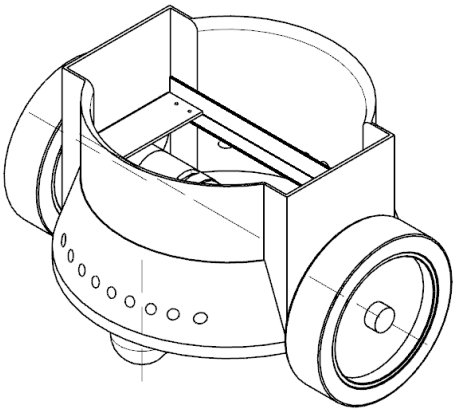
DISEÑO E IMPLEMENTACIÓN DE UN VEHÍCULO AUTÓNOMO TERRESTRE CONTROLADO MEDIANTE INTERFAZ INALÁMBRICA ANDROID-ARDUINO.



MEMORIA PRESENTADA POR:

José Ángel Garrido Sarasol

Máster Universitario en Ingeniería Mecatrónica

DIRECTOR:

Vicente Casanova Calvo

CODIRECTOR:

Julián Salt Llobregat

Septiembre, 2016

ÍNDICE.

1.- INTRODUCCIÓN 8
1.1.- Robots Móviles 8
1.2.- Objetivo 9
1.3.- Problema Planteado 10
1.4.- Relevancia del tema y Aportación 13
1.5.- Estructura del trabajo 14
2.- FUNDAMENTACIÓN TEÓRICA 17
2.1.- Teoría de Control, Control de Velocidad 17
2.2.- Algoritmo de Seguimiento de Trayectoria. 25
2.2.1.- Accionamiento Diferencial 25
2.2.2.- Ventajas e Inconvenientes 29
2.2.3.- Seguimiento de Trayectoria 30
2.2.4.- Follow the carrot 31
2.2.5.- Persecución Pura (*Pure Pursuit*) 33
2.3.- Control de procesos basado en red. 36
3.- ELEMENTOS DEL SISTEMA ROBOT 40
3.1.- Diagrama de Bloques Elementos Sistema Mecánico 40
3.2.- Diagrama de Bloques Elementos Sistema Energético 41
3.3.- Diagrama de Bloques Elementos Sistema Electrónicos 42
3.4.- Descripción-Características Chasis y Bastidor 43
3.5.- Descripción-Características Motores + Encoders 45
3.6.- Descripción-Características Batería y Condensadores 48
3.7.- Descripción-Características Microcontrolador ARM M3(ArduinoDue) 51
3.8.- Descripción-Driver Motores doble Puente en H 53
3.9.- Descripción-Características Modulo Shield wifi Arduino 56
4.- DESARROLLO Y PLAN DE TRABAJO 57
4.1.- Esquema-Cronograma de las tareas Realizadas. 58
4.2.- Identificación de los Motores 60
4.2.1.- Motor 1 (Rueda Derecha), Zona Muerta 61
4.2.2.- Motor 1 (Rueda Derecha), K estática 63
4.2.3.- Motor 1 Tiempo de Establecimiento 65
4.2.4.- Motor 1 FDT 67
4.2.5.- Motor 2 (Rueda Izquierda), Zona Muerta 68
4.2.6.- Motor 2 (Rueda Izquierda), K estática 70
4.2.7.- Motor 2 Tiempo de Establecimiento 72
4.2.8.- Motor 2 FDT 74
4.3.- Cálculo experimental del Regulador PI de Velocidad 75
4.3.1.- Acción Proporcional 76
4.3.2.- Acción Integral 76
4.3.3.- Corrección de la Zona Muerta 77
4.4.- Diseño del Chasis en NX 10 de Siemens 77
4.5.- Flujogramas de Funcionamiento Software 79
4.5.1.- Algoritmo-Persecución + Bucles Control Velocidad (Local). 80
4.5.2.- Algoritmo-Persecución + Bucles Control Velocidad (Local) + Servidor. 85
4.5.3.- Cliente desde Matlab/Simulink (Local Monitorización). 88
4.5.4.- Cliente desde Dispositivo Android (Local Monitorización). 94
4.5.5.- Cliente desde Matlab / Simulink + Algoritmo Seguimiento en modo remoto. . 97
5.- RESULTADOS OBTENIDOS 107

5.1.- Modalidad Funcionamiento Local Cliente Matlab / Simulink. 107
5.1.1.- Trayectoria Cuadrada con las ruedas sin contacto en el suelo. 107
5.1.2.- Trayectoria Cuadrada con las ruedas con contacto en suelo. 108
5.1.3.- Trayectoria figura Lissajous-1 con las ruedas en contacto con el suelo 110
5.2.- Modalidad Funcionamiento Local Cliente Android. 111
5.2.1.- Trayectoria Cuadrada con las ruedas sin contacto en el suelo. 111
5.2.2.- Trayectoria Cuadrada con las ruedas en contacto con el suelo. 112
5.3.- Modalidad Funcionamiento Remoto control basado en Red. 113
5.3.1.- Trayectoria Cuadrada sin Predictor de Smith. 113
5.3.2.- Trayectoria Cuadrada con Predictor Smith. 115
5.3.3.- Trayectoria Lissajous 1-2 sin Predictor de Smith 117
5.3.4.- Trayectoria Lissajous 1-2 con Predictor Smith. 119
6.- CONCLUSIONES 121
7.- BIBLIOGRAFÍA 125
8.- ANEXO 1 BATERIAS DE LITIO 127
8.1.- Descripción 127
8.2.- Carga 127
8.3.- Descarga 129
8.4.- Circuito de Protección 129
8.5.- Asociación de baterías 130
8.6.- Balanceo de las celdas de una asociación de baterías 130
9.- ANEXO 2 ESQUEMA ELECTRÓNICO DRIVER DOBLE PUENTE EN H 131
9.1.- Introducción 131
9.2.- Circuito Integrado L298 133
9.3.- Pines L298 134
10.- ANEXO 3 DIAGRAMAS DE FLUJO 135
10.1.- Introducción Concepto 135
10.2.- Las Reglas para la creación de Flujogramas 135
11.- ANEXO 4 PROGRAMAS 138
11.1.- Funcionamiento Local (Sin Cliente) 138
11.1.1.- **Declaración de variables y librerías utilizadas (Sin Cliente)** 138
11.1.2.- **Configuración Sistema (Sin Cliente)** 140
11.1.3.- **Programa Principal (Sin Cliente)** 141
11.1.4.- **Funciones e Interrupciones (Sin Cliente)** 143
11.2.- Funcionamiento Local + Servidor (Con Cliente) 145
11.2.1.- **Declaración de variables y librerías utilizadas (Comunicación con Cliente)** 145
11.2.2.- **Configuración Sistema (Con Cliente)** 148
11.2.3.- **Programa Principal (Con Cliente)** 149
11.2.4.- **Funciones e Interrupciones (Con Cliente)** 151
12.- ANEXO 5 PROGRAMACIÓN EN ANDROID 154
12.1.- Introducción Conceptos 154
12.2.- Metodos especiales utilizados en la Aplicación Cliente Android 157
12.3.- Código Cliente en Android 160
13.- ANEXO 6 CARACTERÍSTICAS DIMENSIONES MOTOR 168
14.- ANEXO 7 BOLA LOCA 170
14.1.- Despiece 170
15.- PLIEGO INTRODUCIÓN 172
16.- PLIEGO CONDICIONES GENERALES 172
17.- PLIEGO CONDICIONES DE ESPECIFICACIONES TÉCNICAS 172

17.1.- Materiales del Robot **172**
17.1.1.- **Aluminio** 172
17.1.2.- **ABS** 174
17.2.- Motores **175**
17.3.- Baterías de LITIO **176**
17.3.1.- **Introducción** 176
17.3.2.- **Carga** 177
17.3.3.- **Descarga** 178
17.3.4.- **Circuito de Protección** 178
17.3.5.- **Balanceo** 178
17.3.6.- **Características Cargador** 179
17.4.- HARDWARE **182**
17.4.1.- **Microcontrolador ARM M3 (ArduinoDue)** 182
17.4.2.- **Driver doble puente en H** 184
17.4.3.- **Modulo Shield wifi Arduino** 185
17.4.4.- **Dispositivo Android** 186
17.5.- Software **187**
18.- PRESUPUESTO INTRODUCIÓN **189**
19.- PRESUPUESTO RECURSOS UTILIZADOS **189**
19.1.- Recursos Hardware **189**
19.2.- Motores + Chasis + Bastidor + Batería, y varios **190**
19.3.- Licencias de software **190**
19.4.- Personal **190**
19.5.- Instalaciones **191**
20.- PRESUPUESTO DESGLOSE DE COSTES **191**
20.1.- Recursos Físicos **191**
20.2.- Licencias Software **192**
20.3.- Hardware + Material **192**
20.4.- Personal **193**
21.- PRESUPUESTO RESUMEN DEL PRESUPUESTO TOTAL **194**
22.- PLANOS **195**
22.1.- PLANO 1_A CHASIS. **195**
22.2.- PLANO 1_B CHASIS. **195**
22.3.- PLANO 2 BASTIDOR. **195**
22.4.- PLANO 3 CONEXIONES MOTOR-DRIVER-MICROCONTROLADOR **195**

INDICE DE FIGURAS Y TABLAS.

Figura 1.3_1: Modo de Funcionamiento Local 11
Figura 1.3_2: Modo de Funcionamiento remoto(Red) 11
Figura:1.4_1: Modo de Funcionamiento (Red) con cámara exterior 14
Figura 2.1_1: Modelo en tiempo Continuo 18
Figura 2.1_2: Transformada de Laplace 18
Figura 2.1_3: Tabla de Transformadas de Laplace 20
Figura 2.1_4: Izquierda Señal analógica continua. Derecha Señal Digital Discreta 21
Figura 2.1_5: Izquierda Efecto aliasing. Derecha Señal reconstruida sin aliasing 22
Figura 2.2.1_1: Accionamiento Diferencial 26
Figura 2.2.1_2: Sistemas de Referencia 27
Figura 2.2.4_1: Follow the Carrot 31
Figura 2.2.5_1: Pure Pursuit 33
Figura 2.2.5_2: Flujograma algoritmo Pure Pursuit 34
Figura 2.3_1: Sistema Control Continuo 37
Figura 2.3_2: Sistema Control Discreto 38
Figura 2.3_3: Sistema Control basado en Red 39
Figura 3.4_1: Soporte Motor 44
Figura 3.4_2: Unión de los Soportes mediante Perfil de Aluminio 44
Figura 3.5_1: Motor 45
Figura 3.5_2: Dimensiones y conector Motor 47
Figura 3.7_1: Arduino Due 51
Figura 3.8_1: Driver, Motores y Alimentación 54
Figura 3.8_2: Circuito Integrado L298P 55
Figura 3.9_1: Shield wifi Arduimo 56
Figura 4.1_1: Tareas Realizadas 58
Figura 4.1.1_1: Zona Muerta Motor 1 sentido adelante 61
Figura 4.1.1_2: Zona Muerta Motor 1 sentido atrás 61
Figura 4.1.2_1: K estática Motor 1 sentido adelante 63
Figura 4.1.2_2: K estática Motor 1 sentido atrás 64
Figura 4.2.3_1: Motor 1 Constante de Tiempo t sin filtro Velocidad 65
Figura 4.2.3_2: Motor 1 Constante de Tiempo t con filtrado de Velocidad 66
Figura 4.2.5_1: Zona Muerta Motor 2 sentido adelante 68
Figura 4.2.5_2: Zona Muerta Motor 2 sentido atrás 68
Figura 4.2.6_1: K estática Motor 2 sentido Adelante 70
Figura 4.2.6_2: K estática Motor 2 sentido Atrás 71
Figura 4.2.7_1: Motor 2 Constante de Tiempo t sin filtro Velocidad 72
Figura 4.2.7_2: Motor 2 Constante de Tiempo t con filtro Velocidad 73
Figura 4.3.2_1: Ajuste de la acción de Control 77
Figura 4.4_1: Despiece 78
Figura 4.5.1_1: Modo Funcionamiento Local Bucles de Control + Algoritmo seguimiento de Trayectoria 80
Figura 4.5.2_1: Modo de funcionamiento Local + Servidor Robot 85
Figura 4.5.4_1: Modo de funcionamiento Local Cliente Android 94
Figura 4.5.4_2: Interfaz Gráfica Android 96
Figura 4.5.5_1: Control basado en Red diagrama funcional 98
Figura 4.5.5_2: Control basado en Red diagrama temporal-funcional 99
Figura 4.5.5_3: Algoritmo seguimiento trayectoria Bloques Matlab / Simulink 100

Figura 4.5.5_4: Equivalencia del subsistema bloque A Simulink con el código en C 101
Figura 4.5.5_5: Bloque B S-FUNTION + Predictor de Smith 102
Figura 4.5.5_6: Predictor de Smith 103
Figura 4.5.5_7: Equivalencia del subsistema bloque D Simulink con el código C 104
Figura 4.5.5_8: Comparador de Distancia 105
Figura 4.5.5_9: Interfaz de usuario Matlab / Simulink 106
Figura 5.1.1_1: Trayectoria Ruedas Cuadrada 107
Figura 5.1.2_1: Trayectoria Cuadrada (en el suelo) 108
Figura 5.1.2_2: Trayectoria figura Lissajous-1 (en el suelo) 109
Figura 5.1.3_1: Trayectoria figura Lissajous-1 (en el suelo) 110
Figura 5.2.1_1: Trayectoria Ruedas sin contacto con el Suelo Android 111
Figura5.2.2_1: Trayectoria Ruedas sin contacto con el Suelo Android 112
Figura 5.3.1_1: Trayectoria Cuadrada 113
Figura 5.3.1_2: Evolución Temporal Coordenadas xc e yc 114
Figura 5.3.1_3: Evolución Temporal Velocidades Wd y Wi 114
Figura 5.3.1_4: Evolución Temporal D, Th, Vl, W 115
Figura 5.3.2_1: Trayectoria Cuadrada 115
Figura 5.3.2_2: Evolución Temporal xc e yc 116
Figura 5.3.2_3: Evolución Temporal Velocidades Wd y Wi 116
Figura 5.3.2_4: Evolución Temporal D, Th, Vl, W 117
Figura 5.3.3_1: Trayectoria Lissajous-1 117
Figura 5.3.3_2: Evolución Temporal xc e yc 118
Figura 5.3.3_3: Evolución Temporal Velocidades Wd y Wi 118
Figura 5.3.3_4: Evolución Temporal D, Th, Vl, W 119
Figura 5.3.4_1: Trayectoria Lissajous-1 119
Figura 5.3.4_2: Evolución Temporal xc e yc 120
Figura 5.3.4_3: Evolución Temporal Velocidades Wd y Wi 120
Figura 5.3.4_4: Evolución Temporal D, Th, Vl, W 121
Figura 8.2_1: Proceso de carga baterías de Litio 128
Figura 8.2_2: Cargador de baterías de Litio 128
Figura 8.3_1: Proceso de descarga baterías de Litio 129
Figura 9.1_1: Driver Ardumoto 131
Figura 9.1_2: Pines Driver Ardumoto 132
Figura 9.2_1: Formatos del chip 133
Figura 9.2_2: Pines L298 134
Figura 12.1_1: Ciclo de vida de una Actividad oficial de Android 156
Figura 12.2_1: Hilos y Concurrencia 158
Figura 13_1: Dimensiones 168
Figura 13_2 : Conexiones 169
Figura 14.1_1: Despiece Bola Loca 170
Figura 17.1.1_1: Soporte Motor 173
Figura 17.1.1_2: Unión de los Soportes mediante Perfil de Aluminio 173
Figura 17.2_1: Dimensiones Motores 175
Figura 17.3.2_1: Cargador de baterías de Litio 177
Figura 17.4.2_1: Doble Puente en H Ardumoto 184
Figura 17.4.3_1: Shield wifi Arduino 185
Figura 17.4.4_1: Tablet ASUS ZenPad10 186
Tabla 19.1_1: Lista Material Hardware 189
Tabla 19.2_1: Lista Material Motores + Bastidor, varios 190
Tabla 19.3_1: Lista Material Software 190

Tabla 19.4_1: Lista Personal 190
Tabla 19.5_1: Lista Instalaciones 191
Tabla 20.1_1: Lista Costes Recursos Físicos 191
Tabla 20.2_1: Lista Costes Licencias Software 192
Tabla 20.3_1: Lista Costes Hardware + Material 193
Tabla 20.4_1: Lista Costes Personal 193
Tabla 21_1: Resumen Total 194

1.- INTRODUCCIÓN

1.1.- Robots Móviles

En los últimos tiempos los robots móviles o vehículos auto guiados (AVG), han cobrado importancia en todos los procesos de fabricación industrial, cuando se persigue cierta flexibilidad en el propio proceso, para poder elaborar productos distintos con la misma cadena de producción.

Es sabido que existen abiertas importantes líneas de investigación, con avances destacados, y con el objetivo, de no solo aplicar dichos avances en los procesos industriales, sino también en ambientes domésticos (robots que realizan tareas domésticas), en medicina (robots cirujanos), en agricultura (muestreo de suelos y cultivos), en vehículos autónomos y sin conductor, localización etc.

La navegación autónoma que puede plantearse tanto en espacios interiores como en espacios exteriores, requiere de robots móviles dotados de sensores que puedan seguir una trayectoria prefijada y que en el caso de detectar un obstáculo sean capaces de sortearlo, con el fin de llegar al destino y cumplir con el objetivo propuesto.

La plataforma control-hardware-software-sistema mecánico, debe ser la apropiada para solucionar el reto citado anteriormente de poder seguir una trayectoria con precisión, y poder detectar la presencia de un obstáculo para corregir la trayectoria a fin de poder llegar al destino.

El control de velocidad, el control de posición, la lógica adecuada del sistema, la precisión de los sensores y la monitorización del proceso, son todos puntos muy importantes que aseguran el éxito del sistema.

Todos los puntos citados anteriores nos acercan al concepto de inteligencia artificial que se presenta como la solución para los vehículos autónomos.

A pesar de que el concepto de inteligencia artificial es complicado, puesto que podemos entenderlo como la capacidad del robot de poder aprender de los errores y poder de esta forma reprogramarse, el hecho de dar lógica de decisión al robot, control de velocidad o de posición, la aplicación de algoritmos de seguimiento y sensores que permiten detectar cambios en el espacio de trabajo, implican ya cierto grado de aproximación al concepto de inteligencia artificial.

Teniendo en cuenta las exigencias actuales y los nuevos retos que se plantean, el fuerte carácter investigador del departamento de Ingeniería de Sistemas y Automática y el Master de origen, se ha abordado este proyecto con un doble enfoque, investigador, por una parte, y producto vehículo industrial por otro lado.

1.2.- Objetivo

El objetivo de este proyecto es el de diseñar e implementar un sistema basado en un vehículo terrestre (Robot), que sea capaz de cubrir una determinada trayectoria, mediante el control de velocidad de dos ruedas accionadas mediante dos motores de corriente continua (accionamiento diferencial), y un algoritmo de seguimiento de trayectoria (*Pure Pursuit*).

El control de la Trayectoria o la trayectoria a seguir se comunica al Robot mediante interfaz inalámbrica (Wifi). Desde el punto de vista de la comunicación del robot con el mundo exterior, para que este conozca la trayectoria a seguir, habrá siempre un dispositivo externo que podrá ser un PC o un dispositivo Android, que será el cliente y el Robot hará de servidor.

El sistema en conjunto tendrá dos modalidades de funcionamiento, funcionamiento en modo remoto y funcionamiento en modo local. En el caso de funcionar el sistema en modo remoto (Red) el algoritmo de seguimiento de trayectoria se ejecuta en la máquina cliente, y existe una continua comunicación entre cliente y servidor, mientras que en el funcionamiento en modo local una vez comunicada la trayectoria, el servidor (Robot) pasa a la ejecución del seguimiento de la trayectoria, pudiéndose interrumpir la comunicación y el Robot sigue trabajando hasta terminar la trayectoria solicitada.

Es importante reseñar que se trata en conjunto de un sistema robótico móvil de bajo coste, cuyos sensores más importantes son los encoders acoplados a los motores, que son necesarios para poder realizar el control de velocidad y la odometría, asegurando la eficacia del algoritmo de seguimiento de trayectoria. El diseño original podría ser modificado añadiendo sensores detectores de obstáculos y otros como una brújula electrónica, que permitirían por una parte la interrupción o corrección del seguimiento de trayectoria para evitar un obstáculo, y por otra parte la corrección del error acumulado que se puede producir en la orientación.

También indicar que es un sistema de navegación para interiores, que en el caso de una aplicación para exteriores requeriría cambiar la morfología del Robot y cabría la posibilidad de poder utilizar junto con los sensores anteriores

un GPS, que permitiría en todo momento comprobar si la trayectoria del Robot es correcta pudiendo subsanar los errores acumulados en este caso con respecto a la odometría, pasando de un sistema de referencia local a uno global.

Las distintas modalidades de funcionamiento (local o control remoto basado en red), han exigido que la orientación del proyecto sea investigadora, pero buscando la aplicación del sistema en el mundo industrial.

1.3.- Problema Planteado

El control de la trayectoria a seguir se elabora mediante un algoritmo de seguimiento de trayectoria, llamado persecución pura (*Pure Pursuit*) y se tiene que poder comunicar al robot bien desde Matlab/Simulink (PC), a través de un control basado en red, donde el algoritmo de seguimiento de trayectoria está en Matlab/Simulink y el control de velocidad de las ruedas (accionamiento diferencial) en modo local en el Robot, o bien desde un dispositivo Android donde tanto el control de velocidad de las ruedas como el algoritmo de seguimiento de trayectoria están en forma local en el Robot, y desde Android se proporciona la trayectoria o tarea a seguir por el Robot, además de poder visualizar el proceso. Para que de esta forma una vez comunicado al Robot la tarea, poder interrumpir la comunicación desde el dispositivo Android y que el Robot continúe su tarea de forma independiente (enfoque Robot Industrial).

Por lo tanto, tenemos dos modos de funcionamiento Modo control Remoto y Modo control Local.

Tanto en el modo control remoto (Red), como en el caso del control local existen tres subsistemas importantes, dos de ellos son el bucle de control de velocidad para cada una de las ruedas y el algoritmo de seguimiento de trayectoria. El funcionamiento óptimo de estos dos subsistemas asegura el éxito del seguimiento de la trayectoria, y el tercer subsistema es la interfaz gráfica del usuario para poder interaccionar con el sistema.

A continuación, se muestran los diagramas de bloques, Figura 1.3_1 modo de funcionamiento local y Figura 1.3_2 modo de funcionamiento remoto:

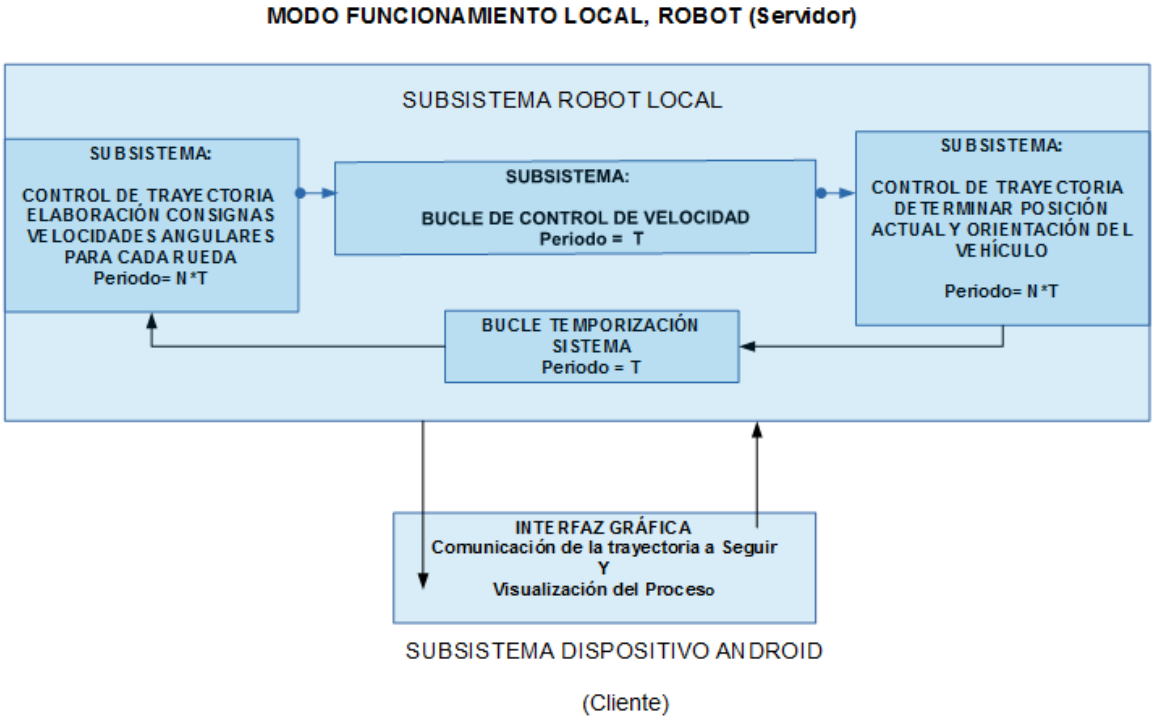


Figura 1.3_1: Modo de Funcionamiento Local

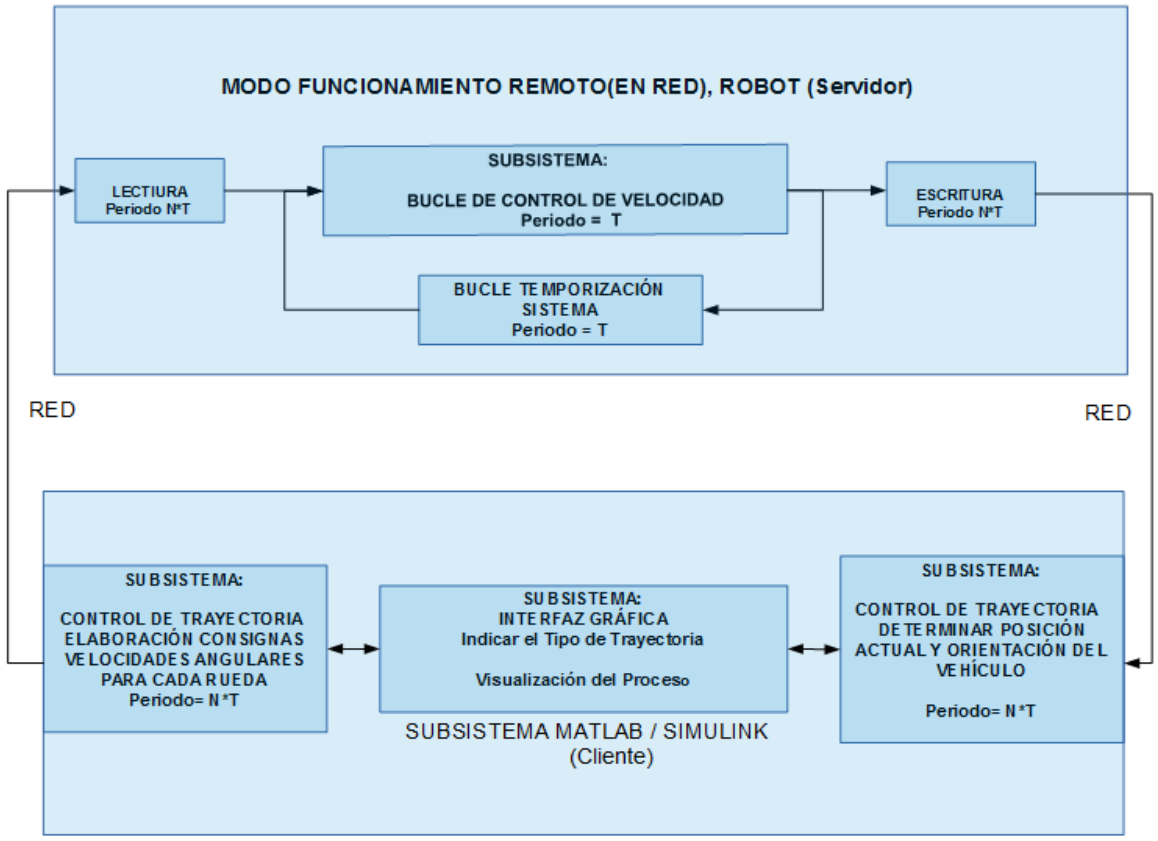


Figura 1.3_2: Modo de Funcionamiento remoto(Red)

El modo remoto (control basado en Red) presenta una serie de ventajas desde el punto de vista investigador, que son:

- La posibilidad de utilizar observadores y predictores que permiten corregir una serie de problemas típicos del control en red, o no, como en el caso de filtrado de ruidos.
- Aplicar tota la teoría de control clásica y moderna, que esta potente herramienta Matlab/Simulink nos proporciona a fin de poder experimentar todos los problemas que presenta el control de un sistema real (Robot) basado en una conexión en red (enfoque investigador).

Hay que tener en cuenta que el Robot tiene que poseer un sistema micro controlado que no consuma excesiva energía, con lo cual no puede llevar a bordo un microprocesador muy potente. Es en este caso donde cobra importancia el control en red, puesto que el PC donde se están ejecutando los algoritmos de seguimiento de trayectoria e incluso el del bucle de control de velocidad pueden ser todo lo complejos que se quiera, ya que la capacidad de cálculo es claramente mayor que la del microcontrolador o sistema micro controlado, y no existe el problema de la limitación energética.

Por otra parte, en el enfoque Robot Industrial posee la ventaja de que al disminuir el tiempo que el Robot está comunicando con el dispositivo Android, el módulo wifi que forma parte del Robot consume mucha menos energía que en el caso de tener que estar comunicando continuamente para poder realizar la trayectoria o tarea.

Existe una tercera posibilidad que no se ha implementado, mediante la cual el dispositivo Android comunicaría con el PC donde esta Matlab/Simulink, el PC mediante Matlab/Simulink, al igual que en el enfoque investigador, es donde estaría el algoritmo de seguimiento de trayectoria que se comunicaría mediante la interfaz inalámbrica con el Robot, que es el que tiene implementado el bucle de control de velocidad.

1.4.- Relevancia del tema y Aportación

En general la idea es la de conseguir cualquier tipo de Robot o vehículo móvil no tripulado.

La aplicación más inmediata de este tipo de vehículos desde el punto de vista en espacios interiores sería la posibilidad de introducir Robots móviles capaces de transportar material o herramientas de un lugar a otro dentro de naves industriales donde se realiza la fabricación de cualquier tipo de producto, como por ejemplo en procesos de fabricación en cadena.

En el caso de navegación exterior no es necesario remarcar la necesidad hoy en día de vehículos inteligentes que pudieran circular por carretera tanto en ciudad como en ruta, que sean capaces de sustituir al ser humano y que, estando conectados a información exterior, a la nube (bases de datos en tiempo real del tráfico), puedan evitar congestiones de tráfico, pudiendo tomar rutas alternativas.

Por otra parte también, desde el punto de vista de la investigación, sirve como planta o sistema para poder aplicar nuevas estrategias de control con el fin de poder evaluar dichas estrategias de control.

La aportación es la de un Robot móvil, que mediante el control de velocidad nos permite aplicar con éxito el algoritmo de seguimiento de trayectoria, que es el que asegura el seguimiento de la trayectoria por parte del Robot.

No hay que olvidar que el robot queda siempre como el servidor y que el cliente en las soluciones adoptadas, puede ser tanto Matlab / Simulink, desde un ordenador, como un dispositivo Android. Y queda abierta la posibilidad de aportar nuevos sensores como acelerómetros o una cámara que darían la solución definitiva al Robot industrial.

Al hilo de lo anterior una posibilidad muy interesante que se está probando e investigando cómo continuación del proyecto en el DISA (Departamento de Ingeniería de Sistemas y Automática), en interiores, es la de utilizar una cámara exterior al Robot móvil, Figura 1.4_1, que pueda facilitar información al Robot para que de esta forma puedan corregirse los errores acumulado de orientación y de medida de distancia (sensorización exteroceptiva).

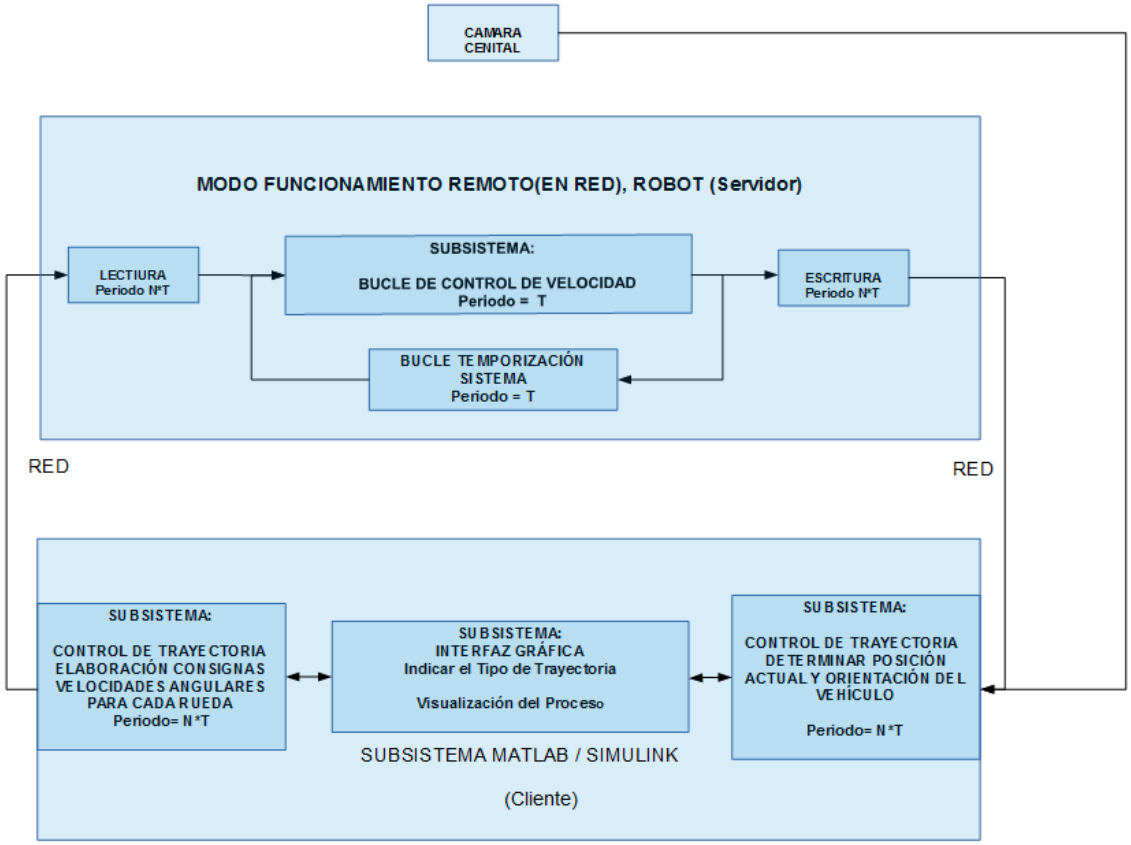


Figura:1.4_1: Modo de Funcionamiento (Red) con cámara exterior

1.5.- Estructura del trabajo

CAPÍTULO 1: **Introducción.**

Consta de una breve introducción a los Robots Móviles, continuando por el objetivo del proyecto con el doble enfoque investigador por una parte y vehículo industrial por otro lado, para pasar a explicar finalmente cual es el problema planteado y la relevancia del tema, así como la aportación ofrecida para la solución del problema planteado.

CAPITULO 2: **Fundamentos Teóricos.**

A pesar de que los fundamentos teóricos necesarios para poder llevar a cabo el proyecto son diversos, puesto que es una solución Mecatrónica, en este apartado se explican tres conceptos teóricos de vital importancia que son y por este orden:

- Teoría de control de Velocidad.

- Algoritmo de Seguimiento de Trayectoria.
- Control de procesos basados en Red.

CAPITULO 3: **Elementos del sistema, Robot.**

En este capítulo para hacernos una idea de todos los elementos que conforman el sistema, primeramente, se muestran tres diagramas de bloques que pertenecen al Diagrama de bloques del sistema Mecánico, el Diagrama de bloques del sistema Energético y el diagrama de bloques del sistema Electrónico, para pasar a continuación a describir:

- Características del Chasis + Bastidor soporte motores.
- Características Motores y Encoders.
- Características Batería y Condensadores.
- Características Microcontrolador ARM Cortex M3 Arduino Due.
- Características Driver doble Puente en H.
- Características Módulo Wifi.

CAPITULO 4: **Desarrollo y Plan de Trabajo.**

Es el capítulo más intenso y comienza con un esquema-cronograma donde se muestran todas las tareas realizadas. Posteriormente se explica el proceso de identificación de cada uno de los motores con todos los pasos dados, para identificar la FDT y la zona muerta de cada uno de los motores, con el objetivo de poder calcular el regulador PI adecuado para cada uno de los motores, que posteriormente se ha ajustado experimentalmente hasta obtener la mejor respuesta.

El paso siguiente es el diseño del chasis y el bastidor, mediante el software NX 10 de Siemens. Finalmente se exponen los flujogramas de funcionamiento de todos los programas desarrollados junto con una explicación en cada uno de ellos.

CAPITULO 5: **Resultados Obtenidos**.

Se exponen y comentan los resultados obtenidos para cada una de las modalidades de funcionamiento del Robot mediante el apoyo con gráficas.

CAPITULO 6: **Conclusiones.**

Se explican todos los logros obtenidos y de forma crítica y constructiva, se aclara cual ha sido el principal problema que ha frenado la obtención de un seguimiento de trayectoria más preciso. A continuación, se plantean las soluciones a dicha problemática mediante dos enfoques de aplicación conjunta que son:

- Sensorización **Propioceptiva**.
- Sensorización **Exteroceptiva**.

También Aplicaciones del Proyecto y proyección futura del mismo.

CAPITULO 7: **Bibliografía.**

CAPITULO 8: **Anexos.**

CAPITULO 9: **Pliego de Condiciones Técnicas.**

CAPITULO 10: **Presupuesto.**

CAPITULO 11: **Planos.**

2.- FUNDAMENTACIÓN TEÓRICA

2.1.- Teoría de Control, Control de Velocidad

Para la implementación de un control de velocidad el primer paso es identificar la planta o sistema, que son cada uno de los motores tanto en carga nominal como en vacío, como se verá en el Capítulo 4, apartado 4.2 Identificación de los Motores, para poder posteriormente calcular el regulador PI.

Por una parte, es necesario entender que todo sistema o planta para poderlo modelar, es preciso encontrar las ecuaciones diferenciales que definen el comportamiento del sistema. Estas ecuaciones están expresadas siempre en el dominio del tiempo y son complicadas de manejar y tratar desde el punto de vista del control.

Para analizar los sistemas y poder calcular el regulador adecuado para controlar la magnitud física que se desea, en este caso la velocidad angular de las ruedas, es necesario en control, aplicar un método que simplifique el manejo de dichas ecuaciones diferenciales. Este método consiste en la transformación de dichas ecuaciones expresadas en el dominio del tiempo, a otras ecuaciones expresadas en el dominio de la frecuencia, y esto lo conseguimos aplica la transformada de Laplace.

A continuación, se muestra un diagrama de bloques con el sistema completo(modelo), donde tanto la planta como el regulador están expresados con funciones algebraicas en el domino de la frecuencia (s), que es el modelo que se ha obtenido del capítulo 4, apartado 4.3 Cálculo experimental de regulador PI de Velocidad (simulación en tiempo continuo).

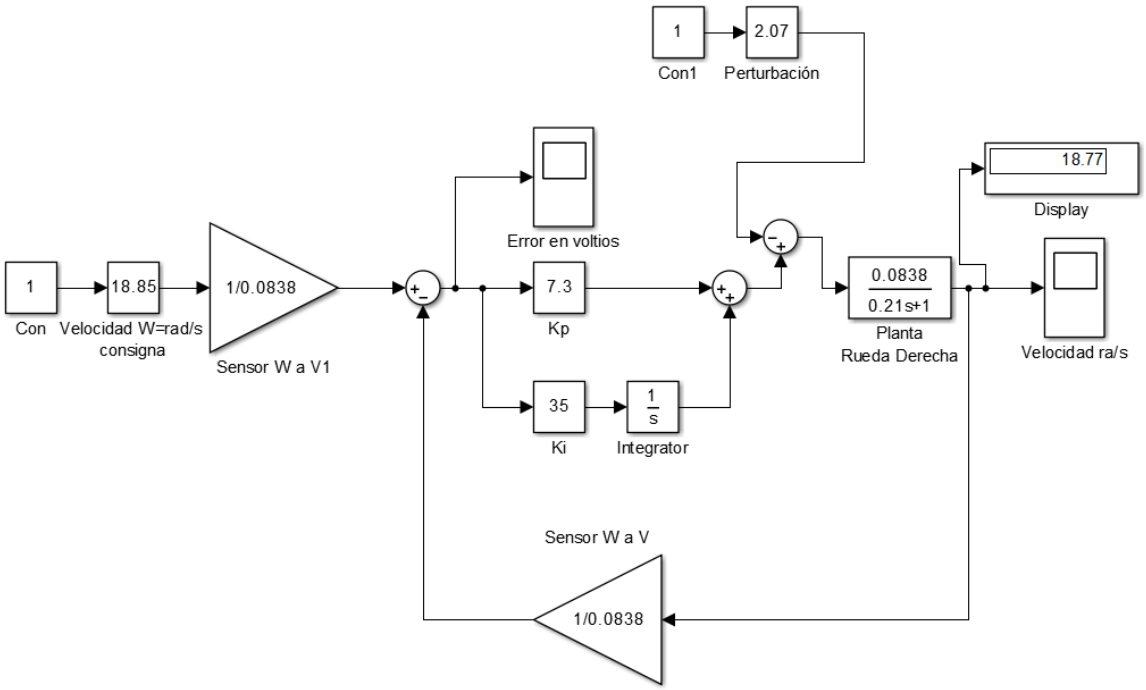


Figura 2.1_1: Modelo en tiempo Continuo

Por otra parte, para entender el control de velocidad mediante un sistema digital y con muestras discretas, es necesario introducir una serie de conceptos relacionados con este tipo de señales.

Pasemos primero a hablar de la transformada de Laplace y posteriormente de las señales digitales discretas.

Transformada de Laplace:

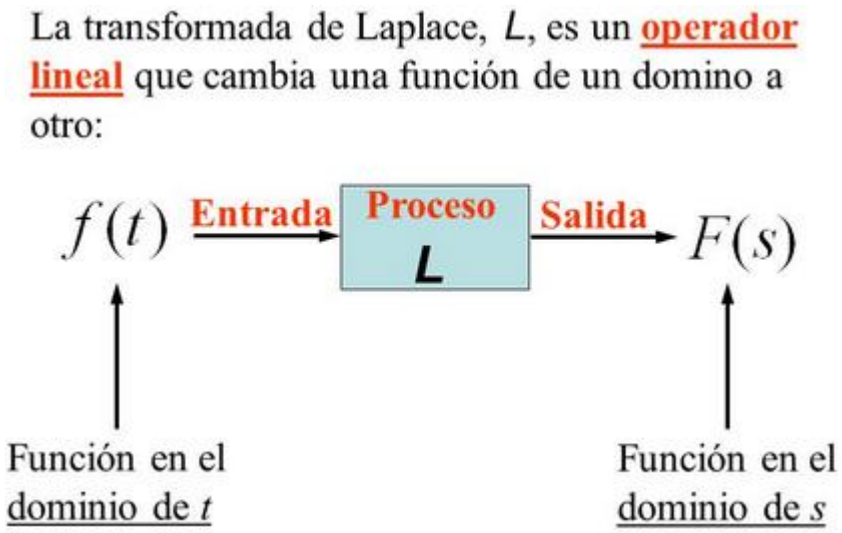


Figura 2.1_2: Transformada de Laplace

Mediante la transformada de Laplace conseguimos transformar las ecuaciones integro-diferenciales en ecuaciones sencillas algebraicas.

Una vez pasadas las ecuaciones al domino de la frecuencia compleja, mediante la transformada de Laplace ya podemos operar con las ecuaciones algebraicas como se aprecia en el diagrama de bloques anterior de la Figura 2.1_2.

Sabida la respuesta del sistema en el domino de la frecuencia aplicamos la Transformada Inversa de Laplace, para saber la respuesta del sistema en el domino del tiempo que es más natural de entender.

La expresión de la transformada de Laplace para cualquier tipo de función continua en el domino del tiempo la expresamos como:

$$F(s) = L\{f(t)\} = \int_0^{\infty} f(t) * e^{-s*t}\, dt$$

Para poder obtener la función equivalente en el domino de la frecuencia, es necesario resolver la integral para una función determinada y aplicar el siguiente límite.

$$\lim_{p \to \infty} \int_0^{p} f(t) * e^{-s*t}\, dt$$

Existe Transformada de Laplace para cualquier función f(t) continua en el intervalo de 0 a ∞ y que el valor absoluto de la expresión [$f(t) * e^{-s*t}$]sea convergente.

A partir de este fundamento teórico y resolviendo la integral anterior para cada una de las funciones en el dominio del tiempo, que en nuestro caso representa la señales que estamos procesando como la velocidad de cada rueda, y aplicando el limite anterior obtenemos una tabla como la que se muestra a continuación Figura 2.1_3, que relaciona la equivalencia de una función en el dominio del tiempo con su respectivo equivalente en el domino de la frecuencia.

$f(t)$	$F(s)$	$f(t)$	$F(s)$
$a_1 f_1(t) + a_2 f_2(t)$	$a_1 F_1(s) + a_2 F_2(s)$	$\delta(t)$	1
$f(at)$	$\frac{1}{a} F\left(\frac{s}{a}\right)$	$u(t)$	$\frac{1}{s}$
$f(t-a)u(t-a)$	$e^{-as} F(s)$	e^{-at}	$\frac{1}{s+a}$
$e^{-at} f(t)$	$F(s+a)$		
$\frac{df}{dt}$	$sF(s) - f(0^-)$	t	$\frac{1}{s^2}$
$\frac{d^2 f}{dt^2}$	$s^2 F(s) - sf(0^-) - f'(0^-)$	t^n	$\frac{n!}{s^{n+1}}$
$\frac{d^3 f}{dt^3}$	$s^3 F(s) - s^2 f(0^-) - sf'(0^-) - f''(0^-)$	te^{-at}	$\frac{1}{(s+a)^2}$
$\frac{d^n f}{dt^n}$	$s^n F(s) - s^{n-1} f(0^-) - s^{n-2} f'(0^-) - \cdots - f^{(n-1)}(0^-)$	$t^n e^{-at}$	$\frac{n!}{(s+a)^{n+1}}$
$\int_0^t f(t)\,dt$	$\frac{1}{s} F(s)$	$\sin \omega t$	$\frac{\omega}{s^2+\omega^2}$
$tf(t)$	$-\frac{d}{ds} F(s)$	$\cos \omega t$	$\frac{s}{s^2+\omega^2}$
$\frac{f(t)}{t}$	$\int_s^\infty F(s)\,ds$	$\sin(\omega t + \theta)$	$\frac{s \sin\theta + \omega \cos\theta}{s^2+\omega^2}$
$f(t) = f(t+nT)$	$\frac{F_1(s)}{1-e^{-sT}}$	$\cos(\omega t + \theta)$	$\frac{s \cos\theta - \omega \sin\theta}{s^2+\omega^2}$
$f(0^+)$	$\lim_{s\to\infty} sF(s)$	$e^{-at} \sin \omega t$	$\frac{\omega}{(s+a)^2+\omega^2}$
$f(\infty)$	$\lim_{s\to 0} sF(s)$		
$f_1(t) * f_1(t)$	$F_1(s) F_2(s)$	$e^{-at} \cos \omega t$	$\frac{s+a}{(s+a)^2+\omega^2}$

Figura 2.1_3: Tabla de Transformadas de Laplace

Si observamos la Figura 2.1_1 Modelo en Tiempo continuo de Matlab / Simulink vemos que tanto la planta como el regulador están expresados en el dominio de la frecuencia (s):

Señales Digitales Discretas:

En un sistema de control analógico o continuo, las señales las podemos representar en forma de funciones continuas, mientras que en un sistema digital como es nuestro caso, las señales las representamos mediante secuencias discretas. Ver ejemplos siguientes Figuras 2.1_4:

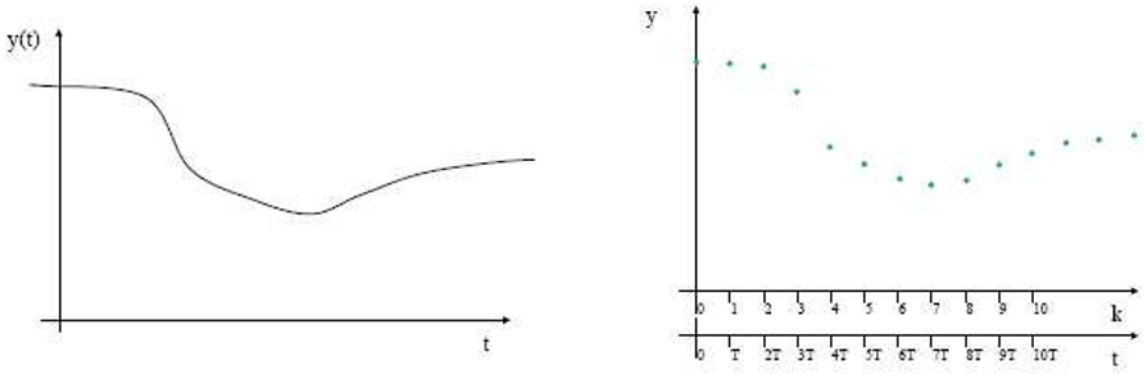


Figura 2.1_4: Izquierda Señal analógica continua. Derecha Señal Digital Discreta

Las señales discretas son muestras de la magnitud que se desea controlar espaciadas en el tiempo con un periodo exacto T, que se conoce como periodo de muestreo. Cada uno de los valores son las muestras y se identifican por el número de muestra *k.* En la figura anterior Figura 2.1_4 a la derecha se observa una serie de muestras $y_{k=\{y(ko),\ y(k1)\ldots\ldots\ldots\ldots\ldots\}}$ que provienen de la señal analógica de la izquierda.

Si queremos saber el tiempo transcurrido en el momento en que se toma la muestra k7, solo tenemos que multiplicar k7•T.

Es evidente que a menor periodo de muestreo mayor es la probabilidad de reconstruir la señal muestreada, consiguiendo así un sistema que digamos tiene los ojos cerrados menos tiempo entre transiciones del periodo de muestreo, reteniendo mayor número de muestras en un mismo tiempo, y evitando la posible pérdida de información cuando se muestrean señales rápidas. Este fenómeno de perdida de información se conoce como (efecto *aliasing*). Esto ocurre porque no cumplimos con el teorema de Nyquist que indica que para evitar este fenómeno y por lo tanto poder reconstruir la señal con precisión, la frecuencia de muestreo debe ser 2 veces mayor que la frecuencia máxima de la señal que se pretende muestrear.

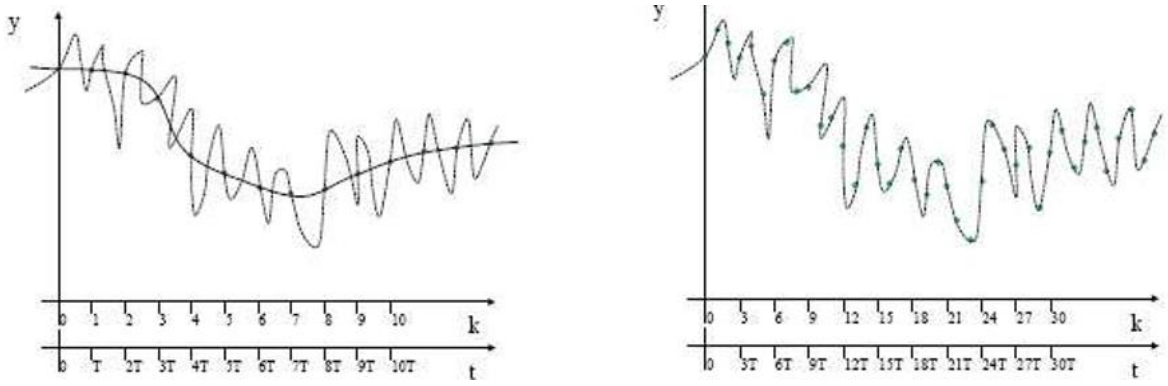


Figura 2.1_5: Izquierda Efecto aliasing. Derecha Señal reconstruida sin aliasing

Para diseñar el regulador sabiendo que el sistema es discreto (digital), lo podemos hacer de dos formas distintas. Trabajar directamente con el sistema discreto en (z), y utilizando la metodología aplicable a estos sistemas discretos, o bien como se ha procedido en este caso, aplicando los conocimientos de sistemas continuos y trabajando como si se tratara de un sistema continuo, para posteriormente discretizar.

Para la discretización del controlador continuo como el que se ha obtenido en el capítulo 4, apartado 4.3 Cálculo experimental de regulador PI de Velocidad, se ha utilizado el método aproximado de la derivada.

Este método consiste en la aproximación por la derivada, mediante la aproximación por la pendiente de la recta que pasa por dos muestras consecutivas.

Si discretizamos en el domino del tiempo (t) tenemos:

$$\frac{d_{y(t)}}{dt} \quad \rightarrow \quad \frac{y_k - y_{k-1}}{T}$$

Si aplicamos la Transformada de Laplace y discretizamos se obtiene en el domino de la frecuencia (s):

$$s * Y_{(s)} \rightarrow \quad \frac{1 - z^{-1}}{T} \quad * Y_{(s)}$$

Luego para obtener el regulador discreto a partir del continuo es suficiente sustituir las s por $\frac{1-z^{-1}}{T}$.

Como veremos en el apartado citado anteriormente 4.3 del capítulo 4 Cálculo experimental de regulador PI de Velocidad, el regulador seleccionado, es un control proporcional-Integral, y como se sabe la función de transferencia del control PI en el dominio de la frecuencia se puede expresar:

$$Gr(s) = K\left(1 + \frac{1}{Ti \bullet s}\right) = Kp + Ki * \frac{1}{s} = \frac{Kp * s + Ki}{s} = Kp\frac{s + \frac{Ki}{Kp}}{s}$$

Si procedemos separando la acción proporcional por un lado y la integral por otro lado y nos centramos en la acción integral tenemos que al discretizar queda:

$$GI(s) = \frac{UMI}{e_{rror}} = Ki * \frac{1}{s} = \frac{UMI}{Ki} = error * \frac{1}{\frac{1-z^{-1}}{T}}$$

$$\frac{UMI}{Ki} = error * \frac{\mathrm{T}}{1 - z^{-1}} = error * \frac{\mathrm{T}}{1 - k_{-1}}$$

$$\mathrm{UMI} * (1 - k_{-1}) = Ki * error * \mathrm{T}$$

$$\mathrm{UMI} - \mathrm{UMI}\ k_{-1} = Ki * error * \mathrm{T}$$

$$\mathrm{UMI} = Ki * error * \mathrm{T} + \mathrm{UMI}\ (\mathrm{k} - 1)$$

Por lo tanto, para calcular la acción integral en el controlador digital, es decir en el software implementado, en cada vuelta de periodo de muestreo del bucle de control de velocidad, lo que hacemos es:

- Primero calculamos el error actual como la diferencia entre la velocidad consigna y la velocidad real a la que circula el vehículo, multiplicado por el periodo de muestreo, tiempo que estamos a ciegas sin tener datos de la señal de velocidad y le sumamos el error calculado y por lo tanto acumulado en las vueltas de bucle anteriores, como se muestra a continuación:

  $Error\ acumulado = \ error * T$ + Error Acumulado anterior

- Y a continuación multiplicamos dicho error por la constante Ki calculada:

Quedando la acción de control como sigue:

$$UMI = Ki * Error\ Acumulado$$

A continuación, se muestra la línea de código que hace esto:

```
Ik=  EM1*Tm + Ik  ;
UMI = KiM1*Ik
```

La acción proporcional no tiene ningún misterio y consiste en multiplicar el error actual por la constante Kp.

2.2.- Algoritmo de Seguimiento de Trayectoria.

Dead Reckoning es un método de seguimiento a estima de trayectoria para vehículos o robots, que teniendo en cuenta la velocidad actual de ambas ruedas, determina la posición actual del robot y además genera las velocidades angulares adecuadas que permiten alcanzar un punto determinado en el plano XY. O más simplemente, es un proceso de estimación de la posición de un vehículo basado únicamente en la velocidad, la dirección y el tiempo transcurrido desde la última posición conocida.

El concepto de navegación a estima se deriva muy probablemente de la expresión "*deduced reckoning* ", utilizada por los marineros del siglo XVII. La mayoría de los vehículos o robots actuales se basan en este tipo de navegación a estima. Una forma sencilla de implementar la navegación a estima es por odometría. La odometría es un método para calcular el desplazamiento realizado por un vehículo en función de la rotación de las ruedas.

Puesto que en interiores no tenemos la posibilidad de utilizar un GPS, nos vemos obligados a obtener el desplazamiento angular o rotación de las ruedas a partir de un encoder ajustado al eje del motor, que genera una señal en forma de pulsos que informa del ángulo girado de las ruedas en función de la cantidad de pulsos que se obtienen en un tiempo. Esta información junto con las ecuaciones cinemáticas del vehículo, nos proporcionan el cambio en la posición del vehículo y la orientación del mismo.

Para la obtención de la posición se necesita el uso de la cinemática directa y para seguir la trayectoria deseada se utiliza lo que se conoce como accionamiento diferencial.

2.2.1.- **Accionamiento Diferencial**

La teoría que se esconde debajo del accionamiento diferencial es bastante sencilla. Todo robot móvil en estado de movimiento gira alrededor de un punto que se encuentra en algún lugar del eje común a ambas ruedas. Este punto se conoce como el centro instantáneo de curvatura, o el centro instantáneo de rotación. Variando la velocidad de las dos ruedas se consigue variar el centro instantáneo de rotación.

En la siguiente Figura 2.2.1_1 se refleja la idea anterior.

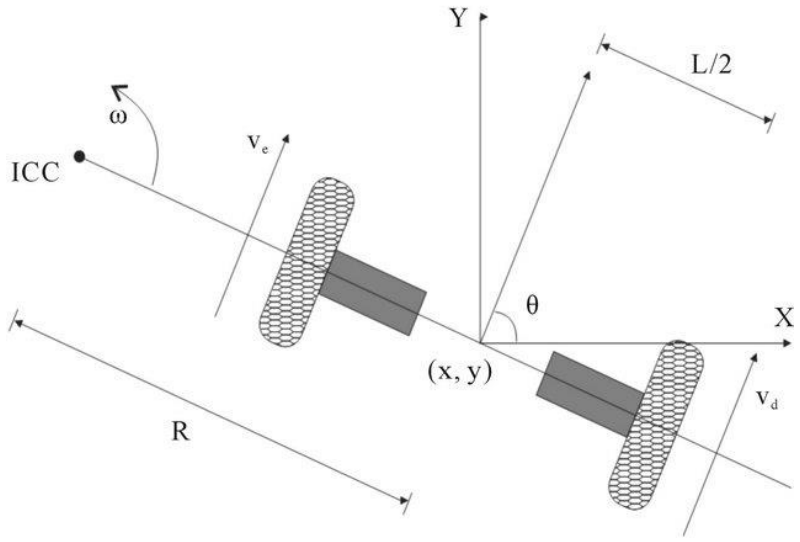


Figura 2.2.1_1: Accionamiento Diferencial

Cada rueda sigue una trayectoria que se mueve alrededor del centro de curvatura ICC, con la misma velocidad angular ω, (velocidad de rotación del vehículo):

$$\boldsymbol{Vd(t) = \omega(t) * \left(R + \frac{L}{2}\right)} \qquad \boldsymbol{Vi(t) = \omega(t) * \left(R - \frac{L}{2}\right)}$$

$$\boldsymbol{R = \frac{1}{2} * \frac{Vd(t)+Vi(t)}{Vd(t)-Vi(t)}} \qquad \boldsymbol{\omega(t) = (Vr(t) - Vi(t))/L}$$

L *es la distancia que separa las dos ruedas.*

Vd *la velocidad lineal de la rueda derecha.*

Vi *la velocidad lineal de la rueda izquierda.*

R *es la distancia desde el punto donde se localiza el centro de curvatura y el punto medio entra las ruedas.*

ω *es la velocidad angular de rotación del vehículo respecto al ICC.*

Tenemos dos casos especiales que se derivan de estas ecuaciones:

Si Vi = Vd el radio de curvatura es infinito y el robot se mueve en una línea recta.

Si Vi= -Vd el radio de curvatura es cero y el robot gira sobre sí mismo.

En todas las demás situaciones el robot gira describiendo una trayectoria alrededor del centro de curvatura a una cierta velocidad angular.

Establecidos los fundamentos anteriores, se pueden derivar las ecuaciones de la cinemática directa del robot.

El concepto clave para el control del robot es cómo cambian las coordenadas (xc, yc) del centro del eje del vehículo y la orientación de este punto.

Θ es el ángulo de orientación, tomado respecto del eje X como se muestra en la imagen siguiente, Figura 2.2.1_2, si está a la derecha del eje X hacia el eje Y, en avance anti-horario, entonces se considera positivo, en caso contrario en sentido horario, Θ se considera un ángulo negativo.

