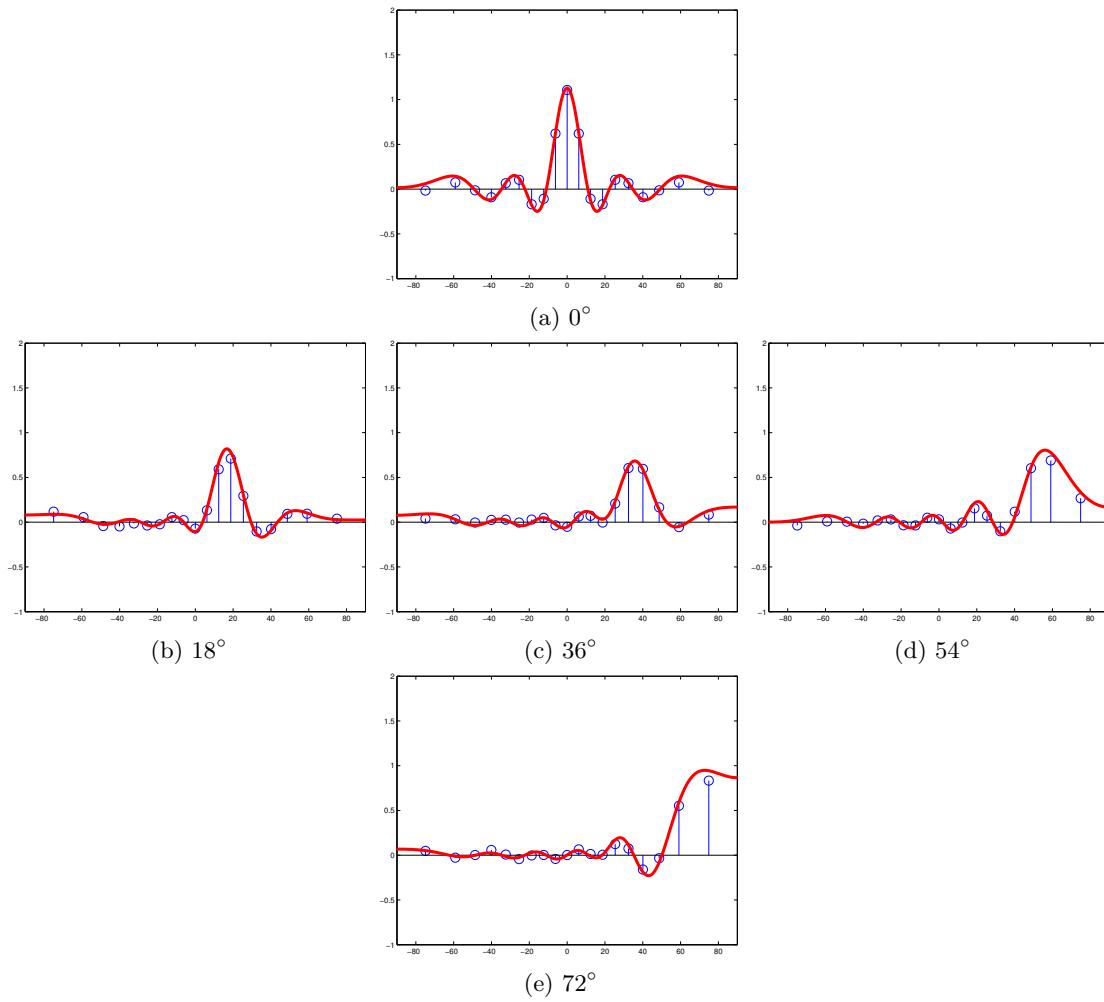


Figura 8.4: Comparación entre la respuesta de potencia dirigida generada por el modelo (línea sólida) y la calculada usando ondas simuladas en el entorno AV16.3. Los resultados para el altavoz en ángulos dados y el array dirigido de  $-90^\circ$  a  $+90^\circ$  se muestran.

