

SISTEMAS DE TIEMPO REAL

Programación de dispositivos y planificador cíclico simple



GRUPO 81

ALBERTO GRACIA GANCEDO 100275479

PABLO HEREDIA DA COSTA 100275956

CONTENIDO

[INTRODUCCIÓN 4](#_Toc372322889)

[DESCRIPCIÓN DE COMPONENTES 5](#_Toc372322890)

[DISEÑO PLANIFICADORES: ARDUINO 9](#_Toc372322891)

[Planificador A 9](#_Toc372322892)

[Planificador B 11](#_Toc372322893)

[Planificador C 13](#_Toc372322894)

[Planificador D 19](#_Toc372322895)

[CÓDIGO IMPLEMENTADO: ARDUINO 21](#_Toc372322896)

[Planificador A 21](#_Toc372322897)

[Planificador B 22](#_Toc372322898)

[Planificador C 23](#_Toc372322899)

[Planificador D 24](#_Toc372322900)

[DISEÑO PLANIFICADORES: VXWORKS 26](#_Toc372322901)

[Planificador A 26](#_Toc372322902)

[Planificador B 27](#_Toc372322903)

[Planificador C 29](#_Toc372322904)

[Planificador D 34](#_Toc372322905)

[CÓDIGO IMPLEMENTADO: VXWORKS 36](#_Toc372322906)

[Planificador A 36](#_Toc372322907)

[Planificador B 37](#_Toc372322908)

[Planificador C 37](#_Toc372322909)

[Planificador D 38](#_Toc372322910)

[PRUEBAS REALIZADAS 39](#_Toc372322911)

[CONCLUSIÓN 40](#_Toc372322912)

TABLAS

[Tabla 1: Tiempos planificador A – Arduino. 9](#_Toc372318585)

[Tabla 2: Tiempos planificador B – Arduino. 11](#_Toc372318586)

[Tabla 3: Tiempos planificador C – Modo selección – Arduino. 13](#_Toc372318587)

[Tabla 4: Tiempos planificador C – Modo acercamiento – Arduino. 15](#_Toc372318588)

[Tabla 5: Tiempos planificador C – Modo parada - Arduino. 17](#_Toc372318589)

[Tabla 6: Tiempos planificador D – Arduino. 19](#_Toc372318590)

[Tabla 7: Tiempos planificador A – VxWorks. 26](#_Toc372318591)

[Tabla 8: Tiempos planificador B – VxWorks. 27](#_Toc372318592)

[Tabla 9: Tiempos planificador C – Modo normal - VxWorks. 29](#_Toc372318593)

[Tabla 10: Tiempos planificador C – Modo frenado - VxWorks. 31](#_Toc372318594)

[Tabla 11: Tiempos planificador C – Modo descarga - VxWorks. 33](#_Toc372318595)

[Tabla 12: Tiempos planificador D – VxWorks. 34](#_Toc372318596)

ILUSTRACIONES

[Ilustración 1: Arduino UNO. 5](#_Toc372318620)

[Ilustración 2: Placa de inserción. 5](#_Toc372318621)

[Ilustración 3: Leds. 6](#_Toc372318622)

[Ilustración 4: Resistencias. 6](#_Toc372318623)

[Ilustración 5: Switch de 3 posiciones. 7](#_Toc372318624)

[Ilustración 6: Resistencia LDR. 7](#_Toc372318625)

[Ilustración 7: Display. 7](#_Toc372318626)

[Ilustración 8: Potenciómetro. 8](#_Toc372318627)

[Ilustración 9: Botón. 8](#_Toc372318628)

[Ilustración 10: Chip 4511. 8](#_Toc372318629)

[Ilustración 11: Distribución tareas CS1 planificador A – Arduino. 10](#_Toc372318630)

[Ilustración 12: Distribución tareas CS2 planificador A – Arduino. 10](#_Toc372318631)

[Ilustración 13: Distribución tareas CS1 planificador B – Arduino. 12](#_Toc372318632)

[Ilustración 14: Distribución tareas CS2 planificador B – Arduino. 12](#_Toc372318633)

[Ilustración 15: Distribución tareas CS1 planificador C – Modo selección distancia - Arduino. 14](#_Toc372318634)

[Ilustración 16: Distribución tareas CS2 planificador C – Modo selección distancia - Arduino. 14](#_Toc372318635)

[Ilustración 17: Distribución tareas CS1 planificador C – Modo acercamiento - Arduino. 16](#_Toc372318636)

[Ilustración 18: Distribución tareas CS2 planificador C – Modo acercamiento - Arduino. 16](#_Toc372318637)

[Ilustración 19: Distribución tareas CS1 planificador C – Modo parada - Arduino. 18](#_Toc372318638)

[Ilustración 20: Distribución tareas CS2 planificador C – Modo parada - Arduino. 18](#_Toc372318639)

[Ilustración 21: Distribución tareas CS1 planificador D – Arduino. 20](#_Toc372318640)

[Ilustración 22: Distribución tareas CS2 planificador D – Arduino. 20](#_Toc372318641)

[Ilustración 23: Distribución tareas planificador A – VxWorks. 27](#_Toc372318642)

[Ilustración 24: Distribución tareas CS1 planificador B – VxWorks. 28](#_Toc372318643)

[Ilustración 25: Distribución tareas CS2 planificador B – VxWorks. 29](#_Toc372318644)

[Ilustración 26: Distribución tareas CS1 planificador C – Modo normal – VxWorks. 30](#_Toc372318645)

[Ilustración 27: Distribución tareas CS2 planificador C – Modo normal – VxWorks. 31](#_Toc372318646)

[Ilustración 28: Distribución tareas CS1 planificador C – Modo frenado – VxWorks. 32](#_Toc372318647)

[Ilustración 29: Distribución tareas CS2 planificador C – Modo frenado – VxWorks. 33](#_Toc372318648)

[Ilustración 30: Distribución tareas planificador C – Modo descarga – VxWorks. 34](#_Toc372318649)

[Ilustración 31: Distribución tareas planificador D – VxWorks. 35](#_Toc372318650)

INTRODUCCIÓN

En esta primera práctica de la asignatura Sistemas de Tiempo Real se pide programar un dispositivo (Arduino Uno) e implementar un planificador cíclico simple para su correcto funcionamiento.

De esta forma se desea que dicho dispositivo simule el sistema electrónico de una carretilla, la cual está controlada por computador desde un servidor externo (VxWorks). Por tanto se pueden distinguir dos partes, una es el Arduino, y la otra el controlador VxWorks.

