MEMORIA TÉCNICA

Desarrollo de sistema inmótico con plataforma Arduino

SUB-PROYECTO: Diseño de sistema de control de persianas y toldos

ÍNDICE

APARTADO **Pág.**

1.- Historia del proyecto:

Este proyecto se enmarca dentro del programa "*Estrategia Europa 2020*" que tiene como uno de sus objetivos fundamentales impulsar la Formación Profesional haciéndola más atractiva y de mayor calidad, lo que refuerza estas enseñanzas y propicia nuevas experiencias en el campo de la innovación, de la calidad, de la creatividad y del espíritu emprendedor.

Se han recibido ayudas a la innovación tecnológica, según la resolución de 5 de abril de 2011, de la Secretaría de Estado de Educación y Formación Profesional (B.O.E. Nº 100, 27 de abril de 2011).

OBJETIVOS:

1.1.- **Colaboración entre diferentes componentes de la comunidad educativa** para conseguir una educación de calidad, según promueve la Ley Orgánica 2/2006, de 3 de mayo, de Educación.

1.2.- **Impulsar y desarrollar acciones y proyectos de innovación y desarrollo**, en colaboración con una empresa, ICTEL, de acuerdo con el Real Decreto 1558/2005, de 23 de diciembre (artículo 6, punto 2b).

1.3.- **Transferir el contenido y valoración de las experiencias** desarrolladas al resto de los centros.

1.4.- **Fomentar el trabajo en equipo** del profesorado (Real Decreto 1538/2006, artículo 18, apartado 2) y la movilidad de profesores y alumnos, creando redes de centros para realizar iniciativas innovadoras dentro del sistema de formación profesional.

Se realizó una primera reunión en Madrid (I. E. S. El Burgo de Las Rozas) para la coordinación de todos los participantes.

El proyecto se divide en nueve sub-proyectos: control energético, control de iluminación, control de persianas y toldos, control de aire acondicionado, control de accesos, gestión de alarmas, control de fichajes de rondas, aplicaciones para IOS y Android y monedero electrónico.

Cada uno de los sub-proyectos estará desarrollado por un centro educativo diferente.

Los módulos utilizados para el proyecto han sido suministrados por la empresa ICTEL Ingenieros.

Los cursos de formación tienen como ponente a Juan Ramón García Vila (ICTEL).

2.- Participantes y su colaboración:

- *Coordinación del sub-proyecto.*
    - Jesús Pérez Esteban.
    - María de los Ángeles Prieto Blanca.
- *Desarrollo de software.*
    - José Antonio Tello Atance.
    - Ángel Francisco Arroyo Sánchez.
- *Recursos y equipamiento.*
    - José Antonio Ventura Bartolomé.
    - José Ángel Alonso Fernández.
    - Flora García Domínguez.
- *Documentación:*
    - José Ramón López Fernández
    - Daniel Villarroel Rodríguez.

3.- Recursos y equipamiento:

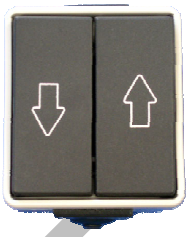
- Ordenadores Personales y portátiles para el desarrollo del software, la creación de la documentación y la programación de los módulos del sub-proyecto.

- Tabletas electrónicas y teléfonos móviles con wifi para el control del sistema a través de la aplicación desarrollada para el sistema inmótico.

- Pasarela DNC a MODBUS TCP, WK0500: con microcontrolador ATMEGA 2560.

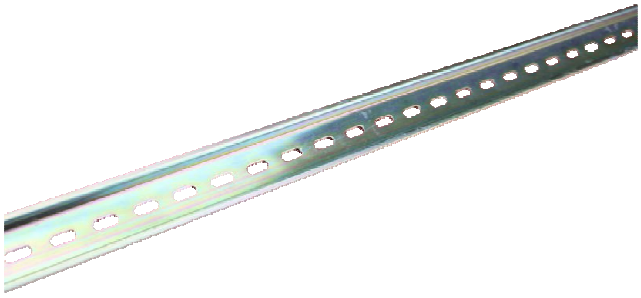
- Autómata para el control de persianas, WK0300CR: con microcontrolador ATMEGA 168 MCU.

- Motor de persiana: M & B, 10Nm/25Kg.