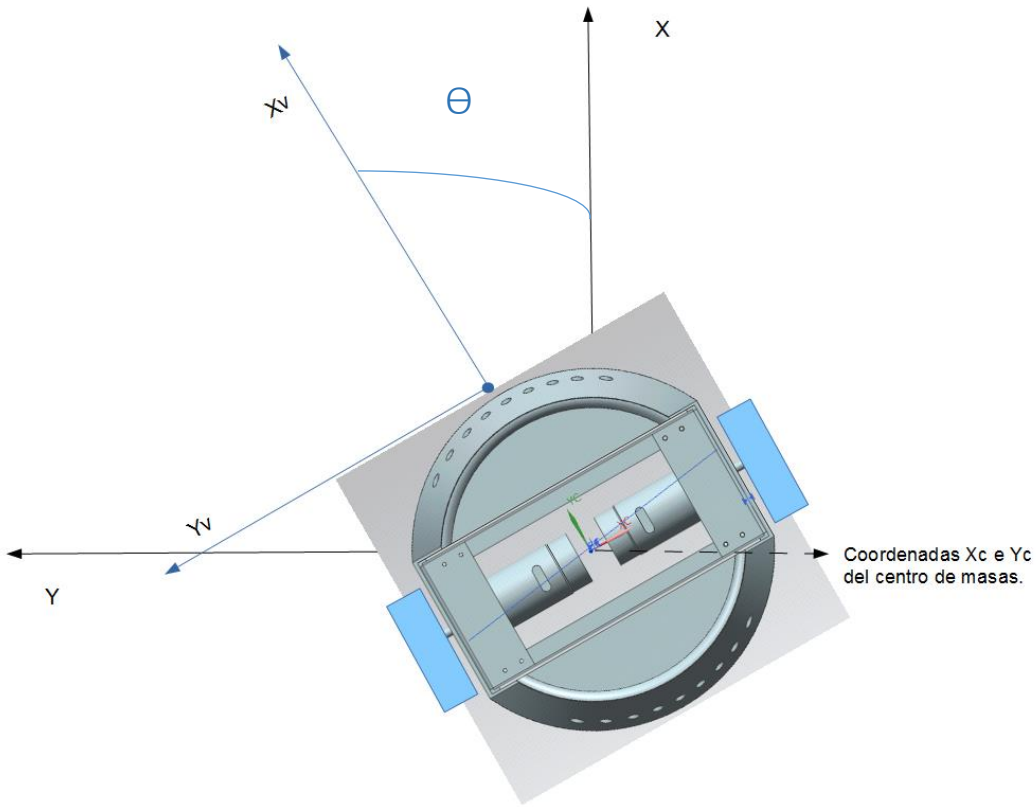


Figura 2.2.1_2: Sistemas de Referencia

Para entenderlo bien, pensamos de la siguiente forma, hay un sistema de referencia fijo, que situamos en un lugar determinado (en la Figura 2.2.1_2 anterior es el formado por los ejes negros X e Y), y con una orientación determinada, y un sistema de referencia móvil que está situado encima del vehículo (en la imagen anterior es el formado por los ejes azules Xv e Yv).

Si m(t) es la velocidad del robot y Ɵ(t) la orientación con respecto al sistema de referencia fijo, ambas en función del tiempo, la solución de la cinemática directa es:

$$\frac{dx}{dt} = \boldsymbol{m(t) * cos\ (\theta\ (t))} \quad (1.1)$$

$$\frac{dy}{dt} = \boldsymbol{m(t) *\ sin\ \ (\theta\ (t))} \quad (1.2)$$

El cambio en la orientación en función del tiempo es el mismo que el de ω (velocidad angular).

$$\boldsymbol{d\theta/dt\ =\ \omega\ =\ (vr - vl)/l} \quad (1.3)$$

Si se integran las ecuaciones anteriores, se obtiene una nueva ecuación que nos indica la nueva orientación (ángulo girado respecto de x), si sabemos el tiempo transcurrido y la anterior orientación.

$$\boldsymbol{\theta(t)\ =\ (vr - vi)t/l\ +\ \theta 0} \quad (1.4)$$

Ya que la velocidad de avance resultante del vector velocidad m(t), de las ecuaciones (1.1) y (1.2) es igual a la media de las velocidades de cada rueda, dichas ecuaciones se pueden expresar como:

$$\frac{dx}{dt} = \boldsymbol{(Vr + Vi)/2 * cos\ (\theta\ (t))} \quad (1.5)$$

$$\frac{dy}{dt} = \boldsymbol{(Vr + Vi)/2 *\ sin\ \ (\theta\ (t))} \quad (1.6)$$

Al integrar las ecuaciones anteriores obtenemos las ecuaciones de la trayectoria:

$$\boldsymbol{x(t) = x0 \ + \frac{1}{2} * \frac{(Vr + Vi)}{(Vr - Vi)} * [sin\left(\left(\frac{Vr - Vi}{l}\right) * t + \theta 0\right) - sin(\theta 0)]}$$

$$\boldsymbol{x(t) = x0 + R * [\ sin(\omega t + \theta 0) \ - \ sin(\theta 0)]} \quad (1.7)$$

$$\boldsymbol{y(t) = y0 \ + \frac{1}{2} * \frac{(Vr + Vi)}{(Vr - Vi)} * [cos\left(\left(\frac{Vr - Vi}{l}\right) * t + \theta 0\right) - cos(\theta 0)]}$$

$$\boldsymbol{y(t) = y0 + R * [\ cos\ (\omega t + \theta 0) \ - \ cos\ (\theta 0)]} \quad (1.8)$$

Estas ecuaciones son la teoría que se esconde tras la navegación a estima, de un Robot con accionamiento diferencial. Si sustituimos Vd y Vi por Sr y Si, que significan los cálculos de desplazamiento por el tiempo, en vez de las velocidades obtenemos las expresiones (1.9) y (2.1).

Es decir, Sr y Si son las distancias recorridas por las Ruedas derecha e izquierda convirtiéndose en la siguiente ecuación (e = v * t):

$$\boldsymbol{x(t) = x0 \ + \frac{1}{2} * \frac{(Sr + Si)}{(Sr - Si)} * [sin\left(\left(\frac{Sr - Si}{l}\right) + \theta 0\right) - sin(\theta 0)]} \quad (1.9)$$

$$\boldsymbol{y(t) = y0 \ + \frac{1}{2} * \frac{(Sr + Si)}{(Sr - Si)} * [cos\left(\left(\frac{Sr - Si}{l}\right) + \theta 0\right) - cos(\theta 0)]} \quad (2.1)$$

Que son las ecuaciones de la cinemática directa utilizadas para vehículos de accionamiento diferencial.

2.2.2.- **Ventajas e Inconvenientes**

La navegación a estima basada en los cálculos odómetricos es más barata que otros métodos como sistemas GPS o de baliza suelta, y además da solución a la navegación en interiores de edificios donde la señal GPS no llega, no requiere de cálculos muy complejos que además proporcionan buena precisión a corto plazo.

También hay que añadir que se pueden obtener grandes tasas de muestreo mediante los potentes microcontroladores modernos a diferencia de otros sistemas de muestreo lento como el GPS.

Los problemas vienen con la acumulación de errores a lo largo del tiempo que pueden ser errores sistemáticos o errores no sistemáticos.

Los errores sistemáticos son creados por los defectos mecánicos, asimetría entre los diámetros de las ruedas o incertidumbre acerca de cuál es la distancia entre ejes más efectiva. Por otra parte, los errores no sistemáticos se deben a causas externas, que no tienen que ver con la cinemática del vehículo, unos ejemplos de estos errores se producen como consecuencia de superficies irregulares que pueden producir deslizamiento de las ruedas.

Los errores sistemáticos son peores que los debidos al entorno, puesto que se acumulan constantemente. A pesar de estos inconvenientes, muchos investigadores creen que la navegación a estima debe formar parte en cualquier sistema de navegación de un robot, sin tener que ser el único medio de medida de la posición o velocidad, pudiendo interaccionar con otros sistemas, a fin de poder cotejar los datos de distintos sensores.

2.2.3.- **Seguimiento de Trayectoria**

El seguimiento de trayectoria por parte de las ruedas es un proceso que se encarga de ajustar las consignas de velocidad de cada una de las ruedas (y por lo tanto la dirección o rumbo), que son aseguradas por cada uno de los bucles de control a lazo cerrado de cada una de las ruedas, en cada instante de tiempo o periodo de muestreo, a fin de seguir una determinada trayectoria o ruta por el Robot.

Un camino o ruta está formado por un conjunto de puntos que representan las coordenadas de posición de una ruta en particular.

Cuando se implementa un algoritmo de seguimiento de trayectoria o ruta, es habitual grabar todas las coordenadas que constituyen el camino. Permitiendo de esta forma la posibilidad de que un ser humano dirija manualmente el Robot durante una cierta trayectoria, de manera que se van guardando los puntos de las coordenadas de la trayectoria realizada por el Robot.

El control (Bucles de control a lazo cerrado de cada Rueda) que forma parte del algoritmo de seguimiento, tiene que ser capaz de manejar las desviaciones de posición u orientación, que pueden ser causadas no solo por errores odómetricos, sino que, también debidas a obstáculos o cambios de pendiente del camino, perturbaciones externas.

Hay muchos tipos de algoritmos de seguimiento de trayectoria, *Follow-the-carrot*, *Pure Pursuit*, *Vector Pursuit* etc. En este proyecto se ha utilizado el *Pure Pursuit*. Pero comentaremos primero como introducción el seguimiento de trayectoria basado en *Follow-the-carrot*.

2.2.4.- **Follow the carrot**

El concepto en el que se basa este algoritmo es muy sencillo, consiste en tener un punto meta, corregir la orientación y dirigir el vehículo hacia dicho punto.

En la imagen siguiente Figura 2.2.4_1, se muestran los conceptos necesarios para poder entender el algoritmo.

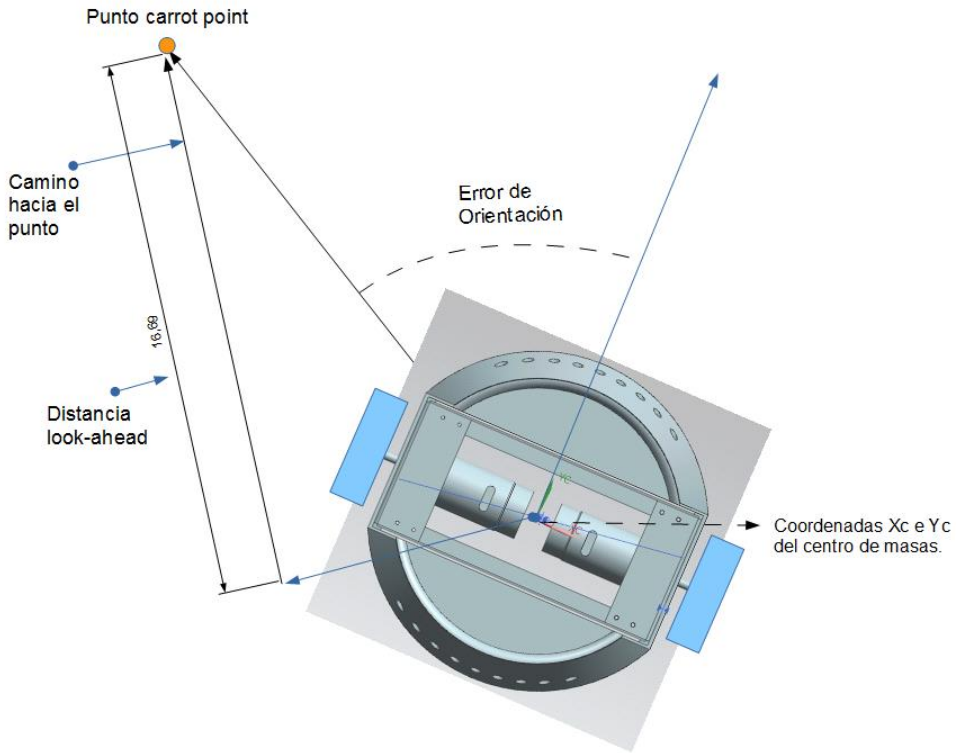


Figura 2.2.4_1: Follow the Carrot

Se dibuja una línea desde el centro del vehículo, que es el sistema de coordenadas perpendicular a la trayectoria. El punto de la Zanahoria o punto objetivo es hacia el que hay que dirigirse.

El parámetro más importante es el error de orientación, que se define como el ángulo formado entre la orientación de partida del vehículo y la línea trazada desde el centro del vehículo origen de coordenas al punto de Zanahoria.

Se aplica una Ley de control proporcional que tiene como objetivo reducir al mínimo el error de orientación entre la orientación del vehículo y el punto de Zanahoria.

Un error de orientación cero significa que el vehículo está apuntando exactamente hacia el punto de la Zanahoria.

La magnitud de giro $\boldsymbol{\phi}$ para corregir el error de orientación se decide como sigue:

$$\boldsymbol{\phi} = kp * eo$$

Donde Kp es la ganancia proporcional y eo es el error en la orientación. Es posible aumentar la precisión del control de trayectoria aplicando acción Integral.

Aunque el enfoque del seguimiento de la Zanahoria es fácil de entender y muy sencillo de poner en práctica, tiene un par de inconvenientes importantes. Por una parte, el vehículo tiende a cortar las esquinas. Esto ocurre porque el vehículo trata de girar hacia el nuevo punto de Zanahoria. Y, por otro lado, el vehículo puede oscilar alrededor de una ruta, bien cuando se cubren pequeñas trayectorias o por utilizar velocidades relativamente altas.

A pesar de que existen modificaciones que podemos hacer al algoritmo para aumentar la eficiencia y la precisión, como basar el ángulo de dirección tanto en el error de orientación como en el de posición, distancia hasta el próximo punto, son tantas las desventajas, que no se utiliza para aplicaciones serias, pero es un buen punto de partida para empezar y sobre todo en temas educativos.

2.2.5.- **Persecución Pura (*Pure Pursuit*)**

El concepto de persecución pura (*Pure Pursuit*.), consiste en calcular la curvatura que llevará el vehículo desde la posición actual a una posición meta. El punto objetivo se determina de la misma forma que para el algoritmo de seguimiento *Follow-the-carrot*.

Se crea un círculo que pasa tanto por el punto objetivo como por el punto de la posición actual del vehículo. Por último, un algoritmo de control elige un ángulo de dirección en relación a este círculo. Lo que ocurre es que el vehículo robot cambia su curvatura por arcos circulares repetidamente empujando el punto objetivo hacia adelante.

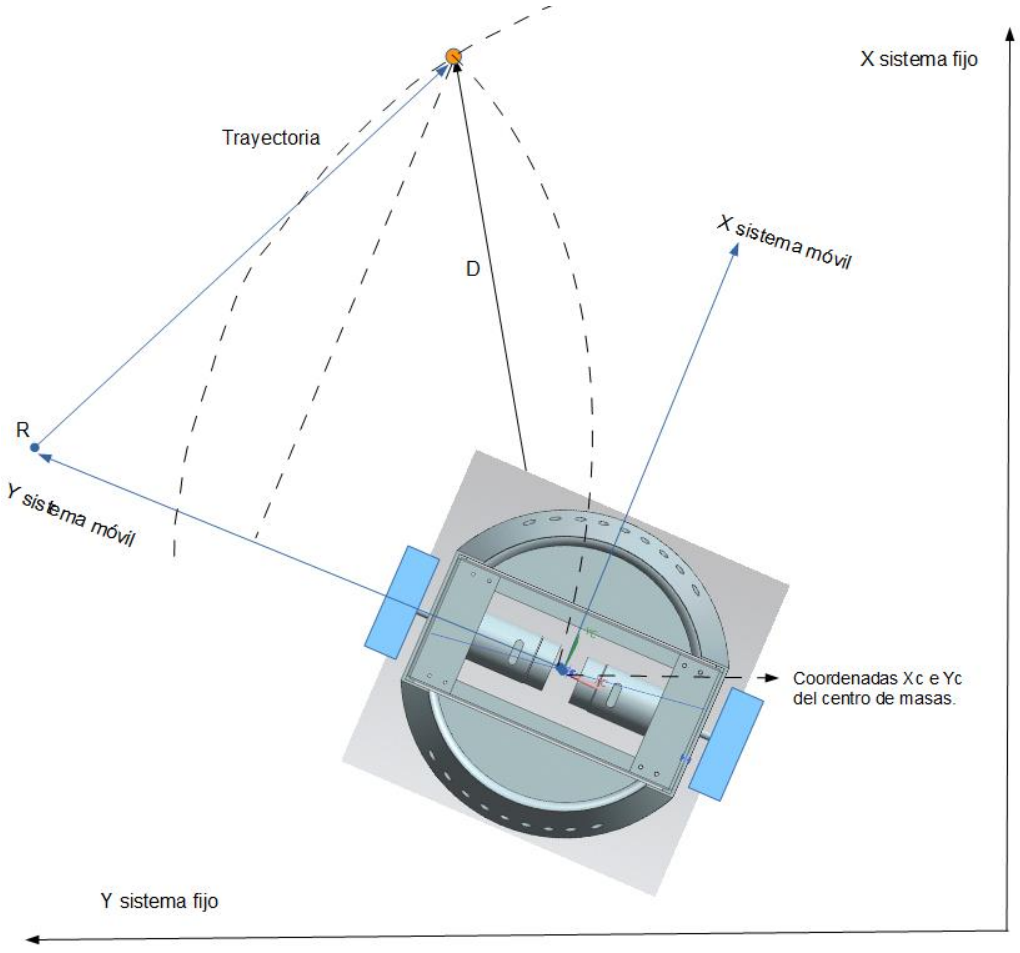


Figura 2.2.5_1: Pure Pursuit

Este algoritmo de persecución pura se muestra en el sistema de coordenadas del vehículo. En el sistema de coordenadas del vehículo se define el (eje x) en la dirección de avance del vehículo, el (eje z) hacia arriba y (eje y) forma un sistema diestro de coordenadas. Por lo tanto, todas las coordenadas utilizadas deben ser en un primer paso convertidas al sistema de coordenadas del vehículo para que el algoritmo funcione correctamente. Afortunadamente es bastante fácil convertir las coordenadas del sistema de referencia externo, al sistema de referencia del vehículo.

Finalmente se muestra a continuación un flujograma Figura 2.2.5_2 con las ecuaciones aplicadas para ir determinando los puntos actuales y las velocidades de cada una de las ruedas para poder ir a los puntos consigna.

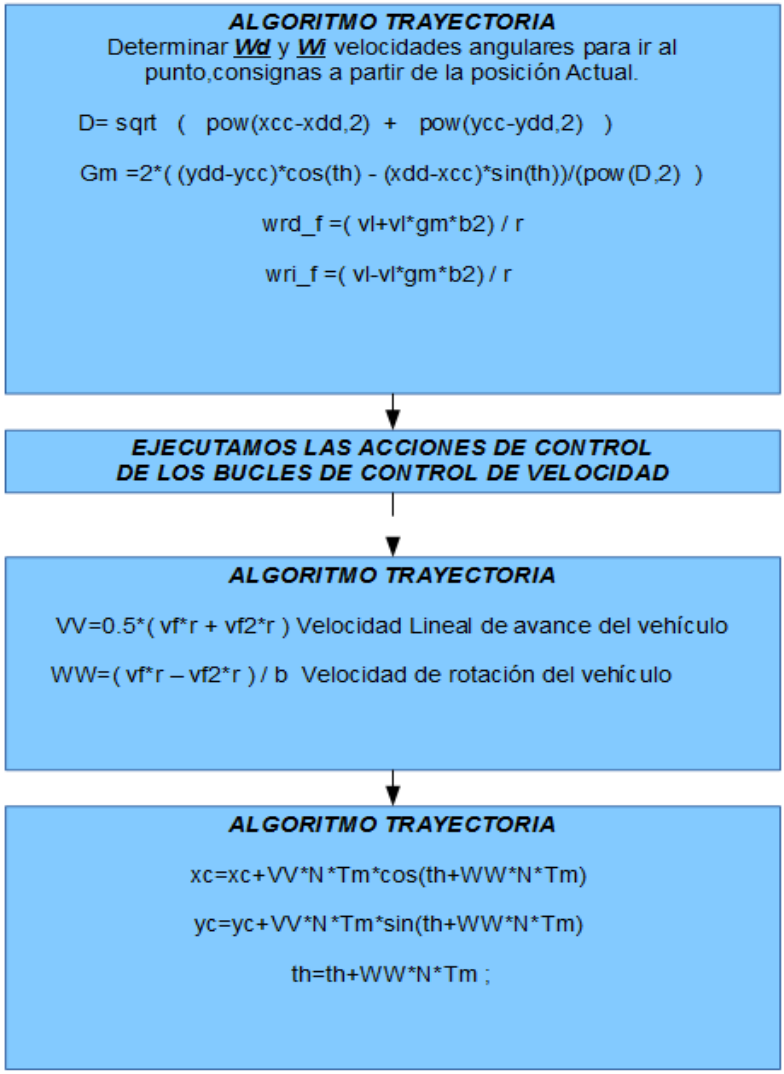


Figura 2.2.5_2: Flujograma algoritmo Pure Pursuit

Wd= wrd_f *es la velocidad angular de la rueda derecha en rad/s determinada por el algoritmo de seguimiento de trayectoria como consigna para ir al punto.*

Wi=wri_f es *la velocidad angular de la rueda izquierda en rad/s determinada por el algoritmo de seguimiento de trayectoria como consigna para ir al punto.*

D *es la distancia desde el centro de masas del vehículo hasta el punto consigna.*

sqrt *es la raíz cuadrada de todo lo que contiene los paréntesis que siguen.*

pow *sirve para elevar al cuadrado el contenido entre paréntesis que viene a continuación.*

Gm y gm *es la inversa del radio de curvatura.*

Xdd = Xd e Ydd = Yd *son las coordenadas del punto consigna al que se quiere ir.*

Xcc = Xc e Ycc = Yc *son las coordenadas del centro de masas del vehículo actuales determinadas por el algoritmo de seguimiento de trayectoria.*

th *es el ángulo de orientación del robot con respecto al eje x del sistema de referencia fijo.*

vl *es la velocidad lineal de avance consigna del Robot prefijada.*

b2 *es la distancia desde el centro del eje da las ruedas hasta el punto medio de la rueda, en la Ilustración 2.2.1_1, es L/2.*

b esta *distancia es el doble de la anterior, es la distancia entre ruedas.*

vf *es la velocidad angular de la rueda derecha determinada por el encoder y el software.*

vf2 *es la velocidad angular de la rueda izquierda determinada por el encoder y el software.*

r *es el radio de las ruedas.*

VV *es la velocidad lineal de avance del vehículo, determinada por el algoritmo de seguimiento de trayectoria.*

WW *es la velocidad de rotación del vehículo respecto al cetro de curvatura determinada por el algoritmo de seguimiento de trayectoria.*

Tm *es el periodo de muestreo (10 ms).*

N *es 10, y sirve para indicar que cada 10 veces el periodo de muestreo se realizan las operaciones indicadas.*

2.3.- Control de procesos basado en red.

Sistemas de Control Basados en Red (SCR), son un tipo especial de sistemas de control, que utilizan como medio de comunicación una red bien sea LAN (Red de área local) o bien sea WAN (Red de área amplia), para transferir información entre el controlador y la planta controlada.

El medio de comunicación para transferir información, es un medio compartido, es decir, es lo que se conoce como un recurso compartido y ello implica, que el medio puede estar saturado por la utilización de dicho medio para otros procesos que no necesariamente tienen que ser procesos de control.

El problema más importante que se desprende de la no utilización de un medio exclusivo para establecer la comunicación, es la disminución del ancho de banda como consecuencia de las comunicaciones que se producen por los distintos sistemas que hacen uso accediendo al medio compartido o red.

Existen varias formas de abordar este problema, como por ejemplo aumentando las prestaciones del medio de comunicación (utilizando sistema de transmisión más rápidos, en el caso de redes LAN que pueden funcionar hasta a 100Mbps e incluso a 1Gbps), pero es imposible cuando se trata de redes WAN, pues nos podemos encontrar en la red, con cualquier medio físico hardware, que puede actuar disminuyendo la velocidad de transmisión (como un efecto embudo).

Otra forma de tratar la problemática sería limitar el acceso al medio compartido, de tal forma que se diera prioridad a ciertas máquinas cuando acceden al medio, estas máquinas serían evidentemente aquellas cuyas tareas fueran el control de cualquier proceso.

Hay un tercer punto de vista, que sería asumir que el medio de comunicación impone una serie de limitaciones que no podemos cambiar, y por lo tanto lo único que podemos hacer es tratar de paliar dicha problemática diseñando un sistema de control apropiado.

Para entender el control basado en Red pasamos a explicar el control de un sistema continuo, a continuación, el control en un sistema discreto y finalmente sistema de control basado en red.

En un **sistema de control continuo**, como el que se esquematiza en la Figura 2.3_1, hay una comunicación bidireccional permanente entre el controlador y la planta controlada. El controlador recibe información continua acerca de la evolución temporal de las variables de interés, a través de un conjunto de

sensores. Del mismo modo, el controlador actúa de forma continua sobre la planta controlada, a través de un conjunto de actuadores, modificando su comportamiento en función de la información recibida y aplicada por éstos.

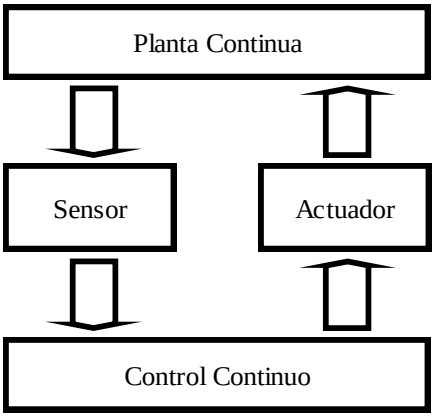


Figura 2.3_1: Sistema Control Continuo

Si en lugar de un sistema de control continuo se trata de un **sistema de control discreto** convencional como el de la Figura 2.3_2, la comunicación se realiza de forma discontinua pero uniformemente distribuida en el tiempo. La etapa de control está implementada en un dispositivo de naturaleza discreta que únicamente es capaz de recibir y generar señales discretas. Son necesarios, por lo tanto, dispositivos de conversión analógico-digital y digital-analógico que actúen de interfaz entre el mundo analógico del proceso bajo control y el mundo digital donde está implementada la ley de control.

La forma más habitual de implementar un control discreto es emplear un periodo de muestreo regular (T), lo cual significa que los instantes de muestreo de las variables de interés y los instantes de actuación del controlador sobre la planta estarán equiespaciados en el tiempo y coincidirán con los múltiplos enteros del periodo de muestreo. Existe, por lo tanto, un único reloj que cada T unidades de tiempo determina la captura de una muestra de las variables de interés, el cálculo de una nueva acción de control y la aplicación de la misma. En una situación más real debería existir una diferencia de un periodo de muestreo entre la captura de una muestra y la aplicación de la acción de control generada a partir de la información que proporciona, ya que el cálculo de la acción necesitará de un tiempo no nulo. Este retraso respecto a la situación ideal habrá de ser tenido en cuenta en el diseño de la ley de control para que no tenga una influencia significativa en las prestaciones del sistema. En cualquiera de los casos, en una estrategia de control discreto convencional los instantes de muestreo y aplicación de acciones están perfectamente determinados y se distribuyen en el tiempo de forma regular.

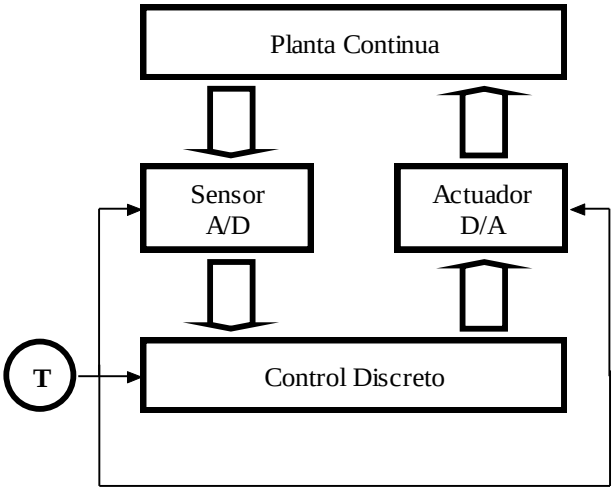


Figura 2.3_2: Sistema Control Discreto

La implementación discreta del control tiene innegables ventajas, entre ellas la posibilidad de implementar estrategias de control complejas que no podrían ser conseguidas con componentes continuos. Sin embargo, tiene el inconveniente de que la comunicación control-planta se interrumpe de forma periódica. Tan sólo en los instantes de muestreo el sistema se comporta como una verdadera estructura de control realimentada, quedando la planta sin control durante todo el periodo intermuestreo. Este inconveniente puede reducirse hasta hacerlo despreciable siempre que se pueda hacer que la duración del intermuestreo sea lo suficientemente corta, aumentando la frecuencia de muestreo. Sin embargo, esto no siempre será posible. Lo más habitual es que el proceso de medida tenga un coste temporal no despreciable, que determina un límite superior a la frecuencia de muestreo. Otras veces, es el coste temporal de la ejecución del algoritmo de control o de las conversiones A/D y D/A, el que determina este límite. Una última posibilidad es que la limitación en la frecuencia de muestreo venga impuesta por el enlace de comunicación, que es el caso del modo de funcionamiento remoto (control basado en red), del presente proyecto, donde el algoritmo de seguimiento de trayectoria se encuentra en el cliente Matlab / Simulink.

En la Figura 2.3_3 se muestra una sencilla representación esquemática de lo que se ha dado en llamar SCR. La diferencia con el sistema discreto convencional de la Figura 2.3_2, es que en este caso la comunicación entre el controlador discreto y la planta continua se realiza a través de un enlace compartido, que es utilizado al mismo tiempo para comunicar a otros dispositivos. Debido a esto, el enlace de comunicación no estará disponible de

forma permanente ya que, en ocasiones, estará ocupado dando servicio a otros dispositivos. Es posible que, cuando el sensor capture una muestra de la variable de interés y pretenda enviarla al controlador, se encuentre con el enlace ocupado y deba permanecer a la espera de que se le conceda el uso del mismo. Del mismo modo, es posible que cuando el controlador ha calculado una nueva acción de control y pretenda enviarla al actuador para su aplicación, el enlace esté siendo utilizado por otro dispositivo y deba esperar a que quede libre. En definitiva, al no emplear un enlace exclusivo existe la posibilidad de que no esté disponible en el momento que se necesite para la transmisión de información, ya sean muestras o acciones de control. Un detalle que de gran importancia y que es un gran inconveniente es que, debido a la naturaleza probablemente irregular del tráfico de información en el enlace, el tiempo que debe esperar un dispositivo para acceder al uso del enlace no será constante. Esto se traducirá en un retraso de acceso variable que, evidentemente tendrá influencia negativa en las prestaciones del sistema.

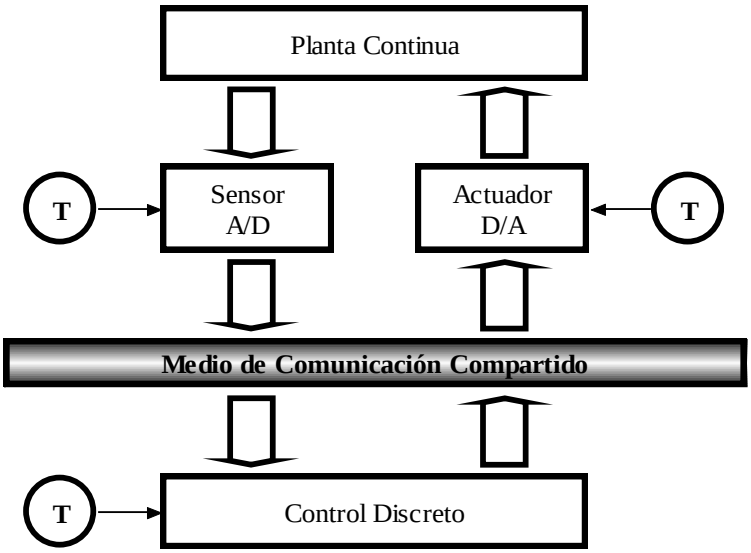
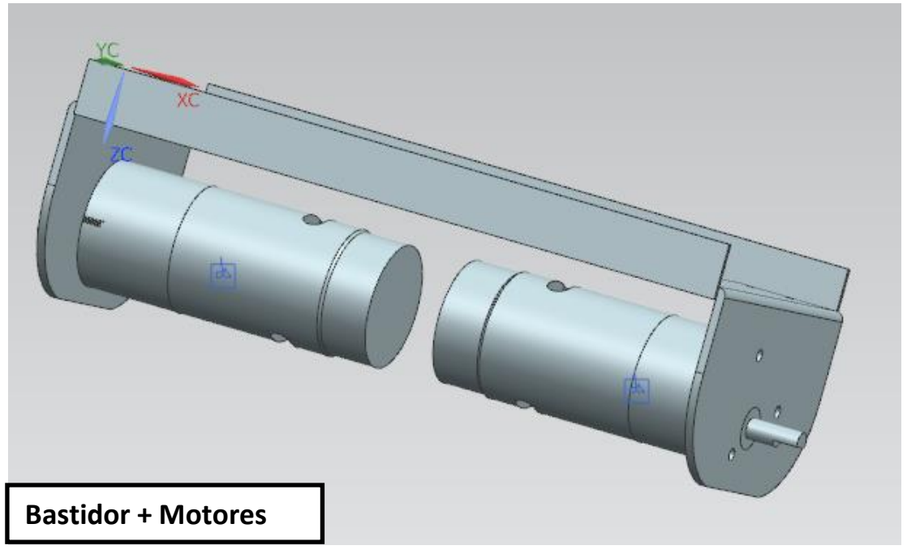


Figura 2.3_3: Sistema Control basado en Red

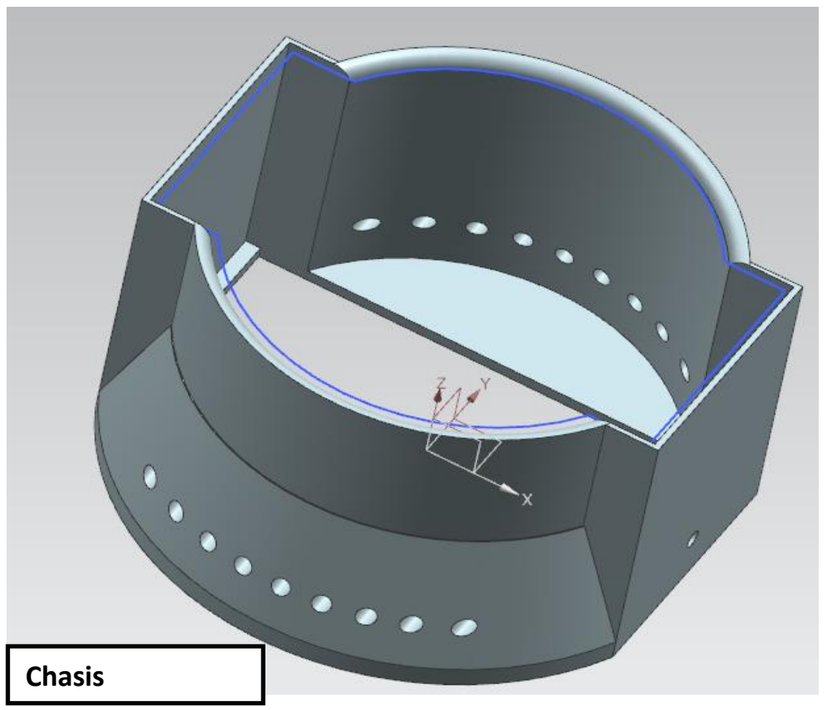
Precisamente para mitigar la problemática que ocasionan estos retrasos de tiempo variables, del flujo bidireccional entre cliente (controlador del seguimiento de trayectoria) y el servidor (planta-robot), se ha utilizado un predictor de Smith, como veremos en el capítulo 4, apartado 4.5.4 Cliente desde Matlab / Simulink + Algoritmo Seguimiento en modo remoto. Aunque el predictor de Smith no soluciona por completo toda la problemática comentada, si mejora considerablemente los resultados obtenidos.

3.- ELEMENTOS DEL SISTEMA ROBOT

3.1.- Diagrama de Bloques Elementos Sistema Mecánico



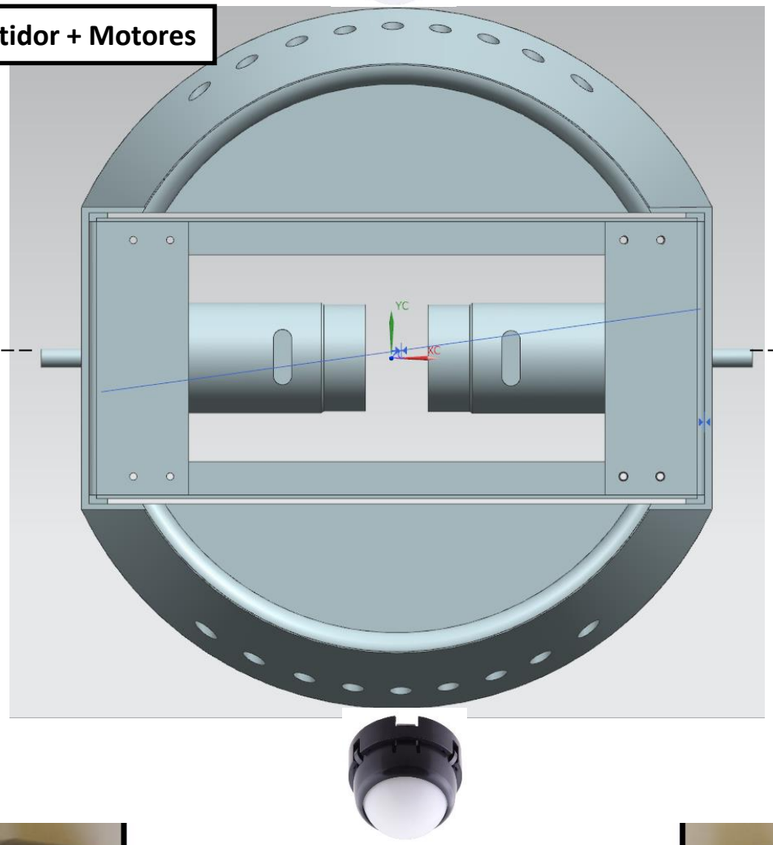
Bastidor + Motores



Chasis

Bolas Locas Apoyos

Chasis + Bastidor + Motores



Ruedas de Goma Radio 51mm

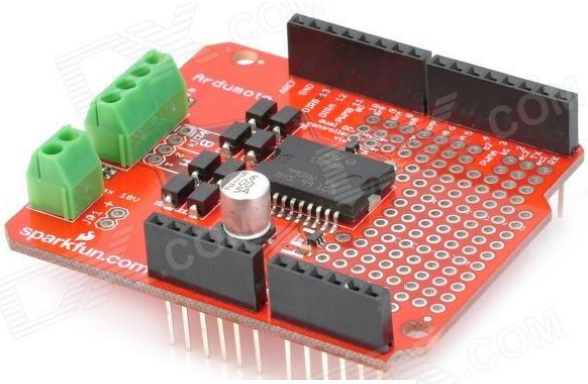
El anterior diagrama de bloque sirve para hacerse una idea, de todas piezas mecánicas que conforman el Robot y la disposición de las mismas.

3.2.- Diagrama de Bloques Elementos Sistema Energético

Batería LI-PO de 2600 mAh

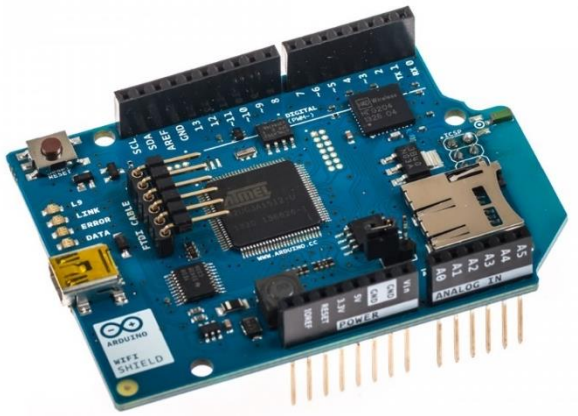
Condensadores Para cubrir los Picos de corriente del módulo Wifi y de Los Motores.

3.3.- Diagrama de Bloques Elementos Sistema Electrónicos



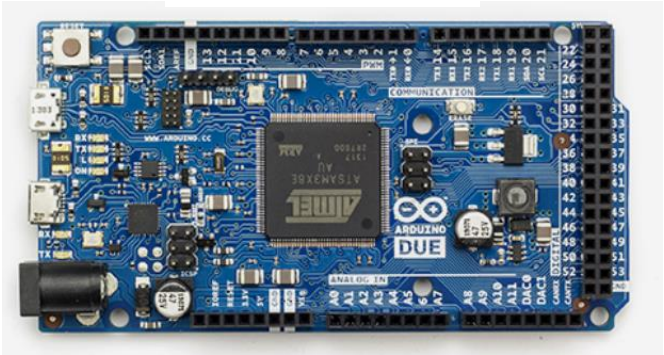
Driver Doble Puente en H

+



Módulo Wifi

+



Microcontrolador ARM M3 Arduino Due

3.4.- Descripción-Características Chasis y Bastidor

El chasis se ha diseñado mediante la Herramienta de trabajo 3D, NX 10 de Siemens, que permite a los diseñadores un muy buen control sobre el modelado geométrico para producir productos con formas muy estilizadas o superficies complejas. Una de las mejoras que incluye la versión 10, es NX *Realice Shape*, que se basa en el modelado por subdivisión, un enfoque matemático para crear geometría 3D con formas suaves y fluidas, pionero en la industria del entretenimiento. Esta herramienta se integra a la perfección con NX y ayuda a reducir el tiempo de desarrollo de productos mediante la eliminación de los múltiples pasos asociados al uso de herramientas independientes para el diseño y desarrollo de ingeniería.

El aumento de la complejidad del producto hace que el modelado 3D sea el método preferido para el diseño de productos en todo el mundo. En algunas industrias, incluyendo las de maquinaria y las de electrónica compleja es más fácil y rápido crear un diseño esquemático inicial en 2D. La nueva solución de desarrollo de concepto en 2D permite a los diseñadores explorar conceptos en 2D para crear nuevos diseños hasta tres veces más rápido. Una vez el diseño está terminado puede ser fácilmente migrado a 3D para completar el modelo.

Después de obtener el modelo del chasis en 3D, NX genera un fichero. prt, es este fichero el que se le pasa a la impresora 3D, que se encarga de levantar(Imprimir) el modelo del chasis.

El material utilizado por la impresora para generar el modelo, es **Acrilonitrilo Butadieno Estireno** o **ABS**, que es un plástico muy resistente al impacto, utilizado especialmente en la industria de la automoción, pero en ocasiones también en la doméstica.

Se le suele llamar plástico de ingeniería porque su elaboración y procesamiento es algo más compleja que en los plásticos comunes.

Este material compuesto soporta bien las temperaturas extremas, especialmente en entornos fríos.

En el apartado Planos **plano 01_A y plano 01_B Chasis**, se especifican las dimensiones exactas del modelo del chasis.

El cuanto al bastidor Figuras 3.4_1 y 3.4_2 está compuesto totalmente por material de Aluminio, consta de dos soportes en forma de L, que han sido diseñados especialmente para fijar el motor con encoder EMG30 mediante tres tornillos de métrica 3mm y profundidad 5mm.

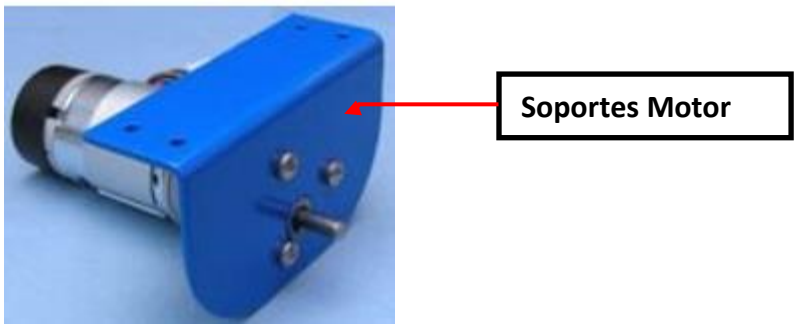


Figura 3.4_1: Soporte Motor

Los dos soportes se han unido mediante dos perfiles de aluminio en forma de L, cuyo objetivo es el de crear una fijación fuerte de los dos soportes a dichos perfiles, a la vez que aseguren la alineación correcta de los ejes de los motores, quedando estos separados 170 mm, (de la cara exterior motor a cara exterior motor). En el apartado planos **plano 02 Bastidor,** se especifican las dimensiones correctas.

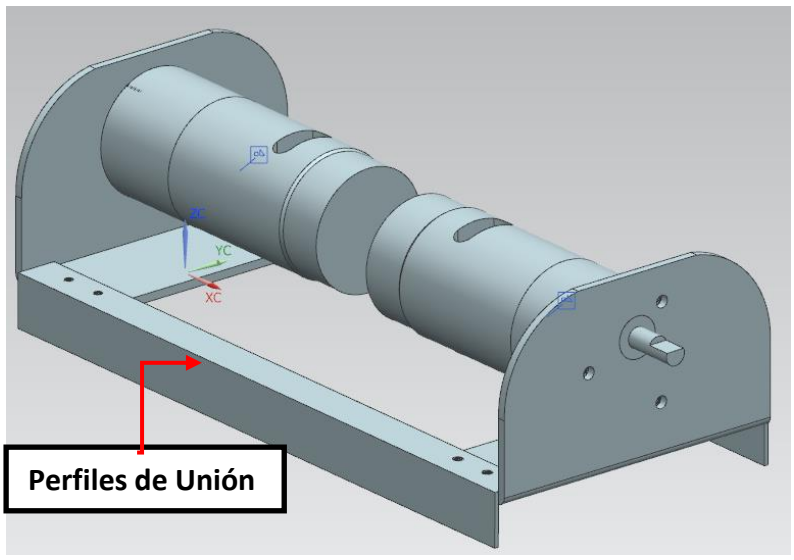


Figura 3.4_2: Unión de los Soportes mediante Perfil de Aluminio

3.5.- Descripción-Características Motores + Encoders

Los motores elegidos son dos motores de bajo coste marca EMG30, de corriente continua de alimentación máxima de 12 voltios y de 170 rpm con carga mientras que sin carga es de 216 rpm (lo cual se consigue con la reductora que lleva incorporado el motor). El par máximo bloqueando el motor (*stall Torque*) es de 1.5 Kg/cm, y el par en carga máximo es de 0.5 Kg/cm. La corriente en bloqueo pude alcanzar 2.5 A, mientras que en vacío es de 0.150 A, siendo con carga nominal de 0.53 A. La potencia nominal es de 4.22 W. Longitud total: 86,6mm. Diámetro motor: 30mm. Diámetro Eje salida: 5mm. Longitud Eje: 9mm. El motor consta de un encoder de cuadrante que manda un tren de impulsos cuando gira el eje del motor, permitiendo así que un circuito externo pueda saber la velocidad real a la que está girando el eje y cuantas vueltas da. El encoder está formado por dos sensores de efecto Hall que proporcionan un total de 360 pulsos por cada vuelta completa del rotor. El motor cuenta con condensadores internos de filtro que ayudan a minimizar el ruido y los parásitos generados por el motor al girar.

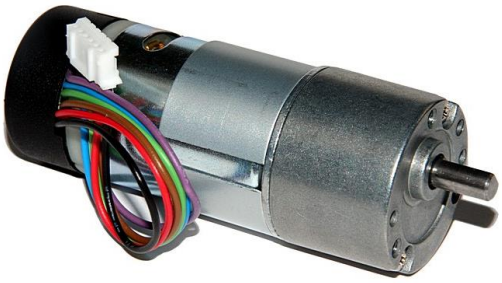


Figura 3.5_1: Motor

Este motor presenta la ventaja de un precio bajo. Y por otra parte cumple las condiciones indispensables de que por un lado el par motor máximo 1.5 Kg/cm • 2 = 3 Kg/cm (dos motores) es superior al par necesario para poder vencer la fuerza de rozamiento y poner el sistema en marcha y que el par motor nominal con una carga nominal es de 0.5 • 2 = 1 Kg/cm.

$$\boldsymbol{Froz} = \mu \boldsymbol{N}$$

El coeficiente de rozamiento para ruedas poco deformables como es el caso, se puede estimar con bastante precisión mediante:

$$\mu \approx \sqrt{\frac{z}{d}} = \sqrt{\frac{1}{100}} = 0.1$$

$$Froz = \mu N = 0.1 \bullet 1.4\ \text{Kg} \bullet 9.8\ \text{m}/s^2 = 1.372\ \text{N}$$

$$T\ (N\ m) = F \bullet r = 1.372\ \text{N} \bullet 0.05 = 0.0686\ N\ m$$

$$T\ \left(\frac{Kg}{cm}\right) = 0.0686\ N\ m = 0.0686 \bullet \frac{1\ Kg}{9.8\ m/s^2}\ m \bullet \frac{100\ cm}{1\ m} = 0.69 \frac{Kg}{cm}$$

Esto implica que la velocidad máxima con esta carga, será un poco menor que la velocidad máxima que puede alcanzar el motor con su carga nominal.

Aproximadamente como la velocidad máxima en carga nominal según el fabricante es de:

$$\omega = = 170\ \frac{vueltas}{minuto} \bullet \frac{1\ minuto}{60\ s} = 2.833 \frac{vueltas}{s}\ \ que\ por\ 2 \bullet \pi = 17.8 \frac{rad}{s}$$

Con lo cual la velocidad máxima de avance o Lineal estará un poco por debajo de:

$$VLineal\ (m/s) = \omega \bullet r = 17.8 \frac{rad}{s} \bullet 0.05m = 0.89 \frac{m}{s}$$

$$VLineal\ \left(\frac{Km}{h}\right) = 0.5 \frac{m}{s} \bullet \frac{1\ Km}{1000\ m} \bullet \frac{3600\ s}{1\ h} = 3.2\ Km/h$$

O puede ser una aproximación, bastante real puesto que el coeficiente de rozamiento cuando el vehículo se está desplazando (dinámico) disminuye con respecto al inicial o estático.

La velocidad anterior es más que suficiente para cumplir con las velocidades adecuadas para el algoritmo de persecución, cuyas velocidades óptimas están en el rango de 0.09 m/s a 0.2 m/s.

El gran inconveniente que presentan estos motores con respecto a unos motores de características similares de la marca Maxon es el gran porcentaje de zona muerta, que como veremos en el apartado de identificación de los motores es del orden del 20 % de 0.89 m/s que viene a ser de 0.178 m/s y esto implica que en el caso del algoritmo de persecución que es interesante ejecutar la curvas a velocidades bajas del orden de 0.1 m/s, estamos dentro de la zona muerta y como veremos tenemos que implementar en el algoritmo de control una solución que

saque al motor de esta zona muerta cuando operemos a estas velocidades bajas que como hemos dicho son muy habituales.

En comparación con el motor Maxon en el que la zona muerta podría quedar en el orden de un 5 %, que mejoraría el comportamiento del vehículo de forma significativa, el inconveniente de estos motores es el precio que oscila en más o menos 500€ que es mucha diferencia con respecto a 37 € que valen cada uno de los motores utilizados.

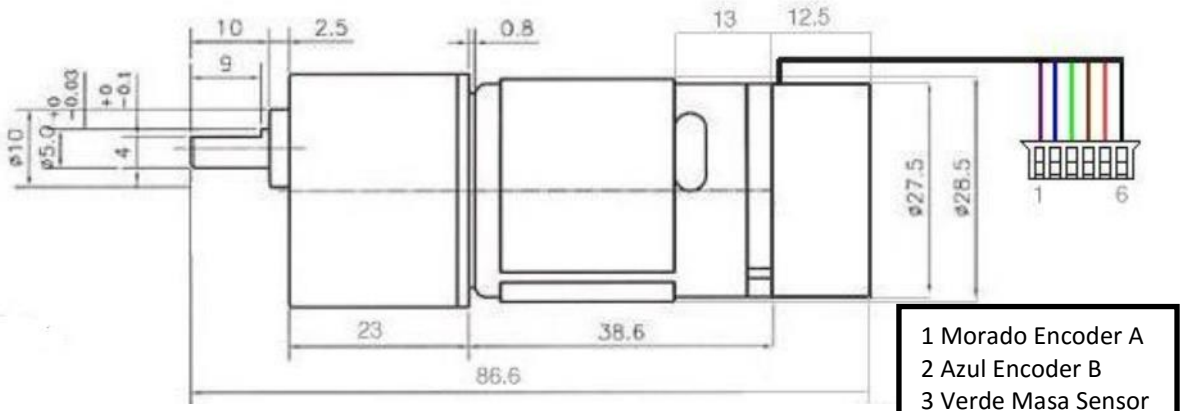


Figura 3.5_2: Dimensiones y conector Motor

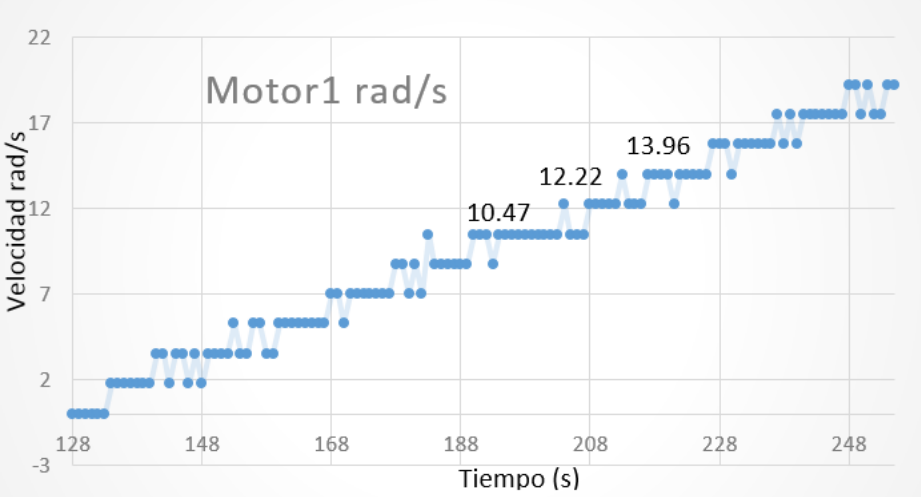
En el apartado Anexos, **anexo 03 Características Dimensiones del Motor,** quedan definidas todas las dimensiones de forma clara con todo detalle y en el **plano 03 Conexiones Driver-Motores-Microcontrolador** se especifica el conexionado del motor al driver y el de las 2 señales A y B a Arduino mediante las resistencias Pull-Up de 10 KΩ. En el Capítulo 4, apartado 4.5, sub-apartado 4.5.1 Flujograma Algoritmo-Persecución + Bucles Control Velocidad (modo-Local), se explica cómo se ha determinado la velocidad a partir de los pulsos generados por el encoder y de la formula siguiente que se comenta en este sub-apartado, podemos deducir que la velocidad mínima que podemos leer (resolución), es para el caso de un pulso por periodo de muestre que se traduce en:

$$\boldsymbol{Velocidad\ m\acute{i}nima}\left(\frac{\boldsymbol{rad}}{\boldsymbol{s}}\right)=\frac{\boldsymbol{1puso}\ *\boldsymbol{0.017453}}{\boldsymbol{10\ ms}}=\boldsymbol{1.7453}$$

Y esto significa, que nuestra velocidad de avance lineal mínima no puede ser menor de:

$$\boldsymbol{Velocidad\ lineal}\left(\frac{\boldsymbol{m}}{\boldsymbol{s}}\right) = \mathbf{1.7453} * \mathbf{0.051} = \mathbf{0,089}\,\frac{\mathbf{m}}{\mathbf{s}}$$

Este resultado teórico coincide exactamente con uno de los experimentos realizados para la identificación de los motores, que se muestra a continuación en la siguiente figura:



Si hacemos la diferencia entre dos escalones consecutivos de la velocidad leída por los encoders, cuando se alimenta el motor con una señal de tensión en forma de rampa obtenemos:

13.96-12.22 = 1.74 rad/s

12.22-10.47 = 1.75 rad/s

Esta es la mayor limitación que presentan estos encoders que vienen adjuntos al motor, que es casi peor que el problema de la no linealidad de la zona muerta.

3.6.- Descripción-Características Batería y Condensadores

La gran cantidad de dispositivos electrónicos y vehículos autónomos alimentados mediante baterías ha aumentado significativamente, y actualmente con la necesidad de sustituir los vehículos de combustión por eléctricos aún se está impulsando más la investigación de baterías más eficientes y de rápida carga.

Las baterías con base de litio son la última generación de baterías de uso popular. Forman ya parte de nuestra vida, estando presentes en Smartphone, tabletas, ordenadores portátiles, etc.

La gran ventaja de estas baterías es:

- Alta densidad de energía.
- Rápida Carga y ligeras.

El inconveniente es la inestabilidad química que presentan tanto a descargas como a cargas, que obliga a utilizar dispositivos electrónicos que protejan a la batería.

En nuestro caso para el proceso de carga se ha utilizado un cargador especial que tienen en cuenta la problemática de este tipo de baterías, realizando un proceso de carga lo más rápido posible, pero regulando el suministro de carga (disminuyendo la corriente aportada) cuando el nivel de tensión se acerca a 12,6 V y corta el proceso asegurando no sobrepasar esta tensión.

Para evitar los picos instantáneos de descarga bruscos se han utilizado los condensadores que a continuación se describen, con el doble objetivo de cubrir picos instantáneos de corriente demandados tanto por los motores como por el módulo Wifi.

Esto se consigue dimensionando bien la capacidad del sistema o batería de condensadores y es debido a la propia naturaleza de los condensadores que pueden responder a una dinámica transitoria mucho más rápidamente, que la batería, a la vez que salvaguardan a la batería.

Por otra parte, está previsto la incorporación de un circuito que mida el nivel de voltaje de la batería a fin de desconectar esta, avisando al usuario de la necesidad de recargar esta, evitando así el problema de la inestabilidad frente a descargas y prolongando la vida de la batería. (**Ver Anexo 1 Baterías de Litio**).

Los consumos en régimen nominal de todo el sistema son aproximadamente de:

Motores Funcionando con carga nominal---------- 2 • 0.53 A = 1.06 A

Arduino Due ----------0.1 A

Modulo Wifi----------0.15 A

Total: 1.31 A

Consumo máximo en un transitorio de funcionamiento:

Cambio brusco de consigna Motores----------2.5• 2 • 0.53 A = 2.65 A

Arduino Due----------0.2 A

Pico Antena Wifi----------0.25 A

Total: 3.1 A

Por una parte, la batería tiene autonomía más que suficiente para mantener al sistema en régimen nominal de funcionamiento puesto que 2600 mAh > 1310 mAh.

Por otra lado la batería durante breves periodos de tiempo si podría aportar más de 2600 mAh , pero sería imposible cubrir de forma reiterativa varios transitorios de funcionamiento como el calculado anteriormente, provocando la desestabilización química, que se produce en este tipo de baterías como se ha comentado anteriormente, que por una parte harían funcionar de forma incorrecta al sistema de forma global, interrupción de la señal inalámbrica Wifi , una respuesta más lenta de los Motores, y por otro lado acortar la vida útil de la batería.

Aquí es donde entran en juego la dinámica de los condensadores:

Estos condensadores de 10000µF pueden aportar a 1 segundo:

$$\boldsymbol{Q = C \bullet V = 0.01F \bullet 12\,V = 0.12\,C\ en\ 1\ segundo \approx 0.12\,A}$$

La solución adoptada en el diseño del vehículo para cubrir los picos esporádicos provocados por los cambios rápidos de consigna es la de asociar tres condensadores en paralelo de 10000 µF, que dan una respuesta a un segundo de 3 • 0.12 = 0.36 A.

En el caso del módulo Wifi con un condensador, es decir con 0.12 A es más que suficiente puesto que el consumo máximo en un pico de estos no sobrepasa los 0.25 A, y no se pretende que el condensador cubra completamente todo el transitorio, al igual que en el caso de los motores , lo que se pretende es que la batería no haga esfuerzos de descarga bruscos por encima de 2600 mA , es decir que esté como trabajando en régimen permanente, y los condensadores suplan el resto de energía necesaria para el buen funcionamiento del sistema durante estos transitorios.

3.7.- Descripción-Características Microcontrolador ARM M3(ArduinoDue)

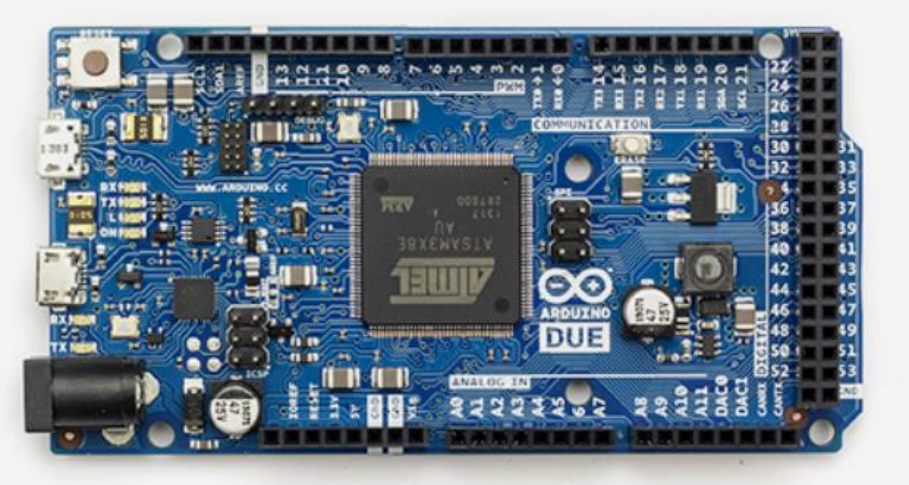


Figura 3.7_1: Arduino Due

Cuando se plantea el problema del Robot móvil se sabe que el microcontrolador elegido tiene que tener implementado un servidor que es el encargado de escuchar la entrada de clientes, en este caso o bien Matlab/ Simulink o bien el dispositivo Android, además tiene que ejecutar el bucle de control de velocidad con un periodo o tiempo de muestro de 10ms, y en el caso del control local, enfoque Robot Industrial, tiene que soportar la ejecución del algoritmo de persecución con un periodo de 100ms.