El primero se encarga simplemente de la parte electrónica y del cálculo de los distintos parámetros a tener en cuenta. Así, se encarga de activar y desactivar los distintos elementos (acelerador, freno, mezclador, etc.) de la carretilla en función de las peticiones recibidas del servidor externo y de facilitar la velocidad de la misma, distancia recorrida y porcentaje de luminosidad entre otras.

La parte de VxWorks es la encargada de controlar, mediante peticiones, cuándo se realiza cada acción, es decir, cuándo la carretilla debe acelerar y cuando frenar, el tiempo que debe estar el cemento en reposo y mezclando, cuando activar las luces, el modo de frenado, etc.

Para la implementación de todo el sistema se ha debido calcular la duración adecuada para cada ciclo, así como la distribución de las tareas dentro de los mismos, lo que se conoce como planificación cíclica. Para ello se ha ido haciendo un desarrollo incremental dividido en distintos apartados, los cuales se describen a lo largo de la presente memoria.

DESCRIPCIÓN DE COMPONENTES

En primer lugar se va a describir los principales componentes electrónicos necesarios y empleados para el desarrollo de esta primera práctica.

* Arduino Uno: Plataforma de prototipos electrónica de código abierto (open-source) basada en un hardware y software flexibles y fáciles de usar. En esta práctica simulará el sistema electrónico de una carretilla.

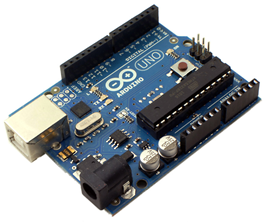


Ilustración 1: Arduino UNO.

* Placa de inserción: También llamada placa de prototipos o “protoboard”, consiste en un tablero con múltiples orificios conectados eléctricamente entre sí siguiendo patrones de líneas. En la práctica nos servirá para insertar los distintos componentes electrónicos, que se explican más abajo, y los cables para establecer las distintas conexiones.

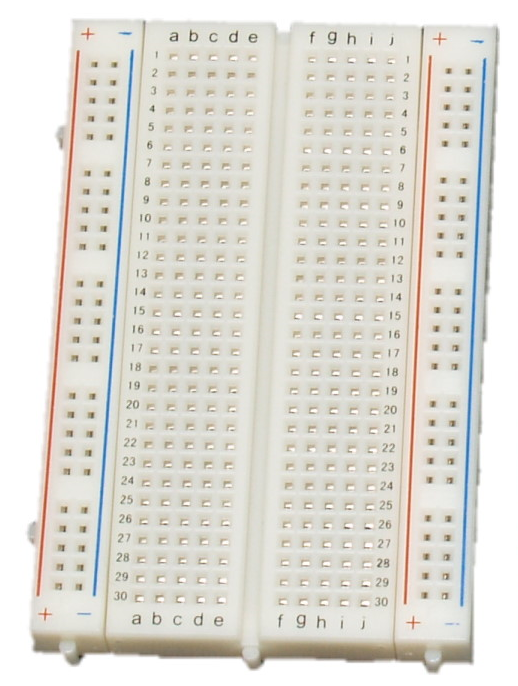


Ilustración 2: Placa de inserción.

* Leds: Se trata de diodos que emiten luz en los cuáles la patilla más larga será el ánodo (electrón positivo) y la patilla más corta el cátodo (electrón negativo). En la práctica son empleados para indicar el estado de la carretilla. De esta forma tenemos:
  + Led verde: Estado del acelerador.
  + Led rojo: Estado del freno.
  + Led amarillo: Estado del mezclador.
  + Led azul: Control de velocidad.
  + Led blanco: Focos.



Ilustración 3: Leds.

* Resistencias: También llamadas resistores, son componentes electrónicos diseñados para introducir una resistencia eléctrica determinada (vendrá dada por las distintas bandas de colores de la misma) entre dos puntos de un circuito electrónico. En la práctica se han utilizado las siguientes resistencias:
  + 220Ω ± 5%
  + 10kΩ ± 5%
  + 1,5kΩ ± 5%

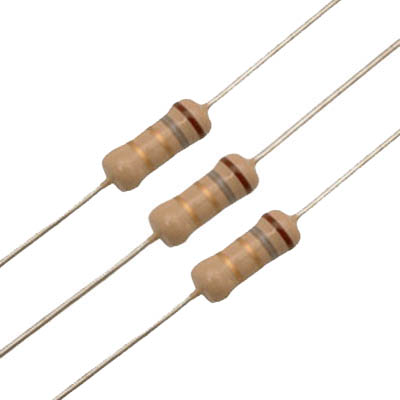


Ilustración 4: Resistencias.

* Switch de 3 posiciones: En la práctica se emplea para determinar cuál es el terreno por el que se mueve la carretilla, pudiendo ser llano, pendiente ascendente o pendiente descendente.

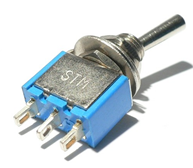


Ilustración 5: Switch de 3 posiciones.

* Resistencia fotosensible (LDR): En la práctica nos sirve para saber si la carretilla está o no dentro de una cueva, es decir, si la resistencia detecta luz es que está en el exterior y por el contrario, si se tapa el sensor, estará en una cueva.



Ilustración 6: Resistencia LDR.

* Display: Se trata de una pantalla LED numérica, en nuestro caso, de un solo dígito. En la práctica se emplea en dos tareas. En la primera de ellas, sirve para mostrar las posibles distancias a seleccionar del próximo depósito en el que se realizará una descarga, y en la otra para ver cuál es la distancia que queda a dicho depósito.

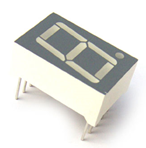


Ilustración 7: Display.

* Potenciómetro: Este componente se emplea para establecer la distancia al próximo depósito.



Ilustración 8: Potenciómetro.

* Botón: Con este componente se selecciona la distancia establecida mediante el potenciómetro y mostrada por el display. De esta forma se establece dicha distancia como la próxima parada a realizar por la carretilla.



Ilustración 9: Botón.

* Chip 4511: Se utiliza para convertir el número binario a señales del display de 7 segmentos descrito anteriormente.



Ilustración 10: Chip 4511.