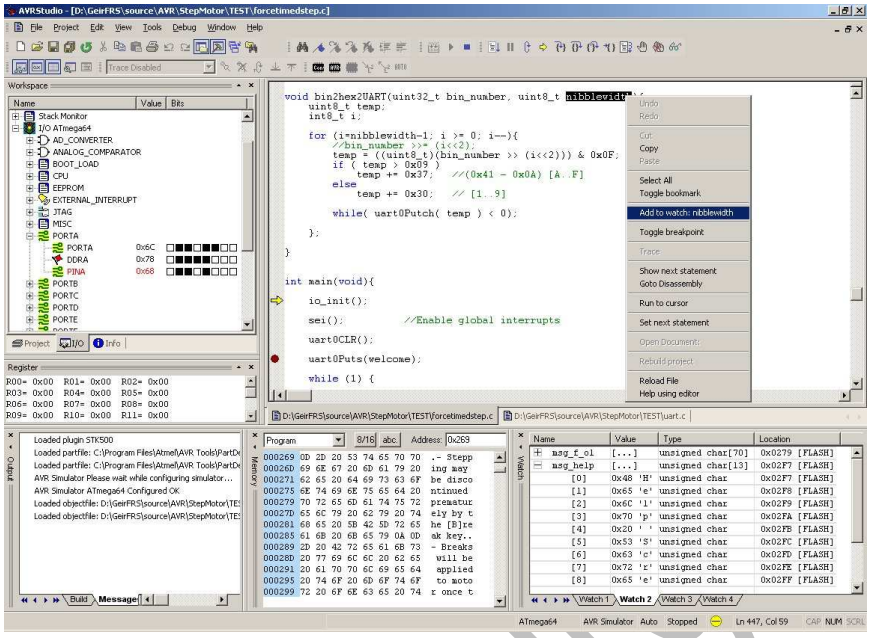
- Pulsador para control de persiana.

- Carriles DIN instalados en tableros para la instalación del sistema.

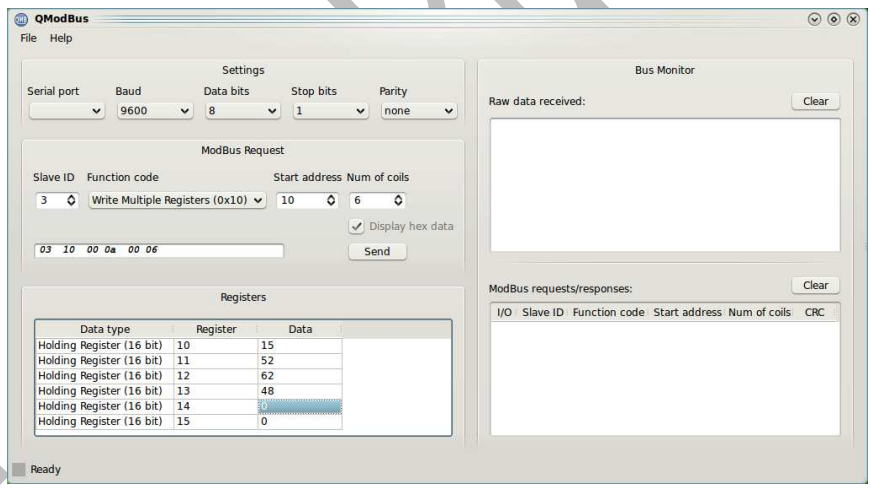


- Programador I2C con puerto USB.