Para dicho acometido se piensa en un Cortex ARM M3 Figura 3.7_1, o un DSP de la familia C2000 de Texas Instruments.

El Arduino Due es la primera placa de desarrollo de Arduino basado en ARM Cortex M3, que posee un microcontrolador de 32 bits, programable mediante el familiar IDE de Arduino. La gran ventaja del entorno de Arduino, es la gran cantidad de librerías desarrolladas tanto para comunicaciones como en nuestro caso para crear los Sockets necesarios que permiten establecer la comunicación mediante el protocolo TCP/IP y UDP, así como otras muchas como la librería del control de los Timers a bajo nivel que posee el Cortex M3 y otras muchas más como las comunicaciones serie I2c o SPI.

Arduino Due dispone de 54 pines digitales de entrada-salida (de los cuales 12 pueden utilizarse para salidas PWM), 12 entradas analógicas, 4 UARTs (Universal Asíncrona Recepción y transmisión Serie), un reloj de 84 MHz, una

conexión USB OTG, 2 DAC (Salidas digital a analógico), 2 TWI, un conector de alimentación, un cabezal SPI, un cabezal JTAG, un botón de reinicio y un botón de borrado. También hay algunas características interesantes como DACs, Audio, DMA, una biblioteca multitarea experimental y más.

Para poder compilar el código de este microcontrolador Cortex se ha utilizado la versión más reciente del IDE de Arduino 1.7.9. Todos los shields que implementen plenamente la disposición Arduino R3 son compatibles directamente (como el Arduino WiFi y Ethernet Shield).

Este microcontrolador junto con la plataforma Arduino se han elegido, por todo lo dicho anteriormente, plataforma software y librerías gratuitas, así como también las importantes prestaciones que ofrece el Cotex ARM M3 que puede ejecutar 84 millones de instrucciones simples por segundo**,(muy apropiado, permite la resolución suficiente para poder leer la velocidad de las ruedas)**, también la posibilidad en nuestro caso, importantísima de poder trabajar mediante Interrupciones y también la posibilidad de no solo poder visualizar resultados en Simulink sino también, la posibilidad de utilizar la modalidad deploy to hardware, que permite hacer pruebas de programación modular y cargarla directamente al microcontrolador .

Características:

- Microcontrolador: AT91SAM3X8E
- Voltaje de operación: 3.3V
- Voltaje de entrada recomendado: 7-12V
- Voltaje de entrada min/max: 6-20V
- Digital I/O Pins: 54 (de los cuales 12 proveen salida PWM)
- Analog Input Pins: 12
- Analog Outputs Pins: 2
- Corriente total de salida DC en todas las lineas I/O: 130 mA
- Corriente DC para el Pin de 3.3V: 800 mA
- Memoria Flash: 512 KB disponibles para aplicaciones del usuario
- SRAM: 96 KB (two banks: 64KB and 32KB)
- Clock Speed: 84 MHz

Esta información técnica se ha obtenido de la página web:
http://arduino.cl/arduino-due/

3.8.- Descripción-Driver Motores doble Puente en H

Este módulo permite controlar dos motores de corriente continua, pudiendo manejar hasta 2 Amp por canal. Se basa en el circuito integrado L298P, que se describe a continuación un poco más abajo. Posee dos leds por canal (Azul y Amarillo) que indican el sentido de giro (Azul avance hacia adelante, amarillo avance hacia atrás).

Las dos entradas que controlan cada uno de los puentes en H para aplicar el PWM a los Motores, son PWMA, conectada a la salida PWM 3 de Arduino, que en nuestro caso se ocupa del control de la rueda izquierda, y PWMB conectada al PWM 11 de Arduino que se ocupa del control de la rueda derecha.

El cambio de polaridad que permite activar el medio puente en H correspondiente al funcionamiento en un sentido de giro, se elige para el PWMA, con el estado lógico del pin digital 12 de Arduino, y para el PWMB se utiliza el estado lógico del pin digital 13 de Arduino. Las salidas de señal de potencia son llamadas A y B, y son estas, las que se conectan a la alimentación de los motores, siendo el valor medio de tensión de estas señales PWM, las que hacen girar más rápido o despacio cada uno de los Motores.

A continuación, en la Figura 3.8_1 se detalla de forma gráfica, todos los pines de entrada y salida del driver tanto hacia Arduino como hacia los Motores.

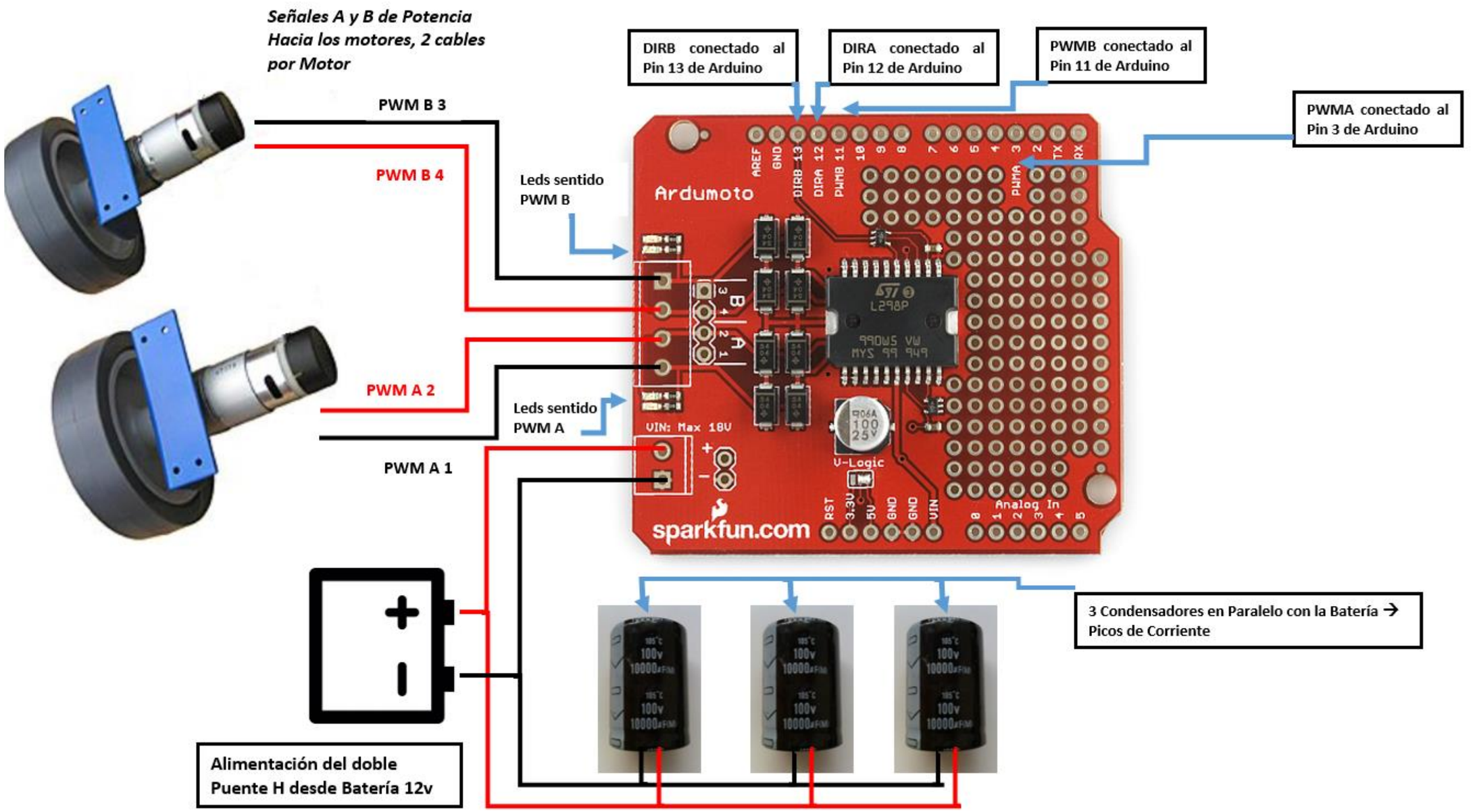


Figura 3.8_1: Driver, Motores y Alimentación

En el apartado Planos, en el apartado Planos, **plano 04 Conexionado Motore-Driver-Microcontrolador**, se especifica con todo detalle el conexionado del driver a los motores, la alimentación de los encoder, la conexión de las señales A y B, en cuadratura a las entradas digitales de Arduino Due, así como las resistencias Pull-Up utilizadas y la alimentación del driver mediante la batería y los condensadores.

Como se puede ver en la Figura 3.8 _1 anterior, el corazón del driver, es el circuito integrado L298P. En el siguiente diagrama se muestra el interior del driver, doble Puente en H y las puertas lógicas que permiten el cambio de polaridad en la alimentación hacia los motores.

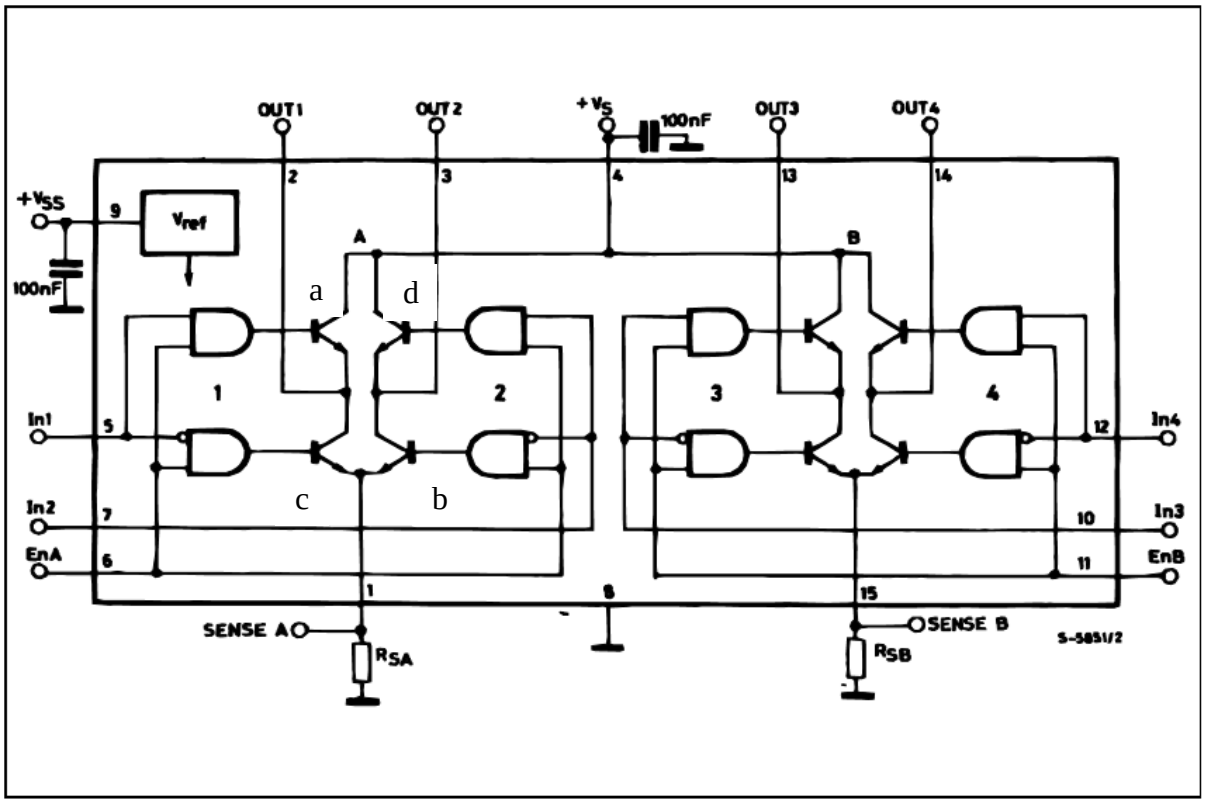


Figura 3.8_2: Circuito Integrado L298P

La señal PWM, de pequeña potencia generada por Arduino en los pines PWM 3 y PWM 11 , son aplicadas directamente a EnA (pin 6 del IC) y EnB (pin 11 del IC) respectivamente.

El bit de dirección DIRA pin digital 12 de Arduino, se aplica a IN1 directamente sin negar la señal y a IN2, con la señal negada, mientras que, el bit de dirección DIRB

pin digital 13 de Arduino, se aplican a IN3 con la señal sin negar y a IN4, con la señal negada.

De esta forma, observando el puente en H A, cuando IN1 vale 1 lógico y IN2 el negado, 0 lógico, lo que se consigue es que conduzcan los transistores de potencia a y b, aplicando esta polaridad a los bornes del motor que, en función de la señal PWM A, conducen más o menos tiempo variando el nivel medio de tensión que se aplica al motor.

En el caso de valer IN1 0 lógico y IN2 el negado, 1 lógico, entonces lo que ocurre es que conducen los transistores de potencia c y d, invirtiendo la polaridad de la señal aplicada al Motor, consiguiendo de esta forma que gire en sentido contrario.

El funcionamiento del Puente en H B es exactamente igual al A.

En el apartado Anexos **anexo 2, Esquema electrónico Driver Puente en H**, se puede analizar con mayor detalle y ver la correspondencia de los pines del driver con Arduino.

3.9.- Descripción-Características Modulo Shield wifi Arduino

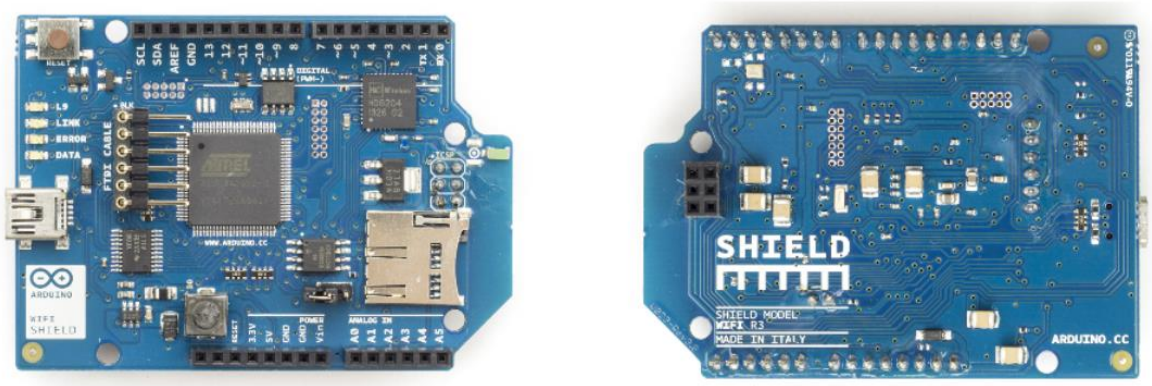


Figura 3.9_1: Shield wifi Arduimo

Este módulo Wifi permite conectar Arduino Due a una red de área local de forma inalámbrica. Se basa en la especificación inalámbrica 802. 11. Este módulo wifi posee dos circuitos integrados importantes por una parte el HDG204 802.11b , es el que posee la antena y por lo tanto, es el encargado de la transmisión y recepción

de datos. La frecuencia de la señal propagada está en el rango de 2400 a 2500 MHz.

Por otro lado, el módulo wifi contiene un AT32UC3, que es un microcontrolador Atmel de 32 bits, que es el que se encarga del empaquetado TCP/IP o UDP.

Una característica importante de este módulo es que la comunicación con Arduino Due se hace a través de la cabecera ICSP que utiliza el bus de comunicaciones SPI, dejando libre la parte superior del módulo wifi para poder apilar en este caso el driver que controla los Motores.

4.- DESARROLLO Y PLAN DE TRABAJO

4.1.- Esquema-Cronograma de las tareas Realizadas.

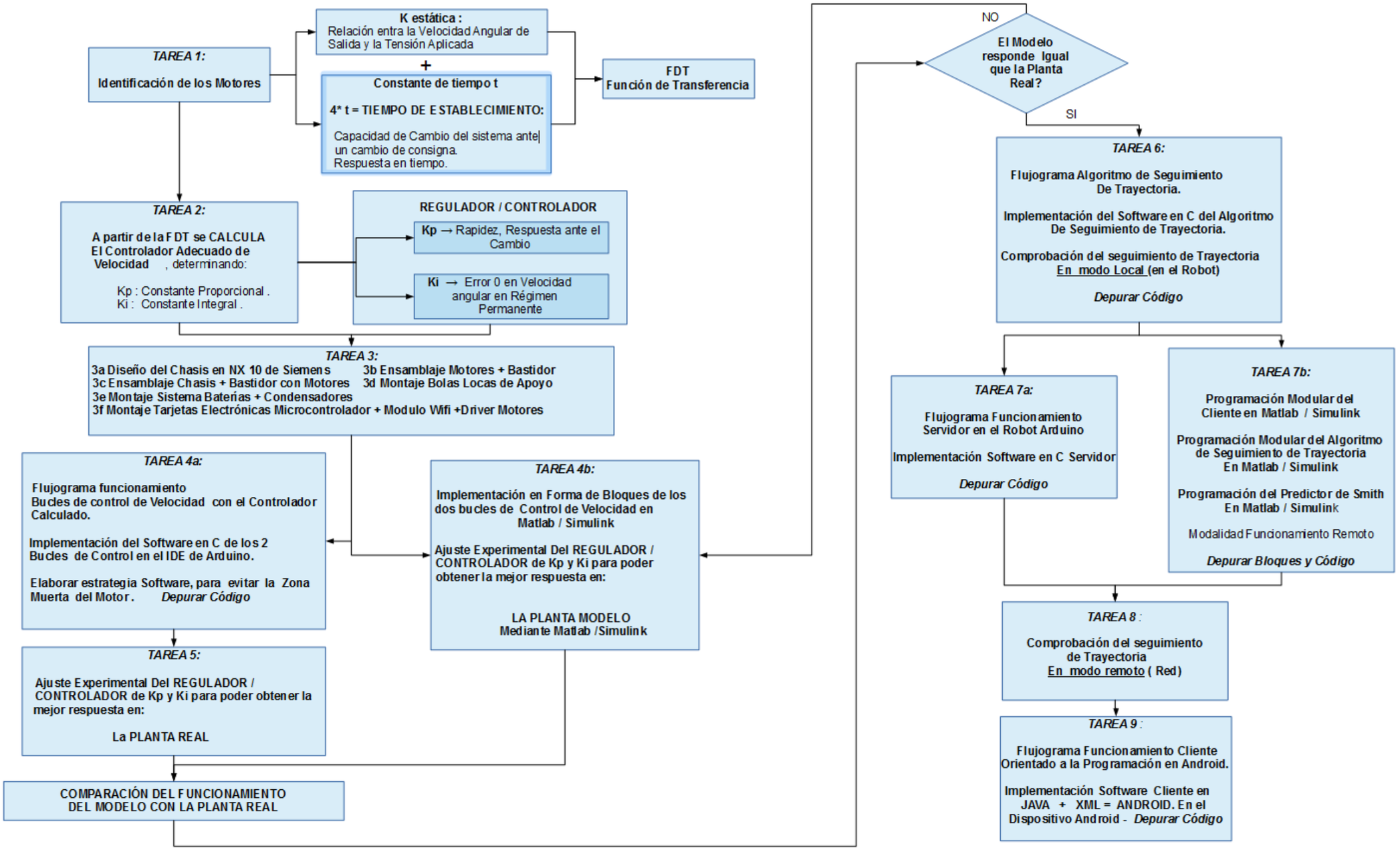


Figura 4.1_1: Tareas Realizadas

Para poder afrontar un proyecto de esta índole y además con el doble enfoque, investigador, por una parte, y vehículo Robot Industrial por otro lado, se ha seguido un proceso relativamente complicado en el que el primer paso es la identificación de los motores junto con el driver controlador o doble puente en H, para determinar la función de transferencia del sistema que permite diseñar el controlador de velocidad de las ruedas.

Dicha identificación no solo sirve para poder elaborar el controlador(regulador) apropiado como se muestra a continuación, sino que, además permite obtener un modelo del sistema o planta, que se puede utilizar para experimentar el proceso real, y es la base para la implementación de un predictor de Smith que puede utilizarse para mitigar los efectos del retraso de señal inherentes siempre en los controles basados en red.

A continuación, se sigue con el cálculo teórico de los dos bucles de control de velocidad con sus respectivos reguladores PI. Dichos bucles de control se tienen que comprobar a fin de buscar la mejor respuesta, y esto se consigue por una parte mediante la elaboración de un flujograma que representa el modo de operar que el software implementado debe conseguir y la implementación del propio software para paso seguido, hacer pruebas experimentales con el objetivo de conseguir el mejor ajuste de las acciones Proporcional e Integral, que aseguren el mejor comportamiento tanto en régimen permanente, como en los transitorios de funcionamiento, y por otra parte se ha comprobado el funcionamiento del modelo y los reguladores ajustando tanto la acción Proporcional como Integral en Matlab / Simulink, permitiendo comparar la respuesta del modelo con la planta o sistema real.

Una vez se poseen los dos bucles de control optimizados y teniendo en cuenta la aplicación de una estrategia software que se encargue de corregir las dos no linealidades que aparecen siempre en el control de un motor de CC, que son la zona Muerta y el límite de saturación, se pasa a la implementación simultanea de un servidor en el dispositivo móvil o Robot, que se encarga de escuchar la entrada de un posible cliente, y la implementación del cliente en un primer momento desde Matlab / Simulink que ha permitido experimentar el control en red y monitorizar todo tipo de respuesta (enfoque investigador).

En esta primera opción siendo el cliente Simulink, el algoritmo de persecución, que como ya sabemos del apartado fundamentos teóricos, es el que permite al Robot seguir una determinad trayectoria, es implementado en Matlab / Simulink. Quedan pues los dos bucles de control de velocidad (uno para cada rueda, accionamiento

diferencial), implementados de forma local en el Robot, y digamos en un nivel superior a través de la red inalámbrica se cierra el control del seguimiento de trayectoria mediante el algoritmo de seguimiento de Trayectoria.

En el segundo enfoque Robot Industrial, tanto el algoritmo de seguimiento de trayectoria como los bucles de control de velocidad de cada uno de los dos motores, quedan en forma local en el servidor Robot, mientras que desde el dispositivo Android, (cliente), solo se encarga de establecer la comunicación, mandar la trayectoria que se desea que ejecute el Robot y monitorizar el correcto seguimiento de la trayectoria.

Es importante recalcar que el enfoque investigador que es el primero que se ha realizado, ha permitido no solo visualizar la respuesta de los motores y por lo tanto la correcta identificación de los mismos, sino que permite implementar un filtro de velocidad, el algoritmo de persecución, medir el retraso de la señal en red, desde la ida de la señal, desde el cliente hasta el servidor y vuelta de nuevo al cliente, para poder implementar el predictor de Smith, así como una valiosa simulación del todo el sistema que interaccionando con el programa NX 10 de Siemens, permite visualizar esta simulación con todo detalle, incluso viendo al vehículo trazar la trayectoria en la simulación.

Tanto el diseño del Chasis como del bastidor mediante NX 10 de Siemens se ha levantado (impreso) mediante una impresora 3D, que posee el Instituto de Automática e Informática (ai2), y se explica en este apartado en el punto 4.3.

4.2.- Identificación de los Motores

Mediante la identificación de los motores se consigue relacionar por una parte la magnitud de salida del motor en $\omega_{salida\ (\frac{rad}{s})}$ con la de entrada $V_{aplicada}$ o $Escalones_{PWM}$, que, indica la relación existente entre ambas, en régimen permanente de funcionamiento y por otra parte cual es la respuesta transitoria del sistema (en tiempo), ante un cambio en la entrada.

Esta identificación de los motores se ha realizado para cada uno de los motores, primero con las ruedas al aire y después con las ruedas en contacto en el suelo permitiendo determinar, además, no solamente la respuesta real del sistema con el peso total del mismo, sino también, cuales son las correcciones necesarias a aplicar para evitar que el motor cuando circula a revoluciones bajas entre dentro de la zona muerta.

4.2.1.- **Motor 1 (Rueda Derecha), Zona Muerta**

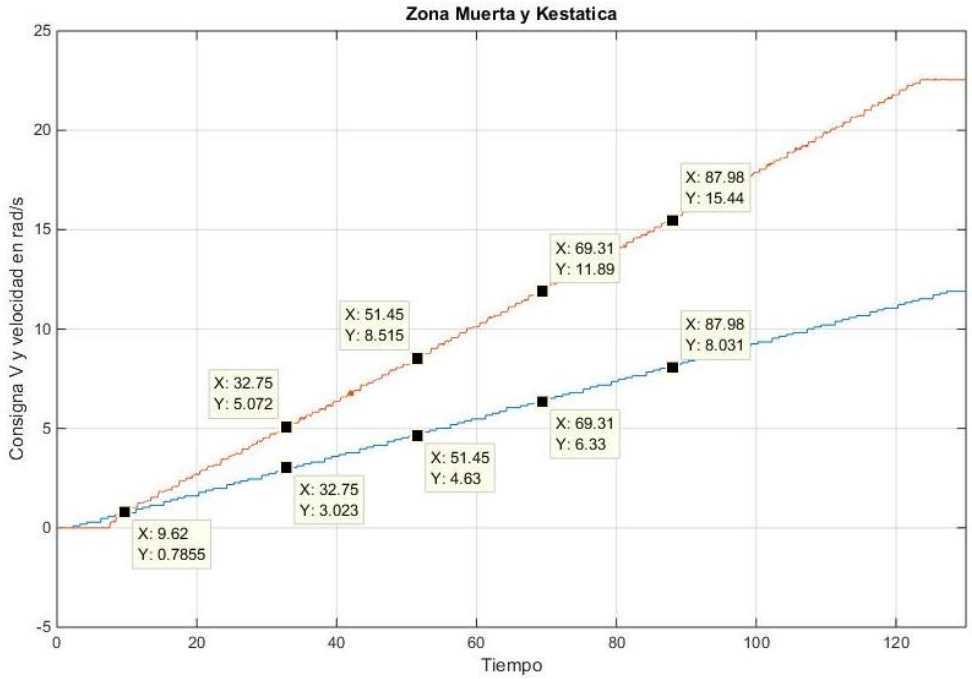


Figura 4.1.1_1: Zona Muerta Motor 1 sentido adelante

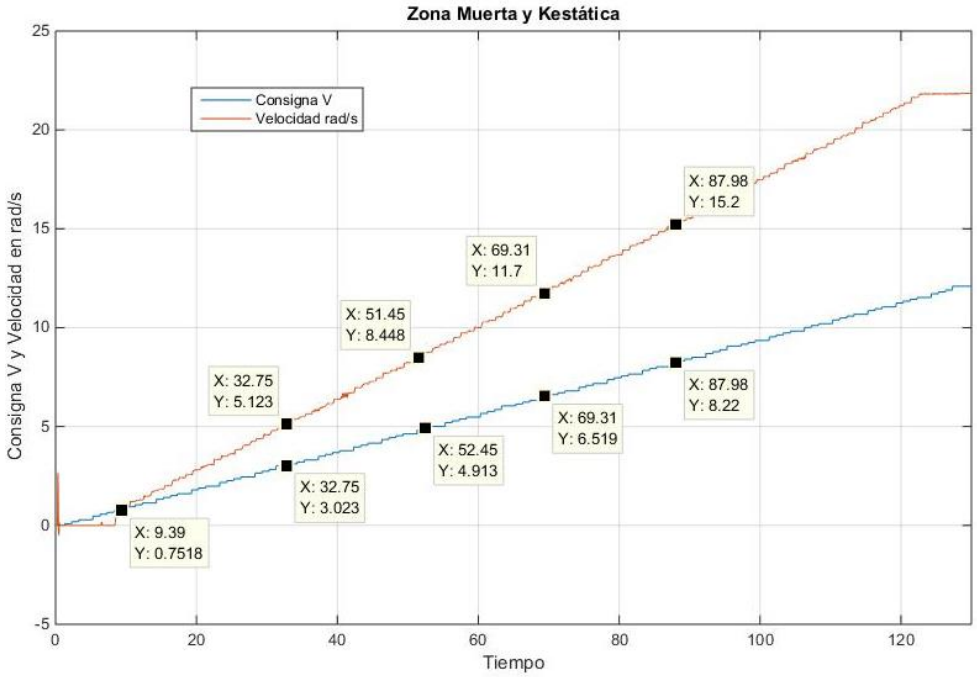


Figura 4.1.1_2: Zona Muerta Motor 1 sentido atrás

En las Figuras 4.1.1_1 y 4.1.1_2 en azul se muestra la consigna en voltios aplicada al motor y en marrón la respuesta en velocidad. Esta zona muerta es la determinada con las ruedas al aire sin contacto en el suelo y por supuesto sin el peso de la batería y la carrocería, y es de aproximadamente:

$$\frac{0{,}788 + 0.7518}{2} = 0.769\ v\ que\ es\ el\ \frac{0.769\ v}{12\ v} = 6{,}4\ \%$$

Si lo expresamos en forma de escalones (PWM) en el rango de 0 a 255 como se han planteado los controladores PI, que se han diseñado para cada uno de los motores:

$$\frac{0.769\ v}{12\ v} \bullet 255 \approx 17\ escalones$$

En las pruebas definitivas con el vehículo en contacto con el suelo, alimentado desde la batería y con el peso real que tiene que arrastrar que es de 1.4 Kg, se demuestra que es necesario aplicar uno 49 escalones del total de 255 para poder sacar el vehículo de la zona Muerta que pasa a ser:

$$\frac{49}{255} \approx 19\ \%$$

Con lo cual queda demostrado que la verdadera prueba de fuego para cualquier vehículo o Robot móvil se realiza con las ruedas en contacto con el suelo y con el peso real que tiene que arrastrar el vehículo.

4.2.2.- **Motor 1 (Rueda Derecha), K estática**

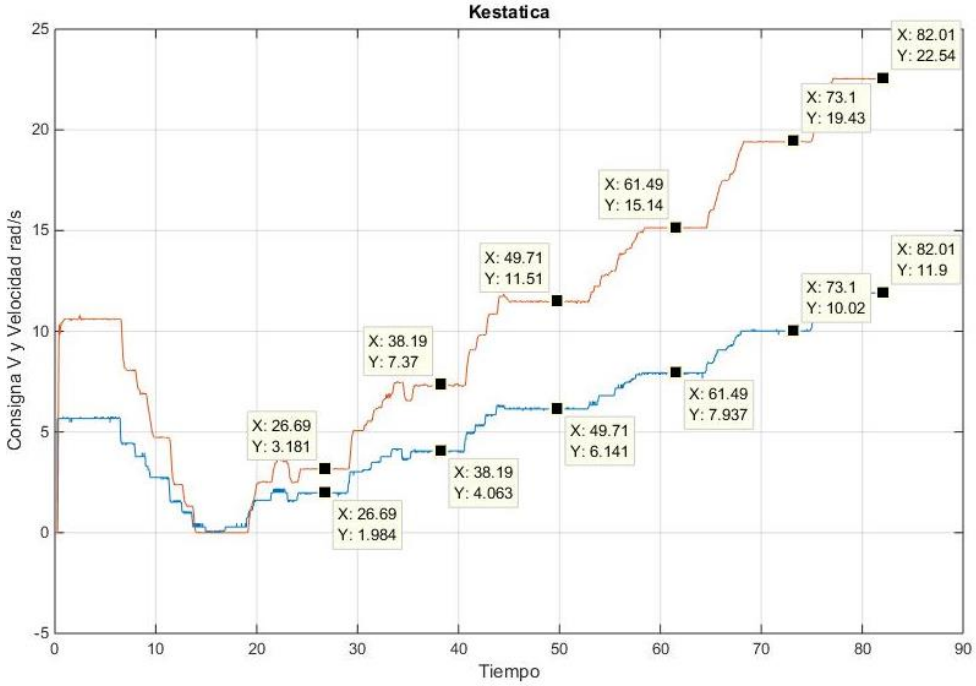


Figura 4.1.2_1: K estática Motor 1 sentido adelante

$$Kest1 = \frac{3.181}{1.984} = 1.6 \quad Kest2 = \frac{7.37}{4.063} = 1.814 \quad Kest3 = \frac{15.14}{7.937} = 1.91$$

$$Kest4 = \frac{19.43}{10.02} = 1.94 \quad Kest5 = \frac{22.54}{11.9} = 1.89$$

$$Kest(media) = \frac{1.6 + 1.814 + 1.91 + 1.94 + 1.89}{5} = \mathbf{1.83}$$

Si expresamos dicha K estática como la relación entre la ω_{salida} y los escalones (PWM) aplicados obtenemos:

$$Kest1 = \frac{3.181}{42} = 0.07423 \quad Kest2 = \frac{7.37}{98} = 0.075 \quad Kest3 = \frac{15.14}{168} = 0.09$$

$$Kest4 = \frac{19.43}{213} = 0.091 \quad Kest5 = \frac{22.54}{252} = 0.089$$

$$Kest(media) = \frac{0.07423 + 0.075 + 0.09 + 0.091 + 0.089}{5} = \mathbf{0.08384}$$

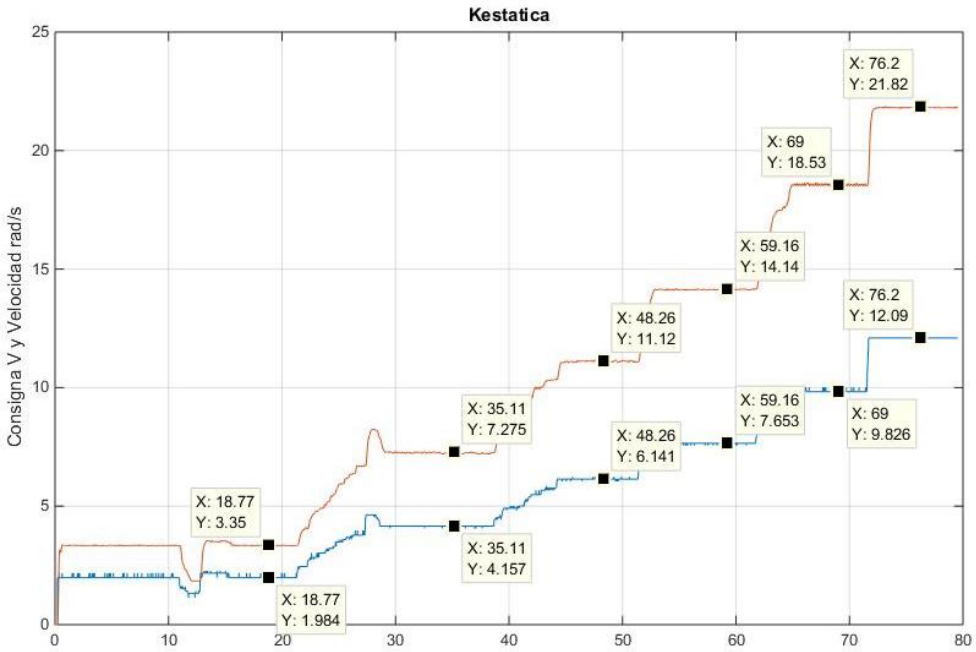


Figura 4.1.2_2: K estática Motor 1 sentido atrás

$$Kest1 = \frac{3.35}{1.984} = 1.688 \quad Kest2 = \frac{7.275}{4.157} = 1.75 \quad Kest3 = \frac{11.12}{6.141} = 1.81$$

$$Kest4 = \frac{14.14}{7.653} = 1.847 \quad Kest5 = \frac{18.53}{9.826} = 1.885$$

$$Kest(media) = \frac{1.688 + 1.75 + 1.81 + 1.847 + 1.885}{5} = \mathbf{1.796}$$

Expresando ahora al igual que antes, la K estática como la relación entre la ω_{salida} y los escalones (PWM) tenemos:

$$Kest1 = \frac{3.35}{42} = 0.0797 \quad Kest2 = \frac{7.275}{88} = 0.0826 \quad Kest3 = \frac{11.12}{130} = 0.0855$$

$$Kest4 = \frac{14.14}{163} = 0.0867 \quad Kest5 = \frac{18.53}{209} = 0.0886$$

$$Kest(media) = \frac{0.0797 + 0.0826 + 0.0855 + 0.0867 + 0.0886}{5} = \mathbf{0.0846}$$

Al igual que en el caso anterior, en las dos Figuras 4.1.2_1 y 4.1.2_2 en el eje de ordenadas tenemos en azul la tensión aplicada al motor, en marrón la respuesta a bucle abierto del motor en velocidad rad/s y en abscisas el tiempo en segundos, quedando clara esta relación para el Motor 1 como:

$$\frac{\omega_{salida\ (\frac{rad}{s})}}{V_{aplicada}} = \frac{1.83 + 1.81}{2} = 1.82$$

O también:

$$\frac{\omega_{salida\ (\frac{rad}{s})}}{Escalones_{PWM}} = \frac{0.08384 + 0.0846}{2} = 0.0842$$

4.2.3.- **Motor 1 Tiempo de Establecimiento**

Para determinar el tiempo que el motor tarda en responder ante una entrada escalón, lo cual es necesario para cuantificar la rapidez con que el sistema responde ante los cambios de consigna se procede:

1 Determinar la constante de tiempo t:

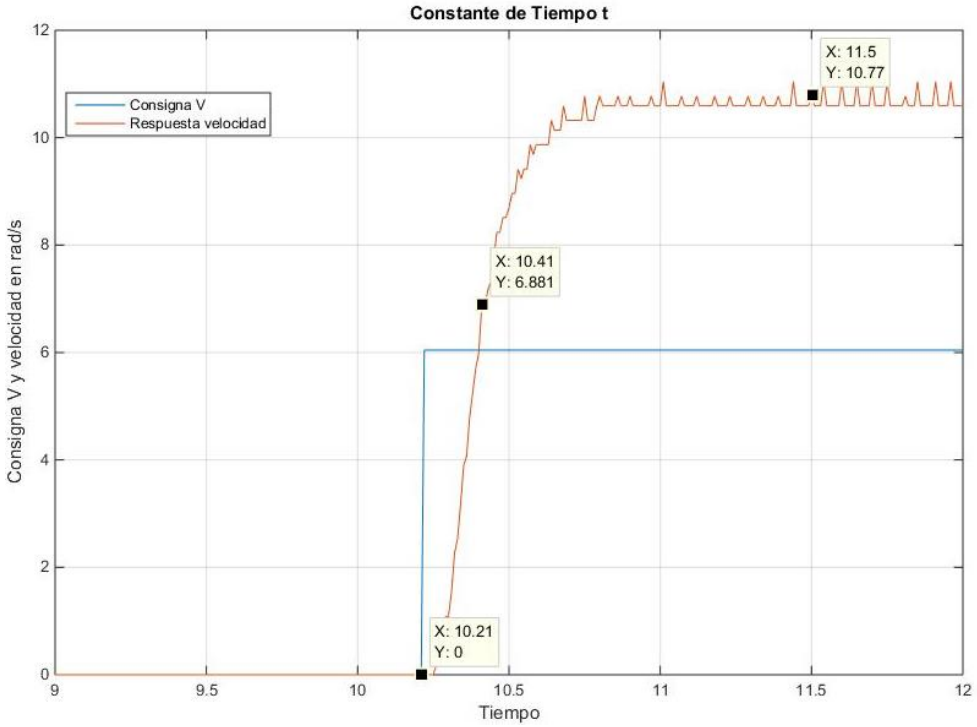


Figura 4.2.3_1: Motor 1 Constante de Tiempo t sin filtro Velocidad

De la Figura 4.2.3_1 sabiendo que en las ordenadas tenemos en azul un escalón de consigna de tensión en voltios aplicada al motor, y en marrón la respuesta en velocidad en $\frac{rad}{s}$, y que en las abscisas tenemos el tiempo en segundos, se determina esta constante de tiempo, como el tiempo que transcurre desde que aplicamos la consigna de tensión hasta que se alcanza el 63% del valor final que alcanza la velocidad para esta consigna.

t = Constante de Tiempo = 10.41 – 10.21 = 0.20 s

El tiempo de establecimiento al 98% es de $te = 4 * t$ = 4*0.20 = 0.800 s **800 ms**

El máximo periodo de muestreo que deberíamos de utilizar es de: $\frac{te}{10} = \frac{0.80}{10} = 0.080\ s\ \approx 100\ ms$

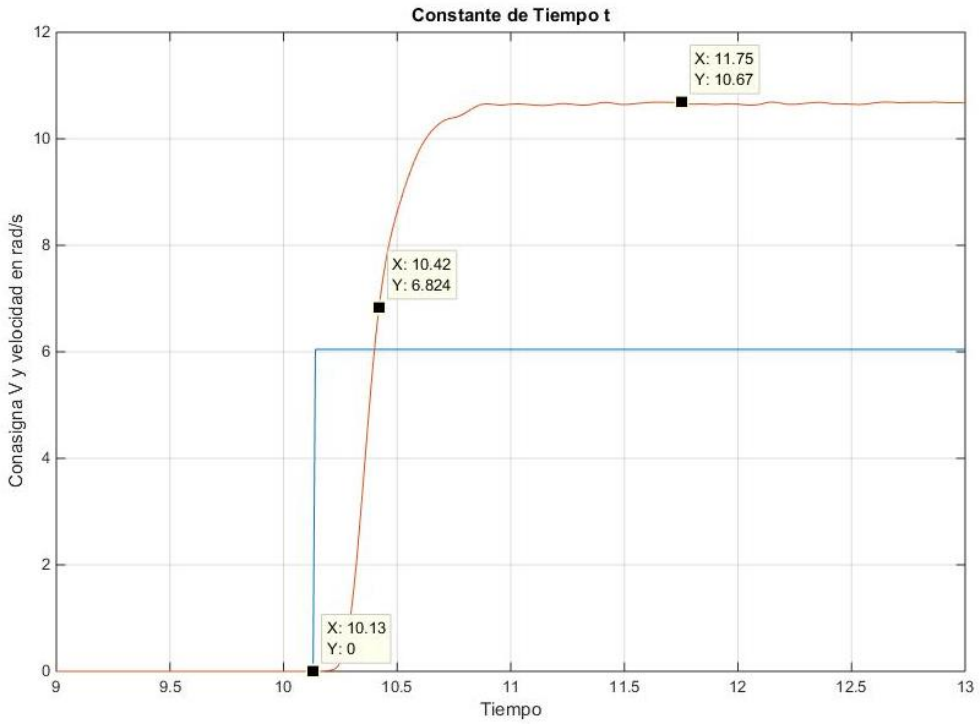


Figura 4.2.3_2: Motor 1 Constante de Tiempo t con filtrado de Velocidad

Como se puede apreciar en la Figura 4.2.3_2 al aplicar un filtrado a la velocidad medida por los encoders, por una parte, queda claro que la señal es mucha más nítida, ya que el ruido mecánico queda eliminado pero la contrapartida, es que el sistema se vuelve más lento, siento la constante de tiempo t y el tiempo de establecimiento los siguientes:

t = Constante de Tiempo = 10.42 – 10.13 = 0.29 s

El tiempo de establecimiento al 98% es de $te = 4 * t$ = 4*0.290 = **1.16 s**

El periodo máximo de muestreo que deberíamos de utilizar es de: $\frac{te}{10} = \frac{1.16}{10} = 0.116\ s\ \approx 120\ ms$

4.2.4.- **Motor 1 FDT**

La función de transferencia para el Motor 1 que servirá como modelo para todas las pruebas experimentales y como referencia para poder calcular el regulador PI queda como:

$\frac{\omega_s}{V_s} = \frac{1.82}{0.29 \cdot s + 1}$ Expresando la velocidad de salida en $\frac{rad}{s}$ con respecto a la tensión aplicada en voltios, y para el caso más desfavorable con el filtrado de la velocidad.

$\frac{\omega_s}{escal_{PWM}} = \frac{0.0846}{0.29 \cdot s + 1}$ Expresando la velocidad de salida en $\frac{rad}{s}$ con respecto a los escalones aplicados en el (PWM) en el rango de 0 a 255, y para el caso más desfavorable con el filtrado de la velocidad.

4.2.5.- **Motor 2 (Rueda Izquierda), Zona Muerta**

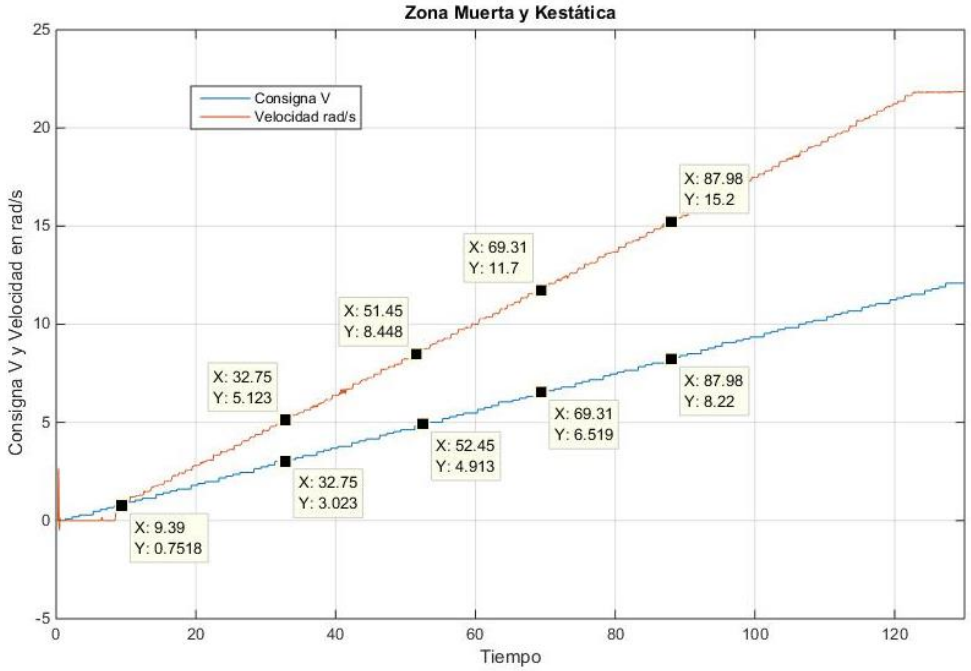


Figura 4.2.5_1: Zona Muerta Motor 2 sentido adelante

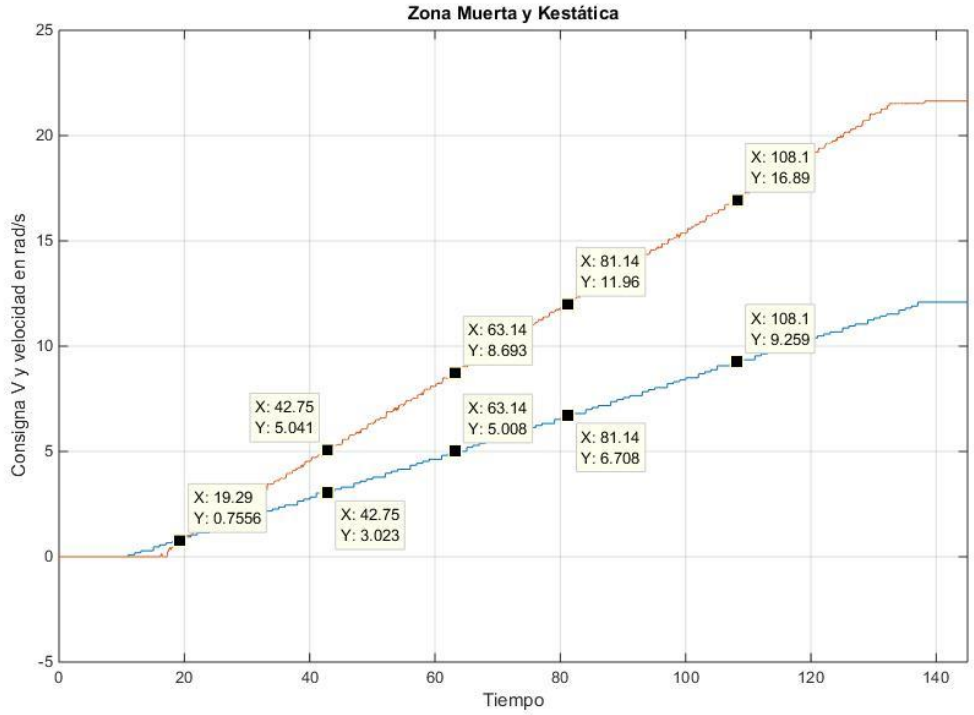


Figura 4.2.5_2: Zona Muerta Motor 2 sentido atrás

Siguiendo la misma dinámica que en los casos anteriores en azul tenemos la tensión aplicada a los motores y en marrón la respuesta en velocidad rad/s, se determina la zona muerta con las ruedas al aire sin contacto en el suelo y por supuesto sin el peso de la batería y la carrocería. La tensión necesaria para sacar al motor de la zona muerta es de aproximadamente:

$$\frac{0{,}7518 + 0.7556}{2} = 0.7537\ v\ que\ es\ el\ \frac{0.7537\ v}{12\ v} = 6{,}3\ \%$$

Si lo expresamos en forma de escalones (PWM) en el rango de 0 a 255 como se han planteado en el controlador PI, que se ha diseñado para cada uno de los motores:

$$\frac{0.7537\ v}{12\ v} \bullet 255 \approx 16\ escalones$$

En las pruebas definitivas con el vehículo en contacto con el suelo, alimentado desde la batería y con el peso real que tiene que arrastrar que es de 1.4 Kg, se demuestra que es necesario aplicar unos 50 escalones del total de 255 para poder sacar el vehículo de la zona Muerta que pasa a ser:

$$\frac{50}{255} \approx 20\ \%$$

4.2.6.- **Motor 2 (Rueda Izquierda), K estática**

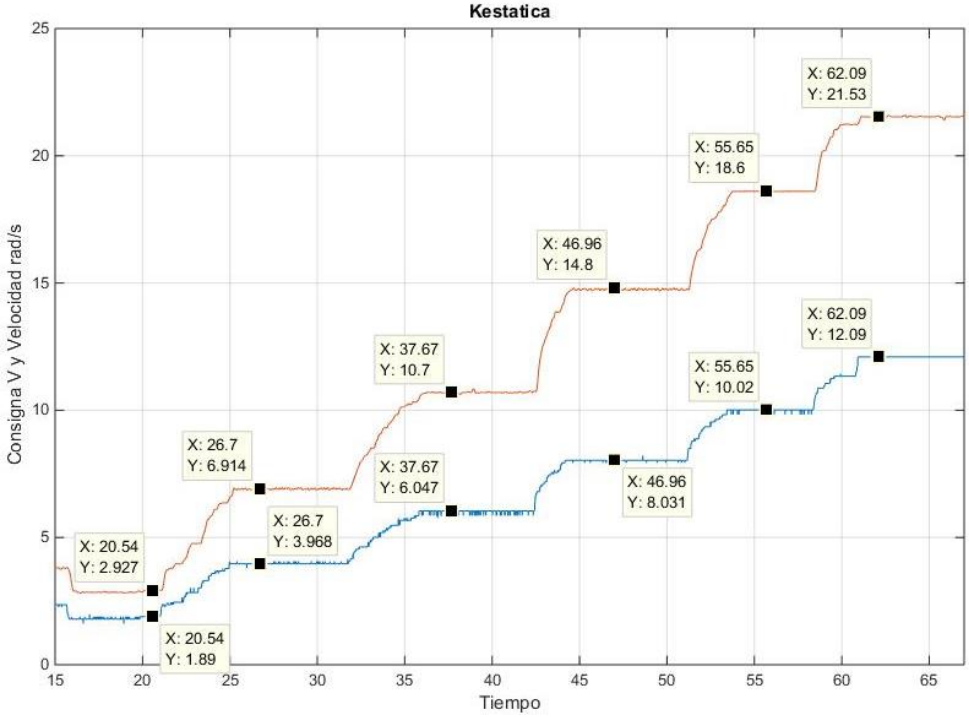


Figura 4.2.6_1: K estática Motor 2 sentido Adelante

$$Kest1 = \frac{2.636}{1.606} = 1.641 \quad Kest2 = \frac{6.941}{4.063} = 1.71 \quad Kest3 = \frac{10.62}{6.047} = 1.7562$$

$$Kest4 = \frac{14.46}{7.937} = 1.822 \quad Kest5 = \frac{18.36}{10.04} = 1.83$$

$$Kest(media) = \frac{1.641 + 1.71 + 1.7562 + 1.822 + 1.83}{5} = \mathbf{1.75}$$

Si expresamos dicha K estática como la relación entre la ω_{salia} y los escalones (PWM) aplicados obtenemos:

$$Kest1 = \frac{2.636}{34} = 0.07753 \quad Kest2 = \frac{6.941}{86} = 0.08 \quad Kest3 = \frac{10.62}{137} = 0.07752$$

$$Kest4 = \frac{14.46}{168} = 0.08607 \quad Kest5 = \frac{18.36}{213} = 0.0862$$

$$Kest(media) = \frac{0.07753 + 0.08 + 0.07752 + 0.08607 + 0.0862}{5} = \mathbf{0.0814}$$

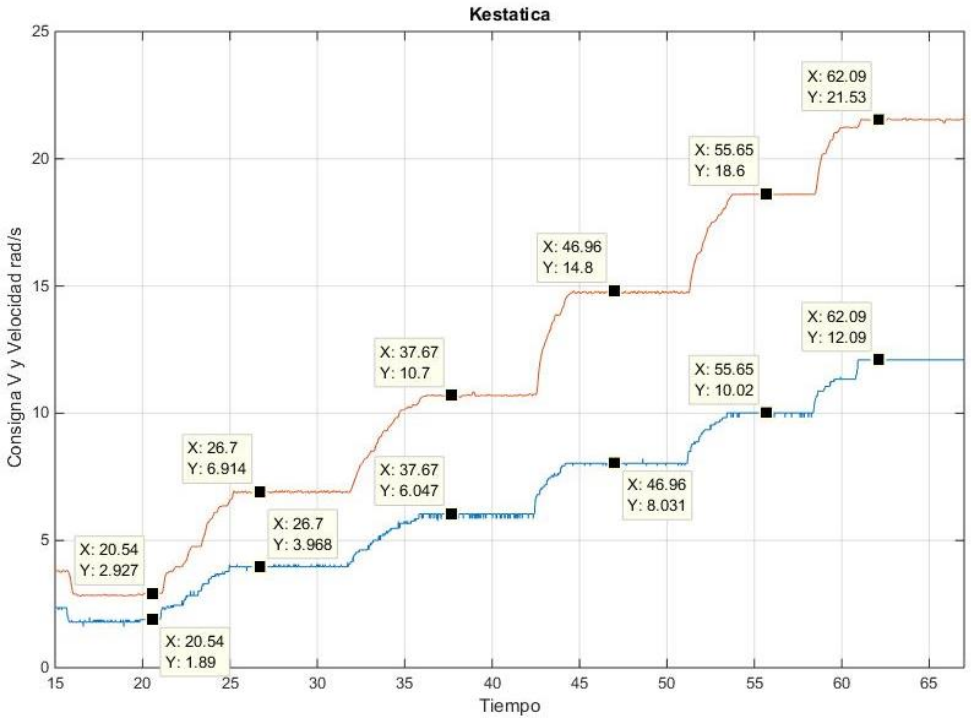


Figura 4.2.6_2: K estática Motor 2 sentido Atrás

$$Kest1 = \frac{2.927}{1.89} = 1.5486 \quad Kest2 = \frac{6.914}{3.968} = 1.74 \; Kest3 = \frac{10.07}{6.047} = 1.67$$

$$Kest4 = \frac{14.8}{8.031} = 1.843 \quad Kest5 = \frac{18.6}{10.02} = 1.856$$

$$Kest(media) = \frac{1.5486 + 1.74 + 1.67 + 1.843 + 1.856}{5} = \mathbf{1.73}$$

Expresando ahora K estática como la relación entre la ω_{salia} y los escalones (PWM) aplicados obtenemos:

$$Kest1 = \frac{2.927}{40} = 0.0731 \quad Kest2 = \frac{6.914}{84} = 0.0823 \; Kest3 = \frac{10.07}{128} = 0.0786$$

$$Kest4 = \frac{14.8}{171} = 0.08656 \quad Kest5 = \frac{18.6}{213} = 0.08732$$

$$Kest(media) = \frac{0.0731 + 0.0823 + 0.0786 + 0.08656 + 0.0862}{5} = \mathbf{0.0813}$$

Al igual que se hizo con el Motor 1 y observando las dos gráficas Figuras 4.2.6_1 y 4.2.6_2 que como ya sabemos en el eje de ordenadas tenemos en azul la tensión aplicada al motor, en marrón la respuesta a bucle abierto del motor en velocidad rad/s y en abscisas el tiempo en segundos, queda clara esta relación para el Motor 2 como:

$$\frac{\omega_{salida\ (\frac{rad}{s})}}{V_{aplicada}} = \frac{1.75 + 1.73}{2} = 1.74$$

O también:

$$\frac{\omega_{salida\ (\frac{rad}{s})}}{Escalones_{PWM}} = \frac{0.0814 + 0.0813}{2} = 0.0838$$

4.2.7.- **Motor 2 Tiempo de Establecimiento**

Al igual que para el Motor 1 y siguiendo los mismos pasos, se determina el tiempo de establecimiento. Que como ya sabemos es el tiempo transcurrido desde que se aplica a la entrada un escalón de tensión y se estabiliza la señal de salida, régimen permanente.

1 Determinar la constante de tiempo t:

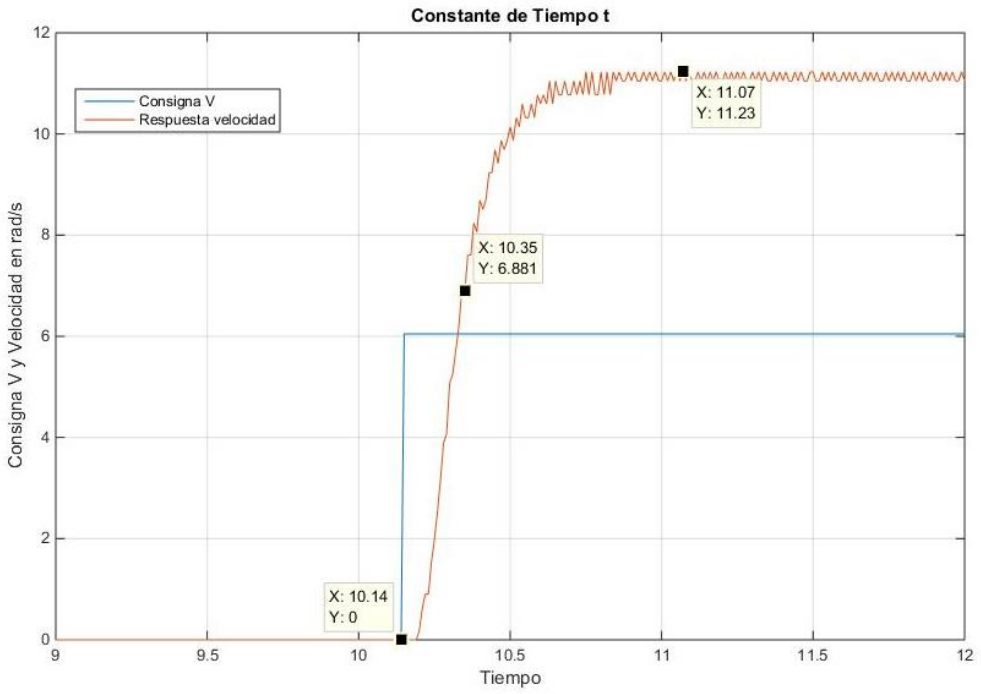


Figura 4.2.7_1: Motor 2 Constante de Tiempo t sin filtro Velocidad

Al igual que para el Motor 1 de la gráfica anterior Figura 4.2.7_1 sabiendo que en las ordenadas tenemos en azul un escalón de consigna de tensión en voltios aplicada al motor, y en marrón la respuesta en velocidad en $\frac{rad}{s}$, y que en las abscisas tenemos el tiempo en segundos, se determina esta constante de tiempo, como el tiempo que transcurre desde que aplicamos la consigna de tensión hasta que se alcanza el 63% del valor final que alcanza la velocidad para esta consigna.

t = Constante de Tiempo = 10.35 – 10.14 = 0.21 s

El tiempo de establecimiento al 98% es de $te = 4 * t$ = 4*0.21 = 0.840 s **840 ms**

El máximo periodo de muestreo que deberíamos de utilizar es de: $\frac{te}{10} = \frac{0.840}{10} = 0.084\ s\ \approx 100\ ms$

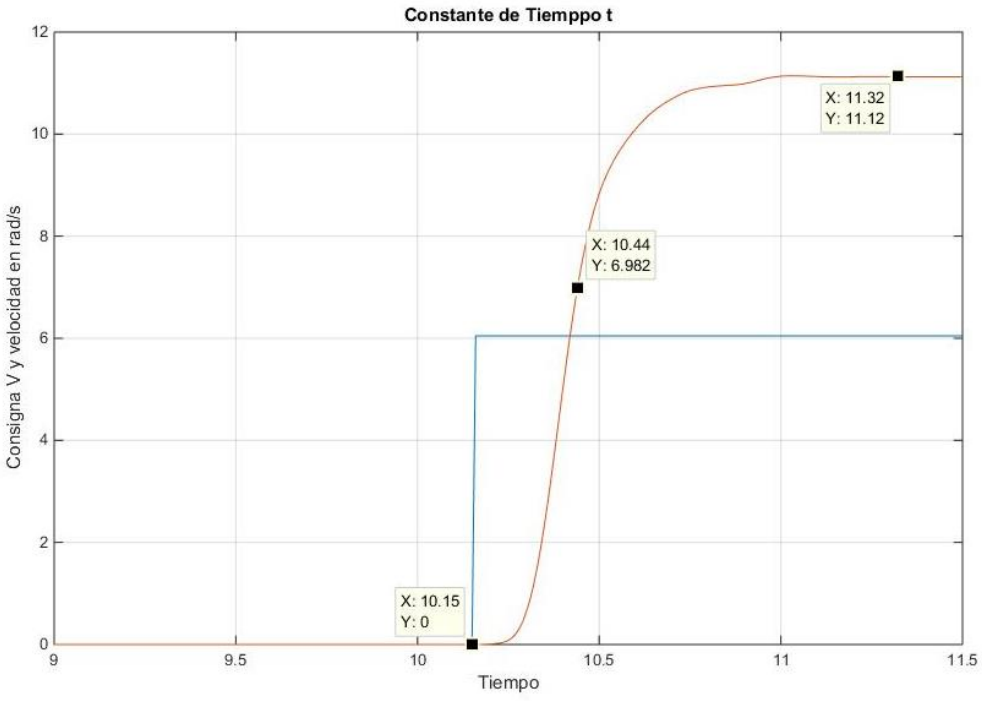


Figura 4.2.7_2: Motor 2 Constante de Tiempo t con filtro Velocidad

Como en el caso anterior del Motor 1 ahora para el Motor 2, se puede apreciar en la gráfica anterior que, al aplicar un filtrado a la velocidad medida por los encoders, por una parte, la señal es mucha más nítida, ya que el ruido mecánico queda eliminado pero la contra partida, es que el sistema se vuelve más lento, siento la constante de tiempo t y el tiempo de establecimiento los siguientes:

t = Constante de Tiempo = 10.44 – 10.15 = 0.29 s

El tiempo de establecimiento al 98% es de $te = 4 * t$ = 4*0.290 = **1.16 s**

El periodo máximo de muestreo que deberíamos de utilizar es de: $\frac{te}{10} = \frac{1.16}{10} = 0.116\ s\ \approx 120\ ms$

4.2.8.- **Motor 2 FDT**

La función de transferencia para el Motor 2 que servirá como modelo para todas las pruebas experimentales y como referencia para poder calcular el regulador PI queda como:

$\frac{\boldsymbol{\omega_s}}{\boldsymbol{V_s}} = \frac{\mathbf{1.74}}{\mathbf{0.29 \cdot s + 1}}$ Expresando la velocidad de salida en $\frac{rad}{s}$ con respecto a la tensión aplicada en voltios, y para el caso más desfavorable con el filtrado de la velocidad.