DISEÑO PLANIFICADORES: ARDUINO

Planificador A

En la siguiente tabla se representan las tareas que son llevadas a cabo por el sistema electrónico de la carretilla (Arduino) en el apartado A, así como sus tiempos. En ella tenemos que T es el período de activación, D es el plazo o tiempo de respuesta y C es el tiempo de ejecución máximo o tiempo de cómputo.

|  |  |  |
| --- | --- | --- |
| **Tareas** | **T/D (ms)** | **C (μs)** |
| Servidor de Comunicaciones | 200 | 352 |
| Act./Desact. Acelerador | 100 | 16 |
| Act./Desact. Freno | 100 | 16 |
| Act./Desact. Mixer | 100 | 16 |
| Calcular y mostrar Velocidad | 100 | 112 |
| Leer Pendiente | 100 | 20 |
| **Uso CPU** | **0,00356** | |

Tabla 1: Tiempos planificador A – Arduino.

El uso de la CPU (U) viene dado por la siguiente fórmula:

De esta forma se tiene que el uso de la CPU es 0,00356 el cual es menor de 1. Esto significa que la planificación puede realizarse.

En primer lugar se calcula el ciclo principal (TM) o hiperperiodo (H) mediante la fórmula:

Así se tiene que el ciclo principal es m.c.m (200, 100, 100, 100, 100, 100) = 200ms.

El ciclo secundario (TS) será de 100ms.

Por tanto se tendrán:

Para ver que la planificación realizada es posible se muestra la distribución de las distintas tareas en ambos ciclos secundarios:

NOTA. Como los tiempos de cómputo son demasiado reducidos, si se muestra la gráfica con los 100ms del ciclo secundario, la distribución de las distintas tareas no sería apreciable. Es por ello que en la gráfica sólo se muestra 1ms de los 100ms.

Ciclo secundario 1 (CS1):



Ilustración 11: Distribución tareas CS1 planificador A – Arduino.

Ciclo secundario 2 (CS2):



Ilustración 12: Distribución tareas CS2 planificador A – Arduino.

Planificador B

Para este apartado se tienen dos nuevas tareas, además de las anteriores. A continuación se presentan todas ellas junto con sus tiempos:

|  |  |  |
| --- | --- | --- |
| **Tareas** | **T/D (ms)** | **C (μs)** |
| Servidor de Comunicaciones | 200 | 352 |
| Act./Desact. Acelerador | 100 | 16 |
| Act./Desact. Freno | 100 | 16 |
| Act./Desact. Mixer | 100 | 16 |
| Calcular y mostrar Velocidad | 100 | 112 |
| Leer Pendiente | 100 | 20 |
| Leer sensor de luminosidad | 100 | 256 |
| Act./Desact. Focos | 100 | 16 |
| **Uso CPU** | **0,00628** | |

Tabla 2: Tiempos planificador B – Arduino.

Calculamos el uso de CPU (U):

Por tanto la planificación puede realizarse. De esta forma tenemos que el ciclo principal (TM) o hiperperiodo (H) es:

El ciclo secundario (TS) será de 100ms. Por tanto el número de ciclos secundarios será 2.

En la siguiente gráfica se muestra una posible distribución de las distintas tareas para cada ciclo secundario:

NOTA. Como los tiempos de cómputo son demasiado reducidos, si se muestra la gráfica con los 100ms del ciclo secundario, la distribución de las distintas tareas no sería apreciable. Es por ello que en la gráfica sólo se muestra 1ms de los 100ms.

Ciclo secundario 1 (CS1):



Ilustración 13: Distribución tareas CS1 planificador B – Arduino.

Ciclo secundario 2 (CS2):



Ilustración 14: Distribución tareas CS2 planificador B – Arduino.

Planificador C

Para la planificación de este apartado se tienen tres planificadores distintos puesto que se pide implementar tres modos de operación de la carretilla. Estos modos son modo selección de distancia, modo acercamiento al depósito y modo parada. A continuación se describe la planificación llevada a cabo para cada uno de los mismos:

* Modo selección de distancia:

|  |  |  |
| --- | --- | --- |
| **Tareas** | **T/D (ms)** | **C (μs)** |
| Servidor de Comunicaciones | 200 | 352 |
| Act./Desact. Acelerador | 100 | 16 |
| Act./Desact. Freno | 100 | 16 |
| Act./Desact. Mixer | 100 | 16 |
| Calcular y mostrar Velocidad | 100 | 112 |
| Leer Pendiente | 100 | 20 |
| Leer sensor de luminosidad | 100 | 256 |
| Act./Desact. Focos | 100 | 16 |
| Selector de la distancia | 100 | 216 |
| Mostrar distancia seleccionada | 100 | 24 |
| Validar distancia seleccionada | 100 | 12 |
| **Uso CPU** | **0,0088** | |

Tabla 3: Tiempos planificador C – Modo selección – Arduino.

Lo primero que hacemos es calcular el uso de CPU (U):

Por tanto la planificación puede realizarse. De esta forma tenemos que el ciclo principal (TM) o hiperperiodo (H) es:

El ciclo secundario (TS) será de 100ms. Por tanto el número de ciclos secundarios será 2.

En la siguiente gráfica se muestra una posible distribución de las distintas tareas para cada ciclo secundario:

Ciclo secundario 1 (CS1): Se muestra 1ms de los 100ms que dura el ciclo.



Ilustración 15: Distribución tareas CS1 planificador C – Modo selección distancia - Arduino.

Ciclo secundario 2 (CS2): Se muestran 2ms de los 100ms que dura el ciclo.



Ilustración 16: Distribución tareas CS2 planificador C – Modo selección distancia - Arduino.

* Modo acercamiento:

|  |  |  |
| --- | --- | --- |
| **Tareas** | **T/D (ms)** | **C (μs)** |
| Servidor de Comunicaciones | 200 | 352 |
| Act./Desact. Acelerador | 100 | 16 |
| Act./Desact. Freno | 100 | 16 |
| Act./Desact. Mixer | 100 | 16 |
| Calcular y mostrar Velocidad | 100 | 112 |
| Leer Pendiente | 100 | 20 |
| Leer sensor de luminosidad | 100 | 256 |
| Act./Desact. Focos | 100 | 16 |
| Mostrar distancia actual | 100 | 124 |
| **Uso CPU** | **0,00752** | |

Tabla 4: Tiempos planificador C – Modo acercamiento – Arduino.

Lo primero que hacemos es calcular el uso de CPU (U):

Por tanto la planificación puede realizarse. De esta forma tenemos que el ciclo principal (TM) o hiperperiodo (H) es:

El ciclo secundario (TS) será de 100ms. Por tanto el número de ciclos secundarios será 2.