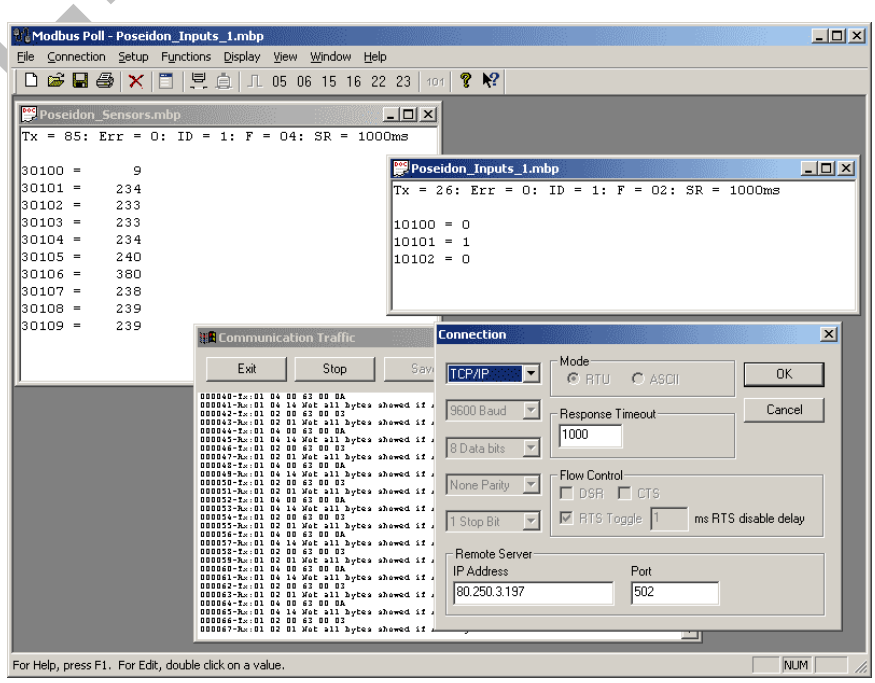
- Entorno AVR Studio. Programación ISP.



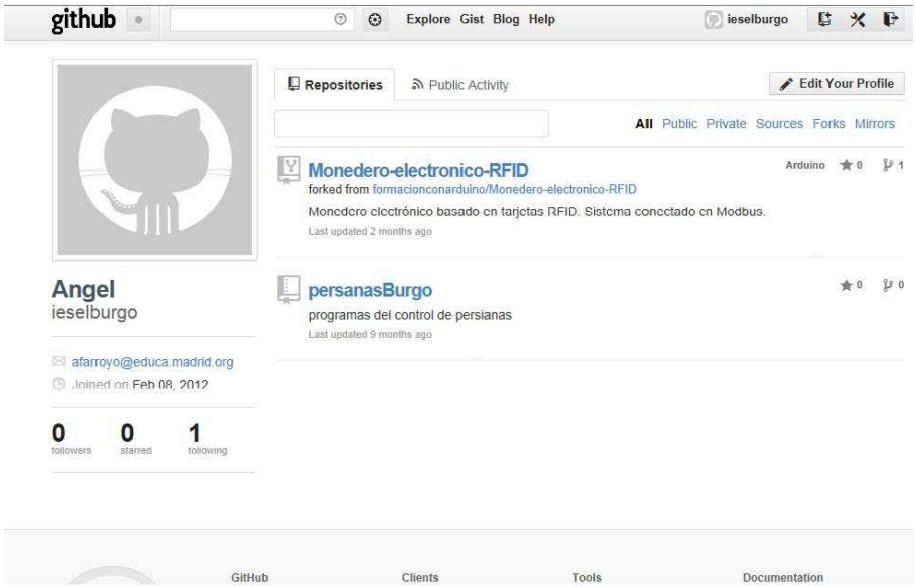
- QModBus.



- Modbus Poll.



- Repositorio GIT para el intercambio de programas e información con los otros centros participantes.



4.- Bases técnicas y recursos metodológicos:

4.1.- **ARDUINO 2560** (ATMEGA) (en el módulo WK0500):

Es el cerebro del sistema.

Características:

- 256 KB de flash
- 4KB de EEPROM. Aquí permanecerán los datos importantes incluso cuando se corte la tensión.
- 8KB de RAM.

Puertos:

- RS232.
- USB.
- I2C.
- Ethernet.
- UART TTL.
- RS 485.

Dos leds y un pulsador.

Alimentación a 24V o por USB.

4.2.- Especificaciones:

La idea es conectar el 2560 (WK0500), por una parte, a Ethernet (Protocolo MODBUS TCP) y por otra a MODBUS RTU.

Las librerías son gratuitas y es un sistema universal y compatible.

Con TCP/IP podemos tener muchos puertos y servicios abiertos por el mismo interfaz.

Con UART TTL podemos hacer un bus para sensores de temperatura (varios puntos de medida de temperatura).

Por USB no vamos a conectar nada, para este proyecto.

Por Ethernet también irá un servidor web (además del MODBUS TCP).

4.3.- Software:

El desarrollo de aplicaciones para PC será libre (cada grupo podrá desarrollar el que quiera). Sólo definiremos para todos los grupos la parte de MODBUS RTU.

Se definirán las estructuras de datos del maestro (2560).