$\frac{\boldsymbol{\omega_s}}{\boldsymbol{escal_{PWM}}} = \frac{\mathbf{0.0838}}{\mathbf{0.29 \cdot s + 1}}$ Expresando la velocidad de salida en $\frac{rad}{s}$ con respecto a los escalones aplicados en el (PWM) en el rango de 0 a 255, y para el caso más desfavorable con el filtrado de la velocidad.

4.3.- Cálculo experimental del Regulador PI de Velocidad

Para obtener unas prestaciones adecuadas en los motores se determina la constante proporcional (Kp) que permita la reacción del sistema adecuada ante cambios transitorios, debidos al cambio de consigna, a perturbaciones, o pequeños cambios en el sistema mecánico, y por otra parte se determina la constante (Ki), que además de ayudar al sistema durante la respuesta transitoria, sumando a la acción proporcional (Kp), permite conseguir error nulo de la magnitud controlada, en este caso velocidad en régimen permanente.

La función de transferencia calculada en el apartado 4.2.4:

$$\frac{\boldsymbol{\omega_s}}{\boldsymbol{escal_{PWM}}} = \frac{\mathbf{0.0846}}{\mathbf{0.21 \cdot s + 1}}$$

La podemos expresar haciendo la siguiente transformación:

$$Gp(s) = \frac{Ke}{t \cdot s + 1} = \frac{\frac{Ke}{t}}{s + \frac{1}{t}} = \frac{\frac{0.0846}{0.21}}{s + \frac{1}{0.21}} = \frac{0.4028}{s + 4.762}$$

Esta es la **FDT** de la Planta o sistema de **la Rueda Derecha**.

A continuación, repetimos el procedimiento anterior para expresar la función de transferencia del motor de la rueda Izquierda:

$$\frac{\boldsymbol{\omega_s}}{\boldsymbol{escal_{PWM}}} = \frac{\mathbf{0.0838}}{\mathbf{0.21 \cdot s + 1}}$$

$$Gp(s) = \frac{Ke}{t \cdot s + 1} = \frac{\frac{Ke}{t}}{s + \frac{1}{t}} = \frac{\frac{0.0838}{0.21}}{s + \frac{1}{0.21}} = \frac{0.399}{s + 4.762}$$

Esta es la **FDT** de la Planta o sistema de la **Rueda Izquierda**.

4.3.1.- **Acción Proporcional**

Para la obtención de la constante Kp, se introduce un escalón de tensión al sistema que aproximadamente fije un cambio en la salida de velocidad de más o menos la mitad de la velocidad máxima que es capaz de alcanzar el motor, y ajustamos la acción de control aumentando dicha constante proporcional de forma que dicha acción máxima este en unos 8 v, que es el 66 % de los 12 v máximos que podemos suministrar al motor, con el objetivo de conseguir el mejor tiempo de establecimiento.

Es decir, se trata de aumentar Kp, obteniendo el menor tiempo de establecimiento sin que llegue a saturar la acción de control.

Esta Kp experimentalmente se ha determinado para ambos motores en Kp = 7.65.

4.3.2.- **Acción Integral**

Una vez ajustada Kp, calculamos Ki de tal forma que se cancele el polo del motor:

Sabiendo que nuestra FDT posee acción proporcional Kp y acción integral Ki podemos expresar como *Gr(s)*:

$$Gr(s) = Kp + Ki * \frac{1}{s} = \frac{Kp * s + Ki}{s} = Kp\frac{s + \frac{Ki}{Kp}}{s}$$

Luego el cero de la FDT del regulador es: $s + \frac{Ki}{Kp}$ si igualamos al polo de la función de transferencia de la planta con este cero de la FDT del regulador:

$$s + \frac{Ki}{Kp} = s + \frac{1}{t} \rightarrow \frac{Ki}{Kp} = \frac{1}{t} \rightarrow Ki = \frac{Kp}{t} = \frac{7.65}{0.21} = 36.4285$$

Conseguimos teóricamente cancelar el polo de la FDT del Motor con el cero de la FDT del regulador y determinamos así el valor de nuestra acción integral Ki.

Sin variar el valor de la constante Kp, se aumenta el valor de la constante Ki buscando más o menos el valor calculado hasta conseguir la respuesta más rápida, sin que la acción de control supere los 10,5 v, asegurándonos así una respuesta rápida, pero con control, es decir sin saturar y con el mínimo número de sobre oscilaciones en el momento en el que se alcanza la consiga de velocidad.

La función de transferencia conjunta Planta + Regulador queda:

$$Gr(s) * Gp(s) = Kp\frac{s + \frac{Ki}{Kp}}{s} * \frac{\frac{Ke}{t}}{s + \frac{1}{t}} = Kp\frac{s + \frac{1}{t}}{s} * \frac{\frac{Ke}{t}}{s + \frac{1}{t}} = \frac{Kp * Ke}{s * t}$$

Para obtener la respuesta que se comenta a continuación en las siguientes gráficas:

Kp, se ajustó a 7.65 y **Ki**, en 40.

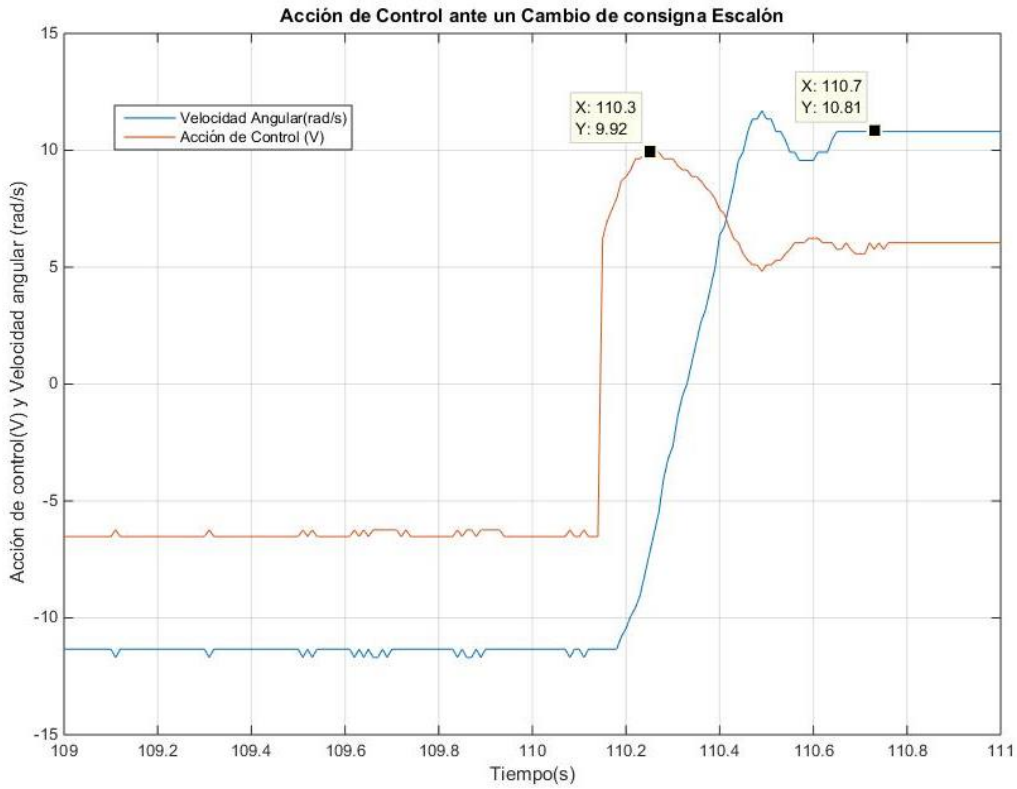


Figura 4.3.2_1: Ajuste de la acción de Control

4.3.3.- **Corrección de la Zona Muerta**

4.4.- Diseño del Chasis en NX 10 de Siemens

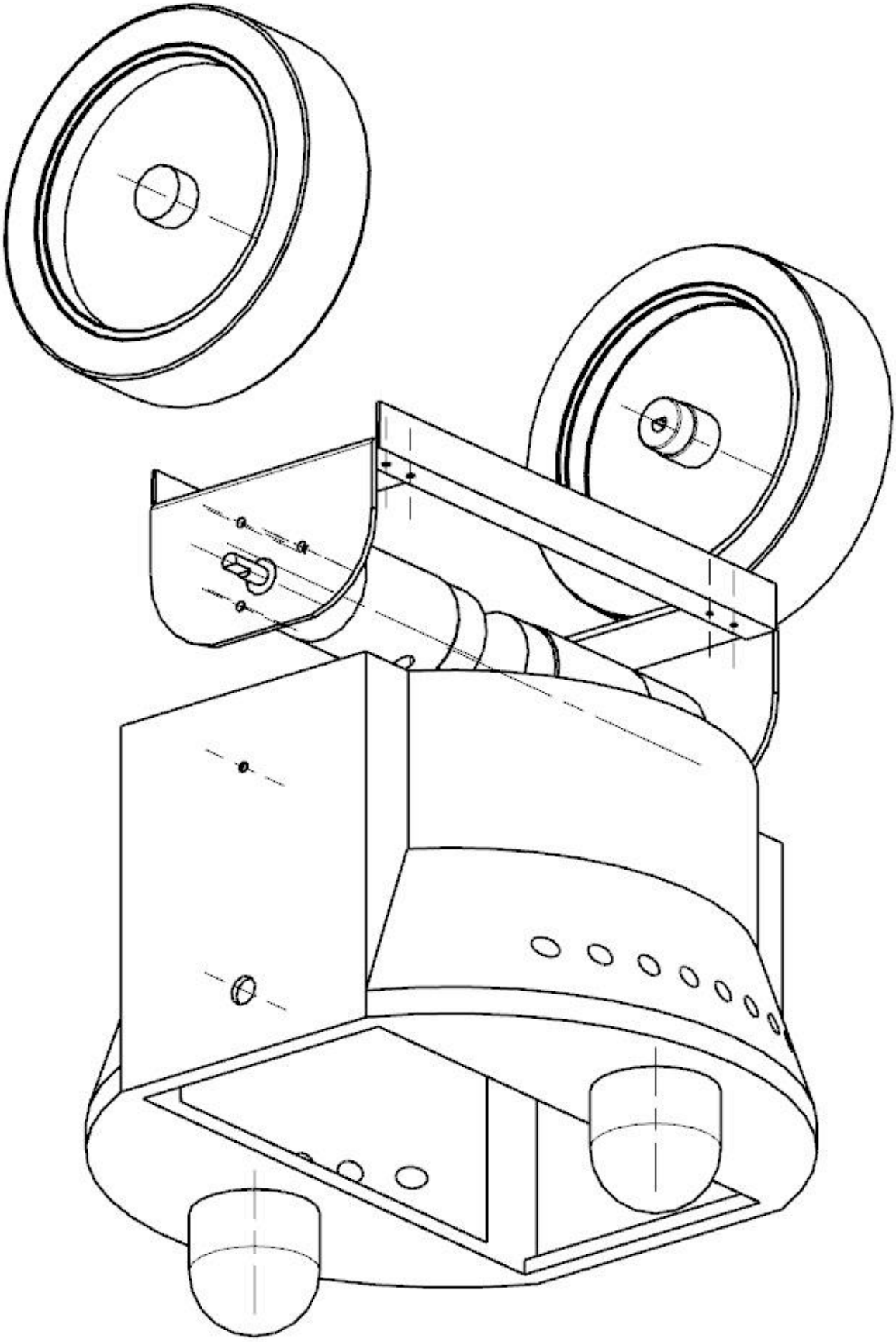


Figura 4.4_1: Despiece

4.5.- Flujogramas de Funcionamiento Software

El paso previo a la implementación de cada uno de todos los programas que se han implementado en este proyecto, ha sido la elaboración de un diagrama de flujo o flujograma, que representa los distintos pasos, operaciones, algoritmos y decisiones lógicas que conforman el programa. Posteriormente se ha implementado el código, y se ha probado.

Al probar el código la primera vez, lo más normal es que no funcione todo lo bien que esperamos, entonces analizando el comportamiento del programa volvemos a flujograma encontramos cual es el problema y una vez localizado el problema, a continuación, se modifica el código y se vuelve a probar el funcionamiento del mismo. Este proceso se ha ejecutado de forma reiterada (depurar código), hasta encontrar la mejor solución.

Cuando se abordan problemas complejos, el flujograma, así como los diagramas en forma de grafcet, son una herramienta imprescindible para conseguir el éxito en el problema, además de que ayuda a asentar en la memoria del programador todos los pasos que se han realizado, teniendo un conocimiento profundo del software desarrollado.

Por otra parte, en el trabajo en grupo multidisciplinar permite la transmisión rápida de todos los pasos que conforman el código, pudiendo resolver o modificar el comportamiento del programa por cualquier ingeniero que aporte una idea nueva, sin tener que ser un entendido o conocedor profundo del lenguaje de programación utilizado.

Otra gran ventaja es que cuando se retoma un proyecto después de un tiempo, se acelera la puesta a punto en el problema gracias a los flujogramas.

Mirar en apartado Anexos **Anexo 3: Diagramas de flujo**, para representar programas.

4.5.1.- **Algoritmo-Persecución + Bucles Control Velocidad (Local).**

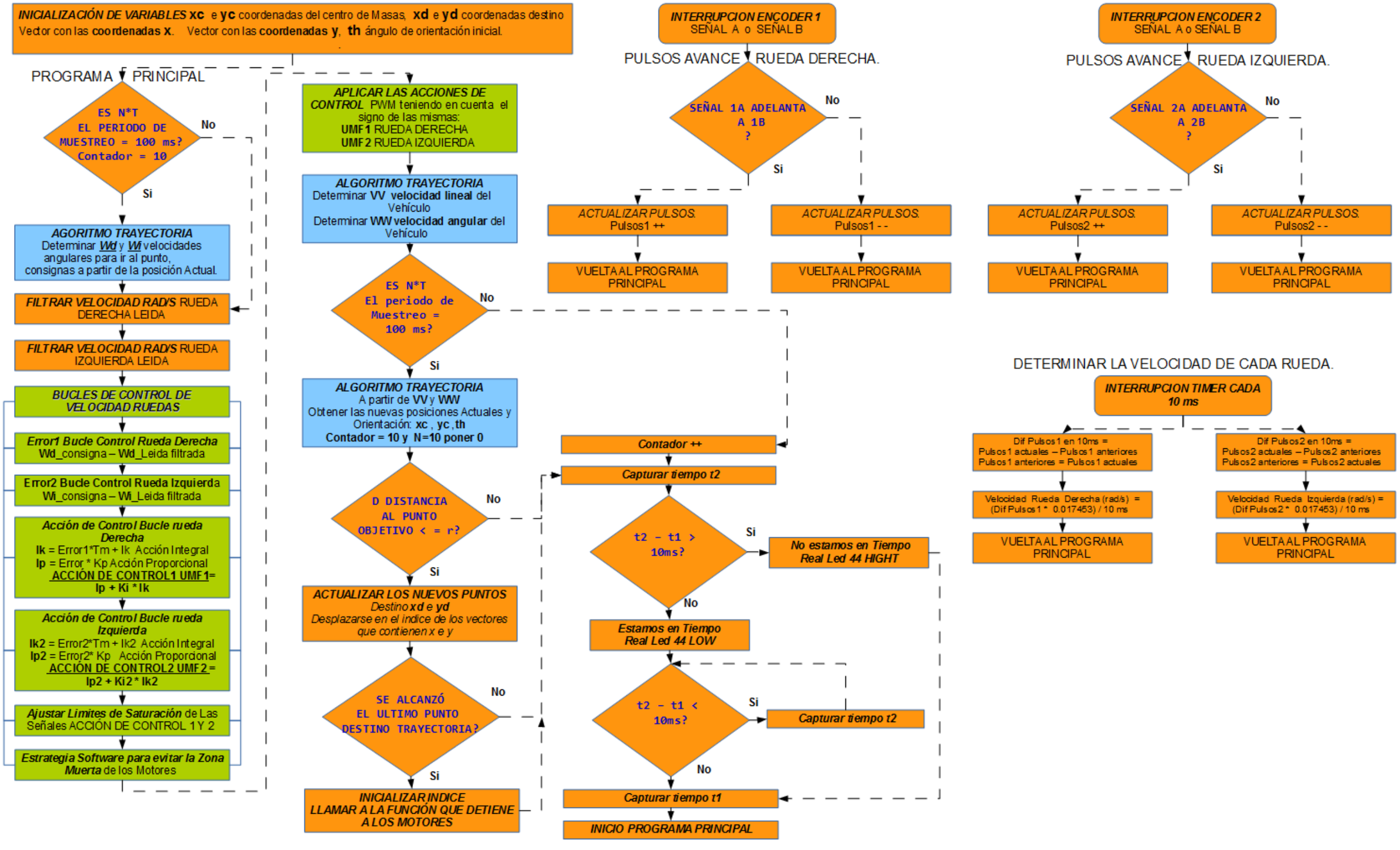


Figura 4.5.1_1: Modo Funcionamiento Local Bucles de Control + Algoritmo seguimiento de Trayectoria

Significado de las abreviaturas o letras del flujograma anterior son:

xc *e* ***yc*** *son las coordenada x e y del centro de masas actuales de cada periodo de muestreo.*

T *es periodo de muestreo base de 10 ms.*

N*T *=100ms es 10 veces el periodo de muestreo base.*

Wd *y* ***Wi*** *son las velocidades angulares(rad/s) consigna determinadas por el algoritmo de seguimiento de trayectoria.*

Wd_Leida *y* ***Wi Leida*** *son las velocidades determinadas por los encoders y el sistema de interrupciones.*

Wd_Leida filtrada *y* ***Wi_Leida filtrada*** *son las velocidades anteriores filtradas por el filtro de segundo orden Butterworth.*

Ik *es la acción integral en el bucle de control de velocidad 1.*

Ik2 *es la acción integral en el bucle de control de velocidad 2.*

Ip *acción de control proporcional en el bucle de control de velocidad 1.*

Ip2 *acción de control proporcional del bucle de control de velocidad 2.*

Kp *contante proporcional tanto para el bucle de control 1 como el 2.*

Ki *contante integral tanto para el bucle de control 1 como el 2.*

UMF1 *Acción de control total Ip + Ik del bucle de control de velocidad 1*

UMF2 *Acción de control total I2p + Ik2 del bucle de control de velocidad 2*

Dif Pulsos*1 pulsos transcurridos en un periodo de muestreo en el bucle de control 1*

Dif Pulsos*2 pulsos transcurridos en un periodo de muestreo en el bucle de control 2*

r *es el radio de la rueda.*

Vector de coordenadas x.

Vector de coordenadas y.

El flujograma de la Figura 4.5.1_1, representa el funcionamiento del robot en modo local, que consta de los bucles de control de velocidad para cada una de las ruedas y el algoritmo de seguimiento de trayectoria, sin ningún tipo de comunicación con el exterior, es decir tiene una trayectoria predefinida. El robot empieza en la posición inicial y va persiguiendo el siguiente punto contenido en los vectores que contienen los puntos de la trayectoria, que son el vector de coordenadas x y el vector de coordenadas y, hasta que se aproxima a dicho punto una distancia igual o menor al radio de las ruedas. Entonces este punto destino cambia al siguiente definido por los vectores de coordenadas y así sucesivamente hasta alcanzar el último punto de la trayectoria, momento en el cual se para el Robot.

Es importante observar que el subsistema que más rápido se ejecuta son los dos bucles de control de velocidad (en color verde), con un periodo T = 10 ms. El algoritmo de seguimiento de trayectoria (en azul), se ejecuta cada 10 veces este periodo de muestreo T=10 ms, es decir (N=10) *T = 100 ms, a excepción de la actualización de las velocidades (VV velocidad lineal de avance del vehículo y WW velocidad angular resultante del vehículo), que se calculan cada T. El resto de bloques en color naranja a excepción de las interrupciones, se ejecutan en cada vuelta de bucle, es decir en T = 10 ms.

Existe un hilo principal de programa y tres hilos múltiples de interrupción. El hilo principal de programa es el que contiene el algoritmo de seguimiento de trayectoria y los bucles de control de velocidad angular para cada una de las ruedas, mientras que los hilos de interrupción son:

- Interrupciones provocadas por las señales A y B del encoder 1.
- Interrupciones provocadas por las señales A y B del encoder 2.
- Interrupción del timer1 cada 10 ms.

El hilo principal se ejecuta de forma continua hasta que se produce una interrupción, se atiende a la interrupción realizando las tareas de esta interrupción y a continuación se vuelve al punto en el que se encontraba la ejecución del programa principal.

En realidad, para determinar la velocidad a partir de los pulsos acumulados en las variables Pulsos1 y Pulsos2 se necesitan 4 interrupciones por encoder para tener la máxima resolución posible. La salida de la señal 1 A del encoder 1 está conectada a la entrada digital pin 40 de Arduino y la señal 1 B del encoder 1 a la entrada digital pin 41, y lo mismo ocurre con el encoder 2, quedando la señal 2 A conectada a la entrada digital 42 y la señal 2 B conectada a la entrada digital 43.

Cada vez que la señal A o B de alguno de los encoders pasa de 0 a 1 lógico o de 1 a 0 lógico se dispara la interrupción, se analiza si la señal A adelanta a B, si es así, se incrementa el valor de la variable Pulsos, y en el caso contrario de que B adelante a A , entonces se decrementa esta variable.

Con lo cual conseguimos leer 4 veces más rápido el incremento de pulsos, que se traduce en una lectura de velocidad 4 veces más rápida que si lo hiciéramos contando el tiempo que trascurre entre pulsos o si simplemente analizáramos cada flanco de subida de una de las señales A o B.

La interrupción del Timer se produce cada 10 ms, en este momento se utiliza el valor de los pulsos contabilizados en las interrupciones anteriores, en la variable Pulsos actual para hacer la operación diferencia con los pulsos de la misma variable Pulsos (Pulsos anteriores) en la vuelta de bucle anterior, determinando así el número de pulsos que transcurren en 10 ms, a partir de aquí la determinación de la velocidad es inmediata como:

$$\boldsymbol{Velocidad}\left(\frac{\boldsymbol{rad}}{\boldsymbol{s}}\right) = \frac{\boldsymbol{Diferncia\ Pulsos} * \mathbf{0.017453}}{\mathbf{10}\ \boldsymbol{ms}}$$

En la expresión anterior, cada pulso equivale a un grado, que son 0.017453 rad.

Las velocidades obtenidas de ambos encoders, a través del sistema de interrupciones son filtradas para eliminar el ruido mecánico, mediante un filtro *Butterworth* de segundo orden calculado en Matlab.

La clave para que todo funcione bien, es que tanto el periodo de muestreo T = 10 ms, como N* T = 100 ms, y el de la interrupción del Timer de 10 ms, se respeten y sean lo más exactos posibles. Los 10 ms del Timer son asegurados por la interrupción, mientras que el periodo T se asegura con el bucle temporizado cuyas unidades de medida son en microsegundos. Por lo tanto, el error máximo será de un microsegundo en el caso de T= 10 ms y como mucho en el caso de N* T = 100 ms, será de 10 microsegundos. Pero lo más importante de todo, además de lo anterior, es que todas las operaciones del bucle de control se ejecuten en menos de 10 ms y las del bucle de control junto con las del algoritmo de seguimiento de trayectoria en 100 ms. Esto se ha comprobado mediante la puesta a nivel alto del

led conectado en la salida digital de Arduino en el pin 44, cuando no se cumple el periodo T, y por lo tanto, no estamos en tiempo real, mientras que, si da tiempo a realizar todas las operaciones, la salida digital 44, permanece a nivel lógico bajo.

Por último, la estrategia software para evitar la zona muerta consiste en analizar en cada vuelta de bucle del programa principal el valor de las acciones de control, si estas acciones en valor absoluto están dentro de la zona identificada como zona muerta, lo que hacemos es convertir la acción de control en justo el valor determinado en los apartados Zona Muerta 4.2.1 y 4.2.5 que es de aproximadamente 50 escalones del PWM, consiguiendo así sacar al motor de la zona muerte, zona en la que las ruedas no se mueven. En el caso de que las acciones de control estén muy cerca de cero, se entiende que la consigna de velocidad deseada es la de cero, es decir estar parado y por lo tanto convertimos las acciones de control en cero, y en el caso de que las acciones de control estén por encima de la zona muerta, entonces no alteramos la acción de control, dejando que dichas acciones de control sean las que tengan que ser, determinadas por los bucles de control de velocidad.

En el Anexo 4 Programación, apartado 4.1 Funcionamiento Local, se puede ver el código C elaborado.

4.5.2.- **Algoritmo-Persecución + Bucles Control Velocidad (Local) + Servidor.**

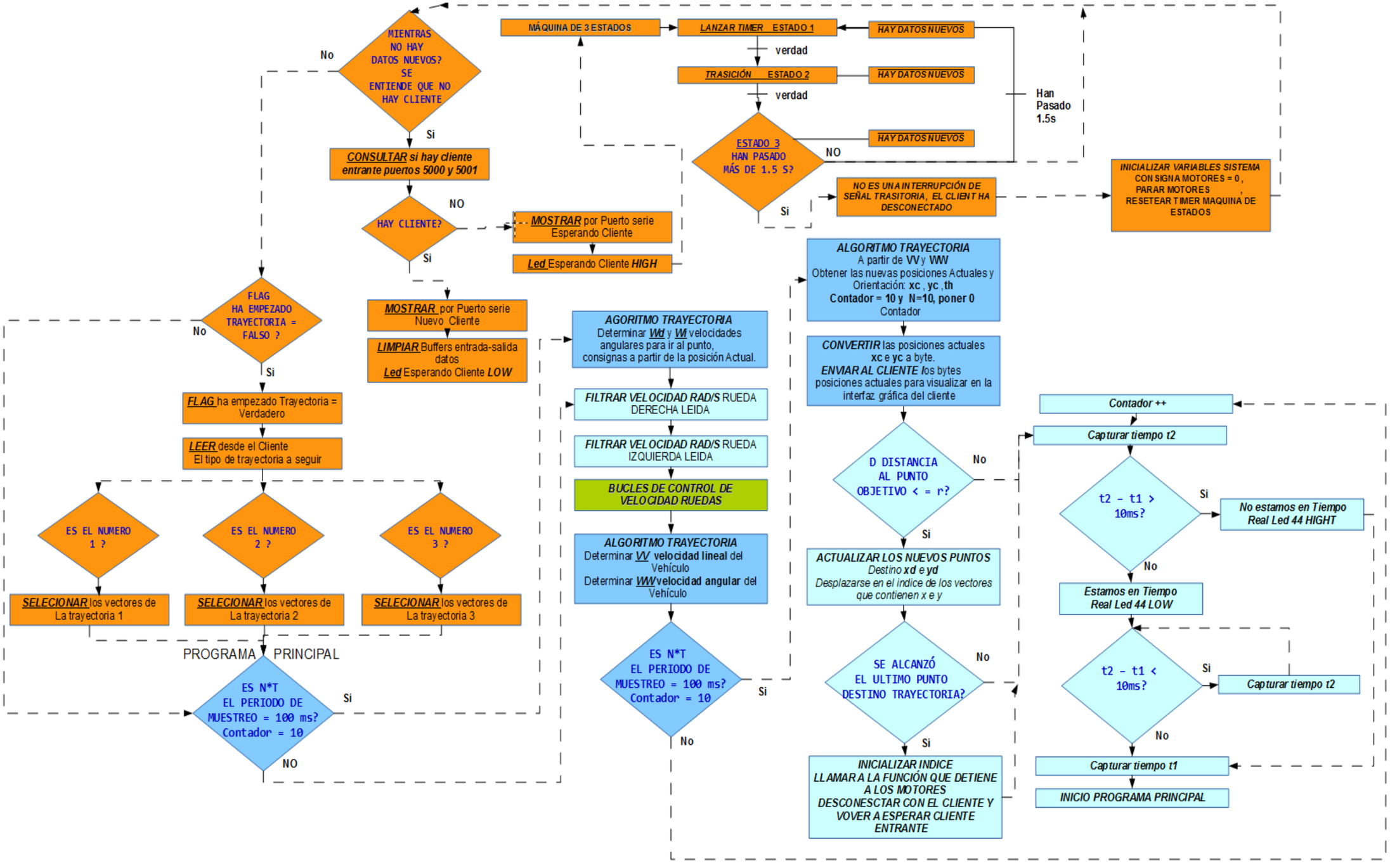


Figura 4.5.2_1: Modo de funcionamiento Local + Servidor Robot

El significado de las abreviaturas o letras utilizados en el flujograma de la Figura 4.5.2_1 son exactamente iguales que los del apartado anterior.

El flujograma anterior pertenece al modo de funcionamiento local con la implementación del servidor que permite establecer la conexión, cuando hay un cliente entrante.

Como se comentó en el apartado 1.3 Problema planteado, el cliente entrante puede ser tanto Matlab / Simulink como el dispositivo Android, el objetivo de esta modalidad de funcionamiento se basa en el planteamiento o enfoque industrial, en el que se le indica al Robot el tipo de trayectoria que se desea que siga y éste la ejecuta. Existen tres trayectorias prefijadas en la memoria del Robot. El cliente solicita una de estas tres trayectorias y una vez el Robot lo sabe empieza la ejecución del programa principal. Como ya se sabe, en el modo de funcionamiento local, además de los dos bucles de control de velocidad que se ejecutan con un periodo de muestreo T = 10 ms, también está implementado el algoritmo de seguimiento de trayectoria que se ejecuta cada N*T = 100ms, en el Robot.

El programa empieza con una pregunta ¿Hay datos entrantes nuevos en los buffers de entrada?, si la respuesta es negativa, cosa que ocurre siempre que no hay clientes entrantes o en el momento inicial de arrancar la aplicación, se pasa a escuchar la posible entrada de clientes a través de los puertos 5000 y 5001. Este proceso de escucha es gestionado a bajo nivel por la librería Wifi.h, y mientras no se detecta ninguna entrada de cliente en el puerto 5000 o en el 5001, se entra en una subrutina de forma permanente hasta evidentemente detectar la entrada de cliente en alguno de estos dos puertos.

La subrutina muestra por el puerto serie (Esperando Cliente), enciende un Led que indica esta espera, y posteriormente se pone en funcionamiento una pequeña máquina de tres estados que se ha planteado e implementado siguiendo las reglas del grafcet. Los tres estados que se repiten funcionan solo si no hay entrada de datos o no hay cliente pidiendo datos. En el momento en que entra un cliente la máquina de tres estados, independientemente del estado actual, se reinicializa y deja de ejecutarse. El objetivo de esta máquina de estados es lanzar un *timer* (temporizador) cuando se pierde la comunicación, que actúa como un temporizador a la desconexión, de tal forma que si transcurren más de 1.5 segundos desde la interrupción de la comunicación se entiende que no hay cliente escuchando y se cierra la comunicación parando al Robot y reiniciado todas las variables de programa. La utilización de la máquina se ha realizado porque, para poder ver la trayectoria que el robot va trazando en la interfaz de usuario en el

cliente, los datos xc, yc y th se van enviando al cliente cada 100 ms, y para poder diferenciar entre una interrupción de la señal esporádica con la interrupción definitiva por parte del cliente, ha sido necesario analizar que las interrupciones por perdida de señal en la wifi suelen esta entre 0,5 segundos a 1 segundo, y por lo tanto una interrupción de señal superior a 1.5 segundos, queda claro que es debida a una interrupción de la comunicación de forma voluntaria por parte del cliente.

Una vez se detecta la entrada del cliente se abandona esta subrutina y se pasa a la pregunta de si había con anterioridad una trayectoria en ejecución, a la cual se responde con el *flag* (HA EMPEZADO LA TRAYECTORIA), si esta bandera indica falso entonces se lee el número que indica una de las tres trayectorias prefijadas que el Robot debe ejecutar, seleccionando así los vectores correspondientes a la trayectoria seleccionada. En el caso de que la interrupción que se había producido con anterioridad hubiera sido por pérdida de señal y no por el cese voluntario del cliente, se retoma la ejecución del programa principal en el punto en el que estaba.

Sea un caso o el otro, aquí empieza el programa principal que se encarga de ejecutar los bucles de control de velocidad y el algoritmo de seguimiento de trayectoria que ya conocemos.

A partir de este momento, la ejecución del programa es exactamente la misma que la explicada en el apartado anterior 4.5.1. La diferencia estriba en que, cada vez que se ejecuta la parte del algoritmo de seguimiento de trayectoria en la que se actualizan las posiciones actuales xc, yc y th, se le comunican al cliente para que éste pueda representar en la interfaz de usuario, la trayectoria que el Robot va ejecutando.

Al igual que en el caso anterior, para ir actualizando los nuevos puntos destino contenidos en los vectores de coordenas x e y, se compara en cada momento si la distancia D, entre el centro de masas y el punto destino es menor o igual que r, que es el radio de la rueda de 51 mm.

Es interesante explicar porque se han utilizado dos puertos para establecer la comunicación (5000 y 5001), como si de dos programas distintos se tratara. El objetivo fundamentalmente ha venido impuesto por los experimentos realizados en el control basado en red. En dichos experimentos, cuando se mandaba consigna desde el algoritmo de seguimiento de trayectoria, que como se ha explicado antes en este tipo de modalidad, dicho algoritmo está en el dispositivo cliente, a veces se producía el fenómeno de cruce de los bytes que iban dirigidos a un bucle de control, hacia el otro bucle de control, con el consiguiente desastre que esto

supone en el seguimiento de la trayectoria por parte del Robot, puesto que las consigas de la rueda derecha se aplicaban a la rueda izquierda y viceversa. La solución de seguridad aplicada, ha sido la de enviar consignas y leer tanto posiciones actuales, en el caso de la modalidad de funcionamiento local, como leer las velocidades de los encoders desde el servidor y enviadas al cliente en el caso de control remoto, utilizando dos puertos distintos, tal y como se ha indicado anteriormente.

En el apartado Anexos, Anexo 4 Programación se puede ver el código C elaborado, 4.2 Funcionamiento Local + Servidor.

4.5.3.- **Cliente desde Matlab/Simulink (Local Monitorización).**

Como hemos visto en el flujograma de la Figura 4.5.2_1, el algoritmo de seguimiento de trayectoria, está en modo local en el Robot. En este modo de funcionamiento lo que se persigue desde el punto de vista del cliente, visto desde Matlab / Simulink es por una parte establecer la conexión para indicar el tipo de trayectoria que se desea que ejecute el Robot, y por otra parte ir leyendo desde el cliente las posiciones actuales, que el algoritmo de seguimiento de trayectoria va generando en el servidor que se encuentra en el Robot. Para este propósito es necesario entender cómo se plantea la programación desde Matlab Simulink.

Para ello hay que diferenciar la programación para crear el cliente, es decir cómo crear el objeto socket que establecerá la comunicación entre las dos máquinas, el cliente (el PC con Matlab / Simulink) y el servidor en el Robot para poder intercambiar el flujo de datos. Dicho socket se implementa programando en Matlab con una S-FUNCTION, mientras que con la programación gráfica o modular basada en Simulink, donde de forma periódica se llama a la S-FUNCTION para por un lado no solo establecer la conexión, sino que también, pasar el byte o información que se le quiere transmitir al servidor, y por otro lado, leer desde la S-FUNCTION el byte recibido y poderlo visualizar en forma gráfica, que en este caso es la trayectoria que va realizando el Robot.

Vamos a empezar por la creación del Socket conexión TCP/IP, programando la S-FUNCTION encargada de crear el Socket para establecer la comunicación y el intercambio de datos.

El código se implementa en un *Script* que tiene una extensión .m con un esqueleto similar al siguiente:

```
                                                    Nombre de la función
Función   [sys,x0,str,ts]              =        sfun_wifi  (t ,x ,u ,flag )

        Valores que devuelve                    Parámetros que recibe

    switch flag,

    case 0,          // Se inicializa la función establecer Socket

    case 2,          // Se escribe al buffer de Salida de datos Transmisión

    case 3,         // Se lee del buffer de entrada de datos Recepción

    case 9 ,          // Se cierran las conexiones

end
```

Este esqueleto consta del nombre de la S-FUNCTION que es una función compleja que requiere de unos parámetros de entrada cuando la llamamos desde Simulink, y devuelve otros parámetros de salida a Simulink tras cada consulta.

La S-FUNCTION es similar a una estructura compleja constituida por métodos, que son invocados en función de un *flag* de entrada, que son los casos del *switch* que se puede ver en el anterior esqueleto.

Los parámetros de entrada desde Simulink a la S-FUNCTION ***sfun_wifi (t, x, u, flag)*** *son*:

t *Tiempo* trascurrido de programa.

x Estado del vector

u Valores que pasamos para enviar al servidor, como por ejemplo tipo de trayectoria, consignas de velocidad angular (bytes), al buffer de salida hacia el servidor.

flag Es el caso al que van dirigidos los parámetros de entrada, que contiene los métodos que queremos que se ejecuten.

Los parámetros de **salida desde la S-FUNCTION** Función [sys,x0,str,ts] hacia Simulink son:

sys Contiene los bytes que se leen en el buffer de entrada en el flag 3. Estos bytes son los que ha enviado el Servidor, en este caso las posiciones que se van actualizando en el algoritmo de seguimiento de trayectoria.

ts Contiene una matriz de dos columnas que contienen los tiempos de muestreo. Para que nuestra S-FUNCTION se ejecute en cada periodo de muestreo de Simulink, este valor es fijado en el flag 0 en ts [0 0].

x0 Es el valor inicial de estados, que en nuestro caso no es necesario y por lo tanto se fija con un vector vacío. x0 = [ ].

str Es un valor que se devuelve para un uso futuro que en nuestro caso no sirve de nada, y queda inicializado como el parámetro anterior str = [ ].

El contenido y los pasos para implementar esta S_FUNCTION pasan a describirse en los siguientes puntos:

1. Crear el objeto que establece la conexión y permite el intercambio (flujo) de datos, siendo Matlab / Simulink el cliente del tipo TCP/IP.

```
ttd = tcpip('192.168.1.3', 5000, 'NetworkRole', 'client');
tti = tcpip('192.168.1.3', 5001, 'NetworkRole', 'client');
```

Al crear el objeto los dos últimos parámetros indica el rol que adoptará nuestra conexión, que en nuestro caso es el de cliente, el primer parámetro es la IP del servidor al que nos vamos a conectar y el segundo el puerto.

Como se indicó en los apartados anteriores, para evitar el cruce de bytes tanto en la recepción como en la transmisión se han creado dos objetos socket TCP/IP con la misma dirección IP, pero con dos puertos distintos, el 5000 para transmisión de los bytes dirigidos a las consignas de velocidades angulares de la rueda derecha, y el 5001 para transmisión de los bytes dirigidos a las consignas de velocidades angulares de la rueda izquierda, cuando se trabaja en modo remoto basado en control en Red. Y para este caso, en el que solo nos interesa la visualización de la trayectoria que va realizando el Robot, que vamos viendo en la interfaz gráfica del cliente, se utiliza el puerto 5000 para enviar las

coordenadas xc (x actual), y el 5001 para enviar las coordenadas yc (y actuales) desde el servidor Robot, que son recibidas por el cliente Matlab / Simulink, para poderlas graficar.

2. Establecer el tamaño en bytes del buffer de entrada y el tamaño en bytes del buffer de salida y abrir la conexión.

```
set(ttd,'OutputBufferSize',1); set(ttd,'InputBufferSize',1);
set(tti,'OutputBufferSize',1); set(tti,'InputBufferSize',1);
fopen (ttd); fopen(tti);
```

La primera línea ttd (puerto 5000), establece que tanto el buffer de salida de datos como el de entrada tienen un tamaño de un byte. La segunda line es lo mismo, pero para el puerto 5001 (tti). A continuación, se abren (establecen) las dos conexiones cliente para comprobar que existe un servidor escuchando y empezar la comunicación.

El hecho de utilizar un byte tanto en la salida en cada transmisión como en la recepción, un byte en cada recepción, y que el tamaño de los buffers de entrada y salida también sean de un byte, se debe a que en control basado en red, y para la visualización en la interfaz gráfica, es muy importante que el retraso, en tiempo, de la señal desde que se envía desde el cliente, llega al servidor y vuelve al cliente debe ser constante o lo más constante posible. Si enviamos solo un byte y recibimos un byte por periodo de muestreo y el buffer tanto de salida, como el de entrada tienen un tamaño de un byte, lo único que puede producirse es la perdida de bytes, pero nuca la acumulación de los mismos que pueden producir un desfase temporal en el retraso de la comunicación variable.

Así pues, aseguramos una comunicación con retrasos lo más constante posible, que hace que el sistema sea previsible, asegurando posteriormente un buen control basado en red, y una representación gráfica constante y con retraso fijo.

El propósito ha sido el de crear un socket del tipo TCP/IP con las ventajas que esto presenta, asegurando en lo posible la llegada a su destino de los bytes enviados y con el orden adecuado.

La desventaja que presenta este tipo de conexión es el retraso que se puede producir con los bytes enviados, que queda solucionado con el envío byte a byte y con el tamaño de los buffers de un byte. De esta forma el protocolo se encarga de asegurar en lo posible la no perdida de bytes y nosotros mantenemos constante el

retraso de la red, aunque esto pueda ocasionar la perdida de algún byte, que siempre es mejor que la posibilitad de ir acumulados bytes en el buffer que posteriormente se traduce en un desfase del tiempo de propagación de los bytes variable.

Ojo, el desfase de la red es constante para nuestro caso, puesto que el ancho de banda no se ve afectado durante las pruebas por la entrada al medio compartido Red, por parte de otras máquinas (PC). Lo que aseguramos es que para un ancho de banda constante nuestros desfases temporales en las señales de control son constantes.

En el anexo 4 Programación, apartado 4.3 Funcionamiento Cliente Matlab / Simulink, se puede ver el código de la S-FUNTION elaborado.

Loas puntos 1 y 2 anteriores, pertenecen al caso cuando el *flag* case vale 0 del esqueleto de la S-FUNCTION introducido al principio, que es cuando arranca la aplicación en Simulink. Simulink llama a la S-FUNCTION y le indica con el valor 0 de este *flag*, que quiere establecer la comunicación y configurar la misma.

3. Ahora una vez inicializado y establecida la conexión podemos escribir o leer en los puertos correspondientes. Para escribir Simulink lo indica a la S-FUNCION poniendo el valor 2 en el *flag* case, pasándole los bytes que se quieren transmitir como se muestra a continuación:

```
fwrite(ttd,u(1),'uint8') ;
fwrite(tti,u(2),'uint8') ;
```

Lo que hacemos en la primera línea de código es indicar a la S-FUNCION que queremos enviar al puerto 5000 (ttd) el byte que hay o se escribe en la entrada del diagrama de bloques de Simulink u(1). Y en la segunda línea de código lo mismo, se indica a la S-FUNCTION que queremos enviar por el puerto 5001 el valor en forma de byte contenido en la entrada u(2) del diagrama de bloques.

Para poder leer los datos enviados desde el Servidor ahora Simulink se lo indica a la S-FUNTION poniendo el valor 3 en el *flag* case y el código para leer es el siguiente:

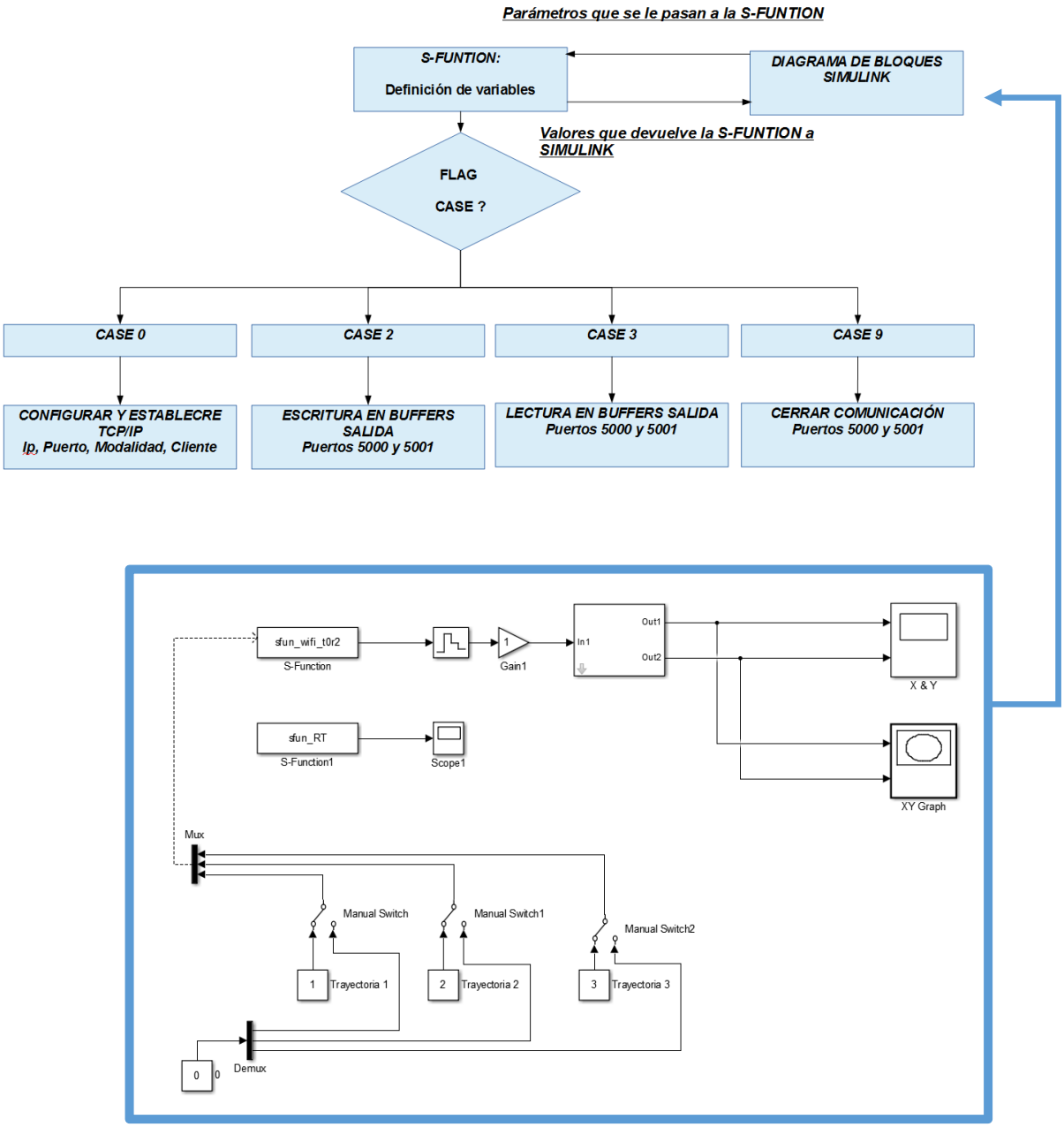
```
sys=[fread(ttd,1,'uint8') fread(tti,1,'uint8')];
```

Lo que hacemos es guardar en el vector sys, los dos datos que se leen desde el buffer de entrada, enviados por el Servidor.

Finalmente, cuando se quiere finalizar la comunicación Simulink establece el *flag* case**,** al valor 9, y el código para este caso es:

```
fclose(ttd) ; fclose(tti) ;
```

A continuación, se muestra un flujograma que indica el funcionamiento de este sistema donde interaccionan la S-FUNTION y el diagrama de bloques de Simulink.



4.5.4.- **Cliente desde Dispositivo Android (Local Monitorización).**

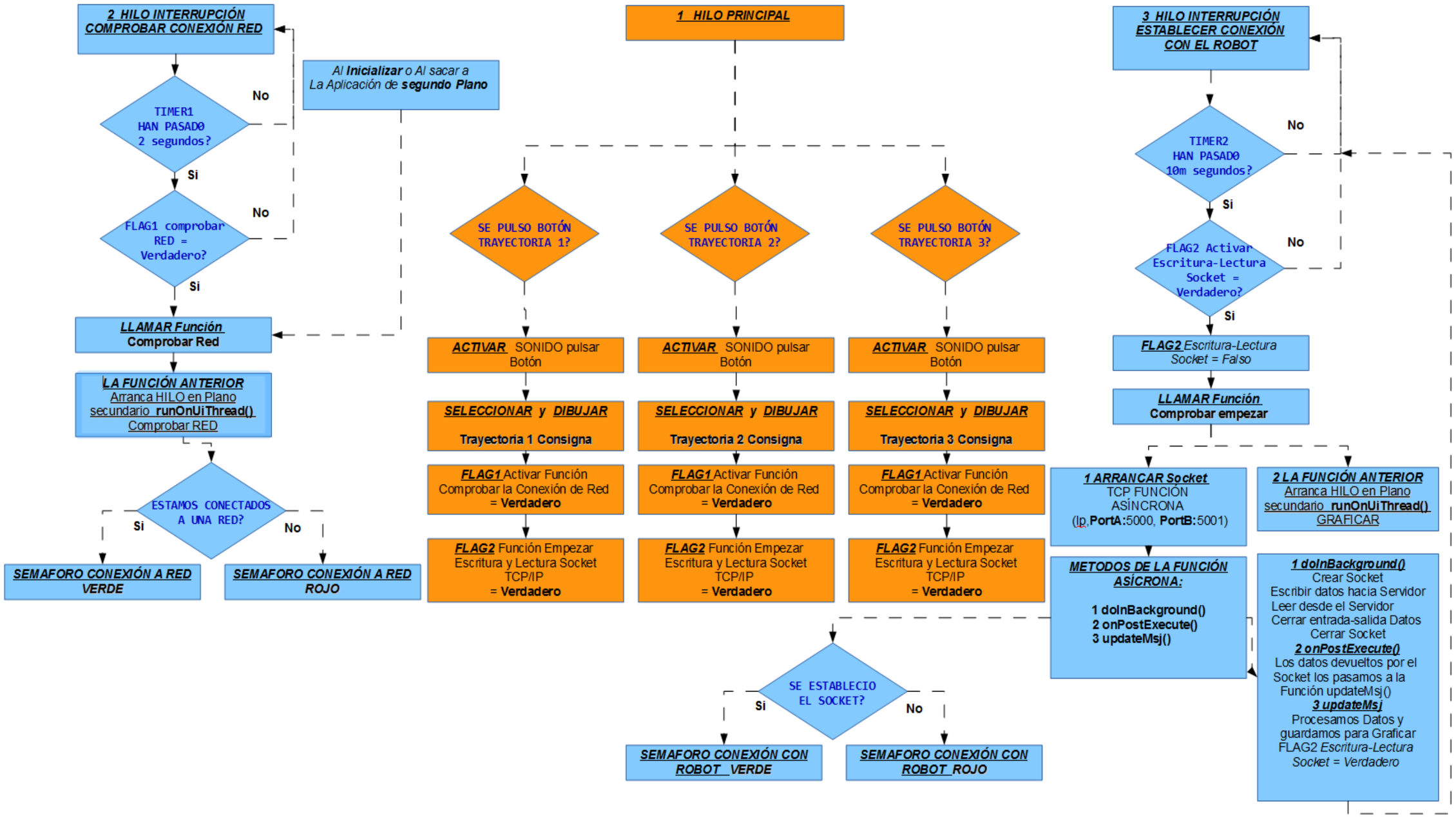


Figura 4.5.4_1: Modo de funcionamiento Local Cliente Android

La programación en Android presenta una serie de peculiaridades y algunas de estas pasaremos a comentarlas, a partir de la explicación del flujograma de la Figura 4.5.4_1.

La programación está fuertemente ligada a lo que se conoce como multi-hilo, cuyo objetivo es el de procesar por el sistema operativo varias aplicaciones a la vez, quedando unas en primer plano y otras en segundo plano. Esto significa que una aplicación tiene que correr sin detener al resto de aplicaciones que el sistema operativo Android está ejecutando, y obliga a que cualquier aplicación tenga un ciclo de vida (Ver anexo 05 Programación en Android), y que las distintas funciones que forman la actividad se ejecute en hilos diferentes.

Dichos hilos son priorizados en función de la forma en que se han implementado, y de esto se encarga el sistema operativo Android. Por eso dentro de una aplicación no podemos implementar todo el código en el hilo principal del programa.

Podemos observar del flujograma anterior que el hilo principal de la aplicación es el encargado de estar escuchando si algún botón de los tres (Trayectoria 1, Trayectoria 2 o Trayectoria 3), ha sido pulsado. Al pulsar cualquiera de estos botones lo que ocurre es, por una parte, se emite el sonido de botón presiona, se dibuja la trayectoria consigna que queremos que el Robot siga, y por otro lado hay dos banderas (FLAG 1 Y FLAG 2), que quedan activadas a nivel lógico verdadero.

El FLAG1, al quedar con valor verdadero permite a la interrupción del Timer1, que se produce cada dos segundos, llamar a la función comprobar red. Esta función red, es la que crea un hilo (*Thread*), tipo ***runOnUiThread*** (ver anexo 5)**,** que finalmente permite comprobar si hay conexión de Red y actuar sobre la interfaz gráfica del usuario, poniendo el semáforo Red en verde si tenemos conexión, o en rojo en el caso de no tener conexión.

El FLAG2, que como se ha comentado, también queda con valor verdadero cuando cualquiera de los tres botones es presionado, permite arrancar un proceso mediante un mecanismo similar al descrito anteriormente, pero un poco más complicado, que será el encargado de establecer la comunicación con el Robot. Estando esta bandera en verdadero, permite al Timer2 que se ejecuta cada 10 mili segundos, llamar a la función empezar.

Es la función empezar la que da la señal de ejecutar la función asíncrona, encargada de llamar a la función Socket para establecer la comunicación TCP/IP con el Robot (método **doInBackground**), recoger los datos (método

onPostExecute), en este caso de las posiciones actuales desde el Socket, y procesar y actualizar los datos recogidos (método **updateMsj**) desde el Socket para guardar en las variables correspondientes, además de volver a poner el FLAG2 en verdadero para que el timer2 pueda volver a llamar a la función empezar.

Finalmente, la función empezar, después de terminar con el Socket, arranca el hilo graficar (tipo **runOnUiThread**), que permite actualizar las posiciones actuales del Robot en la gráfica X e Y, que nos permite comparar la trayectoria que el Robot va trazando, con la trayectoria consigna dibujada anteriormente, en función del botón pulsado.

A continuación, Figura 4.5.4_12 se muestra la interfaz gráfica del usuario para monitorizar el proceso, programada en Android (XML).

En el capítulo siguiente RESULTADOS, se muestran resultados para poder ver la relación entra la trayectoria consigna y la que va trazando el Robot.

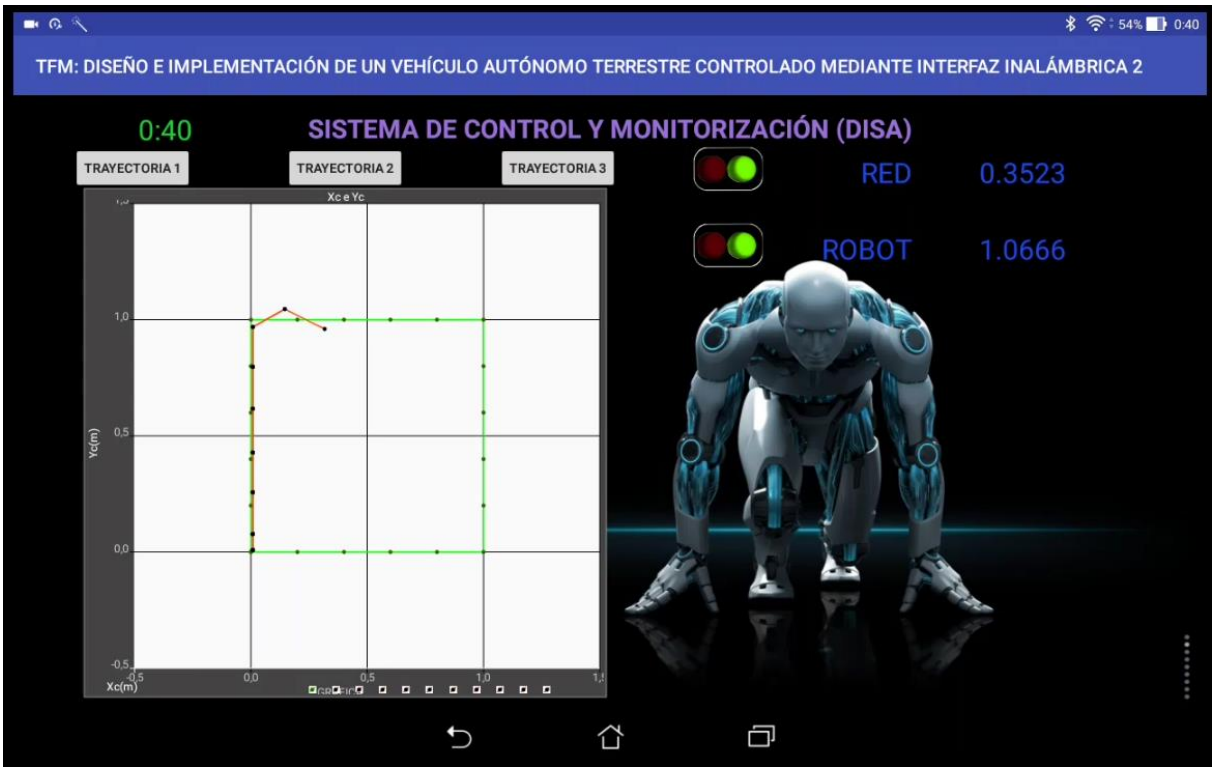


Figura 4.5.4_2: Interfaz Gráfica Android

4.5.5.- **Cliente desde Matlab / Simulink + Algoritmo Seguimiento en modo remoto.**

Es la modalidad de funcionamiento más complicada que se ha implementado, ha permitido la experimentación del control basado en Red, y en esta modalidad el cliente Matlab / Simulink es el que contiene el algoritmo de seguimiento de trayectoria.

En este caso la S-FUNCTION ha sido programada para que los dos bytes de velocidad angular (wd y wi) que elabora el algoritmo de seguimiento de trayectoria sean**,** wd el byte que se envía por el puerto 5000 y wi el byte que se envía por el puerto 5001, como se comentó anteriormente para evitar el cruce de bytes, que van dirigidos como consigna de velocidad a cada uno de los bucles de control de velocidad. Como sabemos están implementados en forma local en el Robot y la S-FUNCTION está programada para que, por los mismos puertos citados anteriormente 5000 y 5001, respectivamente recibir las velocidades angulares (rad/s) reales, wd _leida y wi _leida, determinadas por los encoders y el software de los bucles de control de velocidad.

Recordemos que esta programación consta de dos partes claramente diferenciadas, por una parte, tenemos la S-FUNCTION, que es la encargada de establecer la comunicación con el servidor (configura y crear el Socket) y una vez configurada la comunicación, abrir el flujo bidireccional de bytes intercambiados entre cliente y servidor.

Y por otra parte, tenemos los bloques de Simulink que se encargan de hacer las operaciones necesarias del algoritmo de seguimiento de trayectoria, para que en cada vuelta de bucle de red, en función de las posiciones actuales xc, yc y th, elaboradas por el propio algoritmo a partir de las velocidades ωd _leida y ωi _leida, leídas por el cliente y enviadas por el servidor, por medio del socket establecido por la S-FUNTION, genere las dos consignas de velocidad angular citadas anteriormente (***ωd*** y ***ωi***), que son enviadas al servidor también mediante el socket.

Aquí en este momento, es cuando tenemos que definir e integrar la idea conjunta de los distintos subsistemas que conforman el funcionamiento software del robot. El control de la trayectoria se va a realizar, por una parte, con el control de velocidad ejercido por los bucles de control implementados en el Robot que tratan de asegurar la velocidad constante de cada una de las ruedas, y con el valor a consigna demandado en cada vuelta de bucle, y por otra parte el algoritmo de seguimiento, que es el que fija las consignas, que permiten llegar a cada uno de los puntos que conforma una trayectoria.