*emphasized in figure 8.3.b), and selected to provide different acoustic situations, both in terms of distance and angular position with respect to the arrays. All the graphics show the acoustic power map (predicted or calculated) for a regular two-dimensional grid of 10 cm. The plot is provided from a top view of the room, spanning the full plan at a height of 61 cm above the microphone arrays (this height was the ground truth one for sequence 01). For each speaker position shown, three graphics are plotted:*

- The graphics on the left show the SRP-PHAT acoustic power maps generated by the proposed model (for example, the left graphic in figure 8.5.a for position 1).
- The graphics in the middle show the real SRP-PHAT acoustic power maps calculated using the real acoustic waveforms (for example, the middle graphic in figure 8.5.a for position 1), for a single selected frame.
- The graphics on the right show the average real SRP-PHAT acoustic power maps, averaging for all the frames in which the user was in the given position (for example, the right graphic in figure 8.5.a for position 1).

*The green point represents the real (ground truth) speaker position, and the black dots represent the positions of the four microphones used. The hyperbolic shapes found in the figure are consistent with the fact that the place of points with equal acoustic power value, for a given microphone pair, is a hyperbola (in our two-dimensional case, being a hyperboloid of revolution in the three-dimensional case).*

*From figure 8.5, it can clearly be seen that, again, the predictions closely match the results with real data for the different acoustic conditions, even when the simulations are using fixed and frequency independent average reflection coefficients, and that the acoustic model is based on the simplistic image method model.*

Incluso podemos poner una tabla “apaisada”, como en la 8.3, donde se muestra un resumen de los resultados obtenidos en una serie de experimentos de localización de locutores.