NMODBUS: Librería abierta y gratuita (para MODBUS TCP y RTU).
MODBUS TCP: usa el puerto 502.
Servidor web: usa el puerto 80.

Los equipos remotos deben ser autónomos para que el sistema funcione aunque sea de forma local, sin bus.

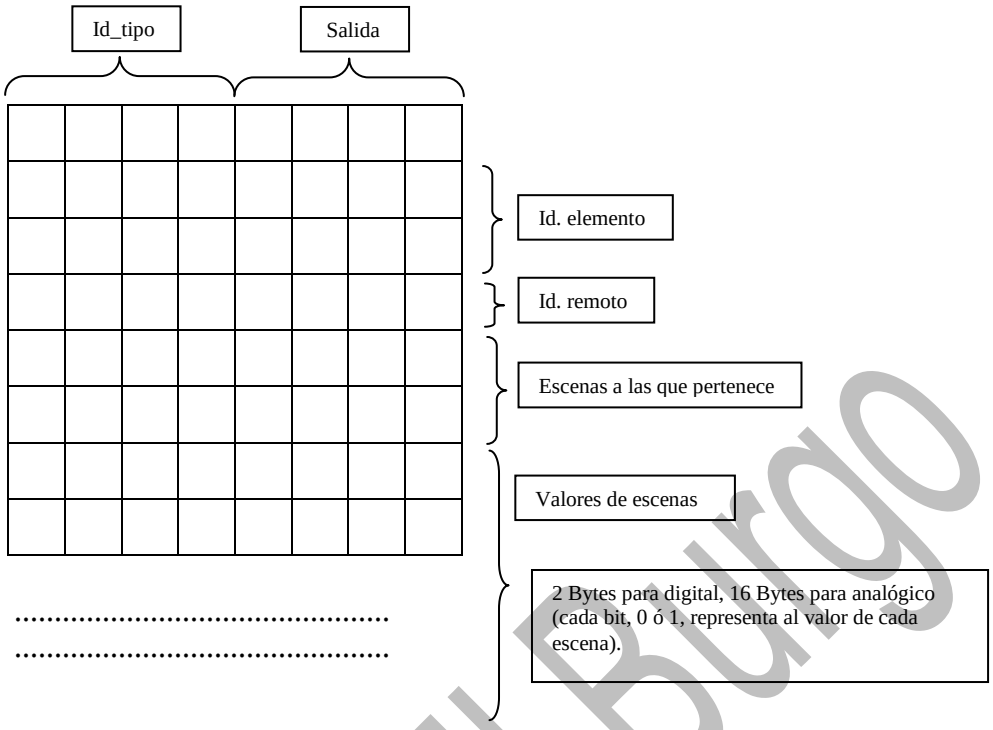
Haremos el sistema escalable y fácilmente integrable. Ampliar o modificar el sistema se hará ampliando o modificando comandos.

Ej.: si queremos subir una persiana (poner un bit a "1"), podemos hacerlo pulsando un botón de la interfaz (en el PC, en Android, en un tablet, etc.) (subir persiana); se enviará un comando al WK0500, que debe conocer el comando y decodificarlo. Después trasladará el comando a la persiana (remoto) adecuada. El remoto de la persiana sólo conocerá los comandos que le afectan (subir y bajar persiana).

Hay que tener en cuenta algunas restricciones:

- El maestro no es un PC por lo que tenemos que tener en cuenta que debemos ocupar poca memoria. En EEPROM no podemos cargar muchos datos pero tampoco dejarlo vacío pues tendríamos que configurar todos los remotos cada vez que queramos usarlos. Queremos que el sistema sea "plug and play".

Elementos a diseñar:



Con 1 Byte (8 bits) podemos tener 256 elementos; con 2 Bytes, 2^{16} elementos. Cada uno con su identidad.

Identidad de remoto: debemos identificar físicamente la conexión (ej.: una lámpara en el remoto 26).

0: dirección broadcast.

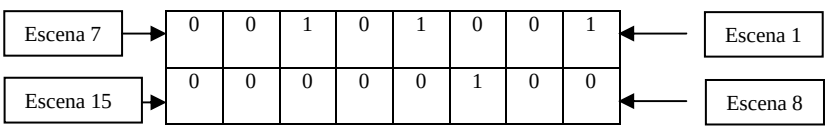
1-255: direcciones de remotos.