Ambos subsistemas se cierran a través del medio de la Red y con soporte inalámbrico cuya velocidad de comunicación es menor y con más perdida de información que con soporte físico cable.

Por lo tanto, es evidente que, desde que se obtienen las velocidades reales wd _leida y wi _leida, sus valores llegan desde el servidor al cliente, pasan por el algoritmo de seguimiento de trayectoria, se determina la posición actual del robot, a partir de esta posición actual (***xc, yc*** y ***th),*** se elaboran las consignas ***ωd*** y ***ωi***, y finalmente se envían al servidor desde el cliente para ser utilizadas por los bucles de control de velocidad. La ejecución de estos procesos implica que pasa un tiempo significativo.

El retardo de red comentado se conoce como (*round-trip time delay*) y es la suma del tiempo que el conjunto sensor-software tardan en darnos la medida de la velocidad actual + tiempo de enviar la lectura al cliente + tiempo en elaborar la acción de control + tiempo de mandar de vuelta al servidor o planta.

El hecho de tener en cuenta este tiempo o no, es determinante para el comportamiento del robot durante el seguimiento de la trayectoria, y posteriormente se explica cómo solucionar esto mediante un predictor de Smith.

El diagrama de bloques de la Figura 4.5.5_1 trata de mostrar y aclarar las ideas anteriores:

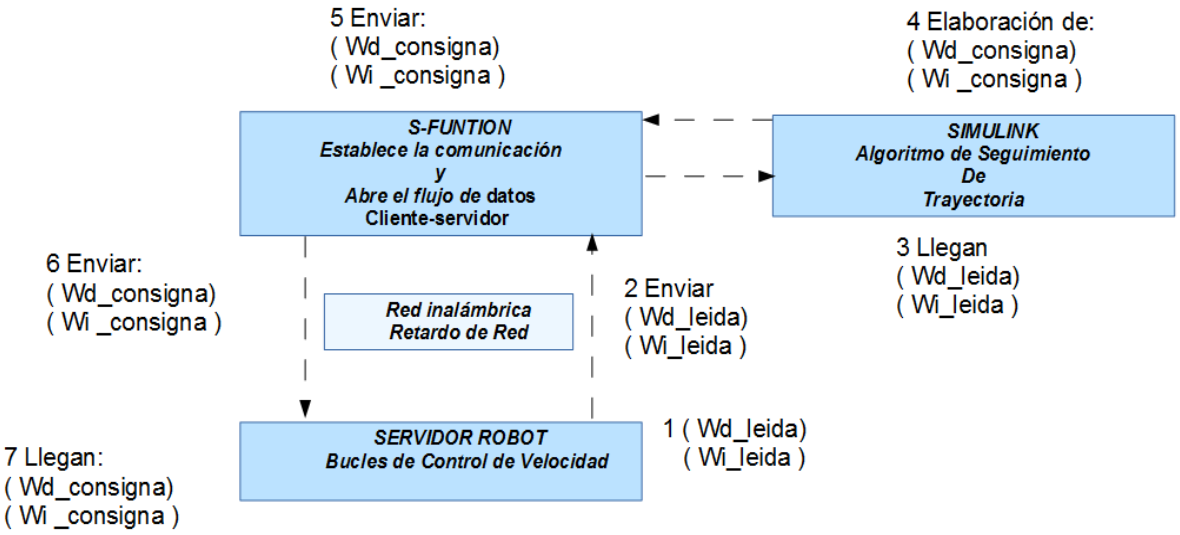


Figura 4.5.5_1: Control basado en Red diagrama funcional

Otra forma de verlo desde una perspectiva funcional-temporal y anidada de los distintos subsistemas que intervienen sería la mostrada en la Figura 4.5.5_2:

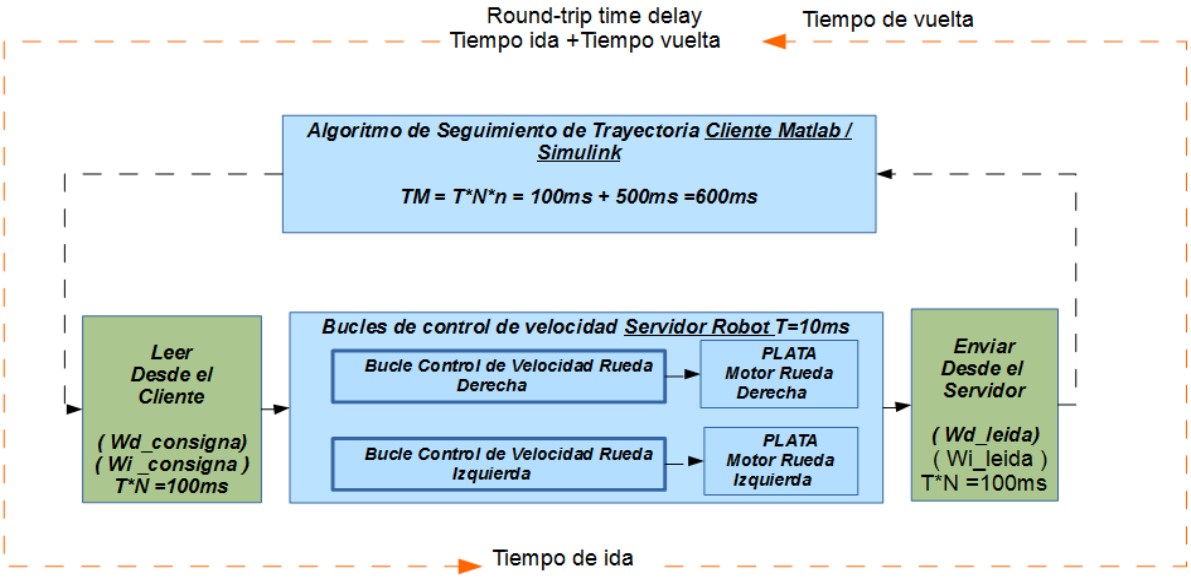


Figura 4.5.5_2: Control basado en Red diagrama temporal-funcional

Podemos ver en el flujograma anterior de la Figura 4.5.5_2 que al final hay un coste temporal entre la ida de las consignas de velocidad y la vuelta al cliente de las velocidades angulares leídas de 600ms que hemos llamado TM.

A continuación, en la Figura 4.5.5_3 se pasa a detallar visto desde Simulink los bloques del algoritmo de seguimiento de trayectoria nombrándolos como A, B, C y D, para identificarlos y poder explicar a continuación el subsistema que engloba cada uno de ellos.

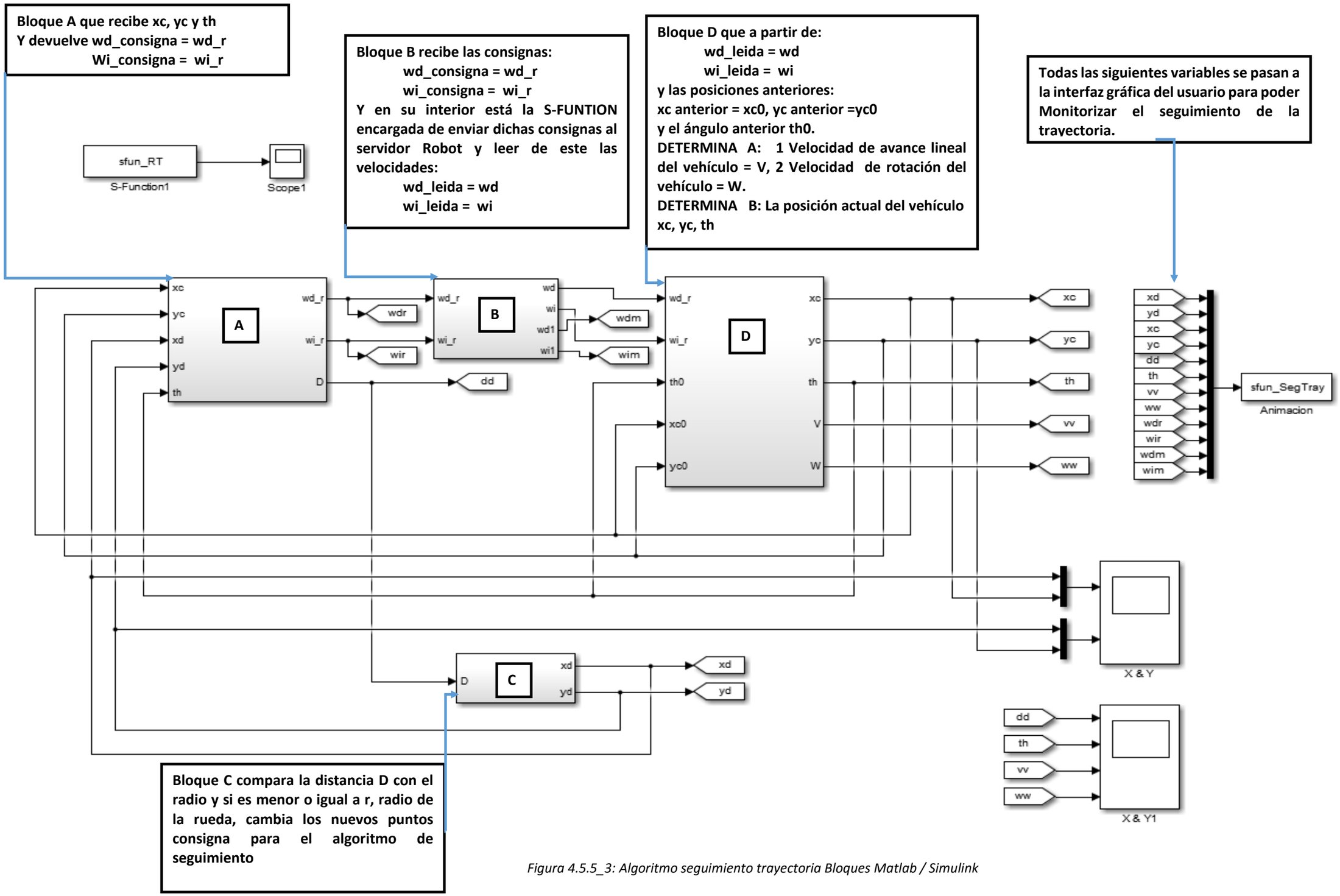


Figura 4.5.5_3: Algoritmo seguimiento trayectoria Bloques Matlab / Simulink

Subsistema interno del Bloque A:

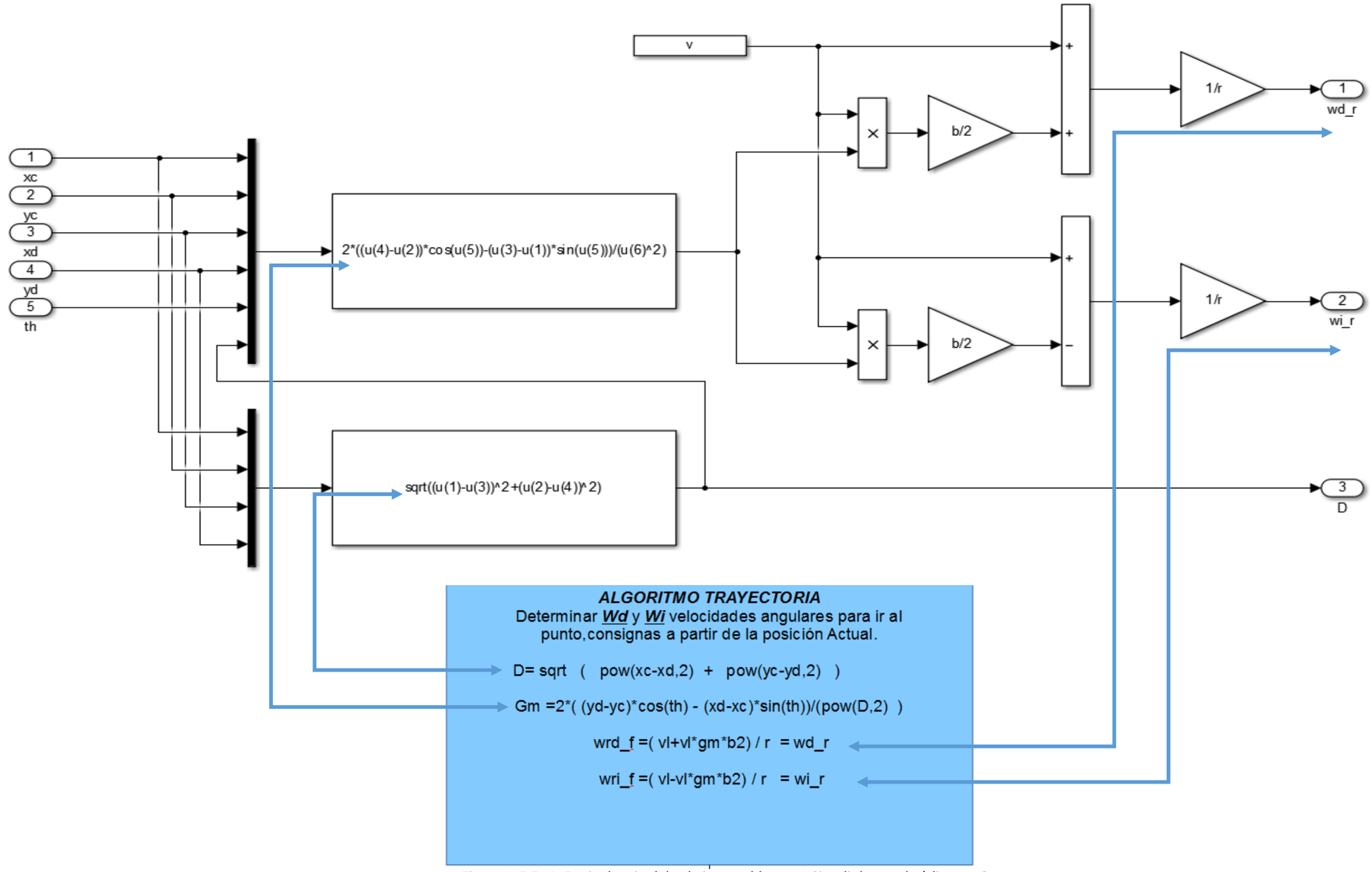


Figura 4.5.5_4: Equivalencia del subsistema bloque A Simulink con el código en C

Subsistema interno del Bloque B:

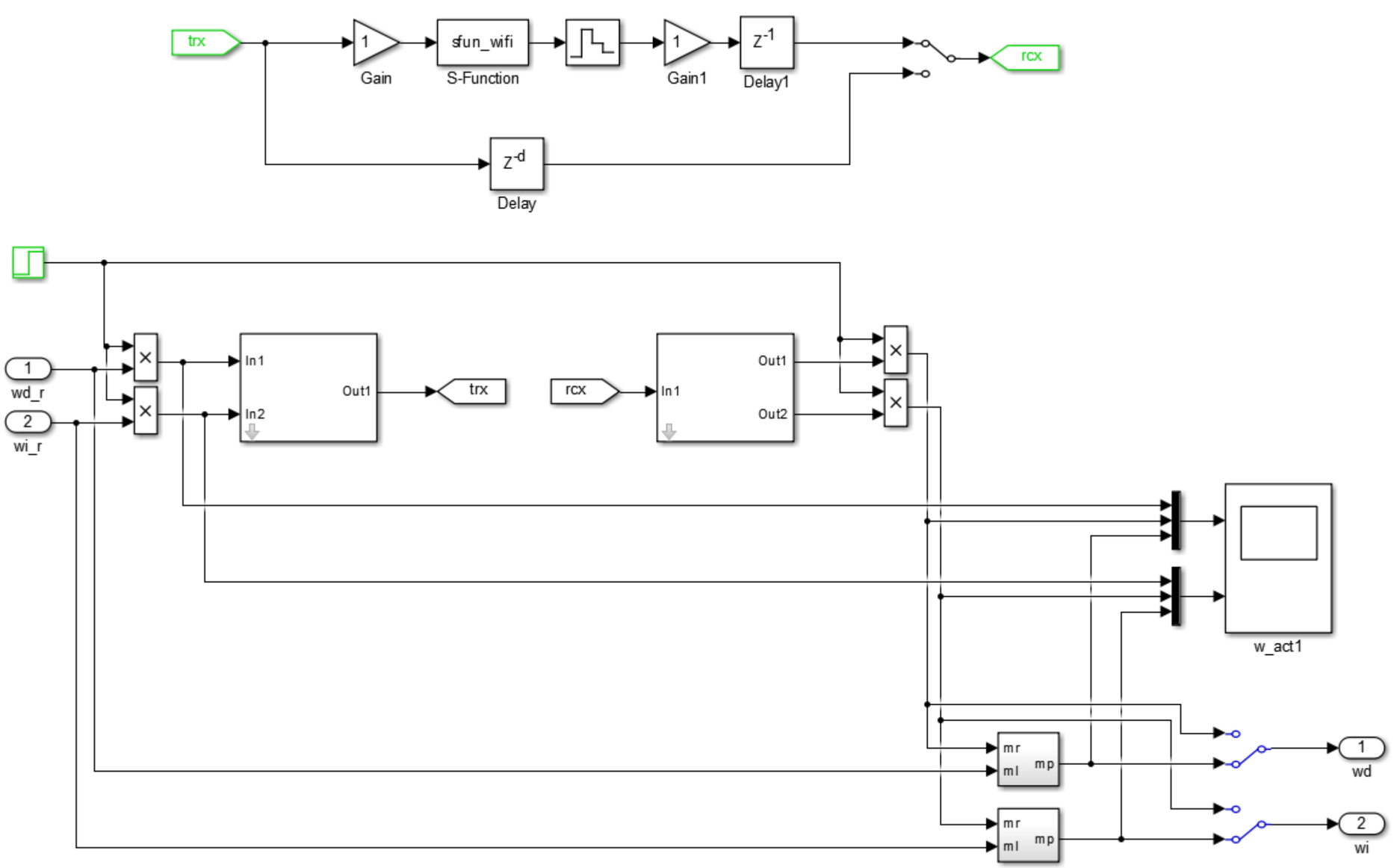


Figura 4.5.5_5: Bloque B S-FUNTION + Predictor de Smith

En la parte superior de la Figura 4.5.5_5 tenemos el bloque equivalente a la S-FUNCTION que establece la comunicación TCP/IP entre el servidor Robot y el cliente Matlab / Simulink.

Y en la parte de abajo tenemos el predictor, que lo que hace cuando la posición de los dos conmutadores de la derecha está hacia abajo, es la de suponer que las velocidades Wd y Wi que determina el algoritmo de seguimiento de trayectoria, han salido a la red, han llegado a la planta, han pasado por los bucles de control de velocidad y se aplicaron a los motores, de forma instantánea. Es como si no existiera el retraso de la red.

Como veremos en el apartado capítulo 5, **apartado 5.3.2 Modalidad Funcionamiento Remoto control basado en Red**, esta estrategia llevada a la práctica, mejora sustancialmente los resultados obtenidos en el seguimiento de la trayectoria.

Un ejemplo de predictor de Smith, es el que se puede ver en la siguiente Figura 4.5.5_6:

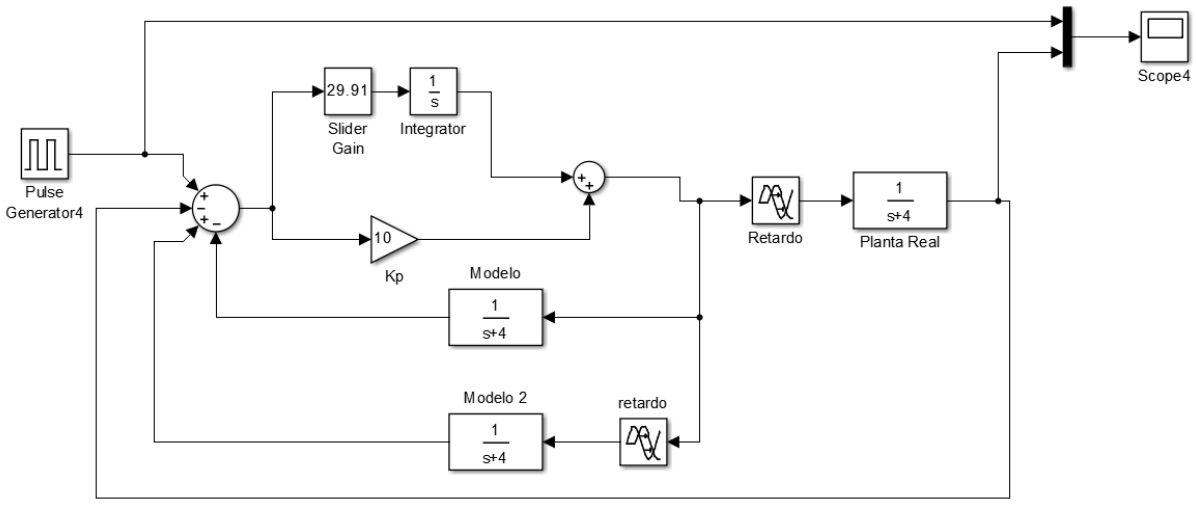
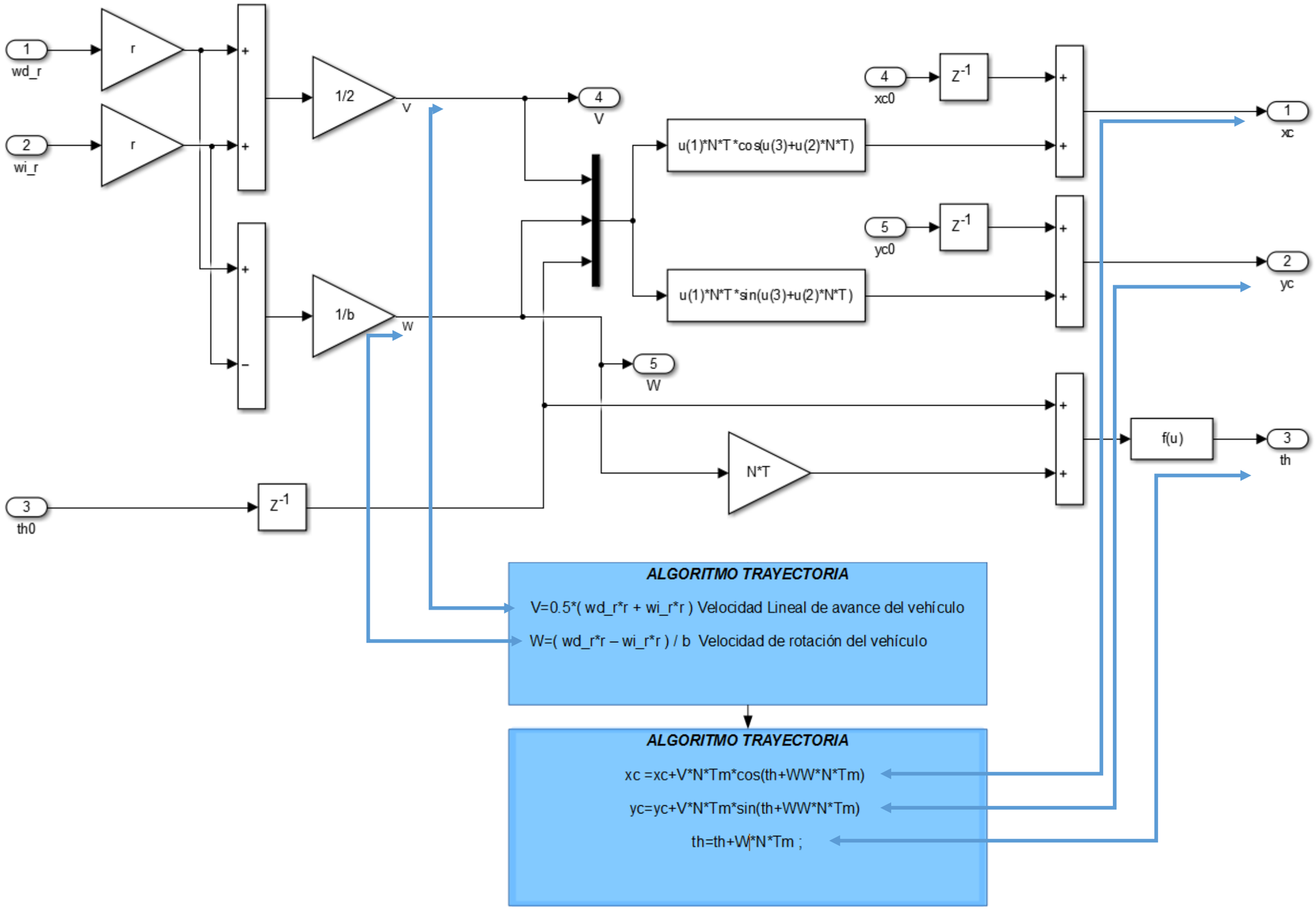


Figura 4.5.5_6: Predictor de Smith

Subsistema interno del Bloque D:

Figura 4.5.5_7: Equivalencia del subsistema bloque D Simulink con el código C



Subsistema interno del Bloque C:

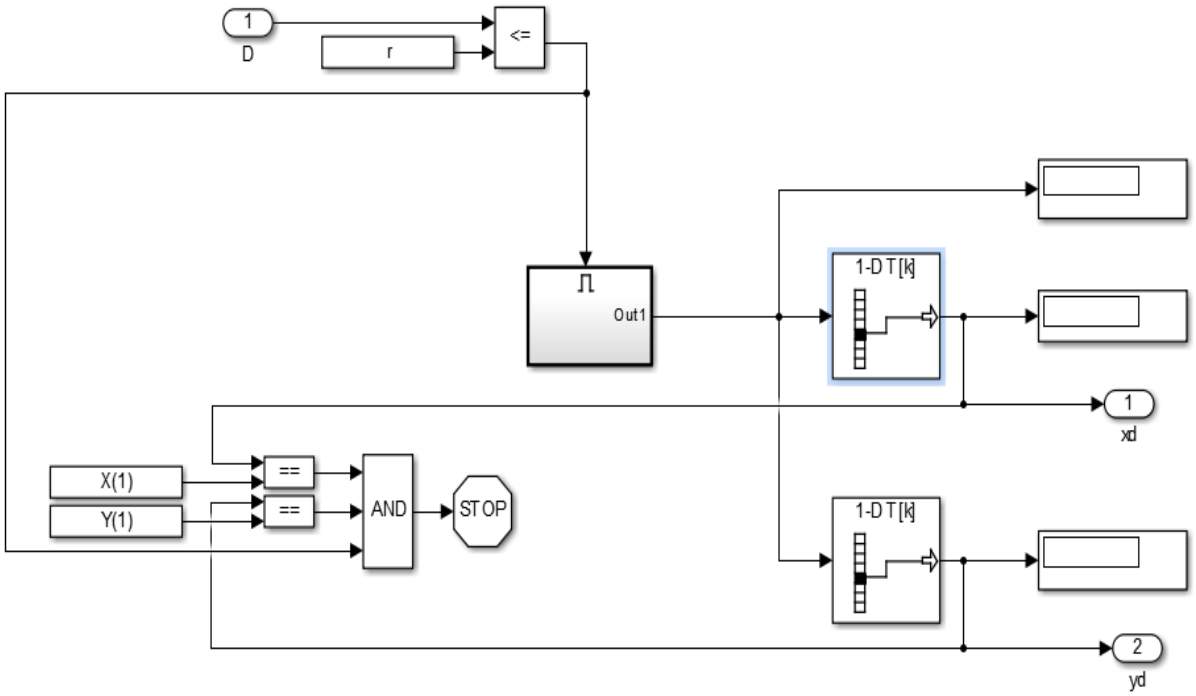


Figura 4.5.5_8: Comparador de Distancia

Este es el bloque encargado de comparar la distancia D (del centro de masas del vehículo hasta el punto consigna actual), para que cuando se cumpla que esta distancia sea menor o igual a r (radio de la rueda), entonces desplazar el punto consigna actual al siguiente contenido en los vectores de la coordenada xd e yd. Finalmente, también se observa que cuando se llega al punto final de la trayectoria se interrumpe la comunicación, la variable stop, es la que se le pasa a la S-FUNCTION para cerrar la comunicación con el servidor.

A continuación, se muestra la interfaz gráfica del usuario para monitorizar el proceso programada en Matlab / Simulink.

En el capítulo siguiente RESULTADOS, se muestran resultados para poder ver la relación entra la trayectoria consigna y la que va trazando el Robot.

Figura 4.5.5_9: Interfaz de usuario Matlab / Simulink

5.- RESULTADOS OBTENIDOS

5.1.- Modalidad Funcionamiento Local Cliente Matlab / Simulink.

5.1.1.- Trayectoria Cuadrada con las ruedas sin contacto en el suelo.

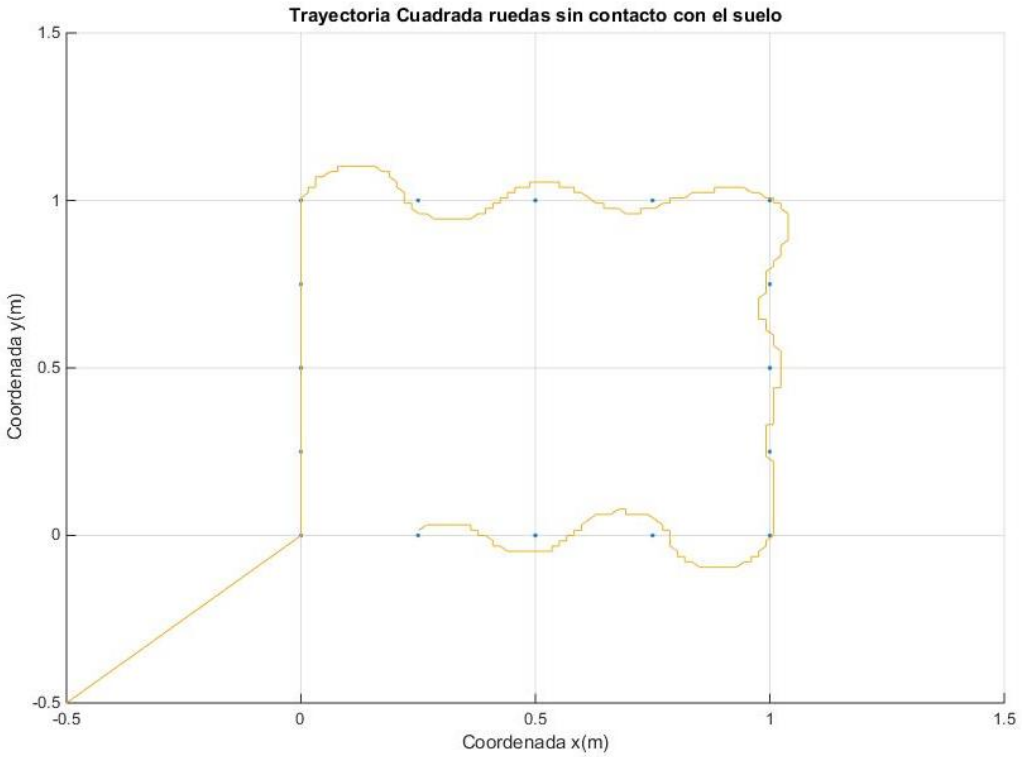


Figura 5.1.1_1: Trayectoria Ruedas Cuadrada

En la Figura 5.1.1_1, la línea de puntos (azules) es la trayectoria consigna y la línea(naranja) continua la trayectoria del robot. Se ha realizado con las ruedas sin contacto con el suelo y sirve para comprobar el funcionamiento correcto del algoritmo de seguimiento de la trayectoria, y la comunicación entre Cliente (Matlab / Simulink) y el Servidor (Robot), que permite la monitorización de la trayectoria seguida. En el siguiente video-vínculo, se puede ver la monitorización de la trayectoria.

- Resultado 1: Monitorización Trayectoria.

Videos\Trayec 5 1 1-1.mp4

5.1.2.- **Trayectoria Cuadrada con las ruedas con contacto en suelo.**

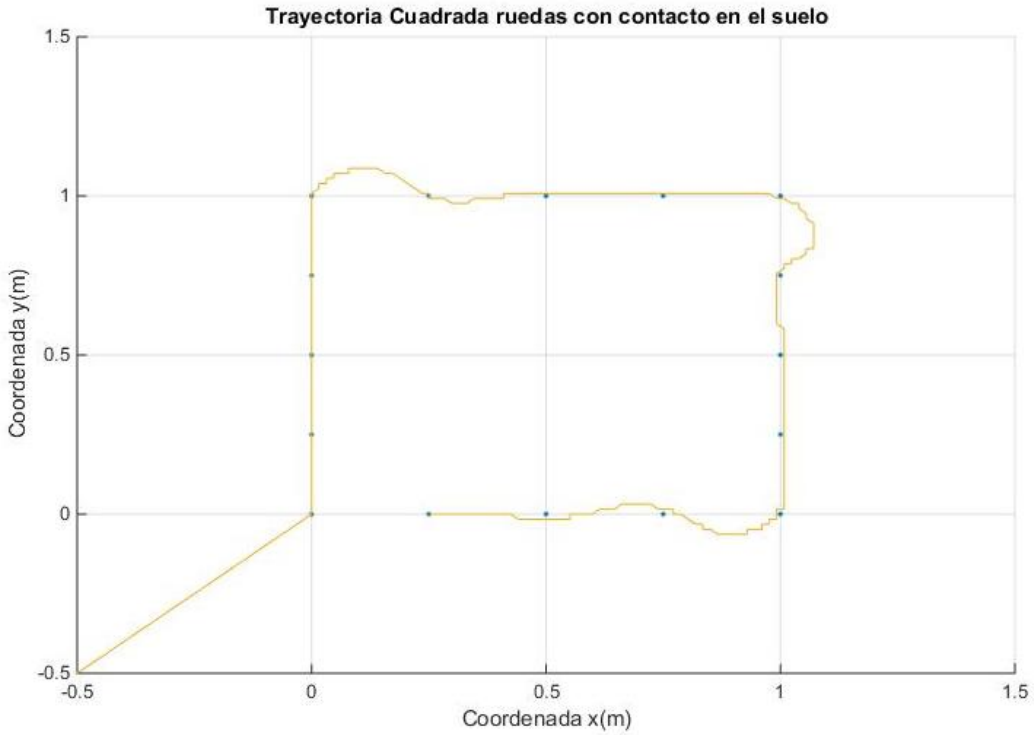


Figura 5.1.2_1: Trayectoria Cuadrada (en el suelo)

En este caso, Figura 5.1.2_1, las ruedas si están en contacto con el suelo.

- Resultado 1: Monitorización Trayectoria.

  Videos\Trayec 5 1 2-1.mp4

- Resultado 2: Video Robot trazando Trayectoria.

  Videos\Robot 5 1 2-1.mp4

- Resultado 3: Evolución temporal de las posiciones del Robot xc e yc.

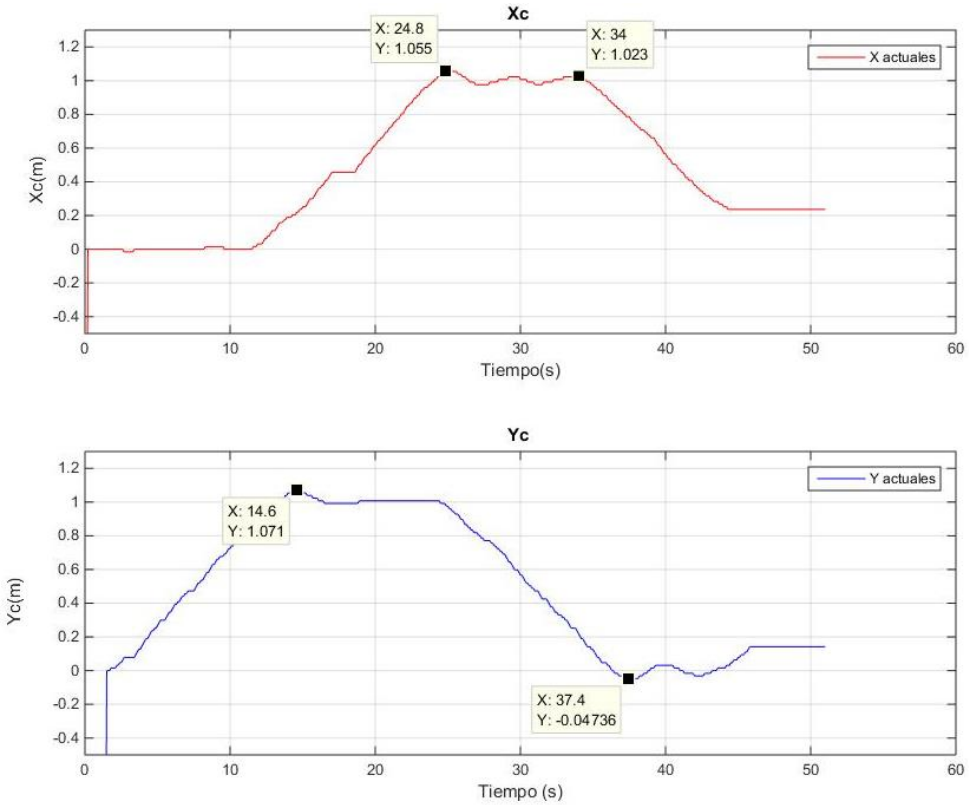


Figura 5.1.2_2: Trayectoria figura Lissajous-1 (en el suelo)

5.1.3.- **Trayectoria figura Lissajous-1 con las ruedas en contacto con el suelo**

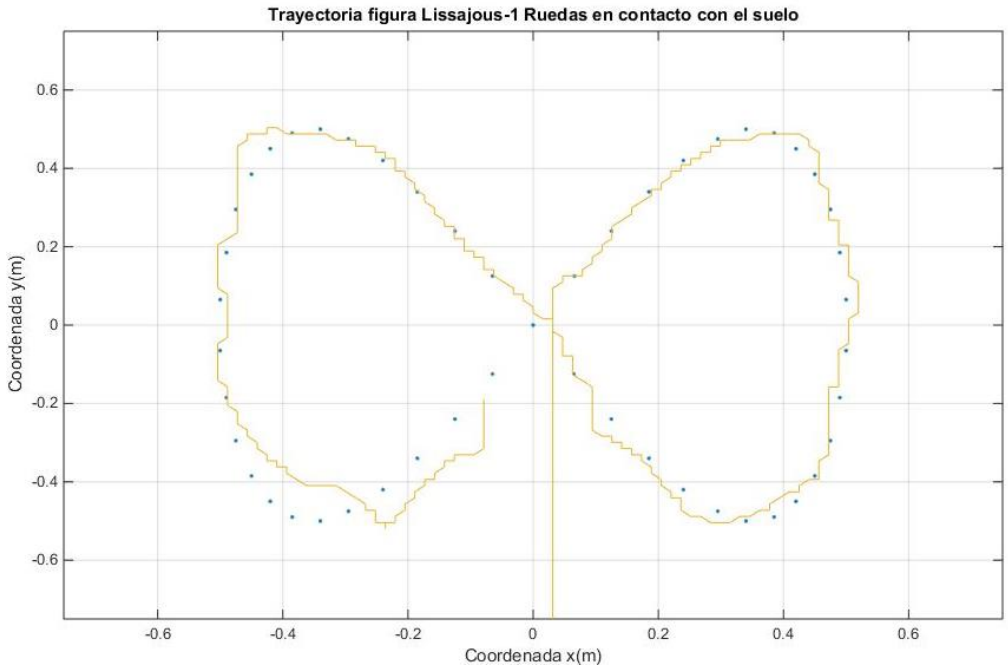


Figura 5.1.3_1: Trayectoria figura Lissajous-1 (en el suelo)

En este caso, Figura 5.1.3-2, las ruedas si están en contacto con el suelo.

- Resultado 1: Video Robot trazando Trayectoria,

Videos\Robot 5 1 3-1.mp4

5.2.- Modalidad Funcionamiento Local Cliente Android.

5.2.1.- Trayectoria Cuadrada con las ruedas sin contacto en el suelo.

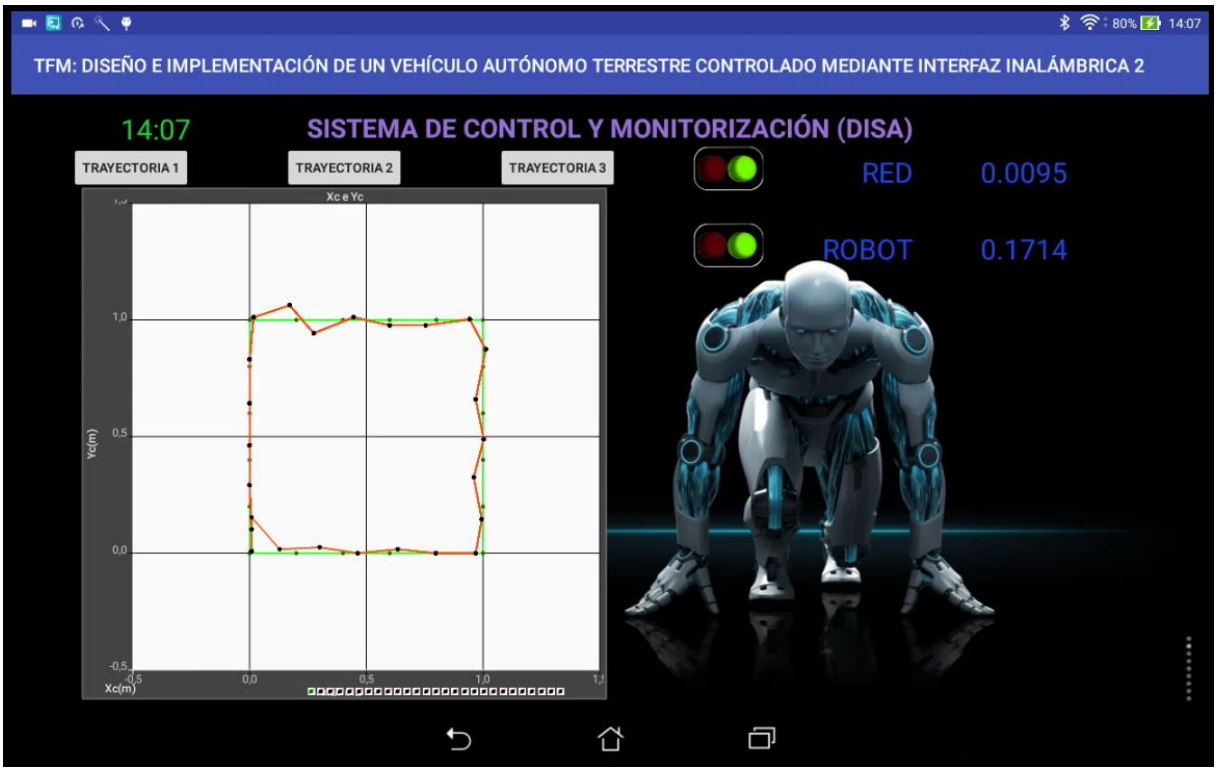


Figura 5.2.1_1: Trayectoria Ruedas sin contacto con el Suelo Android

En la Figura 5.2.1_1 vemos la trayectoria consigna en verde, y en rojo la trayectoria trazada con las ruedas del Robot sin estar en contacto con el suelo. Al igual que en el apartado 5.1.1 Trayectoria cuadrada con las ruedas sin contacto en el suelo, el objetivo es el de comprobar el funcionamiento correcto del algoritmo de seguimiento de la trayectoria, y la comunicación entre Cliente, en este caso Android y el Servidor (Robot), que permite la monitorización de la trayectoria seguida.

La monitorización de la trayectoria la podemos ver en el siguiente video-vínculo:

- Resultado 1: Monitorización Trayectoria.
  Videos\Trayec 5 2 1-1.mp4

5.2.2.- **Trayectoria Cuadrada con las ruedas en contacto con el suelo.**

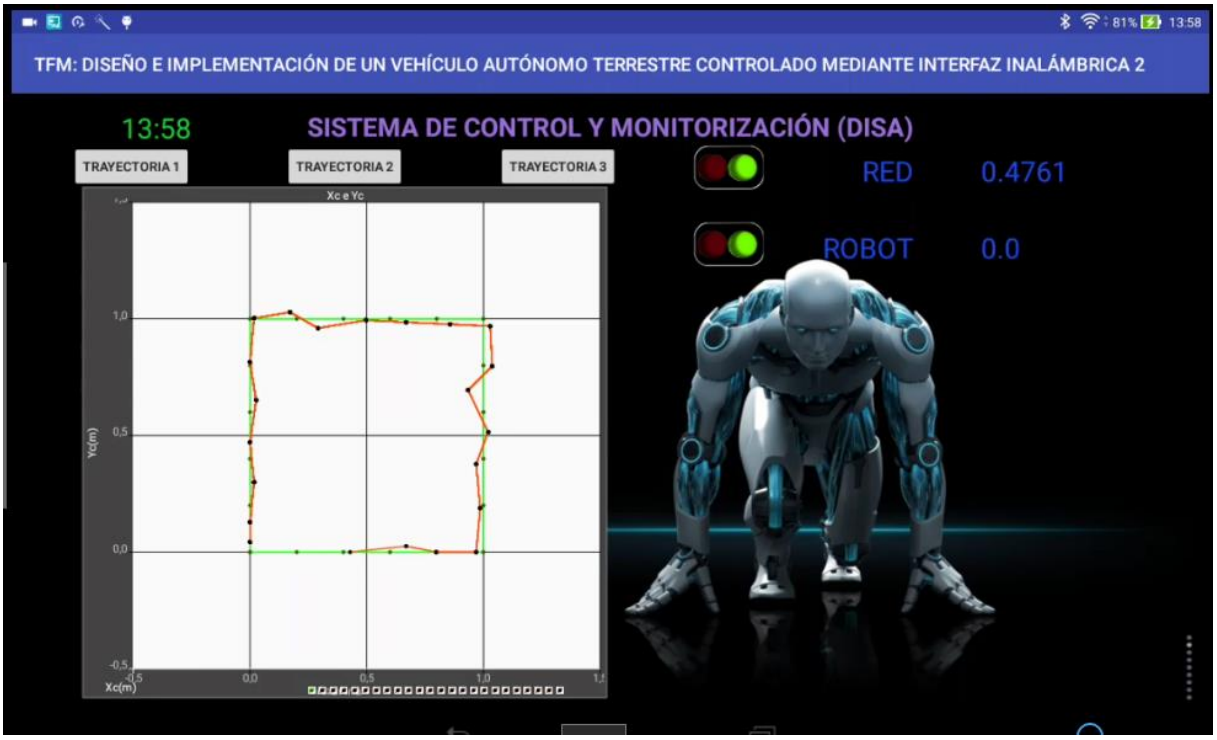


Figura5.2.2_1: Trayectoria Ruedas sin contacto con el Suelo Android

- Resultado 1: Monitorización Trayectoria.

Videos\Trayec 5 2 2-1.mp4

- Resultado 2: Video Robot trazando Trayectoria.

Videos\Robot 5 2 2-1.mp4

5.3.- Modalidad Funcionamiento Remoto control basado en Red.

5.3.1.- Trayectoria Cuadrada sin Predictor de Smith.

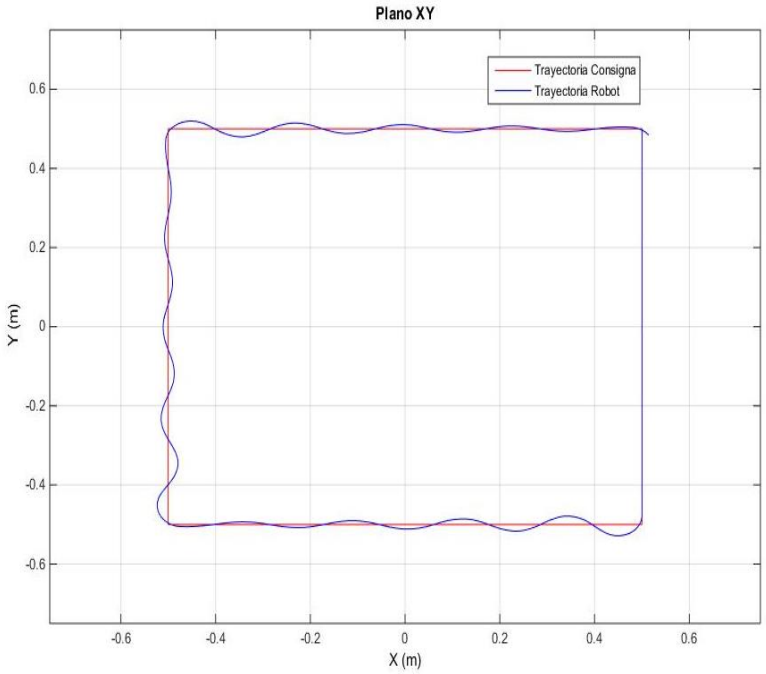


Figura 5.3.1_1: Trayectoria Cuadrada

- Resultado 1: Monitorización Trayectoria.

  Videos\Trayec 5 3 1-1.mp4

- Resultado 2: Video Robot Trazando Trayectoria.

  Videos\Robot 5 3 1-1.mp4

- Resultado 3: Evolución temporal de las posiciones del Robot xc e yc.

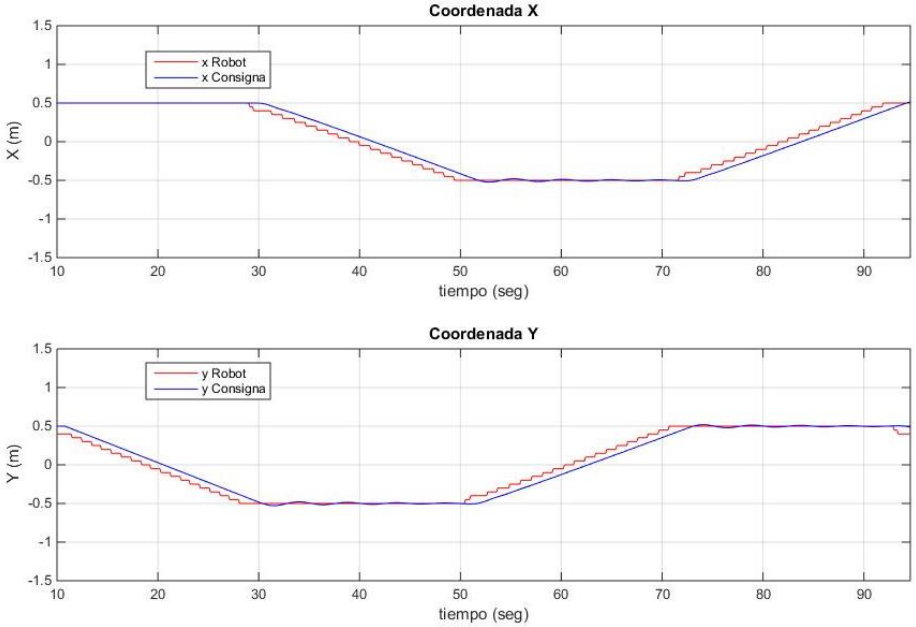


Figura 5.3.1_2: Evolución Temporal Coordenadas xc e yc

- Resultado 4: Evolución temporal de las velocidades Wd y Wi.

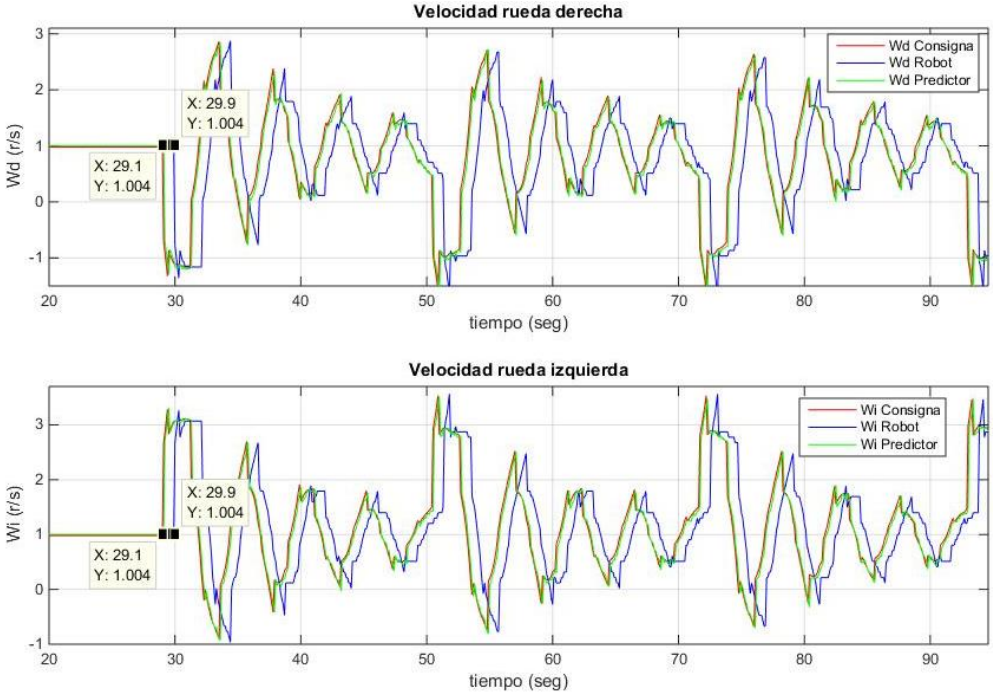


Figura 5.3.1_3: Evolución Temporal Velocidades Wd y Wi

- Resultado 5: Evolución temporal D(m), Th(°), Vl(m/s) y W(r/s)

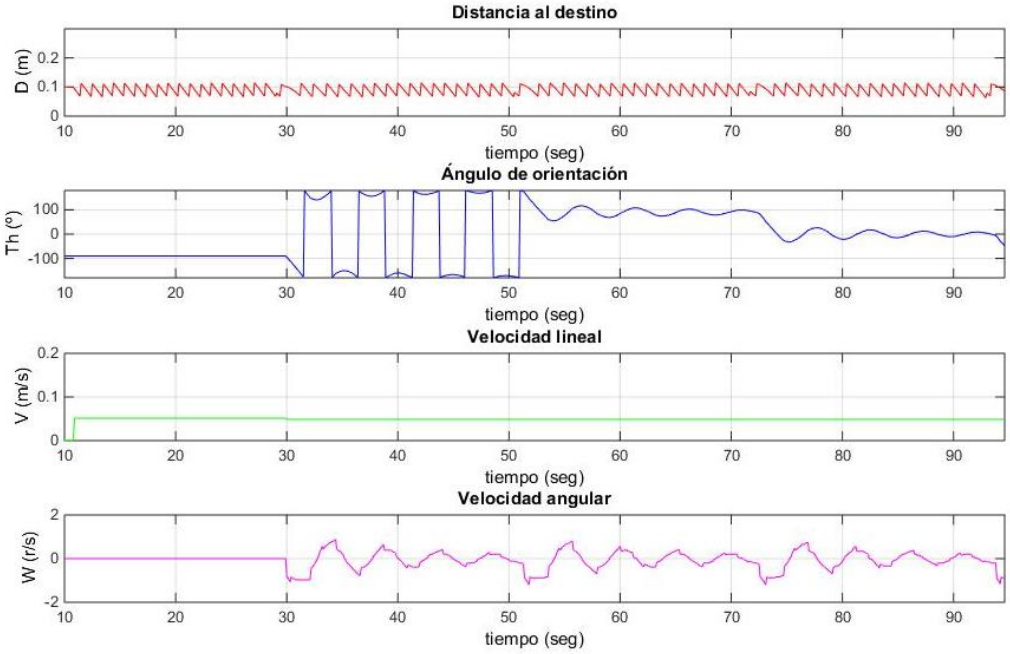


Figura 5.3.1_4: Evolución Temporal D, Th, Vl, W

5.3.2.- **Trayectoria Cuadrada con Predictor Smith.**

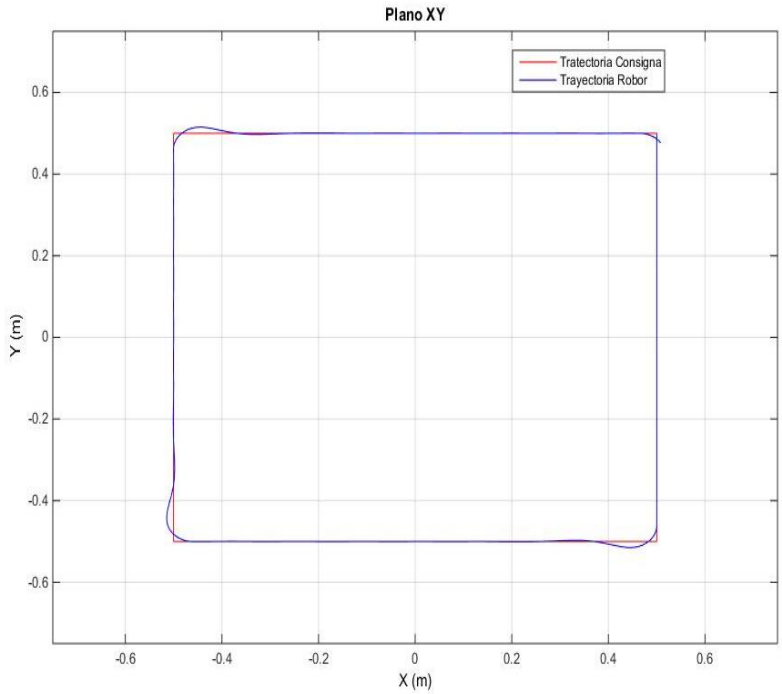


Figura 5.3.2_1: Trayectoria Cuadrada

- Resultado 1: Monitorización Trayectoria.

Videos\Trayec 5 3 2-1.mp4

- Resultado 2: Video Robot Trazando Trayectoria.

Videos\Robot 5 3 2-1.mp4

- Resultado 3: Evolución temporal de las posiciones del Robot xc e yc

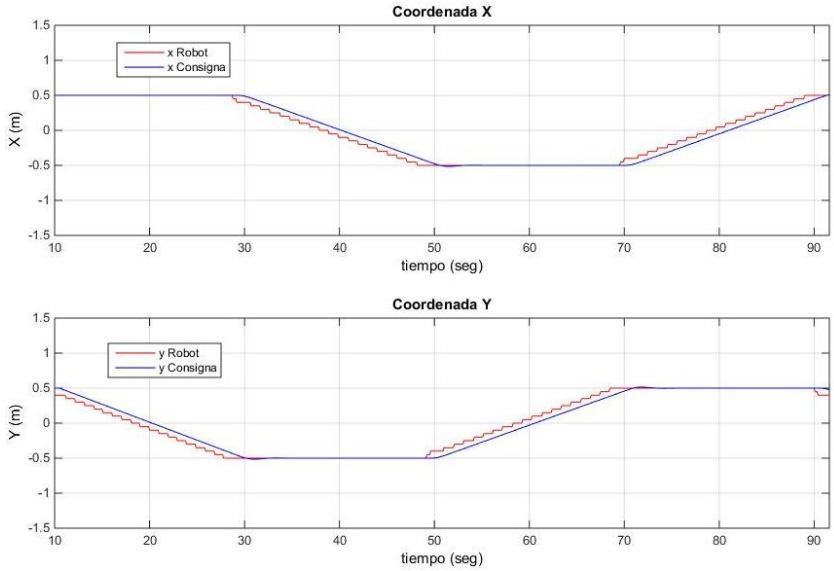


Figura 5.3.2_2: Evolución Temporal xc e yc

- Resultado 4: Evolución temporal de las velocidades Wd e y Wi

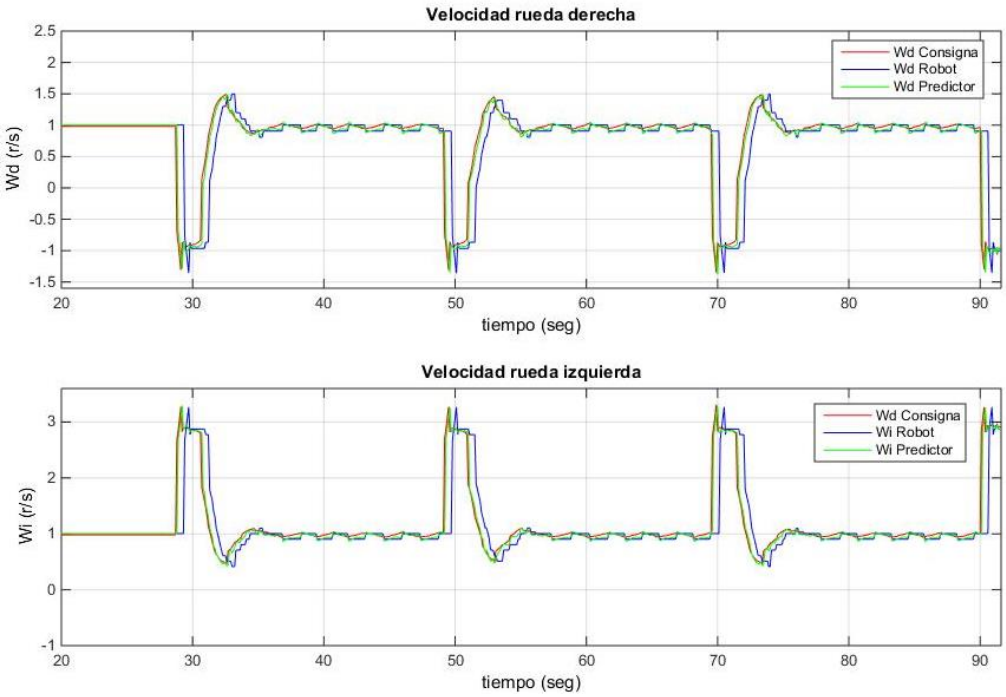


Figura 5.3.2_3: Evolución Temporal Velocidades Wd y Wi

- Resultado 5: Evolución temporal D(m), Th(°), Vl(m/s) y W(r/s

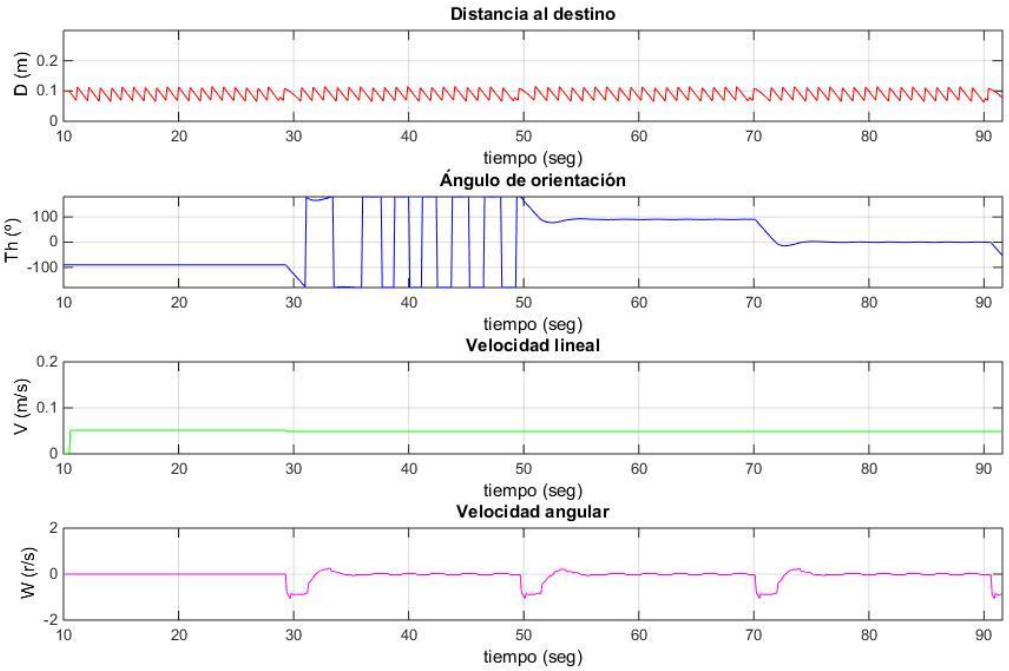


Figura 5.3.2_4: Evolución Temporal D, Th, Vl, W

5.3.3.- **Trayectoria Lissajous 1-2 sin Predictor de Smith**

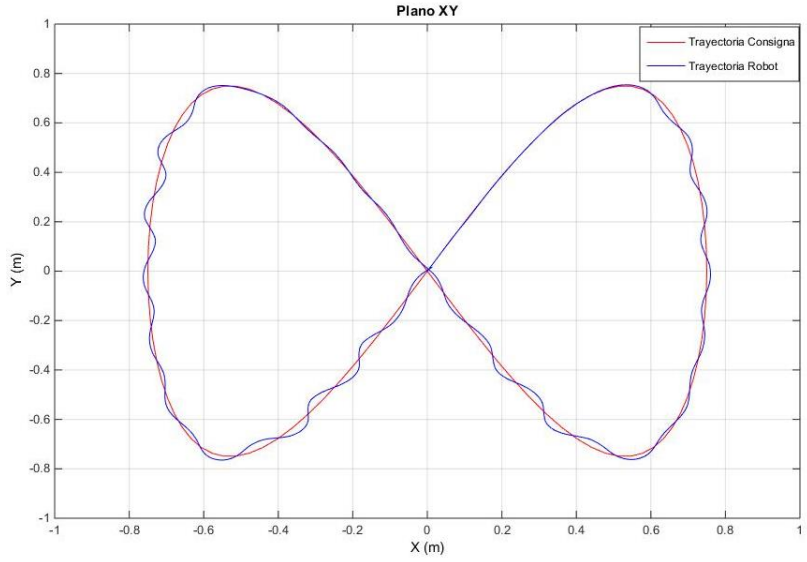


Figura 5.3.3_1: Trayectoria Lissajous-1

- Resultado 1: Monitorización Trayectoria.