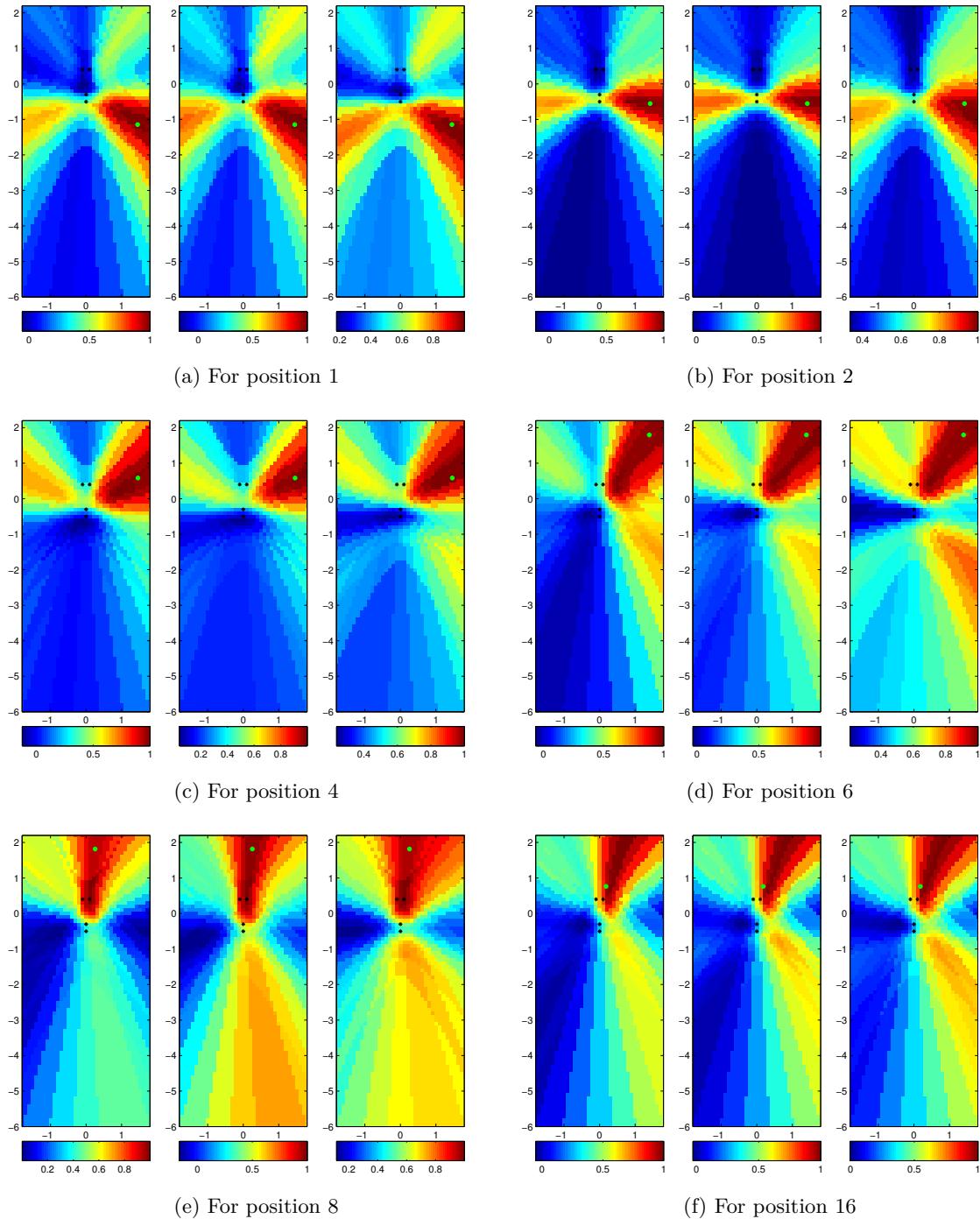


Figura 8.5: Comparison between the SRP-PHAT map predicted by the model (left graphics), the real SRP-PHAT map (middle graphics), and the average (real) SRP-PHAT map (right graphics), for several speaker positions ( $f_0 = 1,5 \text{ KHz}$ ). See figure 8.3.b for geometrical references.

## 8.4 Conclusiones

Blah, blah, blah.

	UKA	ITC	AIT	UPC	TBM
Pcor	57,0 ± 1,4 %	84,0 ± 3,3 %	47,0 ± 3,1 %	20,0 ± 2,5 %	67,0 ± 2,9 %
Bias fine (x:y:z) [mm]	20 : -42 : -75	45 : 27 : -41	-27 : -77 : -40	-59 : 112 : 52	91 : -69 : -38
Bias fine+gross (x,y,z) [mm]	735 : -93 : -258	67 : 439 : -134	17 : -402 : -118	-141 : 255 : 39	474 : -141 : -14
AEE fine [mm] = MOTTP	210	130	266	344	228
Fine+gross [mm]	1201	632	1006	1188	884
Loc. frames	5035	22	995	977	1023
Ref. duration (s)	6287,0	596,0	1143,0	1180,0	1194,0

Tabla 8.3: Resultados TEST CLEAR 2006.



# **Capítulo 9**

## **Conclusiones y líneas futuras**

En este apartado se resumen las conclusiones obtenidas y se proponen futuras líneas de investigación que se deriven del trabajo.

La estructura del capítulo es...

### **9.1 Conclusiones**

Para añadir una referencia a un autor, se puede utilizar el paquete `cite`. En el trabajo [?], se muestra un trabajo...

Y podemos usar de nuevo algún acrónimo, como por ejemplo TD-PSOLA, o uno ya referenciado como Artificial Neural Network (ANN).

### **9.2 Líneas futuras**

Pues eso.



# Apéndice A

## Manual de usuario

### A.1 Introducción

Blah, blah, blah...

### A.2 Manual

Pues eso.

### A.3 Ejemplos de inclusión de fragmentos de código fuente

Para la inclusión de código fuente se utiliza el paquete `listings`, para el que se han definido algunos estilos de ejemplo que pueden verse en el fichero `config/preamble.tex` y que se usan a continuación.

Así se inserta código fuente, usando el estilo `CppExample` que hemos definido en el preamble, escribiendo el código directamente :

```
1 #include <stdio.h>
2
3 // Esto es una función de prueba
4 void funcionPrueba(int argumento)
5 {
6     int prueba = 1;
7
8     printf("Esto es una prueba [
9 }
```

O bien insertando directamente código de un fichero externo, como en el ejemplo ??, usando `\lstinputlisting` y cambiando el estilo a `Cbluebox` (además de usar el entorno `codefloat` para evitar pagebreaks, etc.).

O por ejemplo en matlab, definiendo `settings` en lugar de usar estilos definidos:

```
a = 9;
b = 10;
```

```
c = a+b;
printf(1, 'La_suma_de_
```

O incluso como en el listado ??, usando un layout más refinado (con los settings de <http://www.rafaLinux.com/?p=599> en un lststyle Cnico).