*** NOTA: la dirección 247 no es válida para un remoto. El sistema considerará esta dirección para aquellos remotos no configurados.**

Identidad tipo (Id_tipo): 2^4 = 16 tipos diferentes de elementos (enchufe, lámpara, válvula, etc.).

Salida: 2^4 = 16 salidas (módulos de 16 salidas).

Escenas: cada uno de los 16 bits se pone a 0 ó a 1 si el elemento pertenece a esa escena o no. Tenemos 16 escenas como máximo:

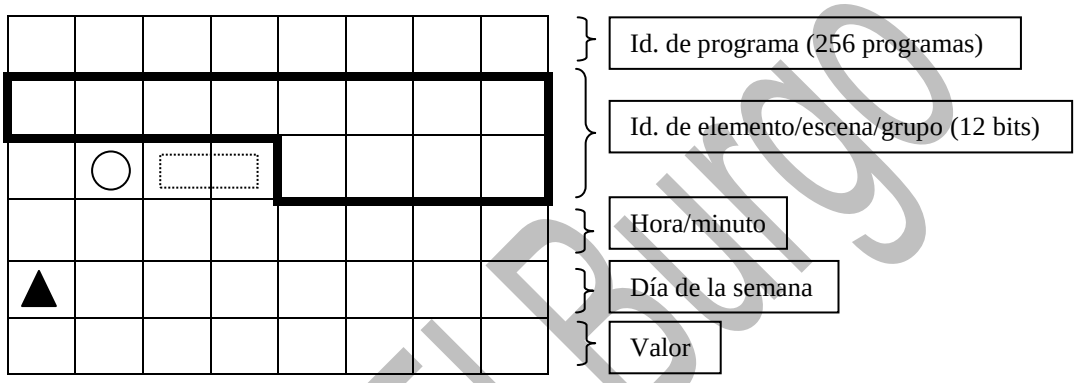


Como vemos, tenemos 16 bits. Se ponen a "1" aquellos que se corresponden con alguna de las 16 escenas a las que pertenezca cada remoto.

Para elementos digitales tendremos 8 bytes por cada elemento (en la EEPROM sólo podemos almacenar 4096 bits = 512 bytes).

Para elementos analógicos tendremos 16 Bytes para cada valor de escena (8 bits por escena).

Programa horario:



▭ : elemento/escena/grupo (01: indica elemento; 10 indica escena; 11 indica grupo).

◯ : Bit = 0 ó 1, según el programa esté habilitado o no.

▲ : Semanal/una vez.

valor : Si se enciendo o apaga o sube o baja,... Si es analógico necesitaremos 8 bits.

Día de la semana: Bit a "1" por cada día de la semana que quiero que se ejecute.

4.4.- Comandos:

Para enviar un comando, la aplicación mandará unos datos a unos registros concretos: Holding Registers (HR) del WK0500.

Todos los comandos comenzarán por Ox20 (a partir de Ox20)

Los primeros 32 comandos se reservan para uso (o propósito) general.

Para funcionar, algunos elementos necesitan pulsos (no activar o desactivar con un bit).

Necesitamos el pulso del elemento (Ox25) más la Identidad del elemento (2 bytes).

Tenemos 11 sub-proyectos. Cada uno de ellos tendrá unos comandos que comenzarán por el mismo número (Ox10, Ox20, Ox30,..., OxA0, OxB0).

El programa para el módulo WK0300 se incluye en el ANEXO I, al final de este documento.

- **Creación de grupos de persianas**. Cada persiana podrá pertenecer a 8 grupos diferentes.
- Alimentación 230 V CA.
- Sistema conectado en bus estándar y abierto. Modbus RTU .
- Conectable e integrable con el sistema inmótico general.
- Hasta 128 equipos en bus sin repetidor.
- Actuaciones sobre:
    - Cada persiana individual.
    - Cada grupo de persianas desde cualquier pulsador (subir todas o bajar todas).
    - Todas las persianas conectadas al bus (subir todas o bajar todas).
- Protección eléctrica contra cortocircuitos en el motor, incluso aunque un relé o ambos se queden pegados.

4.5.- Repositorio GIT:

Tenemos un espacio para ir dejando los programas realizados (repositorio de software). Se basa en GIT y está en el siguiente servidor:

https:/www.github.com/formacionconarduino

Se usa el repositorio para poder compartir archivos con otros usuarios y poder modificarlos. Podremos ver el historial de cambios y el archivo definitivo.