Videos\Robot 5 3 3-1 .mp4

- Resultado 2: Evolución temporal de las posiciones del Robot x e y

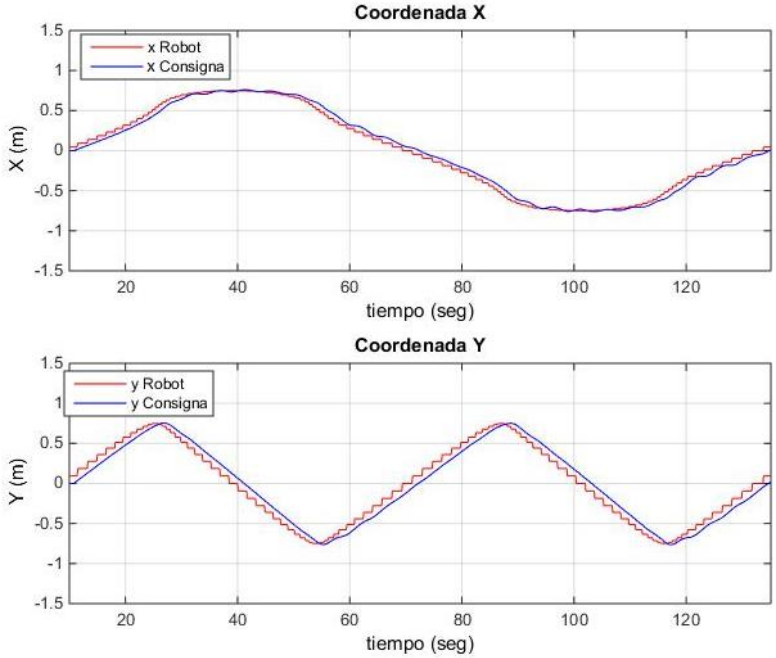


Figura 5.3.3_2: Evolución Temporal xc e yc

- Resultado 3: Evolución temporal de las velocidades Wd e y Wi

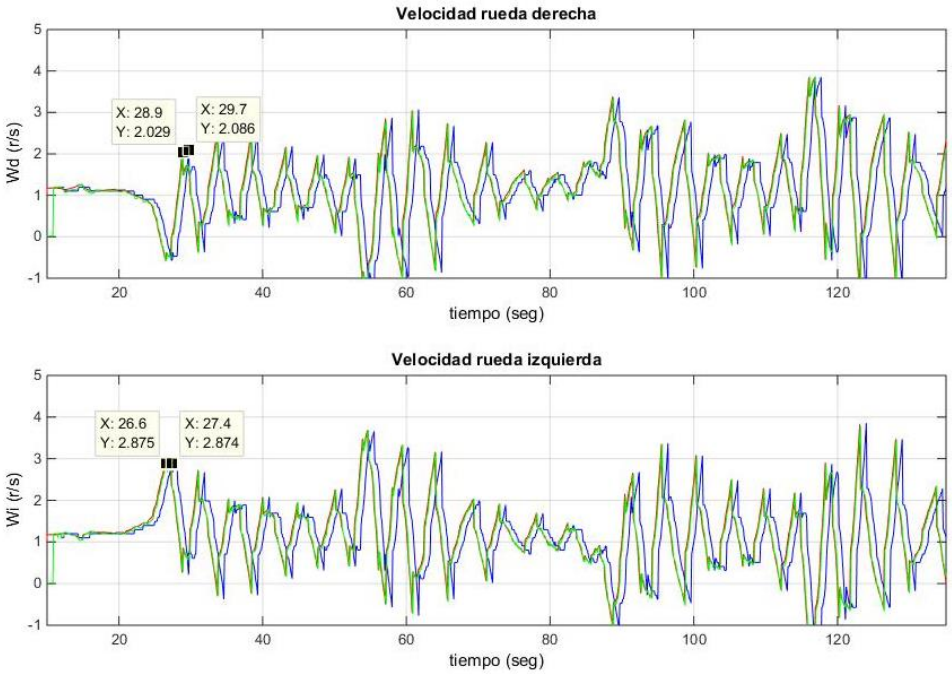


Figura 5.3.3_3: Evolución Temporal Velocidades Wd y Wi

- Resultado 4: Evolución temporal D(m), Th(°), Vl(m/s) y W(r/s)

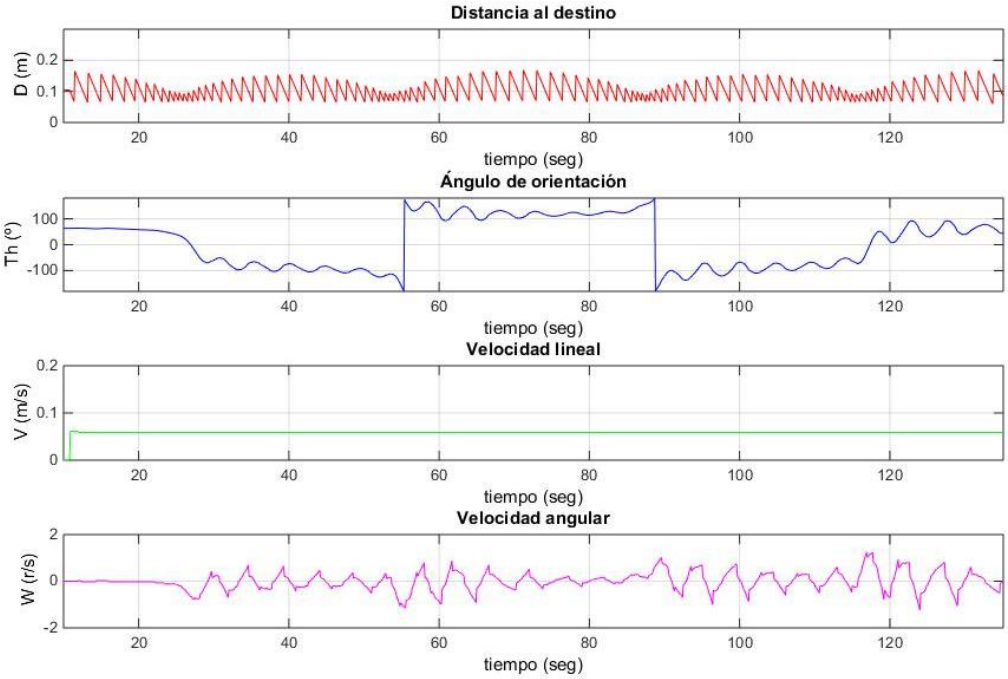


Figura 5.3.3_4: Evolución Temporal D, Th, Vl, W

5.3.4.- **Trayectoria Lissajous 1-2 con Predictor Smith.**

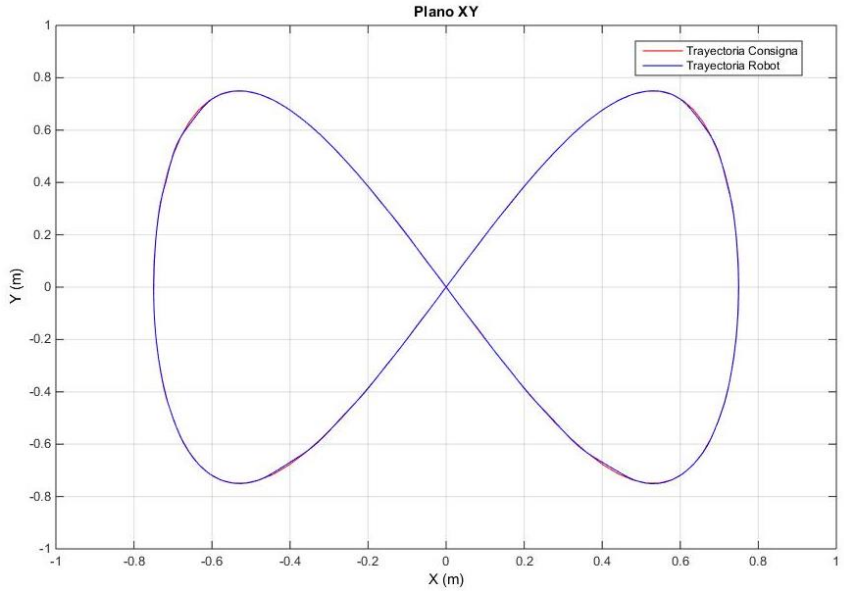


Figura 5.3.4_1: Trayectoria Lissajous-1

- Resultado 1: Monitorización Trayectoria.

Videos\Robot 5 3 4_1.mp4

- Resultado 2: Evolución temporal de las posiciones del Robot x e y.

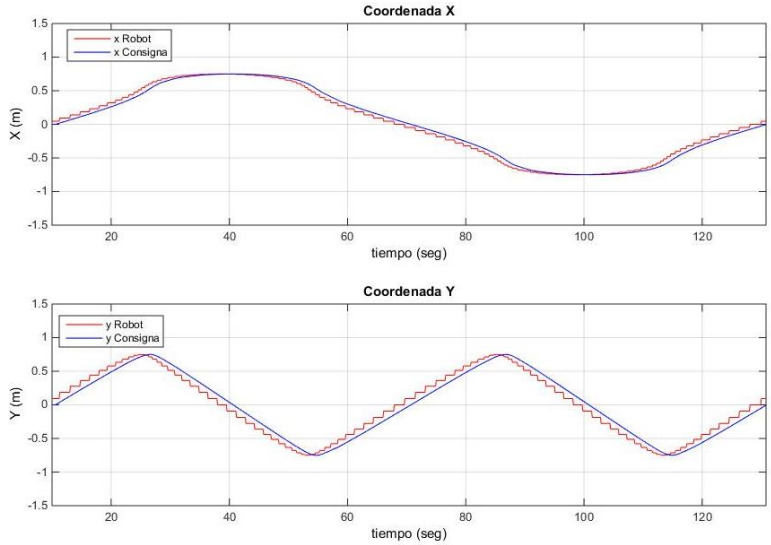


Figura 5.3.4_2: Evolución Temporal xc e yc

- Resultado 3: Evolución temporal de las velocidades Wd e y Wi

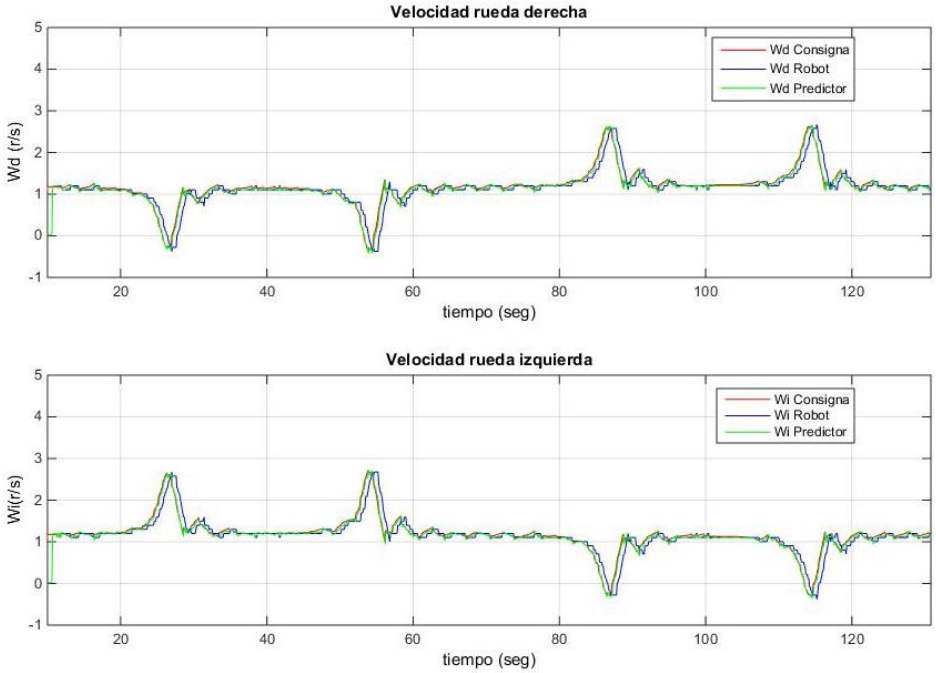


Figura 5.3.4_3: Evolución Temporal Velocidades Wd y Wi

- Resultado 4: Evolución temporal D(m), Th(°), Vl(m/s) y W(r/s)

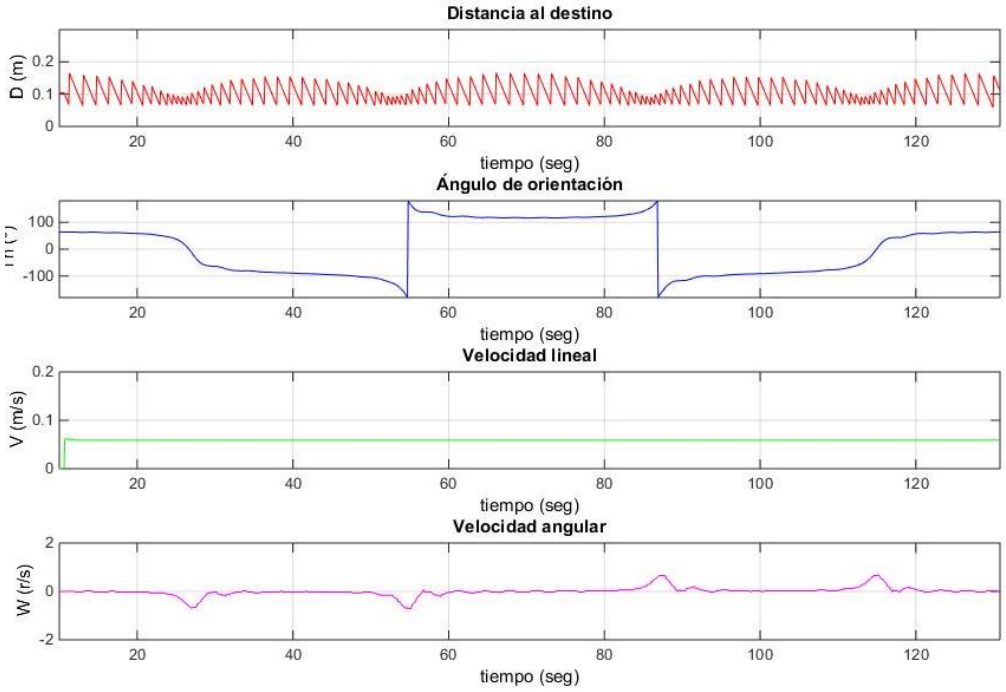


Figura 5.3.4_4: Evolución Temporal D, Th, Vl, W

6.- CONCLUSIONES

A partir de todos los experimentos realizados se obtienen claramente una serie de conclusiones. Que son las siguientes:

A pesar de que se han conseguido varios e importantes objetivos como son la implementación del algoritmo de seguimiento de trayectoria, tanto en modo local como en la modalidad control basado en red, la correcta implementación de los bucle de control de velocidad para cada una de las ruedas, la implementación de una estrategia software que corrige la acción de control para sacar a los motores de la no linealidad de la zona muerta, la implementación de un filtro de velocidad de segundo orden tipo Butterwoth, la implementación del predictor de Smith, la implementación del servidor en el robot que sirve para escuchar la posible entrada de un cliente y establecer la comunicación, la implementación del cliente tanto en Matlab / Simulink , como en el dispositivo Android (dos formas muy distintas de programa pero que en el fondo guarda relación), el diseño del chasis y el bastidor en 3D, mediante la herramienta software NX 10

de siemens, la elección correcta del sistema energético y finalmente el ensamblaje total de todas las tarjetas electrónicas junto con los motores, encoders, bastidor, chasis etc, que dan como resultado final el Robot, no podemos de dejar de lado otras conclusiones a las que se han llegado y que, de forma crítica y constructiva, sirven para que si se sigue en el departamento con el proyecto, dar una solución más óptima aún si cabe, que pasamos a describir a continuación.

- El algoritmo de seguimiento de trayectoria cuando se trabaja en la modalidad local, es claramente más efectivo y por lo tanto los resultados son mejores, que, en el caso de trabajar en la modalidad basada en red, sin el predictor de Smith.

  Esto se debe a que cuando trabajamos en forma local el periodo de muestre, es decir desde que se conoce la posición actual del robot, sabiendo el punto consigna al que nos dirigimos y se actúa para determinar las consignas de velocidad wd y wi y finalmente se aplican a los bucles de control de velocidad, pasan exactamente 100 ms. Por el contrario, en el caso de la modalidad control de seguimiento de trayectoria basado en red, al periodo trascurrido para realizar las acciones anteriores, hay que sumarle el retraso producido por el enlace de comunicación (*Round-Trip Delay*), que en total suman 600 ms, ver Figura 4.5.5_2.

  Si estando en la modalidad control de seguimiento de trayectoria basado en red se activa el predictor de Smith los resultados obtenidos ahora, se acercan mucho a los de la modalidad control local, aunque no llegan a ser exactamente igual de buenos.

- Tanto en una modalidad como la otra, la limitación más importante con la que nos hemos encontrado, ha sido la resolución del sensor de velocidad que como sabemos del capítulo 3, apartado 3.5 es de 1.74 rad/s.

  A pesar de que se ha utilizado un filtro de segundo orden *Butterworth* para tratar la velocidad, este no puede paliar por completo este problema y, por lo tanto, existe siempre una diferencia significativa, entre la

velocidad real y la determinada por el conjunto encoders-software. Esto produce que el algoritmo de seguimiento de trayectoria se piense que ha avanzado más o menos el vehículo con respecto a la realidad, puesto que recordemos que las posiciones actuales que se determinan en cada interacción de bucle, se obtienen a partir de las velocidades obtenidas por el conjunto encoders-software. Esto no solamente produce un error en el avance del vehículo calculado por el algoritmo de seguimiento de trayectoria, sino que también se traduce en un error en la orientación del mismo.

El error de posición que se produce, así como el de orientación, que siempre existirán y además son errores que son acumulativos se pueden minimizar, atacando el problema de dos formas distintas, pero conjuntamente.

Si diferenciamos entre **sensorización propioceptiva**, que son los sensores que están a bordo del Robot (miden variables internas del Robot), como es el caso de los encoders que están integrados en el robot, y **sensorización exteroceptiva**, que serían sensores externos al robot, que son capaces de medir variables del entorno del robot, podemos pasar a describir estas mejoras desde estas dos perspectivas.

Con respecto a la **sensorización propioceptiva**:

Lo primero sería la utilización de encoders con mejor resolución, es sabido que existen encoders que pueden informarnos del giro del motor en decimas de grado, en este caso de 0.1grados la resolución pasaría a ser de:

$$\boldsymbol{Velocidad\ m\acute{i}nima}\left(\frac{\boldsymbol{rad}}{\boldsymbol{s}}\right) = \frac{\boldsymbol{0.1puso} * \boldsymbol{0.017453}}{\boldsymbol{10\ ms}} = \boldsymbol{0.17453}$$

Como vemos cambia significativamente.

Continuando con los sensores propioceptivos, también se podría utilizar un giróscopo para la determinación del ángulo girado (orientación del vehículo). Hay giróscopos que nos permitirían obtener ángulos medidos

de menos de 0.1 grado que mejorarían considerablemente el algoritmo de seguimiento de trayectoria, puesto que podríamos comparar el ángulo determinado por el propio algoritmo de seguimiento de trayectoria con el obtenido por medio del giróscopo, permitiendo corregir la acumulación de error en cuanto a la orientación.

Con respecto a **la sensorización exteroceptiva**:

Una posibilidad utilizada en interiores, y que al hilo de este proyecto se está siguiendo en el departamento de ingeniería de sistemas y automática, es utilizar una cámara exterior (cenital) que mediante una forma geométrica determina puesta sobre el vehículo, pueda determinar con exactitud el ángulo girado en cada interacción de bucle de control de seguimiento de trayectoria (visión artificial), con el fin de poder corregir las desviaciones de la orientación en cada vuelta de bucle.

En el caso de que el vehículo tuviera que trazar trayectorias en exterior, la solución óptima sería la utilización de un buen GPS, que como es sabido pueden llegar a estimar la posición con un error de 1 cm. En este caso el algoritmo de seguimiento de trayectoria se ejecutaría cada 100 ms, y cada 10 veces este periodo, es decir, cada segundo se podrían consultar la posición del GPS, entonces el error acumulado en la posición lo podríamos corregir con la posición indicada por el GPS, de tal forma que el error de posición nunca podría acumularse más de un segundo, en el caso de la orientación utilizaríamos la comparación del ángulo determinado por el algoritmo de seguimiento de trayectoria con el giróscopo comentado anteriormente.

Finalmente, la solución definitiva sería la de incorporar una cámara en el propio vehículo, que, junto con lo dicho anteriormente en el caso de trayectorias exteriores, permitiría al Robot darse cuenta de la presencia de obstáculos, y de esta forma se podría corregir o cambiar la trayectoria a fin de evitar chocar con dicho obstáculo.

7.- BIBLIOGRAFÍA

- **Control Automático. Tiempo Continuo y Tiempo Discreto**. (Editorial REVERTÉ ED / UPV). Julián J.Salt Llobregat, Ángel Cuenca Lacruz, Vicente Casanova Calvo, Antonio Correcher Salvador.
- **MECATRÓNICA, CONTROL Y AUTOMATIZACIÓN**. (Editorial marcombo). Fernando Reyes Cortés, Emilio Vargas Soto.
- Tesis Doctoral: **SISTEMAS DE CONTROL BASADO EN RED. MODELADO Y DISEÑO DE ESTRUCTURAS DE CONTROL**. D. Vicente Casanova Calvo.
- **Documentación oficial Android**. https://developer.android.com/index.html
- **Documentación oficial Arduino**. https://www.arduino.cc/
- **FUNDAMENTOS DE ROBOTICA**. (Editorial MCGRAW-HILL). Antonio Barrientos.
- **Apuntes Diseño Electrónico Avanzado (DIE).** Francisco. J.Gimeno. S. Orts y S. Seguí.
- Tesis Doctoral: ***Path Tracking and obstacle Avoindance for miniature Robot By Martin Lundgren*.**

ANEXOS:

8.- ANEXO 1 BATERIAS DE LITIO

8.1.- Descripción

La evolución y proliferación de dispositivos electrónicos alimentados por baterías ha impulsado en los últimos años el desarrollo de nuevas tecnologías que han permitido mejorar las prestaciones de los aparatos electrónicos. Las baterías con base de litio son la última generación de baterías de uso popular. Forman parte de nuestra vida al estar presentes en los Smartphone, Tablet, ordenador portátil, etc. La tecnología de baterías basadas en Litio es ya una tecnología madura después de varias décadas de desarrollo. Sigue siendo una tecnología que se diferencia de las demás por las múltiples ventajas sin apenas factores negativos. Las principales ventajas son su alta densidad de energía, su rápida carga y su ligereza a la vez que el inconveniente principal es su inestabilidad química frente a sobre-cargas o sobre-descargas que obligan a utilizar sistemas electrónicos que protejan a la batería. Hay varias familias de baterías de litio en función de los materiales utilizados en su construcción, principalmente en el cátodo: Cobalto, Manganeso, Hierro fósforo, Titanio, etc.

8.2.- Carga

Las baterías de Litio requieren una técnica de carga muy específica debido a sus características químicas y eléctricas. La razón es que, a diferencia de otras tecnologías, las baterías de litio se vuelven químicamente inestables cuando alcanzan un voltaje en el que aún no están plenamente cargadas. En otras tecnologías, como las basadas en Níquel o Plomo, la batería alcanza la carga plena antes de que la sobre-carga pueda producir daños normalmente debidos al calor liberado por la energía no almacenada. Sin embargo, las baterías de Litio se vuelven químicamente inestables cuando alcanzan un determinado voltaje mientras que no están aun totalmente cargadas. Por este motivo, se debe utilizar una carga combinada corriente constante/voltaje constante (CC/CV) de forma que la batería se carga a intensidad constante hasta que alcanza el voltaje máximo. En ese punto, el voltaje debe mantenerse constante mientras la intensidad va disminuyendo hasta un nivel señalado en el que se considera cargada la batería. Las baterías de Litio admiten una intensidad de carga de relación elevada con respecto a la Capacidad en comparación a otras tecnologías y tienen un rendimiento muy alto en la carga.

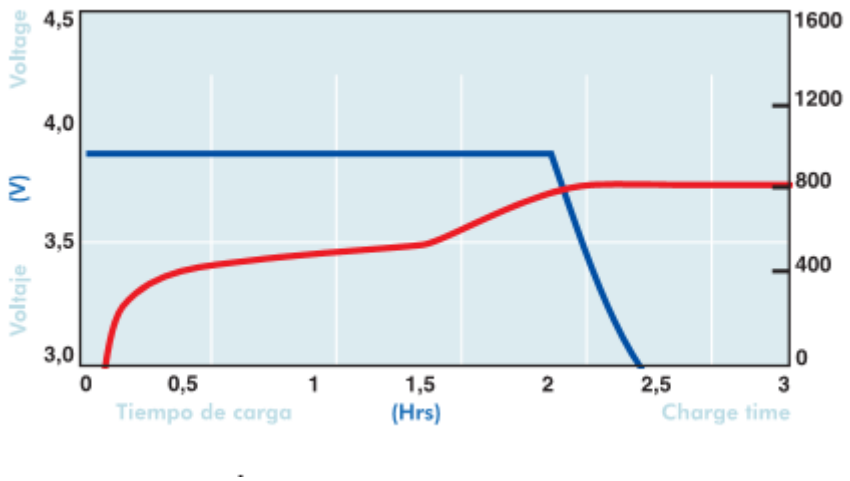


Figura 8.2_1: Proceso de carga baterías de Litio

En nuestro caso la protección en el proceso de carga la aseguramos mediante el cargador especial siguiente:

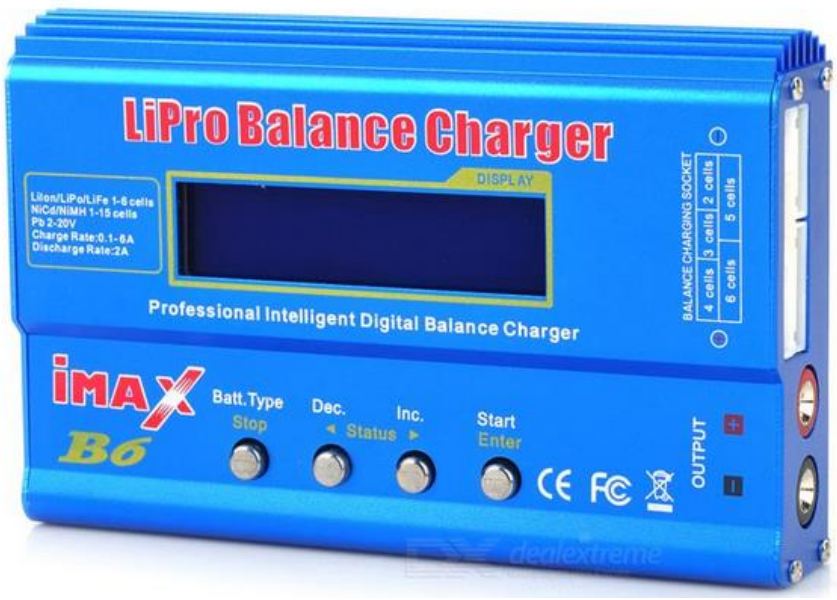


Figura 8.2_2: Cargador de baterías de Litio

Para cargar la batería vamos siempre a la opción Balance (siempre hay que usar esta opción para evitar estropear nuestra batería li-po), no debemos olvidar conectar la clavija blanca en su correspondiente clavija en el cargador. Seleccionamos el tipo de batería y el número de celdas 3,4,7 etc…, en nuestro caso LI-PO de 3 celdas 11.1 voltios.

Y muy importante ajustar la corriente en 1 Amperio para la máxima, y ya podemos empezar el proceso de carga.

8.3.- Descarga

Las baterías de Litio no deben descargarse por debajo de un determinado voltaje. Si esto sucede, la batería se deteriora disminuyendo la capacidad, el número de ciclos de carga y descarga o degradándose el electrolito. La auto-descarga de las baterías de Litio es muy inferior a la de otras tecnologías.

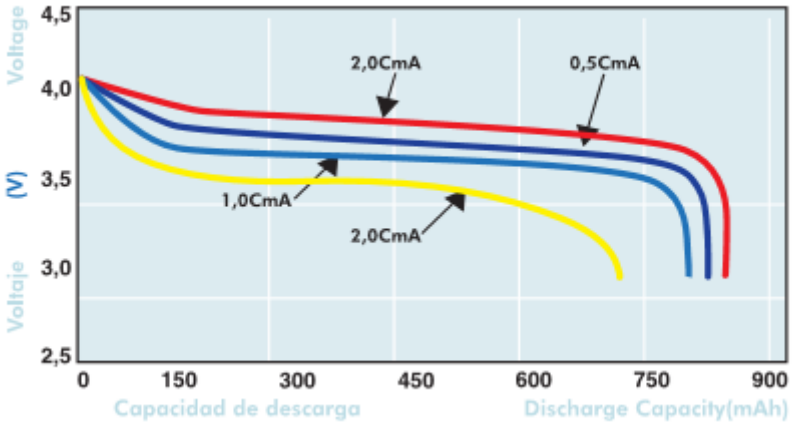


Figura 8.3_1: Proceso de descarga baterías de Litio

Está previsto la instalación de un pequeño circuito electrónico para evitar una descarga profunda, que como se ha indicado en este tipo de baterías puede degradar y acortar la vida de la batería.

8.4.- Circuito de Protección

La vulnerabilidad de las baterías de Litio frente a sobre-voltaje, sobre-descarga y sobre-intensidad entre otros, hace muy recomendable (casi imprescindible) el uso de circuitos electrónicos que controlen los valores de voltaje e intensidad en carga y descarga para evitar daños en la batería. El término PCM sirve para designar un pequeño circuito electrónico que controla los parámetros peligrosos para la batería. El PCM tiene el control y capacidad de desconectar la batería para protegerla tanto en la carga como en la descarga. Son circuitos muy simples y muy eficaces que conviven con las baterías de Litio en casi todas las situaciones.

El voltaje de cada celda está comprendido entre 2.0 a 3.6v en el caso de LiFePo, que es nuestro caso, si no tenemos circuito de protección y trabajamos fuera de este rango, como se comentó antes, podemos dañar la batería.

8.5.- Asociación de baterías

La asociación de baterías en “packs” es frecuente ante la necesidad de conseguir una batería con el voltaje y la capacidad necesarios para cubrir las necesidades de los dispositivos electrónicos. En todas las tecnologías, si bien la asociación en serie es relativamente sencilla y segura, la unión en paralelo es fuente de problemas de funcionamiento del pack. Las baterías de Litio, sin embargo, pueden ser conectadas en paralelo en pequeñas cantidades (2-4). Es fundamental usar PCMs adecuados para asociaciones en serie que monitoricen los niveles de cada batería de forma que si una de las baterías en serie alcanza el nivel máximo o mínimo el PCM debe aislar la batería. La asociación de baterías de Litio en serie con PCM provoca la descompensación de las baterías. Este efecto se debe a que las baterías no son exactamente iguales a pesar de que deben asociarse baterías los más parecidas posible. Al no ser iguales, no se cargan y descargan a la vez y siempre habrá una batería del pack que alcance el nivel máximo de carga o de descarga que provocará la desconexión por el PCM. El desequilibrio de las baterías es un síntoma grave ya que el pack de baterías simulará tener la capacidad de la batería menos descargada del conjunto dando la sensación que el pack tiene menos capacidad de la que realmente podría dar.

8.6.- Balanceo de las celdas de una asociación de baterías

El balanceo es una técnica que se aplica a packs de baterías para corregir los desequilibrios que aparecen entre las diferentes células de un pack. Esta función se puede realizar de muchas maneras. El cargador anterior incorpora un balanceador por batería que descarga una parte de la intensidad de carga cuando la batería está a punto de alcanzar el voltaje máximo. De esta manera se ralentiza la última parte de la carga mientras que las demás baterías del pack se siguen cargando más rápido. De esta manera, en cada carga, se minimiza el desequilibrio entre las baterías. En nuestro caso es imprescindible para la vida útil de la batería.

9.- ANEXO 2 ESQUEMA ELECTRÓNICO DRIVER DOBLE PUENTE EN H

9.1.- Introducción

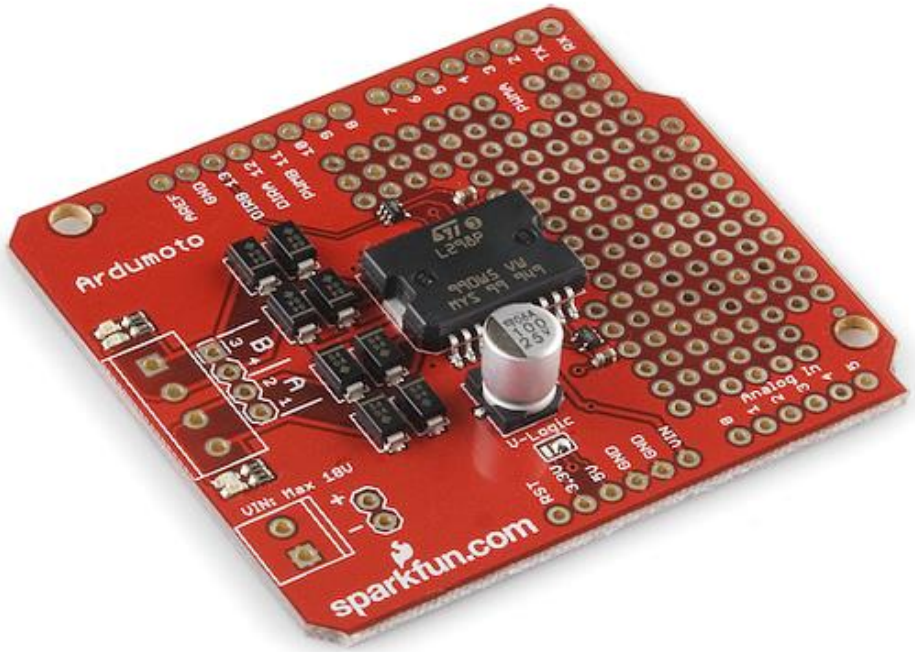


Figura 9.1_1: Driver Ardumoto

Este driver (Ardumoto), es el encargado de aptar la pequeña señal de salida PWM de Arduino a la señal de potencia que llega a la alimentación de los motores.

El driver doble puente en H Ardumoto está diseñado para que adapte perfectamente con los pines de Arduino due. Mediante este driver se pueden controlar dos motores de corriente continua (2 Amperios por canal) o un motor paso a paso bipolar de hasta 2 amperios.

La alimentación de toda la circuitería se realiza a partir de los pines de Arduino, incluye dos leds por canal uno amarillo y otro azul que indican el sentido de giro. Las salidas de potencia hacia el motor están protegidas con diodos para evitar cortocircuitos.

A continuación, se muestra los pines y conexiones del driver:

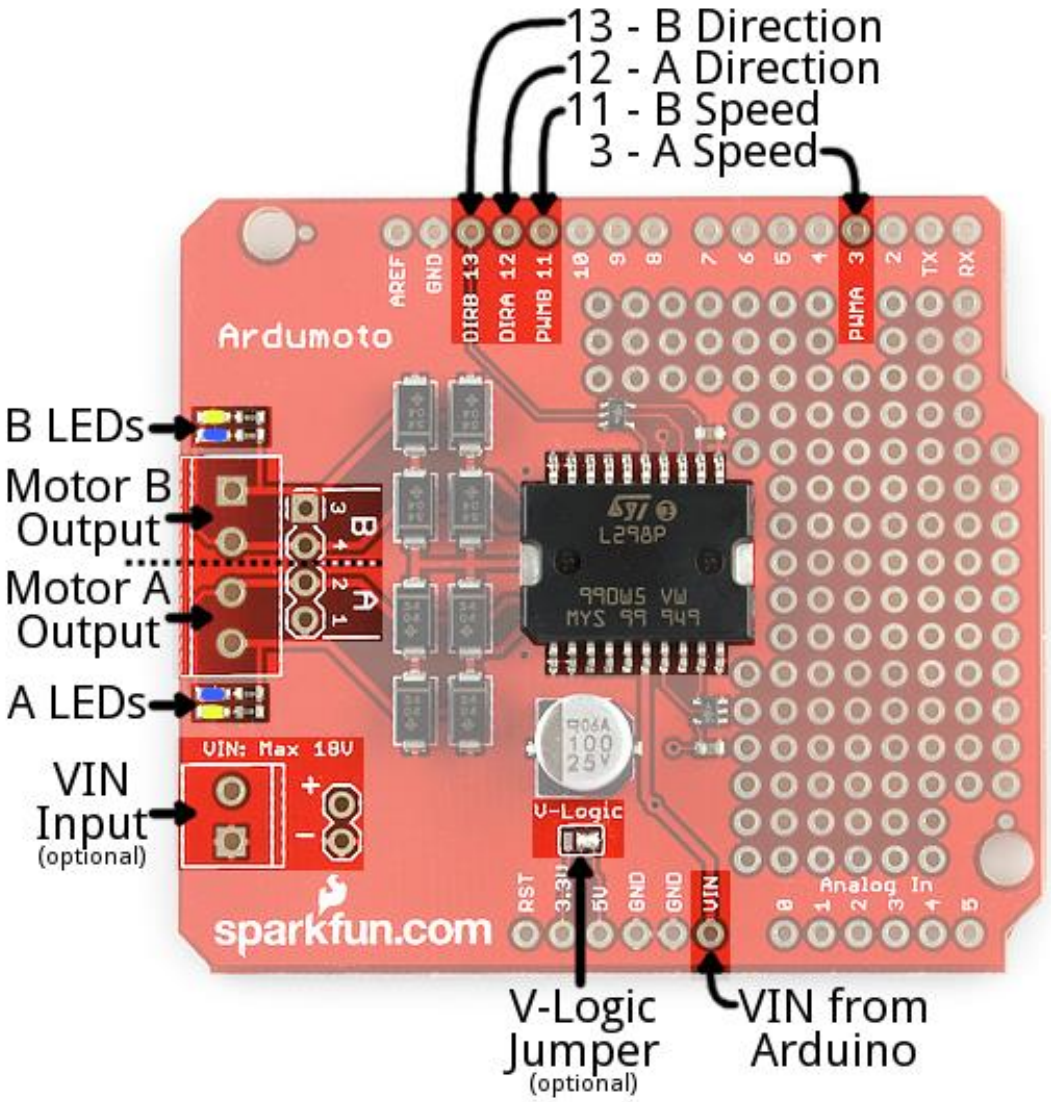


Figura 9.1_2: Pines Driver Ardumoto

Vemos que las señales PWM cuyo valor medio de tensión son las que controlan la velocidad de giro del motor, son PWMA pin 3 y PWMB pin11. Los bits lógicos de cambio de dirección son DIRA pin 12 y DIRB pin 13 y las salidas de señal de potencia son Motor A y Motor B.

El propio driver también sirve para alimentar a Arduino.

A continuación, se muestra el esquema electrónico de driver en su conjunto:

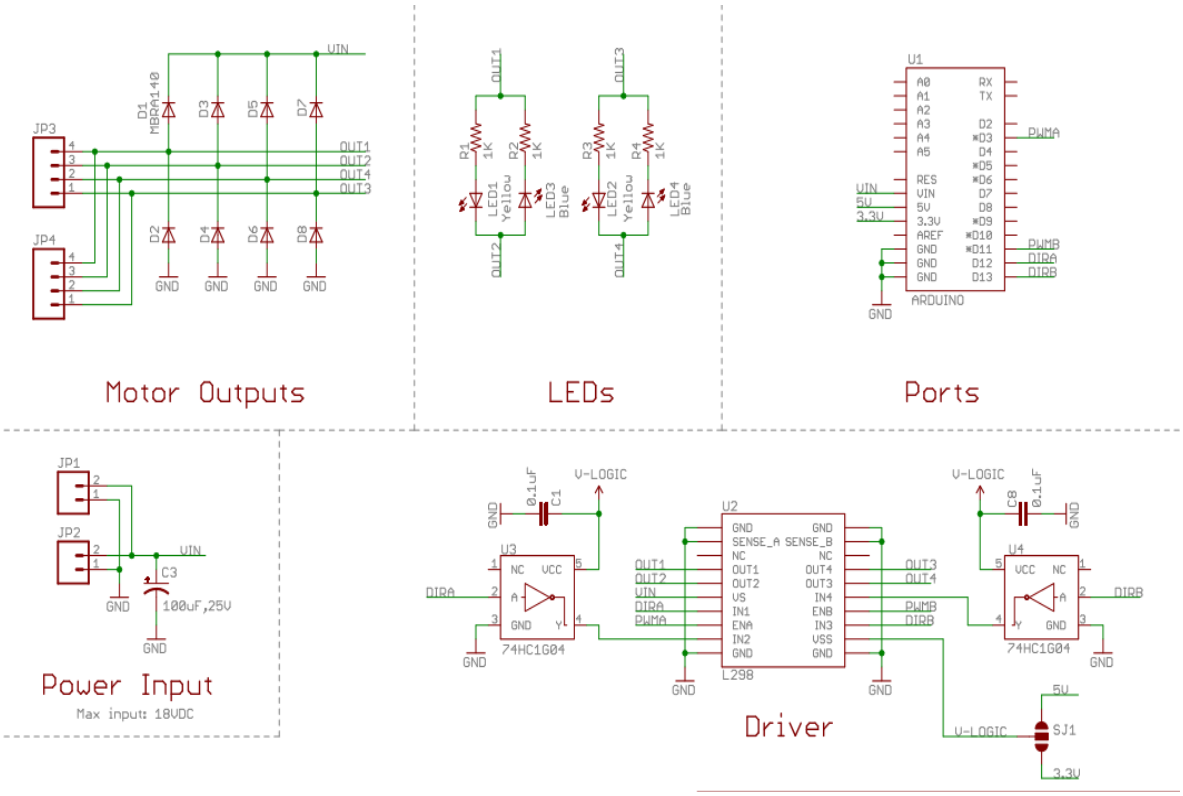


Figura 2.1_3: Esquema electrónico Driver Ardumoto

9.2.- Circuito Integrado L298

El corazón del driver Ardumoto, es el circuito Integrado L298. Este circuito integrado monobloc se fabrica en dos *versiones 15-lead Multiwatt* y *PowerSO20* (formato SMD). Acepta niveles lógicos estándar TTL y soprta cargas inductivas relés, solenoides motores DC y motores paso a paso.

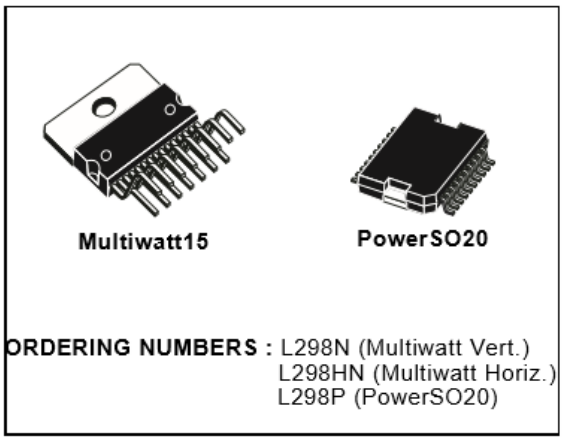


Figura 9.2_1: Formatos del chip

9.3.- Pines L298

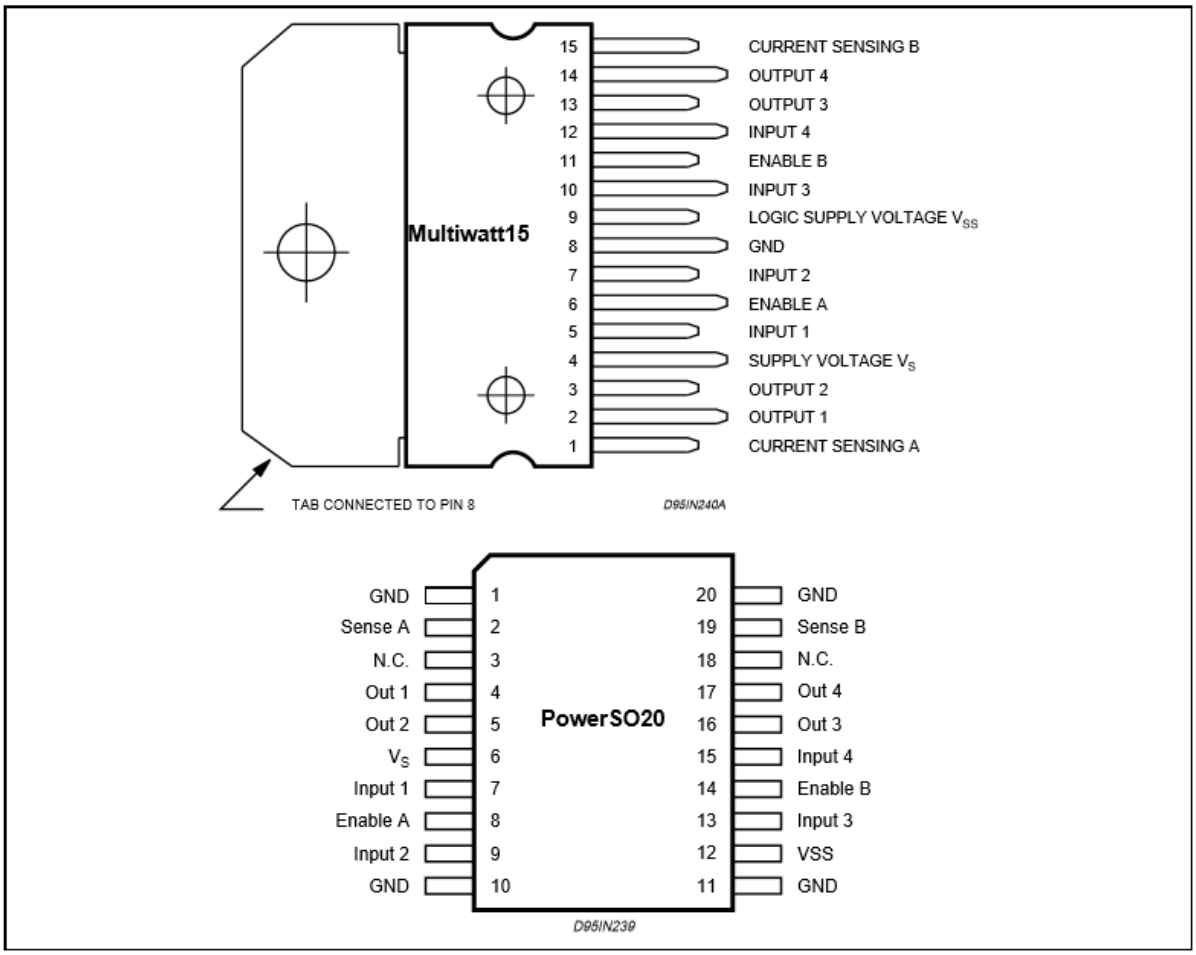


Figura 9.2_2: Pines L298

Para más información se puede consultar en:

http://www.alldatasheet.es/datasheet-pdf/pdf/22437/STMICROELECTRONICS/L298.html

10.- ANEXO 3 DIAGRAMAS DE FLUJO

10.1.- Introducción Concepto

Los diagramas de flujo son una manera de representar visualmente el flujo de datos a través de sistemas de tratamiento de información. Los diagramas de flujo describen que operaciones se realizan en un programa, la secuencia de las mismas y las decisiones lógicas tomadas para solucionar un problema dado.

Un diagrama de flujo u organigrama es una representación diagramática que ilustra la secuencia de las operaciones que se realizarán para conseguir la solución de un problema. Los diagramas de flujo se dibujan generalmente antes de comenzar a programar el código frente a la computadora. Los diagramas de flujo facilitan la comunicación entre los programadores y la gente del negocio. Estos diagramas de flujo desempeñan un papel vital en la programación de un problema y facilitan la comprensión de problemas **complicados** y sobre todo **muy largos**. Una vez que se dibuja el diagrama de flujo, llega a ser fácil escribir el programa en cualquier idioma de alto nivel. Vemos a menudo cómo los diagramas de flujo nos dan ventaja al momento de explicar el programa a otros. Por lo tanto, está correcto decir que un diagrama de flujo es una necesidad para la documentación de un programa complejo.

10.2.- Las Reglas para la creación de Flujogramas

- Los Diagramas de flujo deben escribirse de arriba hacia abajo, y/o de izquierda a derecha.
- Los símbolos se unen con líneas, las cuales tienen en la punta una flecha que indica la dirección en la que fluye la información del proceso, se deben de utilizar solamente líneas de flujo horizontal o verticales (nunca diagonales).
- Se debe evitar el cruce de líneas.
- No deben quedar líneas de flujo sin conectar
- Todo texto escrito dentro de un símbolo debe ser legible, preciso, evitando el uso de muchas palabras.
- Todos los símbolos pueden tener más de una línea de entrada, a excepción del símbolo final.
- Solo los símbolos de decisión pueden y deben tener más de una línea de flujo de salida.

Los símbolos más utilizados en los flujogramas son:

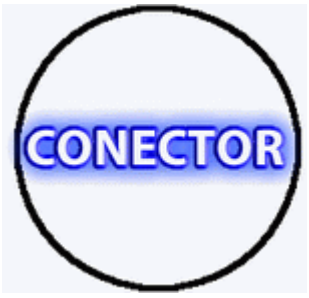
– Terminal (**rectángulo curvilíneo**). Empleado al principio y al final de las acciones del flujograma. Delimita cada bloque de programación.

– Operación **(rectángulo**). Indica la realización de una operación/acción determinada. Por ejemplo, sumar, inicializar unos registros a valor determinado, etc.

– Línea de Flujo **(línea con flecha).** Indica el camino operativo de las uniones entre las operaciones/acciones del flujograma.

– Toma de Decisión **(rombo)**. Permite la bifurcación de la secuencia ordenada de operaciones cuando se cumple una condición determinada (saltos condicionales).

– Etiqueta (**círculo**). Permite el enlace entre Líneas de Flujo de un flujograma que ocupa más de una página. Dentro del circulo debe contener un nombre, que identifica a la etiqueta y en otra página debe aparecer una etiqueta con el mismo nombre, esto indica el enlace entre dichas Líneas de Flujo en diferentes páginas.



– Rutina/Función (**rectángulo de lados dobles**). Permite la identificación de la rutina que es ejecutada en ese punto del programa. Dentro se escribe el nombre de la rutina, el cual debe coincidir con el que contenga el símbolo.