En la siguiente gráfica se muestra una posible distribución de las distintas tareas para cada ciclo secundario:

NOTA. Como los tiempos de cómputo son demasiado reducidos, si se muestra la gráfica con los 100ms del ciclo secundario, la distribución de las distintas tareas no sería apreciable. Es por ello que en la gráfica sólo se muestra 1ms de los 100ms.

Ciclo secundario 1 (CS1):



Ilustración 17: Distribución tareas CS1 planificador C – Modo acercamiento - Arduino.

Ciclo secundario 2 (CS2):



Ilustración 18: Distribución tareas CS2 planificador C – Modo acercamiento - Arduino.

* Modo parada:

|  |  |  |
| --- | --- | --- |
| **Tareas** | **T/D (ms)** | **C (μs)** |
| Servidor de Comunicaciones | 200 | 352 |
| Act./Desact. Acelerador | 100 | 16 |
| Act./Desact. Freno | 100 | 16 |
| Act./Desact. Mixer | 100 | 16 |
| Calcular y mostrar Velocidad | 100 | 112 |
| Leer Pendiente | 100 | 20 |
| Leer sensor de luminosidad | 100 | 256 |
| Act./Desact. Focos | 100 | 16 |
| Leer fin de parada | 100 | 8 |
| **Uso CPU** | **0,00636** | |

Tabla 5: Tiempos planificador C – Modo parada - Arduino.

Lo primero que hacemos es calcular el uso de CPU (U):

Por tanto la planificación puede realizarse. De esta forma tenemos que el ciclo principal (TM) o hiperperiodo (H) es:

El ciclo secundario (TS) será de 100ms. Por tanto el número de ciclos secundarios será 2.

En la siguiente gráfica se muestra una posible distribución de las distintas tareas para cada ciclo secundario:

NOTA. Como los tiempos de cómputo son demasiado reducidos, si se muestra la gráfica con los 100ms del ciclo secundario, la distribución de las distintas tareas no sería apreciable. Es por ello que en la gráfica sólo se muestra 1ms de los 100ms.

Ciclo secundario 1 (CS1):



Ilustración 19: Distribución tareas CS1 planificador C – Modo parada - Arduino.

Ciclo secundario 2 (CS2):



Ilustración 20: Distribución tareas CS2 planificador C – Modo parada - Arduino.

Planificador D

En este apartado se tienen las mismas tareas que para el apartado B excepto la de “Leer sensor de luminosidad”.

|  |  |  |
| --- | --- | --- |
| **Tareas** | **T/D (ms)** | **C (μs)** |
| Servidor de Comunicaciones | 200 | 352 |
| Act./Desact. Acelerador | 100 | 16 |
| Act./Desact. Freno | 100 | 16 |
| Act./Desact. Mixer | 100 | 16 |
| Calcular y mostrar Velocidad | 100 | 112 |
| Leer Pendiente | 100 | 20 |
| Act./Desact. Focos | 100 | 16 |
| **Uso CPU** | **0,00372** | |

Tabla 6: Tiempos planificador D – Arduino.

Calculamos el uso de CPU (U):

Por tanto la planificación puede realizarse. De esta forma tenemos que el ciclo principal (TM) o hiperperiodo (H) es:

El ciclo secundario (TS) será de 100ms. Por tanto el número de ciclos secundarios será 2.

En la siguiente gráfica se muestra una posible distribución de las distintas tareas para cada ciclo secundario:

NOTA. Como los tiempos de cómputo son demasiado reducidos, si se muestra la gráfica con los 100ms del ciclo secundario, la distribución de las distintas tareas no sería apreciable. Es por ello que en la gráfica sólo se muestra 1ms de los 100ms.

Ciclo secundario 1 (CS1):



Ilustración 21: Distribución tareas CS1 planificador D – Arduino.

Ciclo secundario 2 (CS2):



Ilustración 22: Distribución tareas CS2 planificador D – Arduino.

CÓDIGO IMPLEMENTADO: ARDUINO

Planificador A

A continuación se va a describir brevemente cómo se ha realizado la implementación del sketch de Arduino para el apartado A.

En primer lugar hemos definido la configuración de los pines donde van a estar conectados nuestros componentes. De esta forma, por ejemplo para el led que indica si el acelerador está activado o no, se ha utilizado el pin 13.

Después podemos encontrar todas las variables utilizadas a lo largo del código por los distintos métodos implementados. De esta forma se tiene una variable de tipo entero que almacena la velocidad de la carretilla, variables de tipo booleano que guardan el estado del acelerador, freno y mezclador, entre otras donde se almacenan los tiempos. También se declara el número de ciclos secundarios y la duración del mismo en ms (en base a los cálculos realizados en el punto anterior).

Tras esto se tienen los distintos métodos que llevan a cabo las tareas pedidas para este apartado A. Estos métodos son los siguientes:

* **Servidor de comunicaciones (comm\_server):** Este método lo que hace es recibir y dar respuesta a las distintas peticiones recibidas por el controlador. De esta forma se tratan mediante un *“if-else”* todas las posibles peticiones.
* **Act./Desact. Acelerador (accelerateFunction):** Este método simplemente se encarga de activar o desactivar el acelerador en base a la petición recibida por el servidor de comunicaciones. De esta forma decide si se activa o no el led que indica el estado del acelerador (led verde, pin 13).
* **Act./Desact. Freno (brakeFunction):** Este método tiene la misma función que el anterior pero actúa sobre el freno. Se encarga de activarlo o desactivarlo en base a la petición recibida por el servidor de comunicaciones y enciende o apaga el led que indica el estado del freno (led rojo, pin 12).
* **Act./Desact. Mezclador (mixFunction):** Para este método la implementación también es muy parecida a los dos anteriores. En función a la petición captada por el servidor de comunicaciones activa o no el mezclador, encendiendo o apagando el led asociado al mismo (led amarillo, pin 11).
* **Leer Pendiente (readSlopeFunction):** Este método lee de la entrada digital la posición del switch (posición 8 ó 9) y determina cual es el terreno por el que va la carretilla. Si el switch está en la posición 8 quiere decir que avanza por una pendiente descendente, si está en la posición 9 será una pendiente ascendente y si no es ninguna de las anteriores el terreno será llano. Dada la lógica aplicada, se ha empelado un *‘if-else’* para los distintos casos descritos.
* **Calcular y mostrar Velocidad (actualSpeedFunction):** Este método calcula la velocidad que lleva la carretilla en función del estado del acelerador, freno y pendiente, que determinará el tipo de aceleración experimentado por la misma, además del tiempo. Es decir, se ha empleado la fórmula del movimiento rectilíneo uniformemente acelerado (MRUA):

De esta forma simplemente se comprueba el estado de cada componente mencionado anteriormente, si está activado o no, y se sustituye en la fórmula. Así tenemos que la velocidad inicial es la obtenida en el ciclo anterior, el tiempo también vendrá dado por la duración del ciclo y la aceleración por la tabla facilitada en el guion de la práctica (+0,5 en caso de acelerar, -0,5 en caso de frenar y ±0,25 dependiendo del tipo de pendiente.