Existen varios servidores de repositorios:

- SUBVERSION (SVN). Cliente: TORTOISE.
- GIT (se usa cada vez más).

} Tienen comandos distintos pero el concepto es el mismo.

El repositorio es un almacén pero para archivos con código fuente, no para ejecutables.

Necesitaremos instalar un software en nuestro PC para poder usar el repositorio adecuadamente. El directorio con nuestros archivos se enlazará con el almacén de archivos del repositorio (como Dropbox). Es un software libre y lo que subamos es público. Para uso empresarial se usa una versión de pago.

Los archivos se guardan de forma "incremental", es decir, se guarda la información de la modificación realizada y no todo el archivo. Podremos ver el histórico de las modificaciones y recuperarlas cuando se necesite.

Cuando trabajamos, lo haremos sobre la última versión del archivo pero siempre podremos recuperar otras versiones anteriores. Además podemos ver el usuario que realizó las modificaciones, la fecha, etc.).

Puede que el contenido del "almacén" nos parezca un tanto anárquico pero es el gestor del almacén el que lo maneja.

En GIT no hay la relación "cliente-servidor". Todos son clientes. Nos tenemos que bajar el software e instalarlo en nuestro ordenador. El software genera automáticamente nuestro directorio local (o repositorio local). Como firma digital debemos crear una clave pública y una privada.

Claves:
- Privada — Quedará en nuestro PC (no se la damos a nadie).
- Pública — Es la clave para ***github.***

Usuario creado: **ieselburgo.**

password: **********.**

Para generar claves:

ssh -keygen -t rsa -C "email@mail.es"

ssh: safe shell (shell segura).

La clave ".pub" podemos mostrarla a otros que vayan a entrar en el repositorio.

La clave privada no la mostraremos a otros.

Editamos en el bloc de notas el archivo ***id_rsa.pub*** Copiamos el contenido, vamos al repositorio --> Account Settings --> SSH Public Keys, añadimos otra clave pública, ponemos el título que queramos y pegamos la clave.

Nuestro título: **elburgopublico**

Nuestro repositorio creado: **persanasBurgo** (no "persianas").

Las comunicaciones se encriptan con cada clave. Cuando ***github*** manda un archivo encriptado, sólo podrá descifrarlo el usuario que tiene la clave privada adecuada.

Si ***github*** descifra un archivo con una determinada clave pública, sabrá quién lo envió (nuestro PC con su clave privada).

El repositorio se puede manejar en modo texto (como con LINUX) o en modo gráfico.

- Git bush (modo texto).
- Git GUI (modo gráfico).

En cada repositorio se pueden añadir colaboradores (***colaborators***). Se añaden todos los colaboradores que van a acceder a ese repositorio.

A.- TAG:

Cuando tenemos versiones estables de nuestro archivo podemos crear "puntos notables" llamados **tag.** Ej.: versión1.0, versión 2.0,...

Podemos ver las versiones que podemos bajar; pueden estar como ".zip" o "tar.gz".

B.- BRANCH (CONTEXTO):

Cuando trabajamos sobre una versión estable y queremos modificarla, no hace falta hacer un "backup" o copia de respaldo del archivo. Con el repositorio no nos hace falta. La copia estable estará en el contexto "MASTER" y nosotros podemos crear otros contextos (los que queramos) en los que trabajar. Al crear un contexto, automáticamente se realiza una copia de nuestros archivos al nuevo contexto. Los cambios realizados sólo permanecerán en el contexto donde se realizan hasta que no demos la orden en el MASTER de mezclar los contextos.

Podemos trabajar en varios contextos a la vez (varios usuarios pueden estar trabajando a la vez) y cambiar de uno a otro cuando queramos.

C.- CHECKOUT:

Se usa para cambiar de contexto.

D.- MERGE:

Con este comando podemos hacer la mezcla del contenido del contexto MASTER con el de cualquier otro contexto. Respeta el contenido del MASTER y añade las modificaciones realizadas en otro contexto.

Se puede dar el caso de que dos usuarios realicen modificaciones a la vez en contextos diferentes; esto presentaría un conflicto pero el trabajo debe estar bien distribuido entre los usuarios para que esto no ocurra.