Algo muy interesante cuando se trabaja paralelamente con el flujograma y el código del lenguaje de programación que se esté utilizando, es que en el proceso de depuración se recurre al flujograma y al código de forma iterativa. Esto implica la posibilidad de darse cuenta de los errores producidos en el código e incluso la elaboración de distintas estrategias que suelen dar como resultado final un código más eficiente.

http://www.mis-algoritmos.com/aprenda-a-crear-diagramas-de-flujo
Flujogramas: Diseño Electrónico Avanzado [Master en Ingeniería Mecatrónica]

11.- ANEXO 4 PROGRAMAS

11.1.- Funcionamiento Local (Sin Cliente)

11.1.1.- Declaración de variables y librerías utilizadas (Sin Cliente)

```
#include <SPI.h>
#include <DueTimer.h>
#include <math.h>

DueTimer myTimer = DueTimer(3);//Para poder utilizar las Interrupciones Timer----------------------------------------------------------------------------------//

long t1 = 0 ;   // --------Variables del Bucle temporizado-------------------------------------------------------------------------------------------------//
long t2 = 0 ;
int T = 10 ;
int N = 10 ;
float Tm = T/1000.0; // ---Fin Variables del Bucle temporizado-------------------------------------------------------------------------------------------//

float wrd_f = 0.0;    // ---Velocidad Angular rueda derecha gennerada por algotitmo seguimiento-----------------------------------------------------------//
float wri_f = 0.0;    // ---Velocidad Angular rueda izquierda gennerada por algotitmo seguimiento----------------------------------------------------------//

const byte DirM1 = 12;     // Salida digital Arduino Pin 12, sentido de giro Ruda derecha-----------------------------------------------------------------//
const byte DirM2 = 13;     // Salida digital Arduino Pin 13, sentido de giro Ruda izquierda----------------------------------------------------------------//
int PWMA = 3 ;             //Señal PWMA  Rueda izquierda
int PWMB = 11;             //Señal PWMB  Rueda  derecha
bool cambio  = HIGH;       //Estado lógico DirM1 Pin 12
bool cambio2 = HIGH;       //Estado lógico DirM2 Pin 13

float velocidadM1 = 0.0;   //Velocidad Rueda Derecha determinada, encoders-interrupción
float velocidadM2 = 0.0;   //Velocidad Rueda Izquierda determinada, encoders-interrupción

float v = 0.0;        // velocidadM1 después de pasar por el filtro Rueda derecha
float v2= 0.0;        // velocidadM1 después de pasar por el filtro Rueda izquierda

float v_1 = 0.0; //----------------------------Velocidades filtro Rueda Derecha---------------------//
float v_2 = 0.0;
float vf = 0.0;
float vf_1 = 0.0;
float vf_2 = 0.0; //---------------------Fin---Velocidades filtro Rueda Derecha---------------------//

float v_12 = 0.0; //----------------------------Velocidades filtro Rueda Izquierda-------------------//
float v_22 = 0.0;
float vf2 = 0.0;
float vf_12 = 0.0;
float vf_22 = 0.0; //--------------------Fin---Velocidades filtro Rueda Izquierda-------------------//

float EM1 = 0.0;      //Error bucle de control Rueda izquierda
float EM2 = 0.0;      //Error bucle de control Rueda derecha

float UM1f = 0.0;     // Acción de control bucle control Rueda Izquierda
float UM2f = 0.0;     // Acción de control bucle control Rueda Derecha

float UM1fabs = 0.0;  // Acción de control bucle control Rueda Izquierda valor absoluto
float UM2fabs = 0.0;  // Acción de control bucle control Rueda Derecha valor absoluto
```

```
double KpM1 = 7.65 ;        // Constante proporcional Bucle Control Motor Rueda Izquierda
double KiM1 = 40.0 ;        // Constante Integral Bucle Control Motor Rueda Izquierda
double KpM2 = 7.65 ;        // Constante proporcional Bucle Control Motor Rueda Derecha
double KiM2 = 40.0 ;        // Constante Integral Bucle control Motor Rueda Derecha

double Ik  = 0.0 ;          // Acción Integral Bucle Control Motor Rueda Izquierda
double Ik2 = 0.0 ;          // Acción Integral Bucle Control Motor Rueda derecha

long quad = 0;                        //-------Variables Contabilización pulsos encoder Rueda Izquierda------//
long Enc_ant = 0 ;
long Enc_act = 0 ;
int Enc_dif = 0 ;                     //---Fin Variables Contabilización pulsos encoder Rueda Izquierda-----//

long quad2 = 0;                       //-------Variables Contabilización pulsos encoder Rueda Derecha-------//
long Enc_ant2 =0;
long Enc_act2 = 0;
int  Enc_dif2 = 0;                    //---Fin Variables Contabilización pulsos encoder Rueda Derecha------//

int cont = 0;                         // Contador de vueltas de bucle periodo T = 10ms
bool flag = false;                    // Bandera Maquina de Estados condición entrada estado 1

const byte R=35 ;                     //--Combinaciones Led RGB-------------------------------------------------//
const byte G=33 ;
const byte B=31 ;                     //--Fin Combinaciones Led RGB---------------------------------------------//

unsigned long time1 = 0;              // Inicialización Temporizador Máquina de estados
unsigned long intervalo = 1500;       //Intervalo Temporizador Máquina de estados

const float pi=3.1416 ;               // Constante Número Pi
//////////////////////////////////////////////////////////////////////////////////////////////////// Trayectoria Cuadrada
const byte X[]={0,0,0,0,0,0,1,2,3,4,5,5,5,5,5,5,4,3,2,1,0};
const byte Y[]={0,1,2,3,4,5,5,5,5,5,5,5,4,3,2,1,0,0,0,0,0,0};
//////////////////////////////////////////////////////////////////////////////////////////////////Fin Trayectoria Cuadrada
const float vl=0.1 ;                  // Velocidad lineal de Avance del vehículo 0.1
const float b=0.222 ;                 // Distancia entre Ruedas
const float b2=b/2.0 ;                // La mitad de la distancia entre ruedas
const float r=0.051 ;                 // Radio de la Rueda

float xc = (X[0]/10.0)*2.0;           // Inicialización coordenada x inicial
float yc = (Y[0]/10.0)*2.0;           // Inicialización coordenada y inicial

float xd= (X[1]/10.0)*2.0;            // Coordenada xd DESTINO inicial
float yd= (Y[1]/10.0)*2.0;            // Coordenada yd DESTINO inicial

float th= pi/2.0;                     //Ángulo de orientación inicial respecto al eje x sistema Fijo
float D= 0.0;                         //Distancia desde el centro de masas hasta el punto objetivo

float gm=0.0 ;                        //Inversa del Radio de curvatura
float VV=0.0 ;                        //Velocidad lineal de avance vehículo determinada por algoritmo seguimiento
float WW=0.0; ;                       //Velocidad de rotación vehículo determinada por algoritmo seguimiento

bool llegada=false ;                  // Bandera que indica fin de trayectoria
int idx=1;                            // Indice para desplazarse en los vectores coordenas x e y.
int npt=20;                           // Para determinar si hemos llegado al último elemento de los vectores x e y
```

11.1.2.- **Configuración Sistema (Sin Cliente)**

```
void setup() { //----------------------------------------------------------------------------------------------. INICIO SETUP----------------------------------------------------------------------------------------------------------//
     Serial.begin(115200);
     myTimer.attachInterrupt(handler).start(10000);
     pinMode(R,OUTPUT);
     pinMode(G,OUTPUT);
     pinMode(B,OUTPUT);

     delay(2000) ;

     attachInterrupt( 40, encoder1sA, CHANGE);
     attachInterrupt (41, encoder1sB, CHANGE);
     attachInterrupt (42, encoder2sA, CHANGE);
     attachInterrupt (43, encoder2sB, CHANGE);
     digitalWrite(R,HIGH);  digitalWrite(G,HIGH);  digitalWrite(B,HIGH);

     analogWriteResolution(8);
     pinMode(DirM1, OUTPUT );// Pin 12 pin de cambio dirección Rueda Izquiesda
     pinMode(DirM2, OUTPUT );// Pin 13 pin de cambio dirección Rueda Derecha

     pinMode(44,OUTPUT);  //-----Para indicar si se cumple el periodo de Muestreodo---//

}          //----------------------------------------------------------------------------------------------.  FIN SETUP----------------------------------------------------------------------------------------------------------//
```

11.1.3.- **Programa Principal (Sin Cliente)**

```
void loop() {//################################################## INICIO loop ##################################################//
    //Función Algoritmo de seguimiento para determinar wrd_f y wri_f consignas de velocidad angular para poder ir al punto xd e xd----------/

    if (cont>=N) { Vel_Algoritmo_Per (xc,yc,xd,yd,th);}

    vf=Velocidad_Encoder2_filtrada(velocidadM2);  //vf =  Velocidad angular Rueda Derecha filtrada-----------------------------------------------------------/
    vf2=Velocidad_Encoder1_filtrada(velocidadM1); //vf2 = Velocidad angular Rueda Derecha filtrada-----------------------------------------------------------/

    EM1 = wrd_f - vf;//-----  Error Motor Rueda Derecha.   Diferencia entre consigna y velocidad angular Real--------------------------------/
    EM2 = wri_f - vf2;//-----  Error Motor Rueda Izquierda.    Diferencia entre consigna y velocidad angular Real ----------------------------------------/

    Ik=  EM1*Tm + Ik  ;     //---------------------Motor Rueda derecha   Acción de control Integral-----------------------------------------------------/
    Ik2= EM2*Tm + Ik2 ;     //---------------------Motor Rueda izquierda Acción de control Integral-----------------------------------------------------/

    UM1f=KpM1*EM1 + KiM1*Ik  ;   //Acción De Control TOTAL Motor Rueda Derecha   =  Acción de control Proporcional + Acción de control Integral-------------/
    UM2f=KpM2*EM2 + KiM2*Ik2 ;   //Acción De Control TOTAL Motor Rueda Izquierda  = Acción de control Proporcional + Acción de control Integral-------------/

    if(UM1f >255){UM1f =255;}          //--------------------Aplicar la Saturación a la Acción de Control Motor Rueda Derecha-----------------------------/
    if( UM1f <-255) {UM1f =-255;}      //--------------------Se aplica en ambos sentidos-------------------------------------------------------------------/

    if(UM2f >255){UM2f =255;}          //--------------------Aplicar la Saturación a la Acción de Control Motor Rueda Izquierda----------------------------/
    if( UM2f <-255) {UM2f =-255;}      //--------------------Se aplica en ambos sentidos-------------------------------------------------------------------/

    if( UM1f < 0){cambio = LOW;}       //--Analizar el signo de la acción de control Motor Rueda derecha si es <  0 entonces cambio = LOW-------------------/
    if ( UM1f >= 0){cambio = HIGH;}

    if( UM2f < 0){cambio2 = LOW;}      //--Analizar el signo de la acción de control Motor Rueda izquierda si es <0 entonces cambio = LOW-------------------/
    if ( UM2f >= 0){cambio2 = HIGH;}

    if(UM1f >= 0.1*50.0 && UM1f < 55.0)        {UM1f=55.0;}    // Motor Rueda Derecha   Estrategia Software para salir de la Zona Muerta--------------------/
    if(UM1f <= -0.1*50.0 && UM1f > -55.0)      {UM1f=-55.0;}
    if(abs(UM1f)>=0.0 && abs(UM1f)<0.1*50.0)   {UM1f = 0.0;}   // Fin Estrategia Software para salir de la Zona Muerta Motor Rueda Derecha------------------/

    if(UM2f >= 0.1*50.0 && UM2f < 47.0)        {UM2f=47.0;}    // Motor Rueda Izquierda Estrategia Software para salir de la Zona Muerta--------------------/
    if(UM2f <= -0.1*50.0 && UM2f > -47.0)      {UM2f= -47.0;}
    if(abs(UM2f)>=0.0 && abs(UM2f)<0.1*50.0)   {UM2f = 0.0;}   // Fin Estrategia Software para salir de la Zona Muerta Motor Rueda Derecha------------------/

    UM1fabs=abs(UM1f) ;   // Convertir la acción de control en valor absoluto quedando en el Rango de 0 a 255 escalones del PWMB Motor Rueda Derecha---------/
    UM2fabs=abs(UM2f) ;   // Convertir la acción de control en valor absoluto quedando en el Rango de 0 a 255 escalones del PWMA Motor Rueda Izquierda-------/

    digitalWrite(DirM1,cambio2);    //Signo de la Acción de control PWMA Motor Rueda izquierda-----------------------------------------------------------/
    digitalWrite(DirM2,cambio);     //Signo de la Acción de control PWMA Motor Rueda izquierda-----------------------------------------------------------/

    analogWrite(PWMA,int(UM2fabs));  // Acción de control  PWMA + cambio2, aplicada a la Rueda izquierda-----------------------------------------------/
    analogWrite(PWMB,int(UM1fabs));  // Acción de control  PWMB + cambio,    aplicada a la Rueda derecha-----------------------------------------------/
```

```
     VV=0.5*(vf*r+vf2*r); //Determinar la velocidad lineal de avance del vehículo con vf velocidad angular medida de la rueda derecha y con vf2 velocidad-----
     angular medida rueda izquierda--------------/
     WW=(vf*r-vf2*r)/b ;  //Determinar la velocidad angular de rotación del vehículo hacia en centro de curvatura-----------------------------------------/

     if (cont>=N)                        {//##Cada N veces N=10 el periodo de Muestreo T=10ms  ejecutar el contenido entre corchetes, es decir cada 100ms-----/

          Posi_Actu_AlgoritmoPer(VV, WW);//Función  Algoritmo de Seguimiento Trayectoria que recibe los parámetros VV y WW para determinar xc, yc y th ------/
          Serial.print(xc);     Serial.print(" ");    Serial.print(yc);     Serial.print(" ");    Serial.println(D);  // Imprimir P.serie xc yc y th----------/
          cont=0 ;                       // Reinicializar vaiable cuenta periodos de bucle-------------------------------------------------------------------/

     if (D<r)       {// Si nos hacercamos al punto consigna una distanaca menor o igual a r entonces pasar a siguiente punto destino-------------------------/
          idx++;
          if (idx>=npt)    {idx=0 ;}
          if(idx==0) {llegada=true;}
          xd=(X[idx]/10.0)*2.0; yd=(Y[idx]/10.0)*2.0 ;
                              }
                                        }//##Fin cada N veces N=10 el periodo de Muestreo T=10ms  ejecutar el contenido entre corchetes, es decir cada 100ms-/
     else {     cont++ ;}                // En el caso de ser N< 10 No han pasado los 10 periodos T=10msn en total a 10*T = 100ms----------------------------
-------------------------------------------------------------------------------------------/

     if(llegada==true){llegadaPosicion();} //En el caso de llegar al último llamar a la Funcion que para el sistema-----------------------------------------
-------------------------------------------------------------------------/


     t2=micros() ;    //----------------    Bucle Temporización T = 10ms-----------------------------------------------------------------------------------/
     if (t2-t1>T*1000)     {
          digitalWrite(44,HIGH);
               }
     else {digitalWrite(44,LOW);}

     while (t2-t1<T*1000) {t2=micros() ;}
     t1=micros() ; //----------------------Fin  Bucle Temporización T = 10ms-------------------------------------------------------------------------------
}/#############################################################  FIN loop#####################################//
```

11.1.4.- **Funciones e Interrupciones (Sin Cliente)**

```
 //#############################################################################---SUBRRUTINAS DE INTERRUPCIÓN-----#############################################################################/
void handler (){

     Enc_act = quad;   //--------Lectura Velocidad a partir de los encoders Motor Rueda Izquierda-----------------------------------------------------------------#
     Enc_dif = Enc_act - Enc_ant ;
     Enc_ant = Enc_act ;
     velocidadM1 = (Enc_dif*0.017453)/(T/1000.0);


      Enc_act2 = quad2; //-------Lectura Velocidad a partir de los encoders  Motor Rueda Derecha-----------------------------------------------------------------#
      Enc_dif2 = Enc_act2 - Enc_ant2;
      Enc_ant2 = Enc_act2;
      velocidadM2 = (Enc_dif2*0.017453)/(T/1000.0);
  }
//-------------------------------------------------------------ENCODERS-----------------------------------------------------------------------------------------#
void encoder1sA (){
     if(digitalRead(40)==digitalRead(41)){ quad++;} else{quad--;}//----------------------Interrupción flaco de subida-bajada señal A encoder Motor rueda
Izquierda, para contar pulsos = quad-----------------#
}
void encoder1sB (){
     if(digitalRead(40)==digitalRead(41)){ quad--;} else{quad++;}//----------------------Interrupción flaco de subida-bajada señal B encoder Motor rueda
Izquierda, para contar pulsos = quad-----------------#
}
void encoder2sA (){
     if(digitalRead(42)==digitalRead(43)){ quad2++;} else{quad2--;}//----------------------Interrupción flaco de subida-bajada señal A encoder Motor rueda
Derecha, para contar pulsos = quad------------------#
}
void encoder2sB (){
     if(digitalRead(42)==digitalRead(43)){ quad2--;} else{quad2++;}   //----------------------Interrupción flaco de subida-bajada señal B encoder Motor rueda
Derecha, para contar pulsos = quad------------------#
}
 //############################################################################# FIN SUBRRUTINAS DE INTERRUPCIÓN-----#############################################################################/


 //#################################################################### FUNCIONES DE ALGORITMO DE SEGUMIENTO DE TRAYECTORIA ###################################################################/
//--------------------------------------- FUNCIÓN  DEL ALGORITMO SEGUIMIENTO TRAYECTORIA QUE ACTUALIZA CONSIGNAS DE WRD Y WRI ----------------------------//

void Vel_Algoritmo_Per (float xcc, float ycc, float xdd ,float ydd, float thh)   {

     D= sqrt(pow(xcc-xdd,2)+pow(ycc-ydd,2));                    //1 Determinar la distancia desde el centro de masas hasta el punto consigna-----------------#
     gm=2*((ydd-ycc)*cos(thh)-(xdd-xcc)*sin(thh))/(pow(D,2));   //2 Determinar la inversa del radio de curvatura-----------------------------------------------#

     wrd_f=(vl+vl*gm*b2)/r ;                                    //3 determinar la Wderecha   para poder alcanzar el punto destino----------------------------#
     wri_f=(vl-vl*gm*b2)/r ;                                    //4 determinar la Wizquierda para poder alcanzar el punto destino----------------------------#
                                                                                                  }
//---------------------------------------FIN FUNCIÓN  DEL ALGORITMO SEGUIMIENTO TRAYECTORIA QUE ACTUALIZA CONSIGNAS DE WRD Y WRI ------------------------//
```

```
//---------------------------------------- FUNCIÓN DEL ALGORITMO-PERSE QUE ACTUALIZA LAS POSICIONES ACTUALES Xc Yc y Th ---------------------------------/

void Posi_Actu_AlgoritmoPer(float VV, float WW){
     xc=xc+VV*N*Tm*cos(th+WW*N*Tm) ;                          //1 Determinar el nuevo punto coordenada actual del centro de masas xc
     yc=yc+VV*N*Tm*sin(th+WW*N*Tm) ;                          //2 Determinar el nuevo punto coordenada actual del centro de masas yc
     th=th+WW*N*Tm ;                                          //3 Determinar el nuevo ángulo de orientación orientación Actual
  }
//-------------------------------------------------------------FIN  FUNCIÓN DEL ALGORITMO-PERSE QUE ACTUALIZA LAS POSICIONES ACTUALES Xc Yc y Th --------------
-------------------------------------------------/

 //############################################################ FIN  FUNCIONES DE ALGORITMO DE SEGUMIENTO DE TRAYECTORIA#########################################################/


 //#################################################################      FUNCIONES FILTRADO VELOCIDAD       ###############################################################/
//-----------------------------------------------------------FUNCIÓN QUE FILTRA LA VELOCIDAD DEL ENCODER   RUEDA DERECHA-----------------------------------------//
float Velocidad_Encoder2_filtrada( float velocidadM2){//Aplicación Filtro Butterworth-------------------------------------------------------------------------#
     v =   velocidadM2;
     float vf = 0.02008*v + 0.04017*v_1 + 0.02008*v_2 + 1.561*vf_1 - 0.6414*vf_2;
     vf_2 = vf_1;
     vf_1 = vf;
     v_2 = v_1;
     v_1 = v;
  return (vf);                                      }
//-------------------------------------------------FIN  FUNCIÓN QUE FILTRA LA VELOCIDAD DEL ENCODER   RUEDA DERECHA-----------------------------------------//

//------------------------------------------------- FUNCIÓN QUE FILTRA LA VELOCIDAD DEL ENCODER1 RUEDA IZQUIERDA-----------------------------------------//
float Velocidad_Encoder1_filtrada( float velocidadM1){
     v2 =   velocidadM1;
     float vf2 = 0.02008*v2 + 0.04017*v_12 + 0.02008*v_22 + 1.561*vf_12 - 0.6414*vf_22;
     vf_22 = vf_12;
     vf_12 = vf2;
     v_22 = v_12;
     v_12 = v2;
  return (vf2);                   }
//-------------------------------------------------FIN  FUNCIÓN QUE FILTRA LA VELOCIDAD DEL ENCODER   RUEDA DERECHA-----------------------------------------//
 //############################################################ FIN       FUNCIONES FILTRADO VELOCIDAD ###############################################################################//


//-------------------------------------------------------------------------------------------------------------------FUNCIÓN LLEGADA A POSICIÓN-----------------------------------------------------------------------------------------------------------------//
void llegadaPosicion()     {

     Serial.println("llegada2"); // Esta Función se encarga de detener los motores y reinicializar el sistema en general-----------------------------------#
     analogWrite(PWMA,0);
     analogWrite(PWMB,0);
     UM2f = 0.0; v  = 0.0; v_1  = 0.0; v_2  = 0.0; vf   = 0.0; vf_1 = 0.0; vf_2  = 0.0; UM2fabs = 0.0; if(cambio==LOW){cambio = LOW;}else(cambio=HIGH);
     UM1f = 0.0; v2 = 0.0; v_12 = 0.0; v_22 = 0.0; vf2  = 0.0; vf_12= 0.0; vf_22 = 0.0; UM1fabs = 0.0; if(cambio2==LOW){cambio2 = LOW;}else(cambio2=HIGH);
     wrd_f = 0.0;
     wri_f = 0.0;

                       }
//-------------------------------------------------------------------------------------------------------------------FIN       FUNCIÓN LLEGADA A POSICIÓN-----------------------------------------------------------------------------------------------------------------//
```

11.2.- Funcionamiento Local + Servidor (Con Cliente)

11.2.1.- Declaración de variables y librerías utilizadas (Comunicación con Cliente)

```
#include <SPI.h>
#include <WiFi.h>
#include <DueTimer.h>
#include <math.h>

DueTimer myTimer = DueTimer(3);       //Objeto Timer que permte utilizar Timer Micro

char ssid[] = "dlink";                // Nombre de la Red SSID
char pass[] = "abcd1234";             // Pasword de la Red
int status = WL_IDLE_STATUS;          // Estado de la Red

WiFiServer server_dcha(5000);         // Objeto que crea el servidor que escucha puerto 5000
WiFiServer server_izda(5001);         // Objeto que crea el servidor que escucha puerto 5001
WiFiClient client_dcha;               // Objeto para ver si hay datos de entrada o salida
WiFiClient client_izda;               // Objeto para ver si hay datos de entrada o salida

typedef enum  { Est_0,Est_1,Est_2,Est_3, }estado_t;  estado_t Est = Est_3;
estado_t Est = Est_3; // Maquina de estados e inicialización de la misma

long t1 = 0 ;    // ----------------------Variables del Bucle temporizado----------------------------//
long t2 = 0 ;
int T = 10 ;
int N = 10 ;
float Tm = T/1000.0; // --------------Fin Variables del Bucle temporizado-----------------------//

float wrd_f = 0.0;                    // Velocidad Angular rueda derecha gennerada por algotitmo seguimiento
float wri_f = 0.0;                    // Velocidad Angular rueda izquierda gennerada por algotitmo seguimiento


const byte DirM1 = 12;                // Salida digital Arduino Pin 12, sentido de giro Ruda derecha
const byte DirM2 = 13;                // Salida digital Arduino Pin 13, sentido de giro Ruda izquierda
int PWMA = 3 ;                        //Señal PWMA  Rueda izquierda
int PWMB = 11;                        //Señal PWMB  Rueda  derecha
bool cambio  = HIGH;                  //Estado lógico DirM1 Pin 12
bool cambio2 = HIGH;                  //Estado lógico DirM2 Pin 13


float velocidadM1 = 0.0;              //Velocidad Rueda Derecha determinada, encoders-interrupción
float velocidadM2 = 0.0;              //Velocidad Rueda Izquierda determinada, encoders-interrupción

float v = 0.0;                        // velocidadM1 despues de pasar por el filtro Rueda  derecha
float v2= 0.0;                        // velocidadM1 despues de pasar por el filtro Rueda  izquierda

float v_1 = 0.0; //----------------------------Velocidades filtro Rueda Derecha----------------------//
float v_2 = 0.0;
float vf = 0.0;
float vf_1 = 0.0;
float vf_2 = 0.0; //----------------------Fin---Velocidades filtro Rueda Derecha----------------------//
```

```
float v_12 = 0.0; //----------------------------Velocidades filtro Rueda Izquierda-------------------//
float v_22 = 0.0;
float vf2 = 0.0;
float vf_12 = 0.0;
float vf_22 = 0.0; //--------------------Fin---Velocidades filtro Rueda Izquierda-------------------//

float EM1 = 0.0;           //Error bucle de control Rueda izquierda
float EM2 = 0.0;           //Error bucle de control Rueda derecha

float UM1f = 0.0;          // Acción de control bucle control Rueda Izquierda
float UM2f = 0.0;          // Acción de control bucle control Rueda Derecha

float UM1fabs = 0.0;       // Acción de control bucle control Rueda Izquierda valor absoluto
float UM2fabs = 0.0;       // Acción de control bucle control Rueda Derecha valor absoluto

double KpM1 = 7.65 ;       // Constante proporcional Bucle Control Motor Rueda Izquierda
double KiM1 = 40.0 ;       // Constante Integral Bucle Control Motor Rueda Izquierda
double KpM2 = 7.65 ;       // Constante proporcional Bucle Control Motor Rueda Derecha
double KiM2 = 40.0 ;       // Constante Integral Bucle control Motor Rueda Derecha

double Ik  = 0.0 ;         // Acción Integral Bucle Control Motor Rueda Izquierda
double Ik2 = 0.0 ;         // Acción Integral Bucle Control Motor Rueda derecha

long quad = 0;                      //-------Variables Contabilización pulsos encoder Rueda Izquierda------//
long Enc_ant = 0 ;
long Enc_act = 0 ;
int Enc_dif = 0 ;                   //---Fin Variables Contabilización pulsos encoder Rueda Izquierda-----//

long quad2 = 0;                     //-------Variables Contabilización pulsos encoder Rueda Derecha-------//
long Enc_ant2 =0;
long Enc_act2 = 0;
int  Enc_dif2 = 0;                  //---Fin Variables Contabilización pulsos encoder Rueda Derecha------//

int cont = 0;                       // Contador de vueltas de bucle perido T = 10ms
bool flag = false;                  // Badera Maquina de Estados condición esntrada estado 1

const byte R=35 ;                   //--Combinaciones Led RGB-------------------------------------------------//
const byte G=33 ;
const byte B=31 ;                   //--Fin Combinaciones Led RGB---------------------------------------------//

unsigned long time1 = 0;            // Inicialización Temporizador Máquina de estados
unsigned long intervalo = 1500;     //Intervalo Temporizador Máquina de estados

const float pi=3.1416 ;             // Constante Número Pi

////////////////////////////////////////////////////////////////////////////////////////////////////// Trayectori Cuadrada
const byte X[]={0,0,0,0,0,0,1,2,3,4,5,5,5,5,5,5,4,3,2,1,0};
const byte Y[]={0,1,2,3,4,5,5,5,5,5,5,4,3,2,1,0,0,0,0,0,0};
//////////////////////////////////////////////////////////////////////////////////////////////////////Fin Trayectoria Cuadrada
const float vl=0.1 ;                    // Velocidad lineal de Avance del vehículo 0.1
const float b=0.222 ;                   // Distancia entre Ruedas
const float b2=b/2.0 ;                  // La mitad de la distancia entre ruedas
const float r=0.051 ;                   // Radio de la Rueda
float xc = (X[0]/10.0)*2.0;             // Inicialización coordenada x inicial
```

```
float yc = (Y[0]/10.0)*2.0;              // Inicialización coordenada y inicial
float xd= (X[1]/10.0)*2.0;               // Coordenada xd DESTINO inicial
float yd= (Y[1]/10.0)*2.0;               // Coordenada yd DESTINO inicial

float th= pi/2.0;                        //Ángulo de orientación inicial respecto al eje x sistema Fijo
float D= 0.0;                            //Distancia desde el centro de masas hasta el punto  objetivo

float gm=0.0 ;                           //Inversa del Radio de curvatura
float VV=0.0 ;                           //Velocidad lineal de avance vehículo determinada por algoritmo seguimiento
float WW=0.0; ;                          //Velocidad de rotación vehículo determinada por algoritmo seguimiento

byte xc_byte ;                           // Byte que contiene la coordenada x actual para enviar al cliente
byte yc_byte ;                           // Byte que contiene la coordenada y actual para enviar al cliente

bool llegada=false ;                     // Bandera que indica fin de trayectoria
int idx=1;                               // Indice para desplazarse en los vectores coordenasa x e y.
int npt=20;                              // Para determinar si hemos llegado al último elemento de los vectores x e y
```

11.2.2.- **Configuración Sistema (Con Cliente)**

```
void setup() { //----------------------------------------------------------------------------------------------------. INICIO SETUP----------------------------------------------------------------------------------------------------//
      Serial.begin(115200);
      myTimer.attachInterrupt(handler).start(10000);
      pinMode(R,OUTPUT);
      pinMode(G,OUTPUT);
      pinMode(B,OUTPUT);

      delay(2000) ;

      attachInterrupt( 40, encoder1sA, CHANGE);
      attachInterrupt (41, encoder1sB, CHANGE);
      attachInterrupt (42, encoder2sA, CHANGE);
      attachInterrupt (43, encoder2sB, CHANGE);
      digitalWrite(R,HIGH);  digitalWrite(G,HIGH);  digitalWrite(B,HIGH);

      analogWriteResolution(8);
      pinMode(DirM1, OUTPUT );// Pin 12 pin de cambio dirección Rueda Izquiesda
      pinMode(DirM2, OUTPUT );// Pin 13 pin de cambio dirección  Rueda Derecha

      pinMode(44,OUTPUT);  //Para indicar si se cumple el periodo de Muestreodo--------//

      while(WiFi.status() == WL_NO_SHIELD) {
      Serial.println("Buscando presencia SHIELD");
                                  }

 while ( status != WL_CONNECTED) {
      Serial.print("Intentando Conectar WPA SSID: ");
      Serial.println(ssid);
      status = WiFi.begin(ssid, pass);
      delay(10000);
                                  }

 server_dcha.begin();
 server_izda.begin();
 Serial.print("Estamos conectados con : ");
 printCurrentNet();                        // Imprime la Red a la que se conecta
 printWifiData();                           // Imprime la IP asignada por la red

} //----------------------------------------------------------------------------------------------------. FIN SETUP----------------------------------------------------------------------------------------------------//
```

11.2.3.- **Programa Principal (Con Cliente)**

void loop() {//## INICIO loop ##//

```
HayClientes();// Función que comprueva la existencia de cliente
//Función Algoritmo de seguimiento para determinar wrd_f y wri_f consignas de velocidad angular para poder ir al punto xd e xd-------------------------/
if (cont>=N) { Vel_Algoritmo_Per (xc,yc,xd,yd,th);}

vf  =  Velocidad_Encoder2_filtrada(velocidadM2); //vf =   Velocidad angular Rueda Derecha filtrada-----------------------------------------------------/
vf2 = Velocidad_Encoder1_filtrada(velocidadM1);  //vf2 = Velocidad angular Rueda Derecha filtrada-----------------------------------------------------/

EM1 = wrd_f - vf; //---------Error Motor Rueda Derecha.    Diferencia entre consigna y velocidad angular Real ----------------------------------------/
EM2 = wri_f - vf2;//---------Error Motor Rueda Izquierda.  Diferencia entre consigna y velocidad angular Real ----------------------------------------/

Ik=  EM1*Tm + Ik  ;        //--------------------------------------Motor Rueda derecha   Acción de control Integral-------------------------------------/
Ik2= EM2*Tm + Ik2 ;        //--------------------------------------Motor Rueda izquierda Acción de control Integral-------------------------------------/

UM1f=KpM1*EM1 + KiM1*Ik  ;   //Acción De Control TOTAL Motor Rueda Derecha =  Acción de control Proporcional + Acción de control Integral--------------/
UM2f=KpM2*EM2 + KiM2*Ik2 ;   //Acción De Control TOTAL  Acción de control Proporcional + Acción de control Integral-------------------------------------#

if(UM1f >255){UM1f =255;}          //--------------------Aplicar la Saturación a la Acción de Control Motor Rueda Derecha-----------------------------/
if( UM1f <-255) {UM1f =-255;}      //--------------------Se aplica en ambos sentidos-----------------------------------------------------------------/

if(UM2f >255){UM2f =255;}          //--------------------Aplicar la Saturación a la Acción de Control Motor Rueda Izquierda--------------------------/
if( UM2f <-255) {UM2f =-255;}      //--------------------------------------Se aplica en ambos sentidos-----------------------------------------------/

if( UM1f < 0){cambio = LOW;}       //----------Analizar el signo de la acción de control Motor Rueda derecha si es <  0 entonces cambio = LOW----------/
if ( UM1f >= 0){cambio = HIGH;}

if( UM2f < 0){cambio2 = LOW;}      //----------Analizar el signo de la acción de control Motor Rueda izquierda si es <0 entonces cambio = LOW----------/
if ( UM2f >= 0){cambio2 = HIGH;}

if(UM1f >= 0.1*50.0 && UM1f < 55.0)        {UM1f=55.0;}          // Motor Rueda Derecha   Estrategia Software para salir de la Zona Muerta-------------/
if(UM1f <= -0.1*50.0 && UM1f > -55.0)      {UM1f=-55.0;}
if(abs(UM1f)>=0.0 && abs(UM1f)<0.1*50.0)   {UM1f = 0.0;}         // Fin Estrategia Software para salir de la Zona Muerta Motor Rueda Derecha------------/

if(UM2f >= 0.1*50.0 && UM2f < 47.0)             {UM2f=47.0;}          // Motor Rueda Izquierda Estrategia Software para salir de la Zona Muerta---------/
if(UM2f <= -0.1*50.0 && UM2f > -47.0)           {UM2f= -47.0;}
if(abs(UM2f)>=0.0 && abs(UM2f)<0.1*50.0)        {UM2f = 0.0;}         // Fin Estrategia Software para salir de la Zona Muerta Motor Rueda Derecha-------/

UM1fabs=abs(UM1f) ;   // Convertir la acción de control en valor absoluto quedando en el Rango de 0 a 255 escalones del PWMB Motor Rueda Derecha---------/
UM2fabs=abs(UM2f) ;   // Convertir la acción de control en valor absoluto quedando en el Rango de 0 a 255 escalones del PWMA Motor Rueda Izquierda-------/
digitalWrite(DirM1,cambio2);    //Signo de la Acción de control PWMA Motor Rueda izquierda-----------------------------------------------------------/
digitalWrite(DirM2,cambio);     //Signo de la Acción de control PWMA Motor Rueda izquierda-----------------------------------------------------------/


analogWrite(PWMA,int(UM2fabs));  // Acción de control  PWMA + cambio2, aplicada a la Rueda izquierda-----------------------------------------------/
analogWrite(PWMB,int(UM1fabs));  // Acción de control  PWMB + cambio,  aplicada a la Rueda derecha-------------------------------------------------/
```

```
     VV=0.5*(vf*r+vf2*r); //Determinar la velocidad lineal de avance del vehículo con vf velocidad angular medida de la rueda derecha y con vf2 velocidad
     angular medida rueda izquierda--------------/
     WW=(vf*r-vf2*r)/b ;   //Determinar la velocidad angular de rotación del vehículo hacia en centro de curvatura-----------------------------------------/

     if (cont>=N)                        {//###Cada N veces N=10 el periodo de Muestreo T=10ms  ejecutar el contenido entre corchetes, es decir cada 100ms----/

          Posi_Actu_AlgoritmoPer(VV, WW);  //Función  Algoritmo de Seguimiento Trayectoria que recibe los parámetros VV y WW para determinar xc, yc y th ----/
          Serial.print(xc);     Serial.print(" ");    Serial.print(yc);     Serial.print(" ");    Serial.println(D);  // Imprimir P.serie xc, yc y D ----------/
          cont=0 ;                    // Reinicializar vaiable cuenta periodos de bucle-------------------------------------------------------------------------/

     EscribirAlCliente();             // Función que mana al cliente en forma de bytes xc e yc actuales-----------------------------------------------------------/

     if (D<r)      {// Si nos hacercamos al punto consigna una distanaca menor o igual a r entonces pasar a siguiente punto destino-------------------------/
          idx++;
          if (idx>=npt)    {idx=0 ;}
          if(idx==0) {llegada=true;}
          xd=(X[idx]/10.0)*2.0; yd=(Y[idx]/10.0)*2.0 ;
                              }
                                   }//##Fin cada N veces N=10 el periodo de Muestreo T=10ms  ejecutar el contenido entre corchetes, es decir cada 100ms--/

     else {     cont++ ;} // En el caso de ser N< 10 No han pasado los 10 periodos T=10msn en total a 10*T = 100ms-----------------------------------------
-------------------------------------------------------------------------/

     if(llegada==true){llegadaPosicion();} //En el caso de llegar al último llamar a la Función que parar el Sistema-----------------------------------/


t2=micros() ;    //--------------------Bucle Temporización T = 10ms-----------------------------------------------------------------------------------------/

if (t2-t1>T*1000)     {

     digitalWrite(44,HIGH);

                }
else {digitalWrite(44,LOW);}

while (t2-t1<T*1000) {t2=micros() ;}

 t1=micros() ; //-------------------Fin  Bucle Temporización T = 10ms-----------------------------------------------------------------------------------/
                    }/########################################## FIN loop#####################################//
```

11.2.4.- **Funciones e Interrupciones (Con Cliente)**

```
 //#############################################################################---SUBRRUTINAS DE INTERRUPCIÓN-#############################################################/
void handler (){

     Enc_act = quad;   //--------------Lectura Velocidad a partir de los encoders Motor Rueda Izquierda-----------------------------------------------------------#
     Enc_dif = Enc_act - Enc_ant ;
     Enc_ant = Enc_act ;
     velocidadM1 = (Enc_dif*0.017453)/(T/1000.0);


      Enc_act2 = quad2; //------------Lectura Velocidad a partir de los encoders  Motor Rueda Derecha-----------------------------------------------------------#
      Enc_dif2 = Enc_act2 - Enc_ant2;
      Enc_ant2 = Enc_act2;
      velocidadM2 = (Enc_dif2*0.017453)/(T/1000.0);
  }
//-------------------------------------------------------------------------------ENCODERS-------------------------------------------------------------------------/
void encoder1sA (){
     if(digitalRead(40)==digitalRead(41)){ quad++;} else{quad--;}//----------------------Interrupción flaco de subida-bajada señal A encoder Motor rueda
Izquierda, para contar pulsos = quad-----------------#
}
void encoder1sB (){
     if(digitalRead(40)==digitalRead(41)){ quad--;} else{quad++;} //----------------------Interrupción flaco de subida-bajada señal B encoder Motor rueda
Izquierda, para contar pulsos = quad-----------------#
}
void encoder2sA (){
     if(digitalRead(42)==digitalRead(43)){ quad2++;} else{quad2--;}//----------------------Interrupción flaco de subida-bajada señal A encoder Motor rueda
Derecha, para contar pulsos = quad------------------#
}
void encoder2sB (){
     if(digitalRead(42)==digitalRead(43)){ quad2--;} else{quad2++;}//----------------------Interrupción flaco de subida-bajada señal B encoder Motor rueda
Derecha, para contar pulsos = quad------------------#
}
 //############################################################################# FIN SUBRRUTINAS DE INTERRUPCIÓN---#############################################################/


 //############################################################### FUNCIONES DE ALGORITMO DE SEGUMIENTO DE TRAYECTORIA ###########################################################/
//------------------------------------------- FUNCIÓN  DEL ALGORITMO SEGUIMIENTO TRAYECTORIA QUE ACTUALIZA CONSIGNAS DE WRD Y WRI -------------------------//

void Vel_Algoritmo_Per (float xcc, float ycc, float xdd ,float ydd, float thh)   {

     D= sqrt(pow(xcc-xdd,2)+pow(ycc-ydd,2));                                  //1 Determinar la distancia desde el centro de masas hasta el punto consigna-------
-----#
     gm=2*((ydd-ycc)*cos(thh)-(xdd-xcc)*sin(thh))/(pow(D,2));         //2 Determinar la inversa del radio de curvatura-----------------------------------------#

     wrd_f=(vl+vl*gm*b2)/r ;                                          //3 determinar la Wderecha        para poder alcanzar el punto destino------------------#
     wri_f=(vl-vl*gm*b2)/r ;                                          //4 determinar la Wizquierda      para poder alcanzar el punto destino------------------#
                                                                                                    }
//---------------------------------------FIN FUNCIÓN  DEL ALGORITMO SEGUIMIENTO TRAYECTORIA QUE ACTUALIZA CONSIGNAS DE WRD Y WRI -------------------------//
```

```
//---------------------------------------- FUNCIÓN DEL ALGORITMO-PERSE QUE ACTUALIZA LAS POSICIONES ACTUALES Xc Yc y Th ---------------------------------/

void Posi_Actu_AlgoritmoPer(float VV, float WW){
     xc=xc+VV*N*Tm*cos(th+WW*N*Tm) ;                          //1 Determinar el nuevo punto coordenada actual del centro de masas xc
     yc=yc+VV*N*Tm*sin(th+WW*N*Tm) ;                          //2 Determinar el nuevo punto coordenada actual del centro de masas yc
     th=th+WW*N*Tm ;                                          //3 Determinar el nuevo ángulo de orientación orientación Actual
  }
//-------------------------------------FIN  FUNCIÓN DEL ALGORITMO-PERSE QUE ACTUALIZA LAS POSICIONES ACTUALES Xc Yc y Th ---------------------------------/
 //############################################################ FIN  FUNCIONES DE ALGORITMO DE SEGUMIENTO DE TRAYECTORIA ###################################################################/

 //#################################################################       FUNCIONES FILTRADO VELOCIDAD#########################################################################################/
//-----------------------------------------FUNCIÓN QUE FILTRA LA VELOCIDAD DEL ENCODER   RUEDA DERECHA-----------------------------------------------------//
float Velocidad_Encoder2_filtrada( float velocidadM2){//Aplicación Filtro Butterworth-----------------------------------------------------------------------#
     v =   velocidadM2;
     float vf = 0.02008*v + 0.04017*v_1 + 0.02008*v_2 + 1.561*vf_1 - 0.6414*vf_2;
     vf_2 = vf_1;
     vf_1 = vf;
     v_2 = v_1;
     v_1 = v;
  return (vf);                                        }
//-----------------------------------------FIN  FUNCIÓN QUE FILTRA LA VELOCIDAD DEL ENCODER   RUEDA DERECHA-----------------------------------------------//

//-----------------------------------------FUNCIÓN QUE FILTRA LA VELOCIDAD DEL ENCODER1 RUEDA IZQUIERDA-----------------------------------------------//
float Velocidad_Encoder1_filtrada( float velocidadM1){
     v2 =   velocidadM1;
     float vf2 = 0.02008*v2 + 0.04017*v_12 + 0.02008*v_22 + 1.561*vf_12 - 0.6414*vf_22;
     vf_22 = vf_12;
     vf_12 = vf2;
     v_22 = v_12;
     v_12 = v2;
  return (vf2);                   }
//-------------------------------------FIN  FUNCIÓN QUE FILTRA LA VELOCIDAD DEL ENCODER   RUEDA DERECHA-----------------------------------------------//
 //############################################################ FIN       FUNCIONES FILTRADO VELOCIDAD ###########################################################################################/


//-----------------------------------------FUNCIÓN LLEGADA A POSICIÓN----------------------´-------------------------------------------------------------
//
void llegadaPosicion()     {

     Serial.println("llegada2"); // Esta Función se encarga de detener los motores y reinicializar el sistema en general-----------------------------------#
     analogWrite(PWMA,0);
     analogWrite(PWMB,0);
     UM2f = 0.0; v  = 0.0; v_1  = 0.0; v_2  = 0.0; vf   = 0.0; vf_1 = 0.0; vf_2  = 0.0; UM2fabs = 0.0; if(cambio==LOW){cambio = LOW;}else(cambio=HIGH);
     UM1f = 0.0; v2 = 0.0; v_12 = 0.0; v_22 = 0.0; vf2  = 0.0; vf_12= 0.0; vf_22 = 0.0; UM1fabs = 0.0; if(cambio2==LOW){cambio2 = LOW;}else(cambio2=HIGH);
     wrd_f = 0.0;
     wri_f = 0.0;

                          }
//-----------------------------------FIN  FUNCIÓN LLEGADA A POSICIÓN-----------------------------------------------------------------------------------//
```

```
//---------------------------------------FUNCIÓN ESCRITURA AL CLIENTE-----------------------------------------------------------------------------------//
void EscribirAlCliente()   {

     xc_byte=(byte)(((xc+2.0)/4.0)*255) ;
     yc_byte=(byte)(((yc+2.0)/4.0)*255) ;
     client_dcha.write(xc_byte) ;
     client_izda.write(yc_byte) ;

                                }
//-------------------------------FIN  FUNCIÓN ESCRUTURA AL CLIENTE-----------------------------------------------------------------------------------//

 //#####################################################FUNCIONES TESTEAR SI HAY CLIENTES  Y MAQUINA DE ESTADOS#########################################################/
void HayClientes(){
     while(!client_dcha.connected() || !client_izda.connected()){ //Mientras No hay datos de entrada, No cliente, crear objeto cliente y testear  clientes --/
          client_dcha = server_dcha.available();
          client_izda = server_izda.available();
          if(client_dcha || client_izda)             {     // Si al estar escuchando hay clientes entrantes--------------Entonces:--------------------------/
                    Serial.println("Nuevo cliente");      // Imprimir por el puerto serie "Nuevo Cliente"-----------------------------------------------/
                    digitalWrite(R,LOW); digitalWrite(G,HIGH); digitalWrite(B,LOW); Est = Est_3;  // Led indicador de cliente entrante pasa de Rojo a Verde-/
                              }
           else {
                    Serial.println("Esperando cliente"); // En el caso contrario Imprimir por el puerto serie Esperando Cliente----------------------------/
                    digitalWrite(R,HIGH); digitalWrite(G,LOW); digitalWrite(B,LOW); //El Led indicador de cliente entrante permanece en Rojo----------------/
                    //-----------------------------------------------------EMPIEZA MÁQUINA DE ESTADOS-----------------------------------------------------------/
                    // ESTADO_0 SI FLAG ES TRUE
 if(  ( (Est == Est_3) || (Est == Est_2 && (flag==true)) ||  (Est == Est_0 && (Est!=Est_1)) ) && (!client_dcha.connected() && !client_izda.connected() )    )
     {
                          Est=Est_0; time1 = millis(); flag=false;  Serial.println("ESTADO 1");
                                                                                                                                                  }
                    // ESTADO_1
if(  (  (Est == Est_0 && (flag==false)) || (Est == Est_1&& (Est!=Est_2)) ) && (!client_dcha.connected() && !client_izda.connected() )    )
                          {Est=Est_1; Serial.println("ESTADO 2vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv");        }

                    // ESTADO_2
if(  (  (Est == Est_1&&(1)) || (Est == Est_2&& (Est!=Est_0)) ) && (!client_dcha.connected() && !client_izda.connected() )    )
                          {Est=Est_2;
                          Serial.println("ESTADO 3");
                          if (millis()- time1 >=intervalo)                    {
                               digitalWrite(DirM1,HIGH);
                               analogWrite(PWMA,0);
                               digitalWrite(DirM2,HIGH);
                               analogWrite(PWMB,0);
                               client_dcha.flush();
                               client_izda.flush();
                               client_dcha.stop();
                               client_izda.stop();
                               Serial.println("ESTADO 3333");  flag=true;
                               client_dcha = server_dcha.available();
                               client_izda = server_izda.available();          }
                                     }
                                                                                                                                                                  }
               }
//################################################################## FIN       FUNCIONES TESTEAR SI HAY CLIENTES  Y MAQUINA DE ESTADOS ########################################/
```

12.- ANEXO 5 PROGRAMACIÓN EN ANDROID

12.1.- Introducción Conceptos

Esta introducción no pretende ser un curso de Android, si no que pretende comentar las ideas básicas de la programación en Android, para poder entender la aplicación cliente Android de este proyecto, los métodos utilizados y el concepto de hilos de programación que ha sido necesario poner en práctica.

Podemos resumir que en programación Android, toda aplicación está constituida por tres partes claramente diferenciadas, por una parte está el ***Layout.xml*** (vista gráfica), por otro lado la ***Activity(.java)*** (métodos-código lógico) y finalmente un fichero denominado ***AndroidManifest.xml***, que también es tipo Layout.

La finalidad del fichero ***AndroidManifest.xml*** es declarar una serie de metadatos de la aplicación que el dispositivo debe conocer antes de instalar la aplicación. En él se indican el nombre del paquete en el que por defecto se buscarán las actividades que declaremos en nuestra aplicación. El nombre de la aplicación, las actividades que conforman la aplicación y cuál es la principal, el ícono de nuestra aplicación entre otros, y sobre todo los permisos que serán concedidos de manera explícita por el usuario, si da su consentimiento para instalar la aplicación.

Al crear un nuevo proyecto por una parte creamos una *Activity* ***(***Actividad).java, que es la que contendrá todos los métodos de la aplicación principal, es decir toda la parte lógica de la programación y donde están contenidos los distintos hilos de programa. El único método que se sobrescribe (que se crea por defecto) de la clase padre es **onCreate**.

Y lo primero que se hace es llamar al **onCreate** de la clase *Activity.* Esto se hace así porque cuando una actividad se crea, se puede querer crear con información que tenía previamente. Esta información se recibe en el objeto **savedInstanceState**. Imaginemos por ejemplo que cambiamos la posición del móvil a apaisado mientras rellenamos un formulario. Este cambio provoca que la actividad se creé de nuevo, y lo más usable, es que los campos que ya haya rellenado el usuario sigan rellenos con la misma información, y para esto, es necesario este método.

A continuación se realiza algo importante que es vincular la parte digamos del código *Activity*.java con los objetos gráficos que contendrá el Layout.xml(vista gráfica) de nuestraaplicación. Esto se realiza mediante el método **setContentView**. Este método solo se puede llamar desde **onCreate**, y además es un campo immutable. Solo se puede fijar la vista de la actividad una sola vez. Es la forma que tiene **Android** de obligarnos a tener una única vista por actividad.

Acontinuación se muestra el código explicado:

```
package com.autentia.android;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;

public class FirstActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
    }
}
```

Por otro lado tenemos la vista (layout.xml) que es donde se implementa toda la parte gráfica botones, ventanas, sliders etc. Como se comentó anteriormente la *Activity* y el *layout* quedan vinculados mediante el método **setContentView,** que llamamos desde **onCreate.**

A partir de aquí empieza el ciclo de vida de la *Activity.java.* El sistema llama al constructor de la actividad mientras que también inicia la aplicación si es necesario y llama a los siguientes métodos por orden:

- onCreate
- onStart
- onResume

Hasta este momento lo que ha ocurrido es que hemos dada vida a la actividad. Pero en cualquier momento podemos minimizar, dejar la aplicación en segundo plano porque queremos ejecutar otra actividad, y entonces empieza la segunda parte del ciclo de vida de una actividad. Esto ocurre cuando por ejemplo el usuario pulsa el botón de retorno desde la actividad. Los métodos que son llamados y por el siguiente orden son:

- onPause
- onStop

- onDestroy

Ejemplos de acciones que destruirán una actividad:

- Cambio del móvil a apaisado o viceversa
- La actividad ya no se ve en la pantalla, o el sistema está bajo de recursos
- El usuario presiona el botón de retroceso o de Home y sale de la aplicación

Después de esto la aplicación es cerrada.

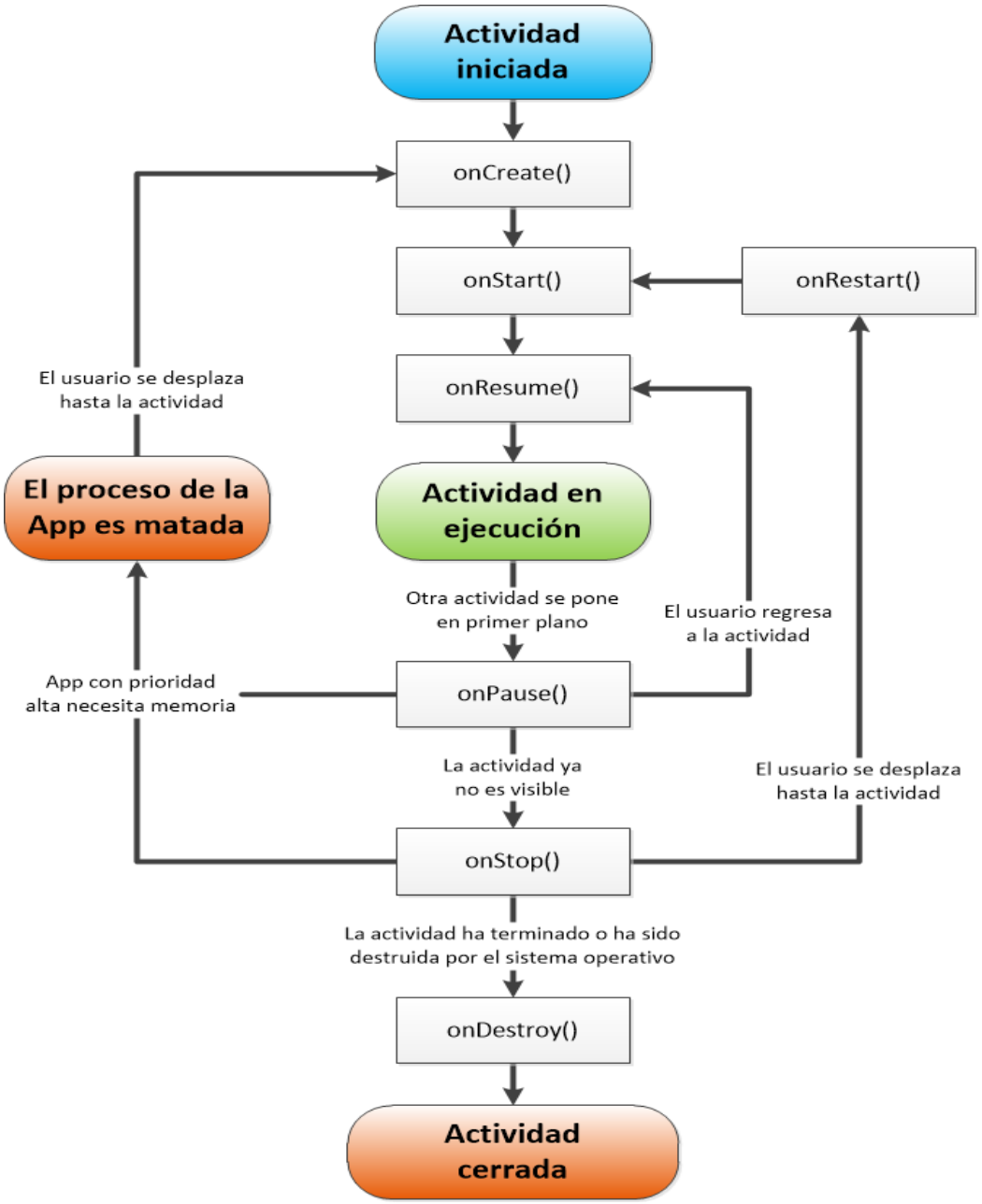


Figura 12.1_1: Ciclo de vida de una Actividad oficial de Android

A continuación, se pasa a describir cada uno de estos métodos del ciclo de vida para posteriormente comentar y justificar los métodos especiales utilizados en la aplicación Cliente Android de este proyecto.

onStart:

Se llama immediatamente después del onCreate. Si nuestra aplicación estaba en segundo plano, **onStart** será llamado cuando la aplicación vuelva a estar en primer plano.

onResume:

Es el último método que se llama antes de que la actividad tenga acceso a la pantalla. Si algún elemento del interfaz gráfico ha cambiado mientras la actividad estaba en segundo plano este método es el sitio para asegurar que el estado está sincronizado. No importa de que estado venga, cuando la actividad vuelva a estar en primer plano este método será llamado.

onPause

Es el primer método que se llama cuando la aplicación se está yendo de la pantalla. Si tenemos bucles, procesos, animaciones que deberían estar corriendo cuando la actividad está en pantalla este método es el idóneo para pararlos. Este método también se llama cuando lanzamos otra actividad desde la que se está ejecutando actualmente. Este método es importante porque puede ser el único en avisarnos de que la actividad o incluso toda la aplicación se está cerrando. En este método deberíamos guardar cualquier información importante a disco, base de datos o preferencias**.**

onStop

Cuando se llama a onStop lo que sabemos es que la actividad está **oficialmente fuera de pantalla**. No siginifica que la actividad se esté apagando, aunque podría ser. Solo se puede asumir que el usuario ha dejado tu actividad por otra. Si estás haciendo algún proceso que solo debería estar corriendo cuando la actividad está en ejecución este es un buen momento para pararla.

onDestroy

Es el último método que se llama antes del final. Es la última oportunidad para limpiar lo necesario antes de que el propio sistema la elimine por completo. Cualquier proceso de *background* que la actividad puede taner corriendo debe pararse. Sin embargo porque este método se haya llamado no significa que la actividad sea borrada. Si tienes algún hilo corriendo, este puede seguir corriendo y consumiendo recursos incluso aunque este método se llame.

12.2.- Metodos especiales utilizados en la Aplicación Cliente Android

runOnUiThread ():

Este método arranca lo que se conoce un hilo secundario de programa cuando se la llama desde el hilo principal de programa, que permite no bloquear a la aplicación principal, pues de ello se encarga el planificador de tareas del sistema operativo. Fundamentalmente este método está pensado para actuar sobre una vista, es decir para cambiar o actualizar un valor de una etiqueta de texto, o como en nuestro caso, ha servido para cada 2 segundos comprobar si hay red, para posteriormente actuar sobre la interfaz gráfica cambiando si hay red, el semáforo Red en verde, y en caso contrario en rojo.

Existe otros métodos similares como **Handler.post(Runnable r)**, que sirven para lo mismo, pero en este caso con la diferencia de que con este método, podemos acceder a los componentes o variables de otros *tread* y no solo al del hilo principal.

Para que un hilo secundario no detenga el hilo principal de programa, el camino correcto a seguir, es renderizar la interfaz de la aplicación y al mismo tiempo ejecutar en segundo plano la otra actividad para continuar con la armonía de la aplicación y evitar paradas inesperadas.

Es aquí donde entran en juego los hilos, porque son los únicos que tienen la habilidad especial de permitir al programador generar concurrencia en sus aplicaciones y la sensación de multitareas ante el usuario. Por eso la programación en Android está muy ligada al concepto de multi-hilo, y existen muchos métodos que permiten ejecutar hilos con propósitos diferentes que se adaptan a las exigencias de la tarea demandad con al hilo.

A continuación se ilustra esta idea con la Figura 12.2_1:

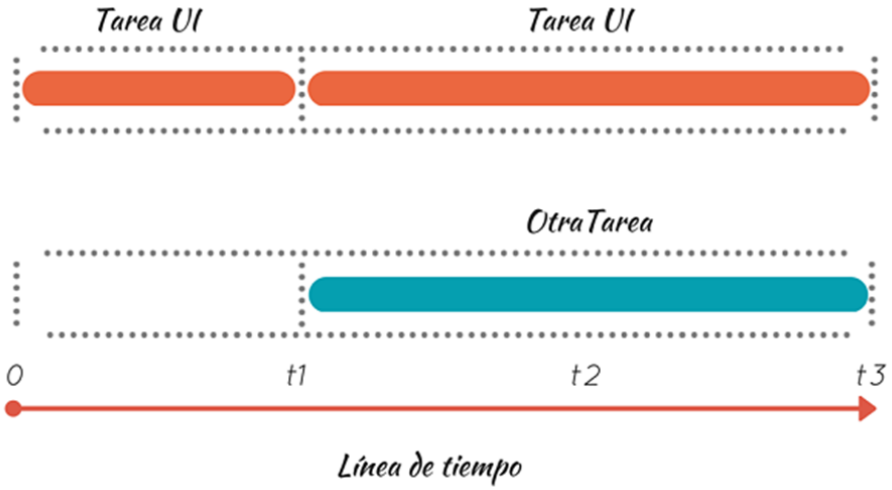


Figura 12.2_1: Hilos y Concurrencia

AsyncTask :

Función o metodo en nuestro programa:

```
private class MySocket extends AsyncTask<String, Integer, int []>
```

Cuando existe la posibilidad de lanzar una tarea bloqueante y largar, como es el caso de lanzar el socket que permite establecer la comunicación en este proyecto, con el Robot, es inprescidible implementarlo de esta forma.

En los casos en los que se ejecutan varias instrucciones que deben presentar cambios en el hilo principal. Si se aplica el enfoque anterior utilizando los metodos **runOnUiThread ():,** por un lado para actualizar la interfaz gráfica, y por otro para llamar al socket, recoger la información y luego actualizarla mediante metodos del tipo **Handler.post(Runnable r)** .

El código para el envío de las ejecuciones tiende a ser muy largo, confuso y poco maleable a la hora de mantenimiento.

Por esta razón ha sido creada la interfaz AsyncTask, cuyo objetivo es liberar al programador del uso de hilos, la sincronización entre ellos y la presentación de resultados en el hilo primario. Esta clase unifica los aspectos relacionados que se realizarán en segundo plano y además gestiona de forma asíncrona la ejecución de las tarea.

Los métodos que conforma la **AsyncTask** son:

onPreExecute(): En este método van todas aquellas instrucciones que se ejecutarán antes de iniciar la tarea en segundo plano. Normalmente es la inicialización de variables, objetos y la preparación de componentes de la interfaz.

doInBackground(Parámetros…): Recibe los parámetros de entrada para ejecutar las instrucciones específicas que irán en segundo plano, luego de que haya terminado onPreExecute(). Dentro de él podemos invocar un método auxiliar llamado publishProgress(), el cual transmitirá unidades de progreso al hilo principal. Estas unidades miden cuanto tiempo falta para terminar la tarea, de acuerdo a la velocidad y prioridad que se está ejecutando.