Además se tienen tres métodos más:

* **Setup**: Este método se encarga de abrir el puerto serie, estableciendo la velocidad de datos en bits por segundo (baudios) en 9600. Además se configuran los distintos pines definidos como entrada o salida mediante la función *pinMode ().*
* **Loop**: En este método simplemente se hace una llamada al planificador implementado (método scheduler), el cual se explica a continuación.
* **Scheduler**: Este método implementa la planificación llevada a cabo en el punto anterior (Ver “Diseño Planificadores: Arduino”). Es el encargado de ir avanzando de un ciclo a otro. Incluye un ‘*switch*’ en el que se contemplan los distintos ciclos secundarios y las tareas llevadas a cabo en ellos según lo planificado.

Planificador B

Para la implementación de este apartado se han mantenido todas las funcionalidades del apartado anterior y se han añadido las nuevas entradas, variables y métodos necesarios para realizar las nuevas tareas pedidas. De esta forma se ha incluido la entrada digital a la que se conecta el led que simula los focos (led blanco, pin 7) y la entrada analógica de la resistencia fotosensible LDR (pin 0). Además se añaden dos nuevas variables, una booleana que indica si los focos están apagados (false) o encendidos (true) y una entera que almacena el valor leído por el sensor de luz.

También se ha modificado el método “*comm\_server*” para incluir la petición de lectura del sensor de luz y la petición de encendido/apagado de los focos. Otro método que se ha modificado es “*setup*” para establecer como salida (OUTPUT) el nuevo pin asociado al led que simula los focos.

Se han añadido dos nuevos métodos que realizan las tareas pedidas para este segundo apartado:

* **Leer sensor de luminosidad (readSensorFunction):** En este método lo que hacemos es leer la intensidad captada por la resistencia fotosensible (de 0 a 1023) y almacenarla en una variable. Con ese valor se hace un mapeo para calcular un número entre 0 y 99 (que será el porcentaje de luminosidad) en función del rango comentado, es decir, de 0 a 1023.
* **Act./Desact. Focos (activateFocus):** Este método comprueba el estado de los focos, es decir, si están activados o no. En base a esto último se encarga de encender o apagar el led asociado (led blanco, pin 7).

Por último se han añadido dichos métodos al planificador (“*scheduler*”) para que sean llevados a cabo por la carretilla.

Planificador C

Para este apartado, además de todo lo implementado hasta el momento, se pide simular tres modos en los que puede encontrarse la carretilla. Para que la explicación sea más legible se detallarán los nuevos métodos implementados según dichos modos:

* Modo selección de distancia: En este modo se tienen tres métodos nuevos.
  + **Selector de la distancia (distanceSelectorFunction):** Este método simplemente lee de la entrada analógica donde se encuentra conectado el potenciómetro y hace un mapeo de dicho valor entre 10000 y 90000, que son las posibles distancias a seleccionar del próximo depósito. También se divide esta distancia calculada entre 10000 y se convierte a entero para obtener un número (de 1 a 9) que se pueda representar posteriormente en el display.

NOTA: Para escoger los valores mínimo y máximo de la función *map ()* se han imprimido los valores ofrecidos por el potenciómetro en sus dos “topes”, es decir, totalmente girado a la derecha y totalmente girado a la izquierda.

* + **Mostrar distancia seleccionada (distanceDisplayFunction):** Este método simplemente llama al método explicado al final es este apartado *“numberDisplayFunction”*, que recibe como parámetro la distancia seleccionada en el método anterior.
  + **Validar distancia seleccionada (distanceValidatorFunction):** En este método se comprueba el estado del botón o pulsador. Si se ha pulsado (HIGH) entonces guardamos la distancia seleccionada anteriormente y cambiamos del modo de selección de distancia a modo de acercamiento.
* Modo acercamiento: Para este modo se ha implementado un nuevo método.
  + **Mostrar distancia actual (actualDistanceFunction):** En este método se hacen las correspondiente medidas de tiempo necesarias para calcular el espacio o distancia recorrida en función de la velocidad que lleva la carretilla en ese instante (MRU). Por tanto se utiliza la fórmula:

Una vez calculada la distancia, se divide entre 10000 para poder mostrarla por el display y se llama al método *“numberDisplayFunction”*, que recibe como parámetro dicha distancia. Se comprueba si la distancia es igual a 0, puesto que cuando ocurre esto, acaba el modo de acercamiento. Si se cumple se cambiará al modo de parada si la velocidad en ese momento es menor que 10m/s o al modo de selección de distancia si la velocidad es mayor o igual a 10m/s.

* Modo parada: En este modo se tiene un nuevo método.
  + **Leer fin de parada (readEndStop):** En este método simplemente volvemos a comprobar el estado del botón. Si está en HIGH significará que lo hemos pulsado para salir del modo parada por lo que volvemos al modo normal. Si no se cumple, es decir, el estado del botón es LOW, es porque todavía no le hemos pulsado para salir del modo parada. En este caso no se cambia de modo y por tanto se sigue en el mismo planificador.

Además de todos los métodos anteriores se han añadido los pines de los nuevos componentes utilizados (potenciómetro: A1, display: D2/D3/D4/D5, botón: D6) y las nuevas variables utilizadas. También se ha modificado el método *“setup ()”* para configurar dichos pines como entrada o salida y el método *“actualSpeedFunction ()”* para que si se está en modo parada la velocidad sea igual a 0.

También se ha implementado un método (numberDisplayFunction) que recibe la distancia entre 1 y 9, y se encarga de convertirla a binario para que el chip 4511 lo traduzca a señal válida para el display.

Por último se modificada el método “*loop ()”* para que según el modo en que se esté, se ejecute un planificador u otro. Y se implementan los planificadores, uno por cada modo, en función de los cálculos realizados en el apartado anterior.