Y podemos reutilizar estilos cambiando algún parámetro, como podemos ver en el listado ??, en el que hemos vuelto a usar el estilo Cnico eliminando la numeración.

Ahora compila usando gcc:

```
$ gcc -o hello hello.c
```

Y también podemos poner ejemplos de código *coloreado*, como se muestra en el ??.

Finalmente aquí tenéis un ejemplo de código shell, usando el estilo BashInputStyle:

```
#!/bin/sh

HOSTS_ALL="gc000 gc001 gc002 gc003 gc004 gc005 gc006 gc007"

for h in $HOSTS_ALL
do
    echo "Running [*] in $h..."
    echo -n " "
    ssh root@$h $*
done
```

## A.4 Ejemplos de inclusión de algoritmos

En la versión actual (abril de 2014), empezamos a usar el paquete algorithm2e para incluir algoritmos, y hay ajustes específicos y dependientes de este paquete tanto en config/preamble.tex como en cover/extralistings.tex (editadlos según vuestras necesidades).

Hay otras opciones disponibles (por ejemplo las descritas en <http://en.wikibooks.org/wiki/LaTeX/Algorithm>), y podemos abordarlas, pero por el momento nos quedamos con algorithm2e.

Incluimos dos ejemplos directamente del manual: uno sencillo en el algoritmo A.1, y otro un poco más complicado en el algoritmo A.2.

**Data:** this text

**Result:** how to write algorithm with LATEX2e  
initialization;

**while** not at end of this document **do**

```
    read current;
    if understand then
        go to next section;
        current section becomes this one;
    else
        go back to the beginning of current section;
```

**Algoritmo A.1:** How to write algorithms

**Data:**  $G = (X, U)$  such that  $G^{tc}$  is an order.  
**Result:**  $G' = (X, V)$  with  $V \subseteq U$  such that  $G'^{tc}$  is an interval order.

```

begin
     $V \leftarrow U$ 
     $S \leftarrow \emptyset$ 
    for  $x \in X$  do
         $NbSuccInS(x) \leftarrow 0$ 
         $NbPredInMin(x) \leftarrow 0$ 
         $NbPredNotInMin(x) \leftarrow |ImPred(x)|$ 
    for  $x \in X$  do
        if  $NbPredInMin(x) = 0$  and  $NbPredNotInMin(x) = 0$  then
             $AppendToMin(x)$ 
    while  $S \neq \emptyset$  do
        REM remove  $x$  from the list of  $T$  of maximal index
        while  $|S \cap ImSucc(x)| \neq |S|$  do
            for  $y \in S - ImSucc(x)$  do
                { remove from  $V$  all the arcs  $zy$  : }
                for  $z \in ImPred(y) \cap Min$  do
                    remove the arc  $zy$  from  $V$ 
                     $NbSuccInS(z) \leftarrow NbSuccInS(z) - 1$ 
                    move  $z$  in  $T$  to the list preceding its present list
                    { i.e. If  $z \in T[k]$ , move  $z$  from  $T[k]$  to  $T[k - 1]$  }
                 $NbPredInMin(y) \leftarrow 0$ 
                 $NbPredNotInMin(y) \leftarrow 0$ 
                 $S \leftarrow S - \{y\}$ 
                 $AppendToMin(y)$ 
        RemoveFromMin(x)
    
```

**Algoritmo A.2:** IntervalRestriction



## Apéndice B

# Herramientas y recursos

Las herramientas necesarias para la elaboración del proyecto han sido:

- PC compatible
- Sistema operativo GNU/Linux [?]
- Entorno de desarrollo Emacs [?]
- Entorno de desarrollo KDevelop [?]
- Procesador de textos L<sup>A</sup>T<sub>E</sub>X[?]
- Lenguaje de procesamiento matemático Octave [?]
- Control de versiones CVS [?]
- Compilador C/C++ gcc [?]
- Gestor de compilaciones make [?]





Universidad de Alcalá  
Escuela Politécnica Superior



ESCUELA POLITECNICA  
SUPERIOR