E.- ADD:

El repositorio es una zona distinta de la del directorio de nuestro ordenador por lo que si añadimos un archivo a este directorio no aparecerá automáticamente en el repositorio. Para añadirlo usaremos el comando **ADD**. Se podrán añadir uno o varios archivos. Recordemos que debemos acceder a **www.github.com/**

F.- PUSH:

Una vez creada una cuenta tendremos espacio en el repositorio donde trabajar. Para crear una copia de nuestro disco duro al repositorio usamos el comando **PUSH. (Subir a internet).** No podemos usar PUSH si no tenemos guardada la última versión; ej.: dos usuarios trabajando sobre un archivo "version1.0"; uno de ellos realiza un PUSH con el archivo "version2.0" y el otro no podrá realizar otro PUSH pues su trabajo se ha realizado sobre la 1.0. Primero debería realizar un PULL, después usar MERGE para aplicar cambios y, por último, hacer PUSH.

G.- PULL:

Para realizar una copia del repositorio en nuestro disco duro (bajar de internet), usaremos el comando **PULL**. Con este comando bajaremos a nuestro ordenador una versión que se mezclará con los cambios realizados en nuestro PC.

H.- FETCH:

Este comando baja la versión tal cual, sin hacer cambios en ella.

I.- COMMIT:

Es como hacer una foto. Es como un **tag** pero sin la importancia de una versión. podremos ver un listado de todos los **COMMIT** realizados.

J.- PATCH:

PATCH del software es una modificación del archivo fuente.

K.- DOWNLOADS:

Podemos bajar la última versión del repositorio como ".zip" o "tar.gz".

L.- Ejemplos de modo texto:

```
$git branch contexto          // Crea un contexto llamado "contexto".
* contexto                    // Estamos en el contexto llamado "contexto".
*master                       // Estamos en el contexto "master".
$git commit -am               //Crea una copia.
$git checkout master          // Cambiamos al contexto "master".
$git merge contexto           // Se realiza una mezcla de contextos en mi PC.
$git tag -a v3.0              // Realiza una versión estable llamada v3.0
                              // abre un editor para escribir una explicación de la
                              // versión. Para salir:    :wq!  (como el editor vi).
$git push alias master --tags  //Sube la versión a internet, incluyendo los tags.
```

// "alias" es el nombre que tenemos del repositorio
// en nuestro PC.

M.- <u>Configuración del repositorio</u>: Global setup.

setup git

git config --global user.name "tu nombre" //Creo que creamos burgo1

git config --global user.email nombre@mail.es

// Esto sólo se realiza una vez, al instalar el repositorio.

git init // Tenemos que estar dentro del directorio donde queremos trabajar.

Next steps:

...

...

git remote add origin git@github.com: ieselburgo/persanasBurgo.git

// Debemos crear un "alias" (en este caso "origin") para sustituir a "git@github.com: ieselburgo/persanasBurgo.git". Es una cadena muy larga.

// Para cada repositorio hay que hacer "init" y poner un "alias".

git add * //Añade todos los ficheros (localmente).

git commit -am comentario //se hace una "foto" cada vez que queramos
//recordar una modificación del archivo.
// Se pueden realizar varios "commit".

git push -u alias master //Se suben todos los commit (histórico) al contexto
//master. Podría ir a otro contexto.

git clone url //Copia un repositorio (de internet) en la carpeta de mi PC.

git pull //Baja la última versión.

git tag -a v1.0 //Crea una versión y abre un editor.

//Con push se sube todo menos los **tags**.

git push --tagss //Para que se suban también los tags.

5.- Actividades realizadas:

- *Reunión de coordinación entre centros.*

  Fecha: 11/11/2011

  Lugar: I. E. S. El Burgo de Las Rozas.

- *Cursos de formación impartidos por la empresa ICTEL*:

FECHAS	LUGAR
24/11/2011 a 26/11/2011	**I. E. S. El Burgo de Las Rozas (Madrid)**
8/02/2012 a 9/02/2012	**Colegio Montecastelo (Vigo)**
16/05/2012 a 18/05/2012	**I. E. S. Tirant Lo Blanc (Gandía)**
27/09/2012 a 28/09/2012	**ICTEL Ingenieros (Vigo)**

- *Reuniones periódicas* entre los participantes del sub-proyecto en el departamento de Electrónica del I. E. S. El Burgo de Las Rozas.