Planificador D

Para la implementación de las funcionalidades pedidas en este apartado no tenemos ningún método nuevo. Simplemente se han realizado pequeñas modificaciones en algunos de los existentes:

* En el método “*loop ()”* añadimos un caso más a los existentes para lanzar el planificador correspondiente al nuevo modo de emergencia.
* En el método “*comm\_server ()*” se añade la posibilidad de nueva petición para entrar en modo emergencia. De esta forma si recibimos dicho mensaje desactivamos el acelerador, activamos el freno y activamos las luces. Además se cambia a modo de emergencia para lanzar su planificador. Dicho planificador lleva a cabo las mismas funciones que el implementado en el apartado B anterior, excepto “Leer sensor luminosidad”.

DISEÑO PLANIFICADORES: VXWORKS

Planificador A

A continuación se muestran en una tabla los tiempos asociados a las distintas tareas que debe llevar a cabo el controlador en este primer apartado:

|  |  |  |
| --- | --- | --- |
| **Tareas** | **T/D (s)** | **C (s)** |
| Leer Pendiente | 10 | 0,9 |
| Leer Velocidad | 10 | 0,9 |
| Encender/Apagar Acelerador | 10 | 0,9 |
| Encender/Apagar Freno | 10 | 0,9 |
| Encender/Apagar Mixer | 15 | 0,9 |
| **Uso CPU** | **0,42** | |

Tabla 7: Tiempos planificador A – VxWorks.

Para ver si la planificación puede llevarse a cabo se calcula el uso de la CPU (U):

Como se cumple podemos seguir. Ahora vemos si se puede reducir el período de la tarea “Encender/Apagar Mixer” para poder utilizar el método de los armónicos para calcular el ciclo secundario. Para ello volvemos a calcular el uso de la CPU para el nuevo período de la tarea mencionada anteriormente.

Por tanto se puede reducir a 10s el período de la tarea “Encender/Apagar Mixer”.

Ahora vamos a calcular el ciclo principal:

Por tanto el ciclo principal ™ es de 10s. El ciclo secundario (TS) será igual que el principal, es decir, 10s. El número de ciclos secundario será lógicamente 1.

De esta forma se tiene que la distribución de tareas en dicho ciclo es la siguiente:



Ilustración 23: Distribución tareas planificador A – VxWorks.

Planificador B

En este segundo apartado se tienen dos tareas más que en el anterior. Dichas tareas junto con las anteriores se muestran en la siguiente tabla:

|  |  |  |
| --- | --- | --- |
| **Tareas** | **T/D (s)** | **C (s)** |
| Leer Pendiente | 10 | 0,9 |
| Leer Velocidad | 10 | 0,9 |
| Encender/Apagar Acelerador | 10 | 0,9 |
| Encender/Apagar Freno | 10 | 0,9 |
| Encender/Apagar Mixer | 15 | 0,9 |
| Leer sensor luminosidad | 6 | 0,9 |
| Encender/Apagar Focos | 6 | 0,9 |
| **Uso CPU** | **0,72** | |

Tabla 8: Tiempos planificador B – VxWorks.

Lo primero se calcula el uso de la CPU (U) para ver si el modelo es correcto:

Como se cumple seguimos con la planificación. Vamos a ver si podemos reducir el período de la tarea “Encender/Apagar Mixer” a 10s y de las tareas “Leer sensor luminosidad” y “Encender/Apagar Focos” a 5s para poder calcular por armónicos el ciclo secundario. Por tanto calculamos de nuevo el uso de CPU (U):

Como se cumple podemos hacerlo. De esta forma se tiene que el ciclo principal es:

Por tanto el ciclo principal ™ es de 10s. El ciclo secundario (TS) será de 5s. El número de ciclos secundario será de 2.

La distribución de tareas para cada uno de los ciclos será la siguiente:

Ciclo secundario 1 (CS1):



Ilustración 24: Distribución tareas CS1 planificador B – VxWorks.

Ciclo secundario 2 (CS2):



Ilustración 25: Distribución tareas CS2 planificador B – VxWorks.

Planificador C

Al igual que para la parte de Arduino, se tienen tres modos de operación (modo normal, modo frenado y modo parada), cada uno con sus tareas y por tanto con una planificación distinta. De esta forma se detallan los tres planificadores según dicho modos:

* Modo normal: Para este modo tenemos las siguientes tareas.

|  |  |  |
| --- | --- | --- |
| **Tareas** | **T/D (s)** | **C (s)** |
| Leer Pendiente | 10 | 0,9 |
| Leer Velocidad | 10 | 0,9 |
| Leer Distancia | 10 | 0,9 |
| Encender/Apagar Acelerador | 10 | 0,9 |
| Encender/Apagar Freno | 10 | 0,9 |
| Encender/Apagar Mixer | 15 | 0,9 |
| Leer sensor luminosidad | 6 | 0,9 |
| Encender/Apagar Focos | 6 | 0,9 |
| **Uso CPU** | **0,81** | |

Tabla 9: Tiempos planificador C – Modo normal - VxWorks.

El uso de la CPU es inicialmente:

Por tanto puede realizarse la planificación. Vemos si podemos reducir el período de la tarea “Encender/Apagar Mixer” a 10s y el de las tareas “Leer sensor luminosidad” y “Encender/Apagar Focos” a 5s para poder calcular el ciclo secundario por armónicos:

Como podemos comprobar se puede hacer. Ahora calculamos el ciclo principal:

Por tanto el ciclo principal será de 10s y el ciclo secundario de 5s. El número de ciclos secundarios serán de 2. De esta forma se tiene la siguiente distribución de tareas en cada ciclo secundario:

* Ciclo secundario 1 (CS1):



Ilustración 26: Distribución tareas CS1 planificador C – Modo normal – VxWorks.

* Ciclo secundario 2 (CS2):



Ilustración 27: Distribución tareas CS2 planificador C – Modo normal – VxWorks.

* Modo frenado:

|  |  |  |
| --- | --- | --- |
| **Tareas** | **T/D (s)** | **C (s)** |
| Leer Pendiente | 10 | 0,9 |
| Leer Velocidad | 5 | 0,9 |
| Leer Distancia | 10 | 0,9 |
| Encender/Apagar Acelerador | 5 | 0,9 |
| Encender/Apagar Freno | 5 | 0,9 |
| Encender/Apagar Mixer | 15 | 0,9 |
| Encender Focos | 30 | 0,9 |
| **Uso CPU** | **0,81** | |

Tabla 10: Tiempos planificador C – Modo frenado - VxWorks.