En nuestro caso se ha utilizado para llamar al método que *socket,* que es el que establece la comunicación con el robot y empieza el flujo bidireccional de información entre el cliente(Android), y el servidor (Robot).

onProgressUpdate(Progreso…): Este método se ejecuta en el hilo de UI luego de que publishProgress() ha sido llamado. Su ejecución se prolongará lo necesario hasta que la tarea en segundo plano haya sido terminada. Recibe las unidades de progreso, así que podemos usar algún View para mostrarlas al usuario para que este sea consciente de la cantidad de tiempo que debe esperar.

onPostExecute(Resultados…): Aquí puedes publicar todos los resultados retornados por doInBackground() hacia el hilo principal. En nuestro caso sirve para actualizar las variables que y el vector que contiene tanto las coordenadas xc como yc, que posteriormente permiten graficar la información en la interfaz gráfica.

https://developer.android.com/reference/android/app/

http://jarroba.com/activity-entender-y-usar-una-actividad/

https://www.adictosaltrabajo.com/tutoriales/inicio-android/#01

12.3.- Código Cliente en Android

```
 package vehculo.joseangel.robot1;

import android.content.Context;
import android.graphics.Color;
import android.media.MediaPlayer;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.AsyncTask;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

import com.androidplot.xy.BoundaryMode;
import com.androidplot.xy.LineAndPointFormatter;
import com.androidplot.xy.SimpleXYSeries;
import com.androidplot.xy.XYPlot;
import com.androidplot.xy.XYSeries;
import com.androidplot.xy.XYStepMode;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Timer;
import java.util.TimerTask;

public class MainRobot extends AppCompatActivity {

private static final String CLASSTAG = MainRobot.class.getSimpleName();

        String ip = "192.168.1.2";
        String PortA = "5000";
        String PortB = "5001";
        byte MserverX = 0;
        byte MserverY = 0;

        String menXs = "";
        String menYs = "";
        double menXd = 0;
        double menYd = 0;

        TextView Xc;
        TextView Yc;

        MediaPlayer mpBoton1;                                   /*SONIDOS*/
        Button bTrayectoria1;                                   /*BOTONES*/
```

```
Button bTratectoria2;
Button bTrayectoria3;
ImageView semRed;
ImageView semRobot;
int contador = 0;

private XYPlot myXYPlot;                                                    /*GRÁFICA*/
double y[] = {0, 0.2, 0.4, 0.6, 0.8, 1, 1, 1, 1, 1, 1, 0.8, 0.6, 0.4, 0.2, 0, 0, 0, 0, 0, 0};
double x[] = {0, 0, 0, 0, 0, 0, 0.2, 0.4, 0.6, 0.8, 1, 1, 1, 1, 1, 1, 0.8, 0.6, 0.4, 0.2, 0};
ArrayList<Double> Vector = new ArrayList<Double>();
ArrayList<Double> Vector2 =new ArrayList<Double>();

public double DatoX, DatoY;
boolean flag = false;
boolean flag2 = false;
boolean flagSocket =false;

@Override
protected void onCreate(Bundle savedInstanceState) {//#####################################INICIO OBJETO PADRE#################################################################//
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main_robot_layout);
bTrayectoria1 = (Button) findViewById(R.id.botonTrayectoria1);

semRed = (ImageView) findViewById(R.id.imageView);
semRobot = (ImageView) findViewById(R.id.imageViewROBOT);

mpBoton1 = MediaPlayer.create(this, R.raw.mp1);
myXYPlot = (XYPlot) findViewById(R.id.Grafica);   //Queda relacionado el objeto xml y el java
 myXYPlot.setDomainStep(XYStepMode.INCREMENT_BY_VAL, 0.5);  //Unidad de incremento en X
myXYPlot.setRangeStep(XYStepMode.INCREMENT_BY_VAL, 0.5);   //Unidad de incremento en Y
myXYPlot.getGraphWidget().getGridBackgroundPaint().setColor(Color.rgb(250, 250, 250));
myXYPlot.getGraphWidget().getDomainGridLinePaint().setColor(Color.rgb(0, 0, 0));   //Grid Verticales
myXYPlot.getGraphWidget().getRangeGridLinePaint().setColor(Color.rgb(0, 0, 0));   //Grid Verticales
myXYPlot.setRangeBoundaries(-0.5, 1.5, BoundaryMode.FIXED);    //Rango a las Y
myXYPlot.setDomainBoundaries(-0.5, 1.5, BoundaryMode.FIXED);  //Rango de las X

for (int i = 0; i < 21; i++)      {

    Vector.add(x[i]);
    Vector.add(y[i]);
                                  }
final XYSeries series1 = new SimpleXYSeries(Vector, SimpleXYSeries.ArrayFormat.XY_VALS_INTERLEAVED, "GRÁFICA");
final LineAndPointFormatter series1Format = new LineAndPointFormatter(
          Color.rgb(13, 255, 0),
          Color.rgb(50, 100, 0),
                /*Color.rgb(0,0,0)*/ null, null);

myXYPlot.clear();
myXYPlot.addSeries(series1, series1Format);

bTratectoria2 = (Button) findViewById(R.id.botonTrayectoria2);
bTrayectoria3 = (Button) findViewById(R.id.botonTrayectoria3);
Xc = (TextView) findViewById(R.id.texViewXC);
Yc = (TextView) findViewById(R.id.textViewYC);

Timer timer = new Timer();
timer.scheduleAtFixedRate(new TimerTask() {
```

```
            @Override
            public void run() {if(flag==true){ ComprobarRED();}
            }
        }, 0, 2000);

        Timer timer2 = new Timer();
        timer2.scheduleAtFixedRate(new TimerTask() {

            @Override
            public void run() {if(flag2==true){empezar(); /*flag2=false;*/}


            }
        }, 0, 100);

 //##############################################################################################//
/*-------------------------SONIDOS Botones Trayectoira-------------------------------*/
    bTrayectoria1.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {
    mpBoton1.start();
    Toast.makeText(getBaseContext(), "Trayectoria 1", Toast.LENGTH_SHORT).show();
    flag = true;
    flag2 = true;
    empezar();

        }
    });
                /*_____________________________________________________________*/
 bTratectoria2.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
        mpBoton1.start();
        Toast.makeText(getBaseContext(), "Trayectoria 2", Toast.LENGTH_SHORT).show();

        }
    });
                /*_____________________________________________________________*/

 bTrayectoria3.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
        mpBoton1.start();
        Toast.makeText(getBaseContext(), "Trayectoria 3", Toast.LENGTH_SHORT).show();

        }
    });
 /*---------------------------------FIN SONIDOS Botones Trayectoira-------------------------*/

}//#################################################################################  FIN OBJETO PADRE ###############################################################################//
```

```
private void ComprobarRED()  {//---------1_1--FUNCIÓN ARRANCA HILO COMPROBAR CADA 2s LA RED---//
    this.runOnUiThread(Red);}

private Runnable Red = new Runnable() {
    @Override
    public void run() {//----------------1_2---ARRANCANDO HILO COMPROBAR RED------------------//

        ConnectivityManager connMgr = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
        if (networkInfo != null && networkInfo.isConnected()) {

            semRed.setImageResource(R.drawable.on);
        } else {
            semRed.setImageResource(R.drawable.off);
        }
    }
};//---------------------------------FIN 1_1 Y 1_2 COMPROBAR RED CADA 2s-----------------------//


private void empezar() {//------2_1--FUNCIÓN ARRANCA HILO lEER CADA 100ms del SOCKET-----------//

    new Thread(new Runnable() {

        @Override
        public void run() {
        //Aqui añadimos 100 nuevas entradas
        for (int i = 1; i < 1000; i++) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {

             if(flag2==true)  {  flag2=false;   empezar2();    }     //Llamamos a la Funcion encargada de Escribir en la GRAFICA

                }
            });                          // Y llamada al socket para enviar consigna AL SERVER------
            //Dormir par frenar la velocidad de entrada de los datos
            try {
                Thread.sleep(100);     //Llamada a la Funcion Temporización----------------------//1800 1200

            } catch (InterruptedException e) {
                e.printStackTrace();
                //Es como manejamos los errores
            }
        }

    }

}).start();

                                }
```

```
private void empezar2(){

        try {
            // Log.v("test", "ENVIAR 0" );

            new MySocket().execute(ip, PortA, PortB);                  //A  LLAMAMOS A LA F.ASÍNCRONA

        } catch (Exception e) {
            Log.v("test", "NO SOCKET 77");
            Log.v("Exception my socket", "Exception:" + e.getMessage());
        }

        this.runOnUiThread(enviar);

        }

private Runnable enviar = new Runnable() {

    @Override
    public void run() {//----------------2_2---ARRANCANDO HILO LEER SOCKET----------//


        XYSeries series2 = new SimpleXYSeries(Vector2, SimpleXYSeries.ArrayFormat.XY_VALS_INTERLEAVED, "");
        LineAndPointFormatter series2Format = new LineAndPointFormatter(
                Color.rgb(255, 76, 0),
                Color.rgb(0, 0, 0),
          /*Color.rgb(0,0,0)*/ null, null);

        myXYPlot.redraw();
        myXYPlot.addSeries(series2, series2Format);

    }
};//--------------------------------FIN 2_1 Y 2_2 LEER SOCKET CADA 100ms----------------------//

private class MySocket extends AsyncTask<String, Integer, int []> {
    private Byte[] result;  // B FUNCION ASINCRONA

    @Override
    protected int [] doInBackground(String... params) {

        String serverIp = params[0];
        String serverPortA = params[1];
        String serverPortB = params[2];
        int [] vector1 = new int[2];
        vector1[0] = -1;
        vector1[1] = -1;
        try {
           // Log.v("test", "ENVIAR 1" );
            vector1= callSocket(serverIp,serverPortA,serverPortB);
            return vector1;

        } catch (Exception e)    {
            Log.v("test", "ENVIAR 1B");
            Log.v("Exception my socket", "Exception:" + e.getMessage());

          return null;
                                }

                                                    }
```

```
    protected void onPostExecute(int [] result) {
        //--------------------------------------------//

        try {
            //Log.v("test", "ENVIAR 3");
            updateMsj(result);
        } catch (Exception e) {
            Log.v("Exception2 my socket", "Exception:" + e.getMessage());

        }

    }
                                                                    }// B FINAL FUNCION ASINCRONA
private void updateMsj(int [] msj) {

if (msj[0] == -1 || msj[1] == -1 ) {
        Toast.makeText(getBaseContext(), "EL SOCKET FALLO", Toast.LENGTH_LONG).show();
        semRobot.setImageResource(R.drawable.off);
        Log.v("test", "ENVIAR445");
        menXs = "PEPE";
        Xc.setText(menXs);
   } else {
        //Toast.makeText(getBaseContext(), "EL SOCKET OK", Toast.LENGTH_LONG).show();
        semRobot.setImageResource(R.drawable.on);

        menXd = (Double.valueOf(msj[0]));

        menYd = (Double.valueOf(msj[1]));
        Vector2.add(menXd*1.08/126);
        Vector2.add(menYd*1.08/126);

        menXs = String.valueOf(  ((menXd)*1.1/126));
        menYs = String.valueOf(  ((menYd)*1.1/126));


        Xc.setText(menXs);
        Yc.setText(menYs);
        flag2=true;



    }
}
private int [] callSocket(final String Ip, final String PorA, final String PorB) {


   Socket socketA = null;
   Socket socketB = null;

   BufferedReader readerPortA = null;
   BufferedReader readerPortB = null;
   int[]  vector = new int[2];
   vector[0] = -1;
   vector[1] = -1;
   //byte mensaje = 1;

   Log.i("INFO", " " + MainRobot.CLASSTAG + " callSocket");
```

```
try {

socketA = new Socket(Ip,Integer.parseInt(PortA));    //Puerto 5002

socketB = new Socket(Ip,Integer.parseInt(PortB));    //Puerto 5001


readerPortA = new BufferedReader(new InputStreamReader(socketA.getInputStream()));

readerPortB = new BufferedReader(new InputStreamReader(socketB.getInputStream()));


    if(readerPortA!= null){try {vector[0] = (int) readerPortA.read();      }
                           catch (Exception e){ Log.v("test", "NO LECTURA 1"); }}
    else{Log.v("test", "BBBBBBBBBBB");}

    if(readerPortB!= null){try{vector[1] = (int) readerPortB.read();         }
                           catch (Exception e){ Log.v("test", "NO LECTURA 2"); }}
    else{Log.v("test", "DDDDDDDDDDDDDD");}


} catch (Exception e) {
    Log.e("INFO", " " + MainRobot.CLASSTAG + " IOEXCEPCIÓN LLAMANDO AL SOCKET", e);
    Log.v("test", "NO SOCKET 1");

}finally {

    try{
        Log.v("test", "1 CERRANDO BUFFERS");
        if (readerPortA != null ) { readerPortA.close();}
        if (readerPortB != null ) { readerPortB.close();}

       }catch (IOException e){
        Log.v("test", "NO SOCKET 4");
       }
       try {

        Log.v("test", "2 CERRANDO SOCKETS");
        socketA.close();
        socketB.close();
       } catch (IOException e) {

        Log.v("test", "NO SOCKET 6");
       }

   }

    return vector;


}//-------------------------------------------------------------------------------------------//
```

```
@Override
public void onRestart(){
    super.onRestart();
    Log.v("test", "Restart");
}

@Override
public void onResume(){

       ConnectivityManager connMgr = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
       NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
       if (networkInfo != null && networkInfo.isConnected()) {

           semRed.setImageResource(R.drawable.on);
       } else {
           semRed.setImageResource(R.drawable.off);
       }

    super.onResume();
    flag = true;
    Log.v("test", "Resume");
}


@Override
public void onPause(){


    super.onPause();
    Log.v("test", "Pause");
}

@Override
public void onStop(){

    super.onStop();
    Log.v("test", "Stop");
}
```

13.- ANEXO 6 CARACTERÍSTICAS DIMENSIONES MOTOR

Motor de 12VDC totalmente equipado con codificadores de cuadratura, caja reductora de 30:1 y condensador de supresión de ruido. Es ideal para aplicaciones de robótica con la posibilidad de un control total por parte del usuario: velocidad, sentido de giro y posicionamiento o desplazamiento del eje. Sus características mas notables son:

- Tensión nominal de 12VDC
- Fuerza 1.5Kg / cm
- Velocidad nominal 170 rpm
- Consumo típico 530mA
- Velocidad sin carga 216 rpm
- Consumo sin carga en vacío 150mA
- Contador del codificador 360 pulsos por vuelta del eje.
- Eje de 5 mm para casquillos de fijación de 5mm.

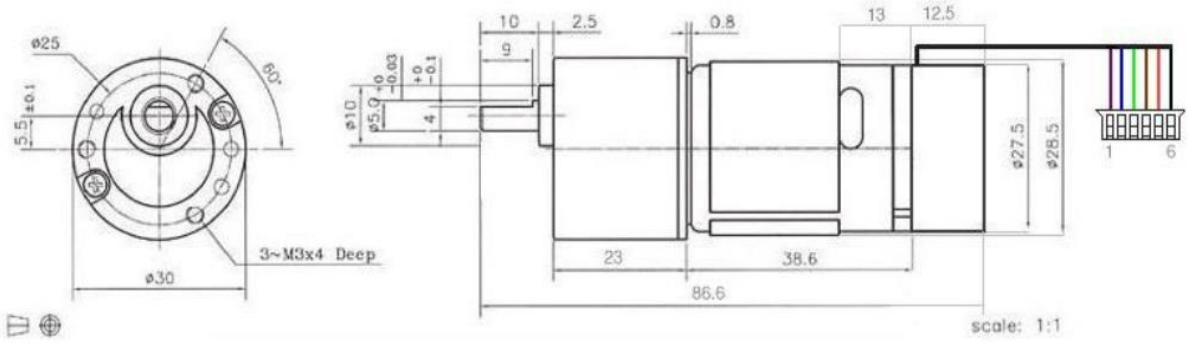


Figura 13_1: Dimensiones

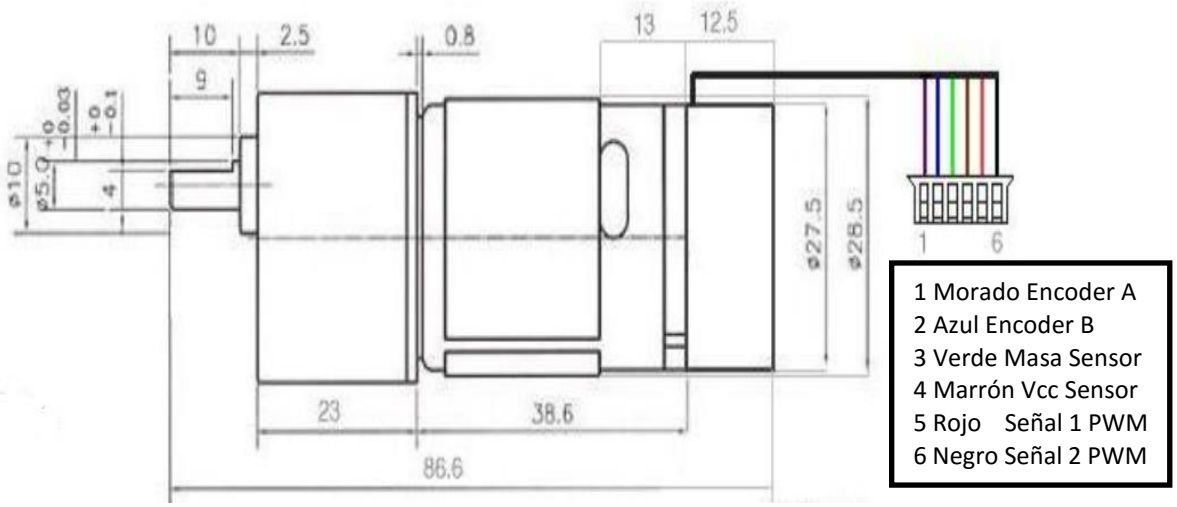


Figura 13_2 : Conexiones

Para poder conectar las salidas del encoder A y B, se han utilizado resistencias Pull-up de 10 KΩ. Ver Plano 04, Conexiones Motor-Driver-Microcontrolador.

14.- ANEXO 7 BOLA LOCA

14.1.- Despiece

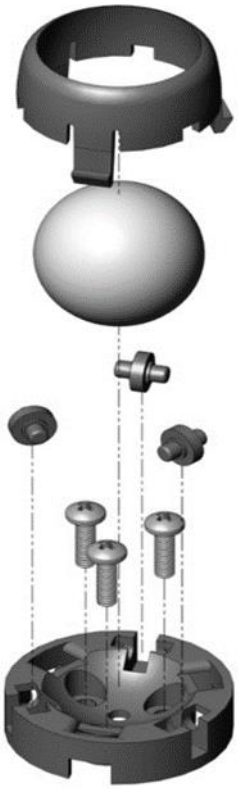


Figura 14.1_1: Despiece Bola Loca

PLIEGO DE CONDICIONES TÉCNICAS:

15.- PLIEGO INTRODUCIÓN

El presente pliego de condiciones tiene por finalidad nombrar las características de cada uno de los materiales y componentes utilizados, ver que estos cumplen con la normativa correspondiente y explicar una serie de directivas básicas para el correcto funcionamiento del vehículo Robot, y muy especialmente se hace hincapié en el proceso de carga, con el fin de prolongar la vida útil de la batería.

Además, durante la realización del proyecto se ha prestado especial atención al marco legal en el cual se engloba el proyecto.

16.- PLIEGO CONDICIONES GENERALES

En lo relativo al marco legal, este proyecto se rige por la *Normativa de Marco de Trabajos De Fin de Grado y Fin de Máster de la Universidad Politécnica de Valencia*, aprobada en Consejo de Gobierno el 7 de marzo de 2013.

Esta normativa nace a consecuencia del *Real Decreto 1393/2007*, por el que se establece la ordenación de las enseñanzas universitarias oficiales, modificado por el *Real Decreto 861/2010* que dispone, con carácter general, que todos los títulos oficiales “concluirán con la elaboración y defensa” de un Trabajo Fin de Grado (TFG) o Trabajo Fin de Máster (TFM), según el caso. Adicionalmente las órdenes ministeriales por las que se establecen los requisitos que deben cumplirse para la verificación de los títulos oficiales que habilitan para las profesiones de Arquitecto, Arquitecto Técnico, Ingeniero o Ingeniero Técnico también incluyen prescripciones generales relativas a la naturaleza de los TFG, condiciones para su presentación y defensa y, en algún caso, composición del tribunal calificador.

17.- PLIEGO CONDICIONES DE ESPECIFICACIONES TÉCNICAS

17.1.- Materiales del Robot

17.1.1.- **Aluminio**

El bastidor, elemento donde quedan fijados los dos motores está compuesto totalmente por material de Aluminio, consta por una parte de dos soportes en forma de L, que han sido diseñados especialmente para fijar el motor con encoder EMG30 mediante tres tornillos de métrica 3mm y profundidad 5mm. (Ver Figura 3.1.1_1).

Figura 17.1.1_1: Soporte Motor

Y por otro lado los dos soportes se han unido mediante dos perfiles de aluminio en forma de L, cuyo objetivo es el de crear una fijación fuerte de los dos soportes a dichos perfiles, a la vez que aseguren la alineación correcta de los ejes de los motores, quedando estos separados 170 mm, (de la cara exterior motor a cara exterior motor). En el apartado planos **plano 02 Bastidor,** se especifican las dimensiones correctas.

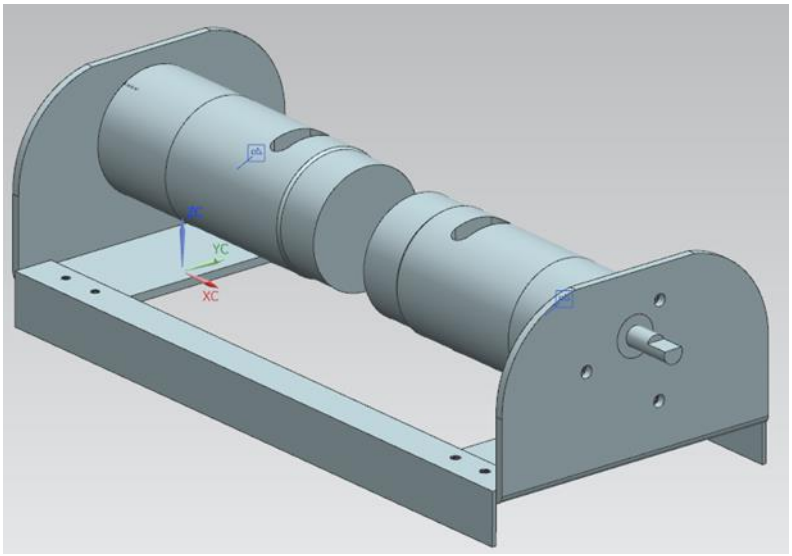


Figura 17.1.1_2: Unión de los Soportes mediante Perfil de Aluminio

El aluminio es un material de baja densidad y alta resistencia a la corrosión. Mediante aleaciones se puede conseguir que adquiera una alta resistencia mecánica. Además, es buen conductor de la electricidad, se mecaniza con facilidad y es relativamente barato.

El aluminio, en todos los casos, corresponde a la aleación 1050 que cumple con las siguientes características:

Composición química:

Esta aleación está formada por un 0.25% de SI, 0.40% de Fe, 0.05% de Cu, 0.05% de Mn, 0.05% de Mg, 0.05% de Zn, 0.03% de Ti, 0.03%, 0.05% de V y 99.5% de aluminio.

Características técnicas

Densidad: 2700 Kg. / m^3

Carga de rotura (Rm): 80 N/m m^3

Alargamiento: 45 %

17.1.2.- **ABS**

El chasis completo se ha diseñado como un único bloque (diseño monocasco), de ABS (**Acrilonitrilo Butadieno Estireno**), que se usa extensivamente en los procesos de fabricación actuales: piezas de Lego, carcasas de electrodomésticos, componentes de automóvil etc. Al tener un punto de fusión alto, se puede utilizar para fabricar contenedores de líquidos calientes, hay que extruirlo a unos 230-260 grados y hay que imprimirlo en impresoras con base de impresión caliente (unas resistencias que calientan la base dónde se deposita el material).

El material utilizado por la impresora para generar el modelo, es Acrilonitrilo Butadieno Estireno o ABS, que es un plástico muy resistente al impacto, utilizado especialmente en la industria de la automoción, pero en ocasiones también en la doméstica.

Al llegar al punto de fusión el ABS desprende gases que en concentraciones altas pueden ser nocivos. Se puede utilizar sin problemas en casa o en la oficina, pero para evitar las concentraciones altas no se recomienda tener varias impresoras funcionando en un espacio pequeño y sin ventilar.

El ABS se puede mecanizar, pulir, lijar, limar, agujerear, pintar, pegar etc. con extrema facilidad, y el acabado sigue siendo bueno. Además, es extremadamente resistente y posee un poco de flexibilidad. Todo esto hace que sea el material perfecto para **aplicaciones industriales**.

Se le suele llamar plástico de ingeniería porque su elaboración y procesamiento es algo más compleja que en los plásticos comunes.

Este material compuesto soporta bien las temperaturas extremas, especialmente en entornos fríos.

17.2.- Motores

Motor de 12VDC totalmente equipado con codificadores de cuadratura, caja reductora de 30:1 y condensador de supresión de ruido. Es ideal para aplicaciones de robótica con la posibilidad de un control total por parte del usuario: velocidad, sentido de giro y posicionamiento o desplazamiento del eje. Sus características más notables son:

- Tensión nominal de 12VDC
- Fuerza 1.5Kg / cm
- Velocidad nominal 170 rpm
- Consumo típico 530mA
- Velocidad sin carga 216 rpm
- Consumo sin carga en vacío 150mA
- Contador del codificador 360 pulsos por vuelta del eje.
- Eje de 5 mm para casquillos de fijación de 5mm.

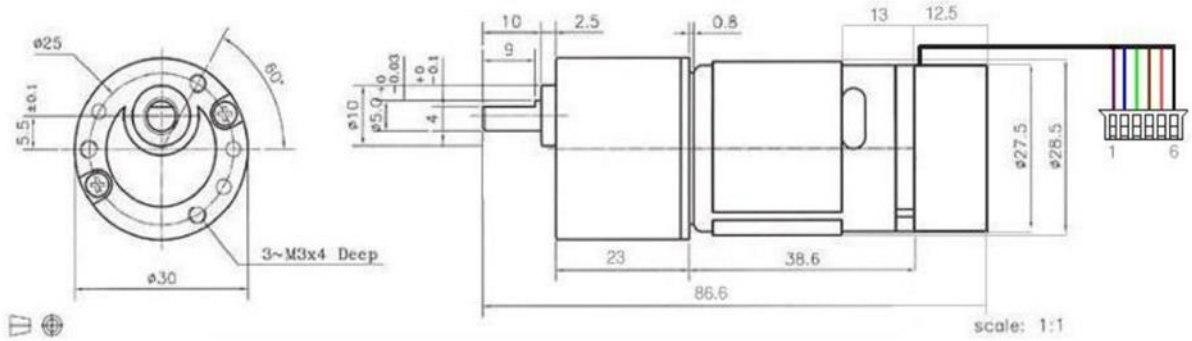


Figura 17.2_1: Dimensiones Motores

17.3.- Baterías de LITIO

17.3.1.- Introducción

La evolución y proliferación de dispositivos electrónicos alimentados por baterías ha impulsado en los últimos años el desarrollo de nuevas tecnologías que han permitido mejorar las prestaciones de los aparatos electrónicos. Las baterías con base de litio son la última generación de baterías de uso popular. Forman parte de nuestra vida al estar presentes en los Smartphone, Tablet, ordenador portátil, etc. La tecnología de baterías basadas en Litio es ya una tecnología madura después de varias décadas de desarrollo. Sigue siendo una tecnología que se diferencia de las demás por las múltiples ventajas sin apenas factores negativos. Las principales ventajas son su alta densidad de energía, su rápida carga y su ligereza a la vez que el inconveniente principal es su inestabilidad química frente a sobre-cargas o sobre-descargas que obligan a utilizar sistemas electrónicos que protejan a la batería. Hay varias familias de baterías de litio en función de los materiales utilizados en su construcción, principalmente en el cátodo: Cobalto, Manganeso, Hierrofósforo, Titanio, etc.

En nuestro caso la familia a la que pertenece nuestra batería, es la de batería de polímero de iones de litio, de ion de litio polímero o más comúnmente batería de polímero de litio (abreviadamente Li-poli, Li-Pol, LiPo, LIP, PLI o LiP) son pilas recargables (células de secundaria), compuestas generalmente de varias células secundarias idénticas en paralelo para aumentar la capacidad de la corriente de descarga, y están a menudo disponibles en serie de "packs" para aumentar el voltaje total disponible.

En nuestro caso la batería es exactamente del tipo batería de ion-litio con un cátodo de fosfato de hierro-litio: LiFePO.

Resumen Características Batería LiPo:

Nivel de Voltaje: 3.7V/célula.

Máximo voltaje de carga: 4.2V/célula

Corriente de carga rápida: 1C o menos.

Mínimo voltaje de descarga, no menos de: 3V/célula o más alto.

17.3.2.- **Carga**

Las baterías de Litio requieren una técnica de carga muy específica debido a sus características químicas y eléctricas. La razón es que, a diferencia de otras tecnologías, las baterías de litio se vuelven químicamente inestables cuando alcanzan un voltaje en el que aún no están plenamente cargadas. En otras tecnologías, como las basadas en Níquel o Plomo, la batería alcanza la carga plena antes de que la sobre-carga pueda producir daños normalmente debidos al calor liberado por la energía no almacenada. Sin embargo, las baterías de Litio se vuelven químicamente inestables cuando alcanzan un determinado voltaje mientras que no están aun totalmente cargadas. Por este motivo, se debe utilizar una carga combinada corriente constante/voltaje constante (CC/CV) de forma que la batería se carga a intensidad constante hasta que alcanza el voltaje máximo. En ese punto, el voltaje debe mantenerse constante mientras la intensidad va disminuyendo hasta un nivel señalado en el que se considera cargada la batería. Las baterías de Litio admiten una intensidad de carga de relación elevada con respecto a la Capacidad en comparación a otras tecnologías y tienen un rendimiento muy alto en la carga.

En nuestro caso la protección en el proceso de carga la aseguramos mediante el cargador especial de la Figura 17.3.2_1:

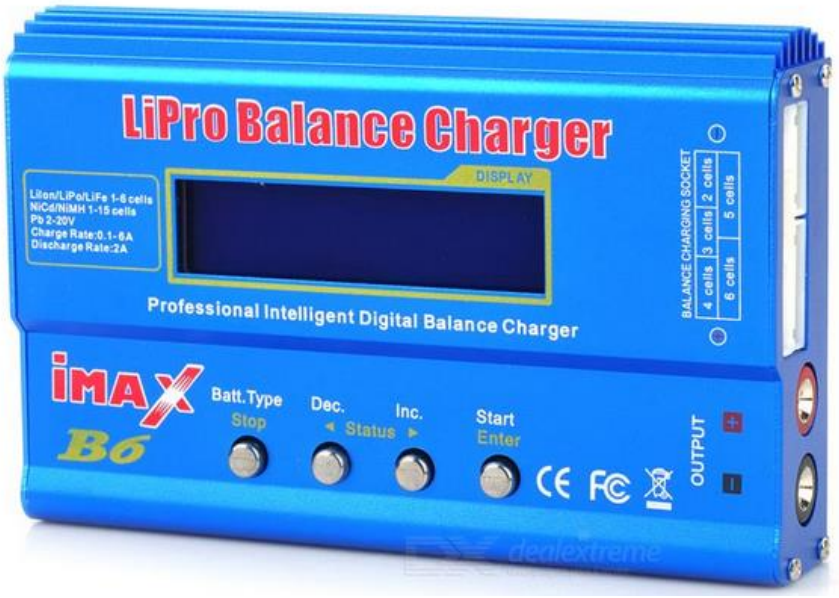


Figura 17.3.2_1: Cargador de baterías de Litio

Para cargar la batería vamos siempre a la opción Balance (siempre hay que usar esta opción para evitar estropear nuestra batería li-po), no debemos olvidar conectar la clavija blanca en su correspondiente clavija en el cargador (Ver Figura 3.3.6_1). Seleccionamos el tipo de batería y el número de celdas 3,4,7 etc. En nuestro caso LI-PO de 3 celdas 11.1 voltios.

Y muy importante ajustar la corriente en 1 Amperio para la máxima, y ya podemos empezar el proceso de carga.

17.3.3.- **Descarga**

Las baterías de Litio no deben descargarse por debajo de un determinado voltaje. Si esto sucede, la batería se deteriora disminuyendo la capacidad, el número de ciclos de carga y descarga o degradándose el electrolito. La auto-descarga de las baterías de Litio es muy inferior a la de otras tecnologías.

Está previsto la instalación de un pequeño circuito electrónico para evitar una descarga profunda, que como se ha indicado en este tipo de baterías puede degradar y acortar la vida de la batería.

17.3.4.- **Circuito de Protección**

La vulnerabilidad de las baterías de Litio frente a sobre-voltaje, sobre-descarga y sobre-intensidad entre otros, hace muy recomendable (casi imprescindible) el uso de circuitos electrónicos que controlen los valores de voltaje e intensidad en carga y descarga para evitar daños en la batería. El término PCM sirve para designar un pequeño circuito electrónico que controla los parámetros peligrosos para la batería. El PCM tiene el control y capacidad de desconectar la batería para protegerla tanto en la carga como en la descarga. Son circuitos muy simples y muy eficaces que conviven con las baterías de Litio en casi todas las situaciones

El voltaje de cada celda está comprendido entre 2.0 a 3.6v en el caso de LiFePo, que es nuestro caso, si no tenemos circuito de protección y trabajamos fuera de este rango, como se comentó antes, podemos dañar la batería.

17.3.5.- **Balanceo**

El balanceo es una técnica que se aplica a packs de baterías para corregir los desequilibrios que aparecen entre las diferentes células de un pack. Esta función se puede realizar de muchas maneras. El cargador de la Figura 3.3.2_1 incorpora un balanceador por batería que descarga una parte de la intensidad de carga

cuando la batería está a punto de alcanzar el voltaje máximo. De esta manera se ralentiza la última parte de la carga mientras que las demás baterías del pack se siguen cargando más rápido. De esta manera, en cada carga, se minimiza el desequilibrio entre las baterías. En nuestro caso es imprescindible para la vida útil de la batería.

La carga incorrecta, sin utilizar un cargador con las características que tiene el de la Figura 3.3.2_1, puede implicar la explosión de la batería

17.3.6.- **Características Cargador**

ESPECIFICACIONES:

Voltaje de operación: De 11 a 18 Voltios, Corriente Continua. AC 125 240 voltios 50HZ.

Potencia: ……..…….Máxima en Carga: 50 Vatios.

Máxima en Descarga:……. 5 Vatios.

Corriente de carga: ……..……...0.1 a 5 Amperios.

Corriente de descarga: …….0.1 a 1 Amperio.

Corriente drenaje para equilibrar baterías LiPo: ....…….……300 mAh/célula.

Número de células de baterías de NiCd/NiMH: …….....…1 a 15 células.

Número de células de Li-íon/Polímero: ……...……..1 a 6 en serie.

Voltaje para baterías de Plomo: ……...……...2 a 20 Voltios.

Peso: ……..……220 gramos.

Dimensiones: ……..……133 x 87 x 33 mm.

SOFTWARE DE OPERACIÓN OPTIMIZADO:

El Cargador B6 presenta una función AUTO, que ajusta la corriente de alimentación, durante el proceso de carga y descarga. Especialmente para las baterías de litio, para impedir la sobrecarga, lo que puede dar lugar a una explosión, debido a la posible negligencia del usuario. El cargador puede desconectar automáticamente el circuito, y activar la alarma cuando detecta un mal funcionamiento.

Todos los programas de este cargador, están controlados a través de un sistema de control de dos vías, para obtener un máximo de seguridad y minimizar los problemas. Todos los ajustes pueden ser configurados por el usuario.

BALANCEADOR INTERNO DE BATERÍA DE LITIO INDEPENDIENTE:

El Cargador B6 emplea un equilibrador interno de baterías de Lipo. Con lo cual no es necesario conectar un equilibrador de baterías externo para balancearlas.

DESCARGA DE LAS BATERÍAS, CON BALANCEADOR INDIVIDUAL DE LAS BATERÍAS:

Durante el proceso de descarga, el Cargador B6 puede monitorizar y equilibrar, cada célula de la batería por separado. Si el voltaje individual de cualquier célula es anormal, se mostrará un mensaje de error y el proceso será automáticamente interrumpido.

ADAPTÁBLE A VARIOS TIPOS DE BATERÍAS DE LI-PO:

El Cargador B6 se puede adaptar a varios tipos de baterías, de Lítio: como Li-Ion, Li-Po, o la nueva serie de baterías LiFe, que es nuestro caso.

MODALIDAD RÁPIDA O DE ALMACENAMIENTO DE BATERIAS DE LÍTIO:

La modalidad de carga de las baterías de Lítio varían, la carga “f*ast*” (Rápida) reducen la duración del tiempo de carga, mientras “Store” (Almacenamiento), puede controlar el voltaje final da su batería, para almacenarla por un largo período, y proteger así la batería alargando la vida útil de la misma.

MÁXIMA SEGURIDAD:

Sensibilidad Delta-Peak: Programa de terminación de carga automático, basado en el principio de la detección de la tensión del Delta-Peak. Cuando el voltaje de la batería supera el límite, el proceso de carga se interrumpirá automáticamente.

LIMITE AUTOMÁTICO DE LA CORRIENTE DE CARGA:

Se puede determinar o limitar, al cargar sus baterías de NiCd o NiMH, la corriente máxima de carga, esto es útil para las baterías de NiMH de baja impedancia y capacidad, en el modo de carga “AUTO”.

LIMITE DE CAPACIDAD:

La capacidad de carga se calcula siempre, como la corriente de carga multiplicada por el tiempo. Si la capacidad de carga excede del límite, el proceso finalizara automáticamente cuando este determinado el valor máximo por el usuario.

CONTROL DE TEMPERATURA:

La reacción química interna de la batería, causará un aumento de su temperatura. Si se supera el límite de temperatura prefijado, el proceso finalizará.

Esta función estará disponible, por la conexión opcional de una sonda de temperatura, no incluida en el cargador.

LIMITE DE TIEMPO DEL PROCESO:

También puede limitar el tiempo del proceso, para evitar cualquier posible defecto.

MONITOR DEL VOLTAJE DE ENTRADA:

Para proteger la batería del vehículo, si utiliza su batería como fuente de alimentación del cargador, contra una posible descarga, su voltaje se monitoriza continuamente. Si el voltaje de entrada cae por debajo de un límite determinado, el proceso finalizará.

ALMACENAR DATOS EN MEMORIA:

Los datos de hasta cinco baterías, se pueden almacenar para usarlos según su conveniencia. Se pueden guardar los datos, relacionados con el programa de configuración de la batería, de la carga o la descarga. Estos datos pueden ser recuperados, en cualquier momento sin ningún tipo de programa especial.

CARGA / DESCARGA CÍCLICA:

Se pueden programar de 1 a 5 procesos, cíclicos y continuos de carga>descarga, o también de descarga>carga, para refrescar o balancear la batería, y también para estimular la actividad de la misma.

Como se comentó en nuestro caso, es necesario cargar siempre con la modalidad "Programa de carga y balanceado", en la figura siguiente Figura 3.3.6_1 se muestra la forma correcta de conectar para cargar la batería con el cargador.

Figura 17.3.6_1: Forma correcta de conectar la batería

17.4.- HARDWARE

17.4.1.- **Microcontrolador ARM M3 (ArduinoDue)**

Características:

- Microcontrolador: AT91SAM3X8E
- Voltaje de operación: 3.3V
- Voltaje de entrada recomendado: 7-12V
- Voltaje de entrada min/max: 6-20V
- Digital I/O Pins: 54 (de los cuales 12 proveen salida PWM)
- Analog Input Pins: 12
- Analog Outputs Pins: 2
- Corriente total de salida DC en todas las lineas I/O: 130 mA
- Corriente DC para el Pin de 3.3V: 800 mA

- Memoria Flash: 512 KB disponibles para aplicaciones del usuario
- SRAM: 96 KB (two banks: 64KB and 32KB)
- Clock Speed: 84 MHz

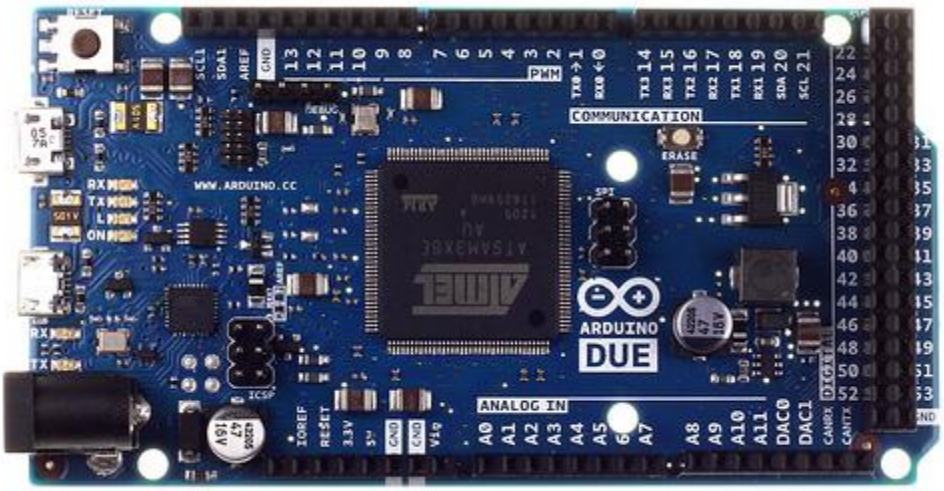


Figura 17.4.1_1: Arduino Due

17.4.2.- **Driver doble puente en H**

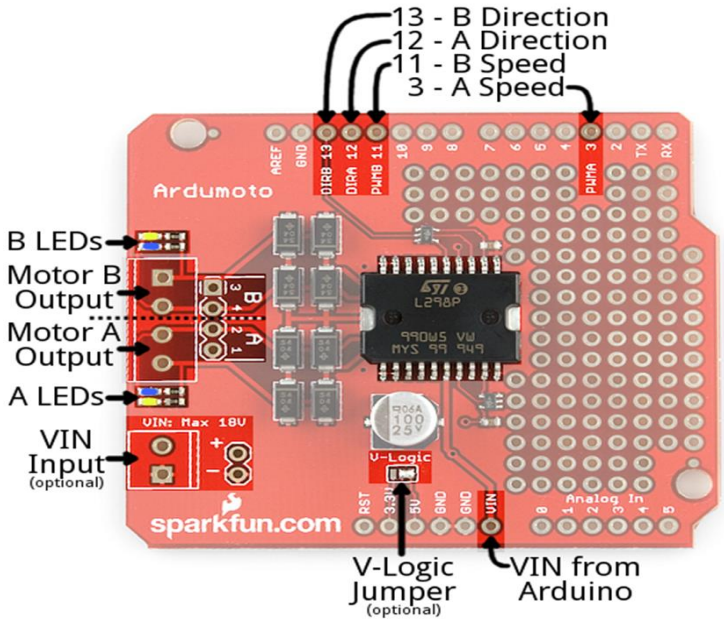


Figura 17.4.2_1: Doble Puente en H Ardumoto

Este módulo permite controlar dos motores de corriente continua, pudiendo manejar hasta 2 Amp por canal. Se basa en el circuito integrado L298P, que se describe a continuación un poco más abajo. Posee dos leds por canal (Azul y Amarillo) que indican el sentido de giro (Azul avance hacia adelante, amarillo avance hacia atrás).

Las dos entradas que controlan cada uno de los puentes en H para aplicar el PWM a los Motores, son PWMA, conectada a la salida PWM 3 de Arduino, que en nuestro caso se ocupa del control de la rueda izquierda, y PWMB conectada al PWM 11 de Arduino que se ocupa del control de la rueda derecha.

17.4.3.- **Modulo Shield wifi Arduino**

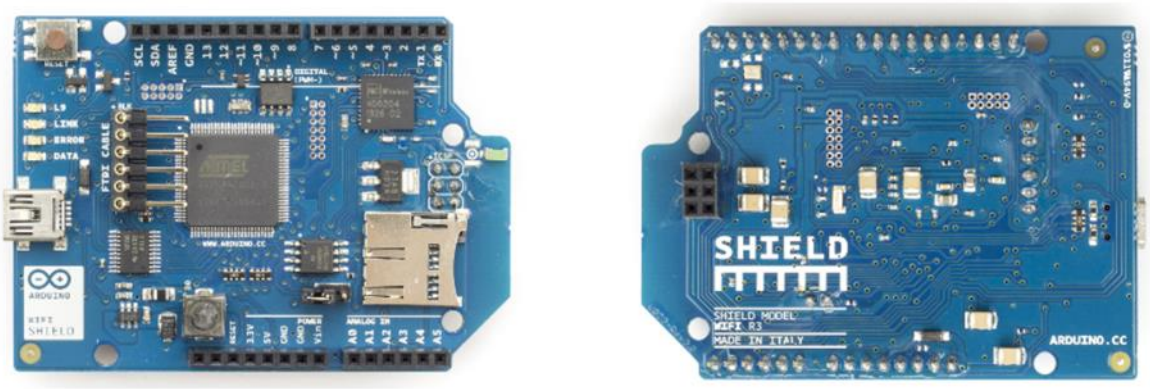


Figura 17.4.3_1: Shield wifi Arduino

Este módulo Wifi permite conectar Arduino Due a una red de área local de forma inalámbrica. Se basa en la especificación inalámbrica 802. 11. Este módulo wifi posee dos circuitos integrados importantes por una parte el HDG204 802.11b , es el que posee la antena y por lo tanto, es el encargado de la transmisión y recepción de datos. La frecuencia de la señal propagada está en el rango de 2400 a 2500 MHz.

Por otro lado, el módulo wifi contiene un AT32UC3, que es un microcontrolador Atmel de 32 bits, que es el que se encarga del empaquetado TCP/IP o UDP.

Una característica importante de este módulo es que la comunicación con Arduino Due se hace a través de la cabecera ICSP que utiliza el bus de comunicaciones SPI, dejando libre la parte superior del módulo wifi para poder apilar en este caso el driver que controla los Motores.

17.4.4.- **Dispositivo Android**

Figura 17.4.4_1: Tablet ASUS ZenPad10

Dimensiones	251.6 x 172 x 7.9 mm
Peso	510 gramos
Pantalla	IPS de 10.1 pulgadas con ASUS TruVivid
Densidad de píxeles	160 ppp
Procesador	Intel Atom x3-C3200 de cuatro núcleos y 64 bits a 1.2 GHz
RAM	2 GB
Sistema operativo	Android 5.0.2 con ZenUI
Almacenamiento	32 GB ampliables con tarjetas microSD
Cámaras	Trasera de 5 megapíxeles y frontal de 2 megapíxeles
Batería	4.890 mAh
Otros	Wi-Fi / LTE, Bluetooth 4.0, Giroscopio, Sensor de luz, GPS, Glonass

17.5.- Software

Los programas informáticos utilizados, todos ellos en su versión compatible con Windows 10, son:

- Matlab / Simulink 14b
- NX 10 Siemens
- Android Studio 2.1.1
- Microsoft Office 2016
- Autocat 2008
- OpenOffice Draw
- CamStudio
- Google Chrome
- Logitec WebCam Software

PRESUPUESTO:

18.- PRESUPUESTO INTRODUCIÓN

A continuación, se presenta el presupuesto estimado para este proyecto. En este presupuesto se tienen en cuenta los costes de los equipos, licencias de software, el personal y las instalaciones. En el apartado siguiente se enumeran los distintos recursos utilizados y posteriormente se realiza un desglose de su coste, tanto por hora de uso como de una manera total en el proyecto.

Para realizar el cálculo de la tasa horaria de los equipos, se ha realizado una estimación de la vida útil de los mismos.

En lo referente a las horas trabajadas se tiene en cuenta que el desempeño de este proyecto se ha producido en el marco de una *Beca de colaboración de tipo A* convocada por la propia Universidad, por lo que la estimación de las horas trabajadas que es utilizada para el cálculo de los costes se desprende del propio contrato de colaboración:

$$Horas_{trabajadas} = 80 * \frac{horas}{mes} * 3\ meses = 240\ horas \quad (1)$$

19.- PRESUPUESTO RECURSOS UTILIZADOS

19.1.- Recursos Hardware

Descripción	Unidades
Arduino Due Coterx ARM 80Mhz	1
Driver Ardumoto Doble Puente en H	1
Módulo Shield Wifi Arduino	1
Cargador Baterías LiPro Balance B6	1

Tabla 19.1_1: Lista Material Hardware

19.2.- Motores + Chasis + Bastidor + Batería, y varios

Descripción	Unidades
Motor CC EMG30	2
Cartucho ABS Impresora 3D	1
Aluminio Bastidor Soporte	1
Batería LI-PO 2600 mAh	1
Condensadores Electrolíticos 1000µF	4
Cableado	1

Tabla 19.2_1: Lista Material Motores + Bastidor, varios

19.3.- Licencias de software

Descripción	Unidades
Licencia Matlab 2014b	1
Licencia NX 10 Siemens	1
Licencia Microsoft Office 2016	1

Tabla 19.3_1: Lista Material Software

19.4.- Personal

Descripción	Unidades
Becario	1

Tabla 19.4_1: Lista Personal

19.5.- Instalaciones

Descripción	Unidades
Despacho	1
Impresora 3D	1
PC HP Pavilion 550-131ns	1

Tabla 19.5_1: Lista Instalaciones

20.- PRESUPUESTO DESGLOSE DE COSTES

En lo referente a los equipos físicos, se considera el coste completo de la cámara cenital. Por otra parte, se asume que el 1 ordenadores personales, la impresora 3D, tienen una vida útil de 5 años cada uno. De esta forma se calcula el coste de estos componentes para el proyecto como una parte proporcional de su total, es decir, se considera que el coste total del equipo hace referencia a sus, aproximadamente, 60 meses de vida útil y que únicamente se hace uso de 3 meses de esos 60:

20.1.- Recursos Físicos

$$\boldsymbol{Cote_{equipo}}(€) = \frac{\boldsymbol{Total\ Equipo}}{\boldsymbol{60\ mes}} * \boldsymbol{3\ meses}$$

Descripción	Coste total (€)	Coste Proyecto (€)
PC HP Pavilion 550-131ns	502	25.1
Impresora 3D	-	10
Logitech Webcam Pro 900	43	43
Total		**78,1**

Tabla 20.1_1: Lista Costes Recursos Físicos

20.2.- Licencias Software

Para calcular el coste de las licencias de software se toma el coste de una licencia anual y se calcula una parte proporcional a ésta, asumiendo que en un año laboral se trabajan en torno a 1820 horas y que son 240 las trabajadas. Asumiendo entonces que una licencia anual puede equivaler a una licencia por 1820 horas, se halla la tasa horaria:

$$\boldsymbol{Tasa_{horaria}}\left(\frac{€}{\mathrm{h}}\right) = \frac{coste\ licencia\ anual\ €}{horas\ anuales\ trabajadas\ 1820\ h}$$

Descripción	Coste anual (€)	Tasa horaria (€/h)	Coste Proyecto (€)
Licencia Matlab 2014b	500	0,275	66
Licencia NX 10 Siemens	603	0,331	79,5
Licencia Microsoft Office 2016	80	0,044	43
Total			**188,5**

Tabla 20.2_1: Lista Costes Licencias Software

20.3.- Hardware + Material

Descripción	Unidades	Precio (€) unidad	Subtotal (€)
Motor CC EMG30	2	37,35	74.7
Cartucho ABS Impresora 3D	1	69,8	69,8
Aluminio Bastidor Soporte	1	4,35	4,35
Batería LI-PO 2600 mAh	1	22	22
Condensadores Electrolíticos 1000µF 25 V	4	0,18	0,72

Cableado	1	3,75	3,75
Cargador Baterías LiPro Balance B6	1	16,42	16,42
Arduino Due Coterx ARM 80Mhz	1	45	45
Driver Ardumoto Doble Puente en H	1	29	29
Módulo Shield Wifi Arduino	1	84,58	84,58
TOTAL			**350.32**

Tabla 20.3_1: Lista Costes Hardware + Material

20.4.- Personal

Descripción	**Sueldo mensual €/mes**	**Meses**	**Coste/Proyecto €**
Becario	470	3	1413

Tabla 20.4_1: Lista Costes Personal

21.- PRESUPUESTO RESUMEN DEL PRESUPUESTO TOTAL

Concepto	**Coste (€)**
Recursos Físicos	78,1
Licencias Software	188,5
Hardware + Material	350.32
TOTAL, SIN PERSONAL	616,92
PERSONAL	1413
TOTAL, CON PERSONAL	**2029,92**

Tabla 21_1: Resumen Total

PLANOS:

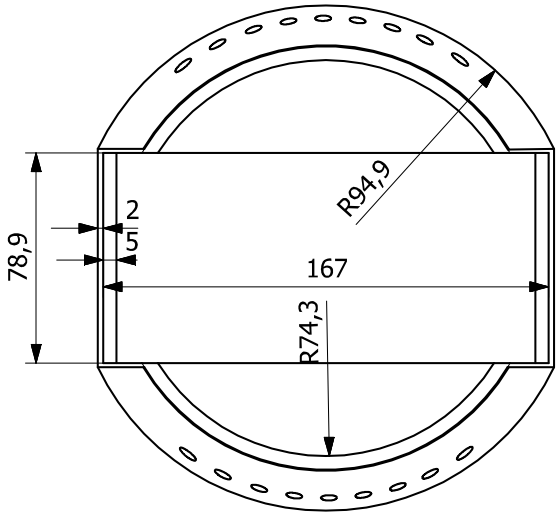
22.- PLANOS

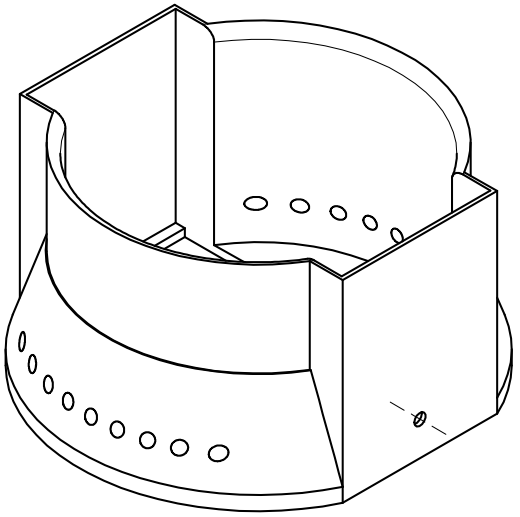
22.1.- PLANO 1_A CHASIS.

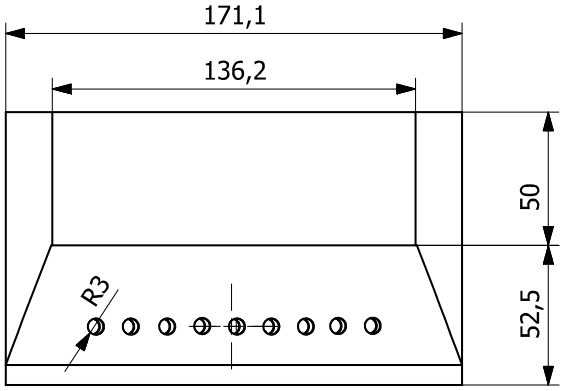
22.2.- PLANO 1_B CHASIS.

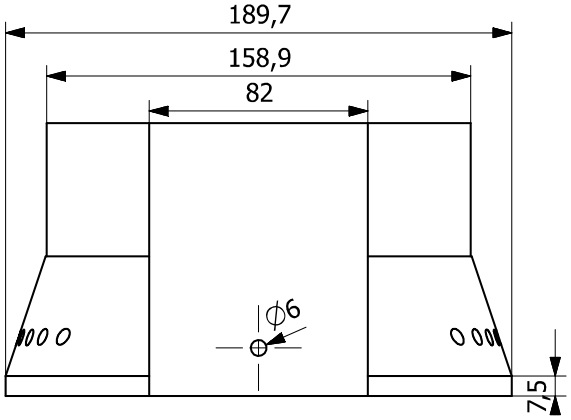
22.3.- PLANO 2 BASTIDOR.

22.4.- PLANO 3 CONEXIONES MOTOR-DRIVER-MICROCONTROLADOR



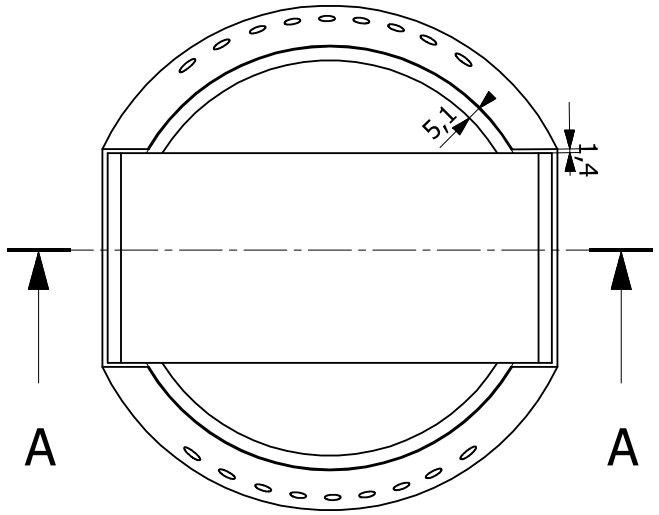


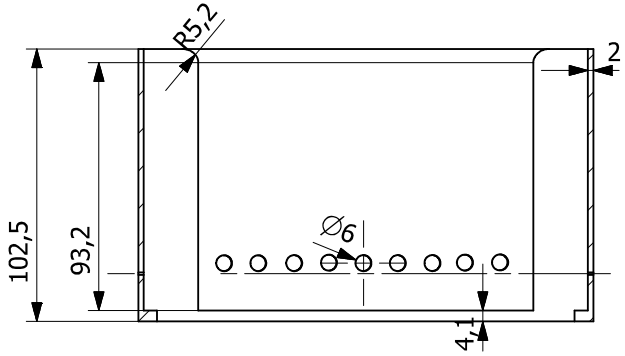




	Unidades	Nombre	ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DEL DISEÑO
Dibujado	mm	José Ángel Garrido Sarasol	
Departamento	DISA		
Máster	MÁSTER UNIVERSITARIO EN NGENIERÍA MECATRÓNICA		
Escala 1:2	PLANO 1_A CHASIS		DISEÑO E IMPLEMENTACIÓN DE UN VEHÍCULO AUTÓNOMO TERRESTRE CONTROLADO MEDIANTE INTERFAZ INALÁMBRICA ANDROID-ARDUINO

A3

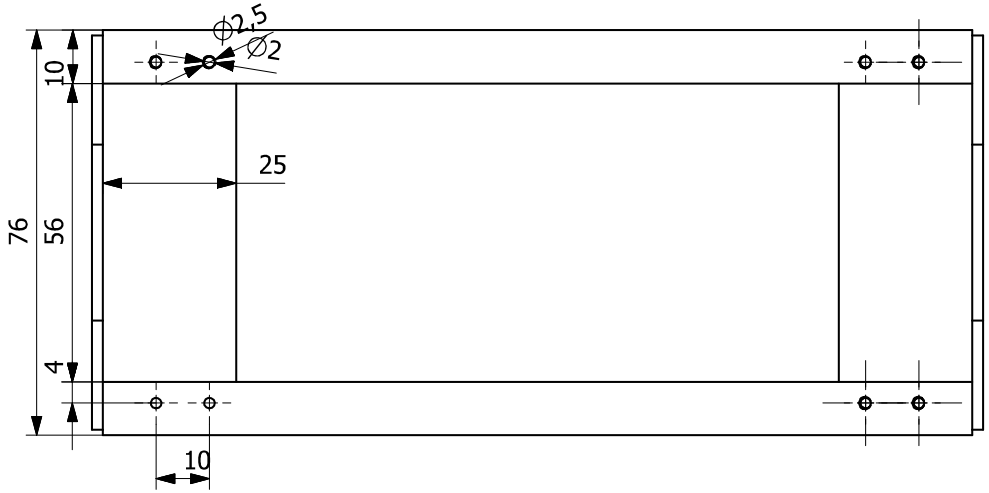


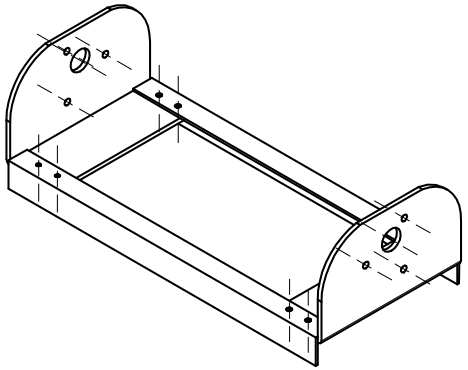


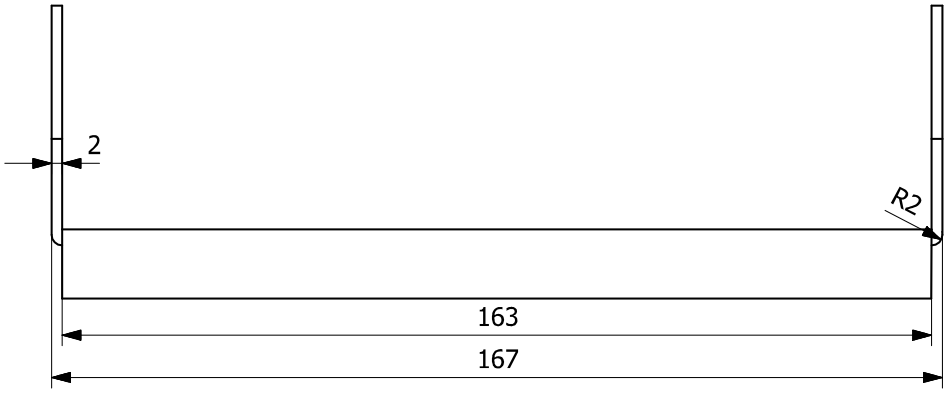
SECTION A-A

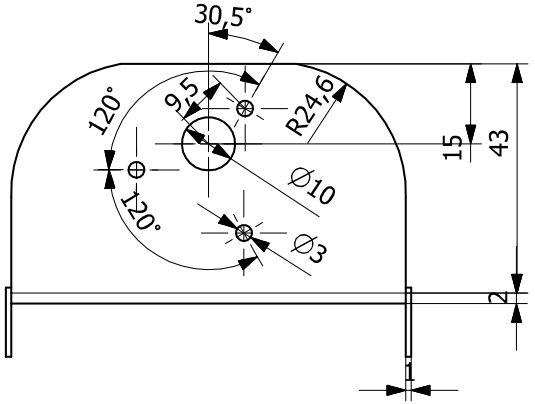
	Unidades	Nombre	
Dibujado	mm	José Ángel Garrido Sarasol	ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DEL DISEÑO
Departamento	DISA		
Máster	MÁSTER UNIVERSITARIO EN NGENIERÍA MECATRÓNICA		
Escala 1:2	PLANO 1_B CHASIS		DISEÑO E IMPLEMENTACIÓN DE UN VEHÍCULO AUTÓNOMO TERRESTRE CONTROLADO MEDIANTE INTERFAZ INALÁMBRICA ANDROID-ARDUINO

297.0 x 210.0



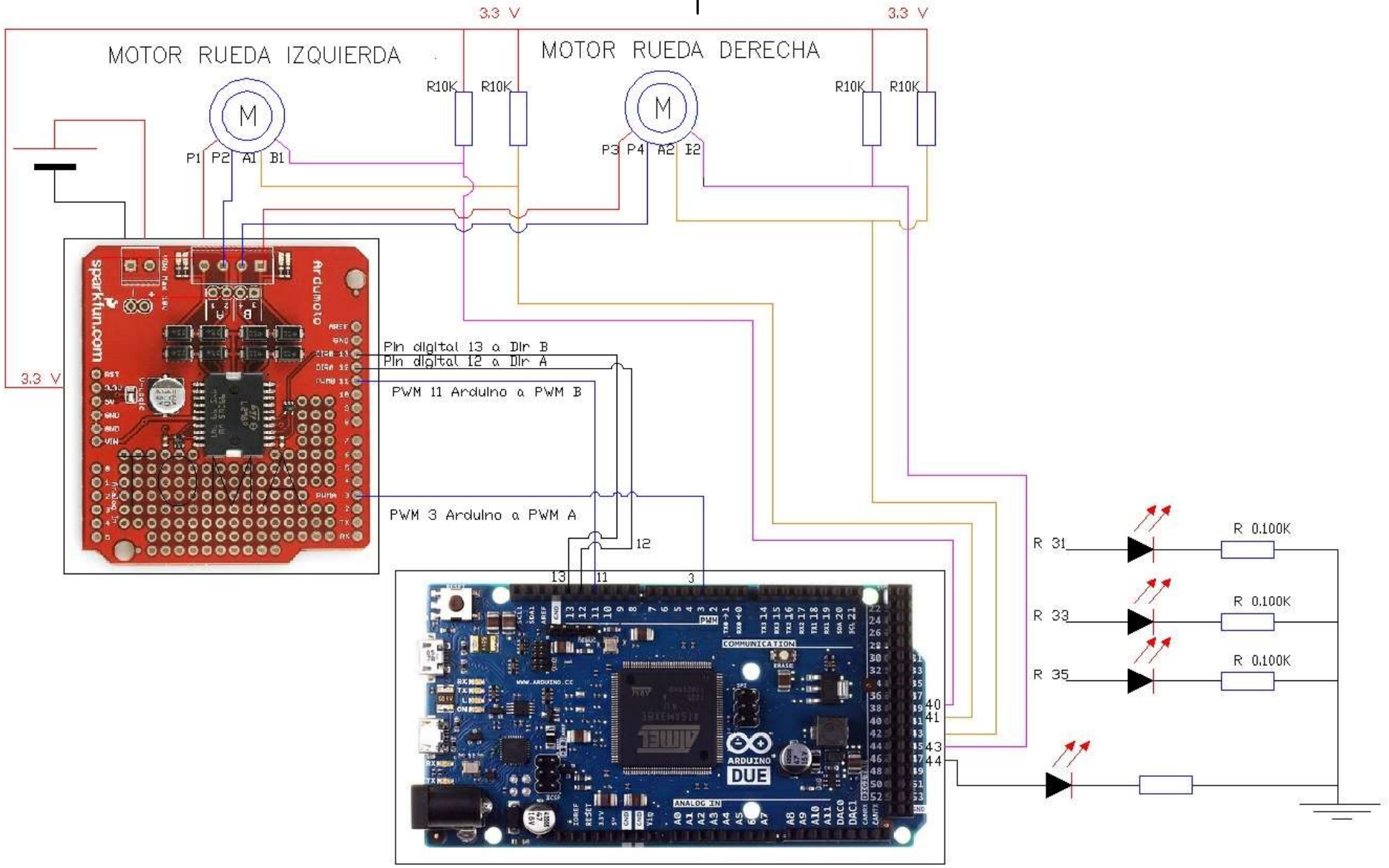






	Unidades	Nombre	
Dibujado	mm	José Ángel Garrido Sarasol	ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DEL DISEÑO
Departamento	DISA		
Máster	MÁSTER UNIVERSITARIO EN NGENIERÍA MECATRÓNICA		
Escala 1:1	PLANO 2: BASTIDOR		DISEÑO E IMPLEMENTACIÓN DE UN VEHÍCULO AUTÓNOMO TERRESTRE CONTROLADO MEDIANTE INTERFAZ INALÁMBRICA ANDROID-ARDUINO

A3



	Unidades	Nombre	
Dibujado	mm	José Ángel Garrido Sarasol	ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DEL DISEÑO
Departamento	DISA		
Máster	MÁSTER UNIVERSITARIO EN NGENIERÍA MECATRÓNICA		
Escala 1:2	PLANO 3 CONEXIONADO MOTOR-DRIVER-MICROCONTROLADOR		DISEÑO E IMPLEMENTACIÓN DE UN VEHÍCULO AUTÓNOMO TERRESTRE CONTROLADO MEDIANTE INTERFAZ INALÁMBRICA ANDROID-ARDUINO

A3