El uso de la CPU es inicialmente:

Por tanto puede realizarse la planificación. Comprobamos si podemos reducir el período de las tareas “Encender/Apagar Mixer” y “Encender Focos” a 10s para poder calcular el ciclo secundario por armónicos:

Como podemos comprobar se puede hacer. Ahora calculamos el ciclo principal:

Por tanto el ciclo principal será de 10s y el ciclo secundario de 5s. El número de ciclos secundarios serán de 2. De esta forma se tiene la siguiente distribución de tareas en cada ciclo secundario:

* Ciclo secundario 1 (CS1):



Ilustración 28: Distribución tareas CS1 planificador C – Modo frenado – VxWorks.

* Ciclo secundario 2 (CS2):



Ilustración 29: Distribución tareas CS2 planificador C – Modo frenado – VxWorks.

* Modo descarga:

|  |  |  |
| --- | --- | --- |
| **Tareas** | **T/D (s)** | **C (s)** |
| Leer fin de descarga | 5 | 0,9 |
| Encender/Apagar Mixer | 15 | 0,9 |
| Encender Focos | 5 | 0,9 |
| **Uso CPU** | **0,42** | |

Tabla 11: Tiempos planificador C – Modo descarga - VxWorks.

Se tiene que el uso de la CPU inicialmente es el siguiente:

Por tanto puede realizarse la planificación. Comprobamos si podemos reducir el período de la tarea “Encender/Apagar Mixer” a 5s:

Por tanto se puede hacer. Ahora calculamos el ciclo principal:

Por tanto el ciclo principal será de 5s, al igual que el ciclo secundario. De esta forma el número de ciclos secundarios será de 1. De esta forma se tiene la siguiente distribución de tareas:



Ilustración 30: Distribución tareas planificador C – Modo descarga – VxWorks.

Planificador D

En este apartado tenemos las siguientes tareas:

|  |  |  |
| --- | --- | --- |
| **Tareas** | **T/D (s)** | **C (s)** |
| Leer Pendiente | 10 | 0,9 |
| Leer Velocidad | 10 | 0,9 |
| Encender/Apagar Acelerador | 10 | 0,9 |
| Encender/Apagar Freno | 10 | 0,9 |
| Encender/Apagar Mixer | 15 | 0,9 |
| Encender Focos | 10 | 0,9 |
| Activar modo emergencia | 10 | 0,9 |
| **Uso CPU** | **0,6** | |

Tabla 12: Tiempos planificador D – VxWorks.

Calculamos el uso de la CPU (U):

Por tanto puede realizarse la planificación. Comprobamos si podemos reducir el período de la tarea “Encender/Apagar Mixer” a 10s para poder calcular el ciclo secundario por armónicos:

Como podemos comprobar se puede hacer. Ahora calculamos el ciclo principal:

Por tanto el ciclo principal será de 10s, al igual que el ciclo secundario. De esta forma tenemos un único ciclo secundario, en el que se define la siguiente distribución de tareas:



Ilustración 31: Distribución tareas planificador D – VxWorks.

CÓDIGO IMPLEMENTADO: VXWORKS

Planificador A

Para empezar se definen los elementos principales para llevar a cabo la planificación en base a los cálculos anteriores. Estos elementos son el número de ciclos secundarios y su duración tanto en segundos como en nanosegundos.

Después se tienen las distintas variables globales usadas por los métodos implementados. Para este apartado se tiene una variable para guardar la velocidad, otra para el estado del mezclador (activado o no) y otra para el tiempo de mezclado. También se declaran las variables para los mensajes tanto de petición como de respuesta.

A continuación vamos a ver los distintos métodos implementados (recordamos que se han realizado en base a los proporcionados):

* **Leer velocidad (readSpeedFunction):** Este método manda a la carretilla una petición que se corresponde con la consulta de la velocidad actual. Se recibe la respuesta y se muestra por el display (consola).
* **Leer pendiente (readSlopeFunction):** Este método manda la petición correspondiente a la consulta de la pendiente. Se recibe el mensaje de respuesta y dependiendo del tipo de pendiente leída se muestra el dígito correspondiente por consola.
* **Encender/Apagar acelerador (acceleratorControlFunction):** Este método se encarga de mandar la petición para encender el acelerador o apagarlo en función de la velocidad leída por el método anterior. Si esta es mayor de 55m/s se manda petición para activar el acelerador, por el contrario, si es menor de 55m/s, se manda desactivarlo. Además dependiendo del caso se muestra un 0 ó un 1 por la consola.
* **Encender/Apagar freno (brakeControlFunction):** Este método opera de forma similar al anterior pero con las peticiones correspondiente para activar o desactivar el freno.
* **Encender/Apagar mezclador (mixerControlFunction):** En este método se obtiene el tiempo real actual y se calcula la diferencia con el tiempo en el que se empezó a mezclar o a reposar el cemento. En base a esta diferencia se decide activar o no el mezclador. Como se pide que esté tanto activado como desactivado entre 30s y 60s, hemos contemplado que se actúe en cuanto dicha diferencia supere los 30s para mandar la petición de activado, si el estado actual es desactivado, y la petición de desactivado si el estado actual es activado.
* **Controller:** Este método es el que implementa la planificación llevada a cabo en el punto anterior. Para este apartado sólo se tiene un único ciclo secundario en el que se ejecutan todos los métodos descritos anteriormente.

Planificador B

En este apartado, además de toda la implementación anterior, se añaden nuevas variables y métodos para las tareas pedidas, además de realizar modificaciones en la planificación.

De esta forma y en función a los cálculos realizados, se definen como número de ciclos secundarios 2, y como duración de dicho ciclo 5s. Se añade también una variable para el almacenar la cantidad de luz (en porcentaje) recibida mediante la petición correspondiente.

A continuación se explican brevemente los nuevos métodos implementados:

* **Leer sensor luminosidad (readSensorFunction):** Este método envía una petición para conocer la cantidad de luz leída por el sensor de la carretilla. Si dicho porcentaje es mayor o igual a 50 entonces se muestra por consola que no hay oscuridad. Por el contrario, si el porcentaje es menor de 50, se muestra que está oscuro.
* **Encender/Apagar focos (focusControlFunction):** En este método se manda la petición de encender o apagar los focos de la carretilla en función del porcentaje de luz leído en el método anterior. Si es mayor o igual de la mitad se mandará petición para apagarlos. Si es menor de la mitad se manda petición para encenderlos. Además se muestra por consola el estado de los focos (0 apagados, 1 encendidos).

Además se modifica el método *“controller ()”* para implementar el nuevo planificador. El modo de avanzar entre los distintos ciclos secundarios es similar al del apartado anterior. Ahora se tienen dos ciclos secundarios entre los que se distribuye la ejecución de los distintos métodos en base a la planificación realizada en el punto anterior.

Planificador C

En este apartado, partiendo de toda la base anterior ya creada, se tiene como objetivo descargar el cemento de la carretilla en un depósito. Para ello se distinguen tres modos de estado de la carretilla:

* Modo normal: la carretilla continuará el mismo sistema de ejecución que en el apartado B explicado anteriormente. Para ello se ha creado un método parecido al *“controller ()”* de los apartados anteriores, que como hemos mencionado gestiona las mismas funcionalidades que en el caso del apartado B.
* Modo frenado: la carretilla comenzará a frenar cuando la distancia al punto de descarga sea menor de 11000 metros.
* Modo parada: es el lugar donde la carretilla descarga la carga que lleva.

NOTA: Los 11000 metros, distancia a la que se comienza a frenar, se obtienen tras realizar una serie de cálculos de movimiento rectilíneo uniformemente acelerado (MRUA), suponiendo la situación del peor caso. Los cálculos determinan que la distancia en el peor de los casos de frenada es de 9600m, pero a esa cantidad hay que sumarle el margen de error a causa de la ejecución de los ciclos, lo que lleva a una distancia de error de 1400m. Por tanto se tienen los 11000m mencionados (9600 + 1400).

Además se modifican y añaden los siguientes métodos:

* **acceleratorControlFunction ():** a este método implementado anteriormente se le añade la funcionalidad de controlar también la velocidad en la zona de frenado. Dicha velocidad es de 2.5 m/s. Para no repetir lo mismo que en el caso del modo normal, cuando va a 55 m/s de media, se crea una variable global que va cambiando según el modo en el que la carretilla se encuentre.

NOTA: Nuevamente los 2.5m/s se obtienen mediante cálculos de movimiento rectilíneo uniformemente acelerado (MRUA), suponiendo el peor de los casos. Dicho cálculo se realiza para que una vez que la carretilla esté dentro de la velocidad adecuada para parar a descargar la carga y no se salga de ella.

* **focusControlFunction ():** a este método que también ha sido creado anteriormente se le añade la restricción de que las luces estén siempre encendidas si estamos en un modo que sea distinto al normal.
* **readDistanceFunction():** este método se encarga de medir constantemente la distancia para entrar en el modo normal si dicha distancia es mayor que 11000\*, en el modo de frenado si es menor o igual que 11000 y mayor que 0, o en modo de parada si la distancia es igual a 0.
* **readDischargeEndFunction():** método que realiza la función de pasar del modo de parada al modo normal si la descarga de la carretilla ha finalizado.

Para terminar se implementan los distintos planificadores en función a los cálculos hecho en el punto anterior y distribución de tareas.

Planificador D

En este apartado además de las funcionalidades anteriores tenemos un método para mandar la petición de activar el nuevo modo de emergencia (“**emergencyModeActivationFunction**”).

Además se modifican los siguientes métodos:

* **acceleratorControlFunction ():** Si se está en modo emergencia se manda petición para desactivar el acelerador.
* **brakeControlFunction ():** Si se está en modo emergencia se manda petición para activar el freno.

Para los focos ya se tiene implementado en el apartado C.

PRUEBAS REALIZADAS

Para las pruebas llevadas a cabo para comprobar el correcto funcionamiento de todos los apartados se han realizado tanto pruebas unitarias, es decir, cada parte con su propio simulador, como pruebas de integración, con el Arduino conectado al sistema VxWorks. Para las pruebas unitarias distinguimos las dos partes principales:

* Arduino: Para probar de forma individual su correcto funcionamiento, se ha utilizado el propio software ofrecido por el fabricante, y se han ido cargando los distintos sketch correspondientes a cada apartado. De esta forma, mediante el monitor serial, se han ido realizando las distintas peticiones y viendo su respuesta en el Arduino y los componentes conectados al mismo. Por ejemplo, una prueba sería poner la petición de encender el acelerador (GAS: SET) y ver que se enciende el led asociado correctamente. Además, para el ejemplo citado también se debería comprobar si la velocidad va aumentando (SPD: REQ) debido al uso del acelerador.
* VxWorks: Para probar el correcto funcionamiento de esta parte, se ha utilizado el simulador. De esta forma se ha ido viendo si lo que se muestra por el display es correcto o no y si se corresponde con el estado de los componentes conectado al Arduino. Por ejemplo, una prueba realizada ha sido ver qué ocurre si la carretilla entra en una cueva. El correcto funcionamiento implicaría que en el Arduino se enciende el led asociado a los focos (led blanco).

Además y como ya hemos mencionado, se han realizado pruebas de integración. Estas han consistido en ver la respuesta tanto del Arduino como del display mostrado por VxWorks según la ejecución del programa. Siguiendo con el ejemplo anterior de la luminosidad, una prueba es tapar la resistencia fotosensible. De esta forma el sistema remoto, al mandar la petición de leer luminosidad, y recibir el porcentaje de la misma calculado por el Arduino, mandará la orden de encender los focos de la carretilla y además en el display se mostrará que está oscuro.

CONCLUSIÓN

Con la realización de esta práctica se han aplicado en un caso real los conocimientos adquiridos en la asignatura. De esta forma se ha comprobado la importancia de los cálculos realizados sobre papel en los ejercicios hechos en clase y la correcta distribución de las distintas tareas.

Nos ha resultado una práctica entretenida ya que todo lo que se iba programando se podía ver de una forma muy directa, observando el comportamiento del sistema. Debemos decir que ha requerido bastante tiempo para su realización y ha habido algunas funcionalidades que nos han resultado algo complejas de implementar. Lo positivo es que al ser una práctica tan visual, y cada tarea llevar asociados unos componentes, la detección de los distintos errores era más fácil de llevarla a cabo. Además hemos aprendido cosas básicas sobre electrónica mediante el montaje de todo nuestro sistema.

También se ha podido ver cómo funciona un sistema de tiempo real y lo importante que es realizar de forma correcta su planificación y medir de forma exacta todos los tiempos para su correcto funcionamiento.

En resumen, y para finalizar la realización de esta primera práctica, decir que ha sido muy útil para reforzar nuestros conocimientos sobre la asignatura y entender mejor la planificación cíclica.