6.- Resultados y productos:

Sub-proyecto (control de persianas y toldos):

6.1.- Programación del máster WK0500:

Programaremos el módulo WK0500 a través del bus I2C, utilizando un programador y un PC conectado al mismo por medio de una conexión al puerto USB:

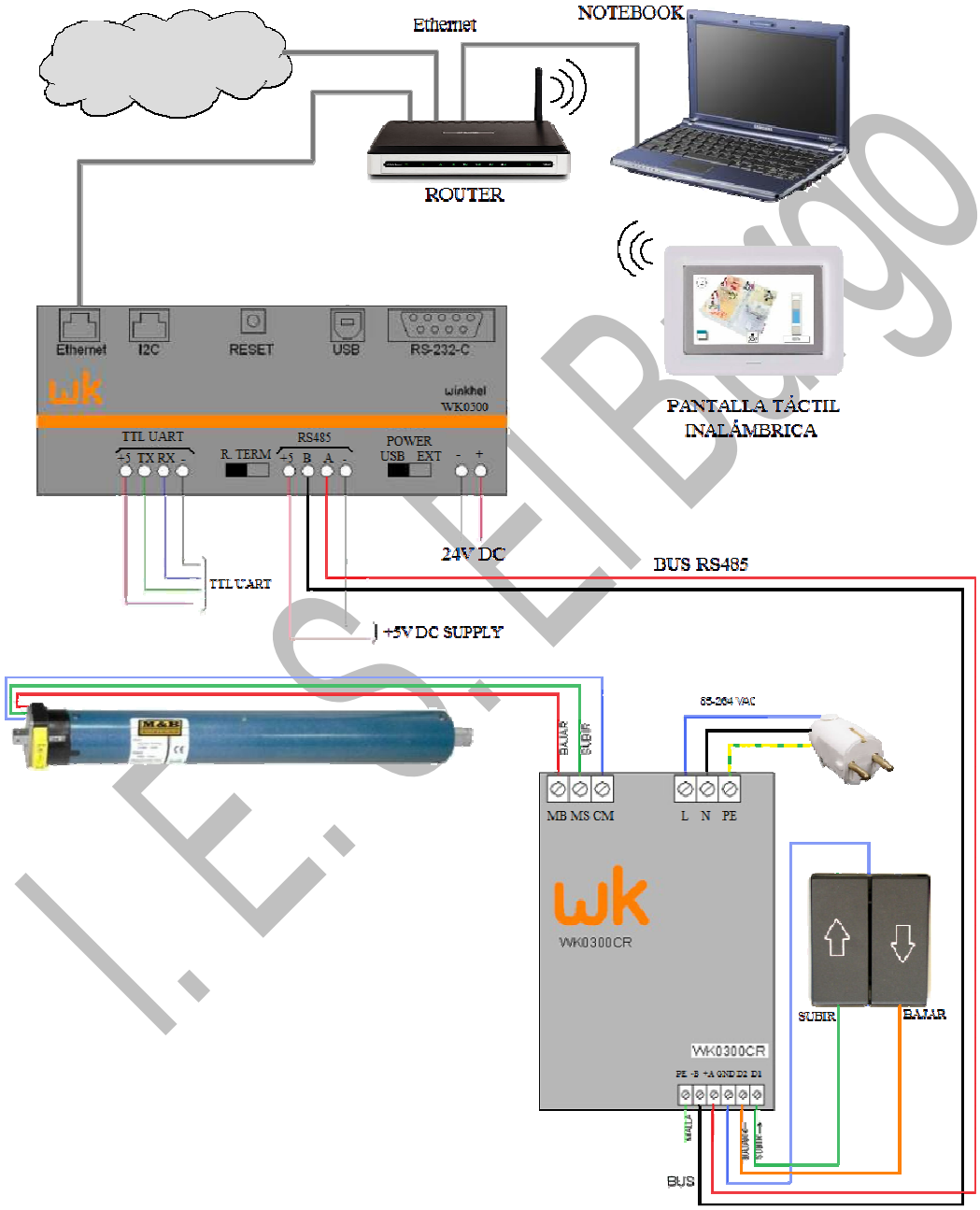
MÁSTER WK0500

6.2.- Programación del módulo de persianas WK0300:

Programaremos el módulo para persianas, WK0300, a través del bus I2C, utilizando un programador y un PC conectado al mismo por medio de una conexión al puerto USB.

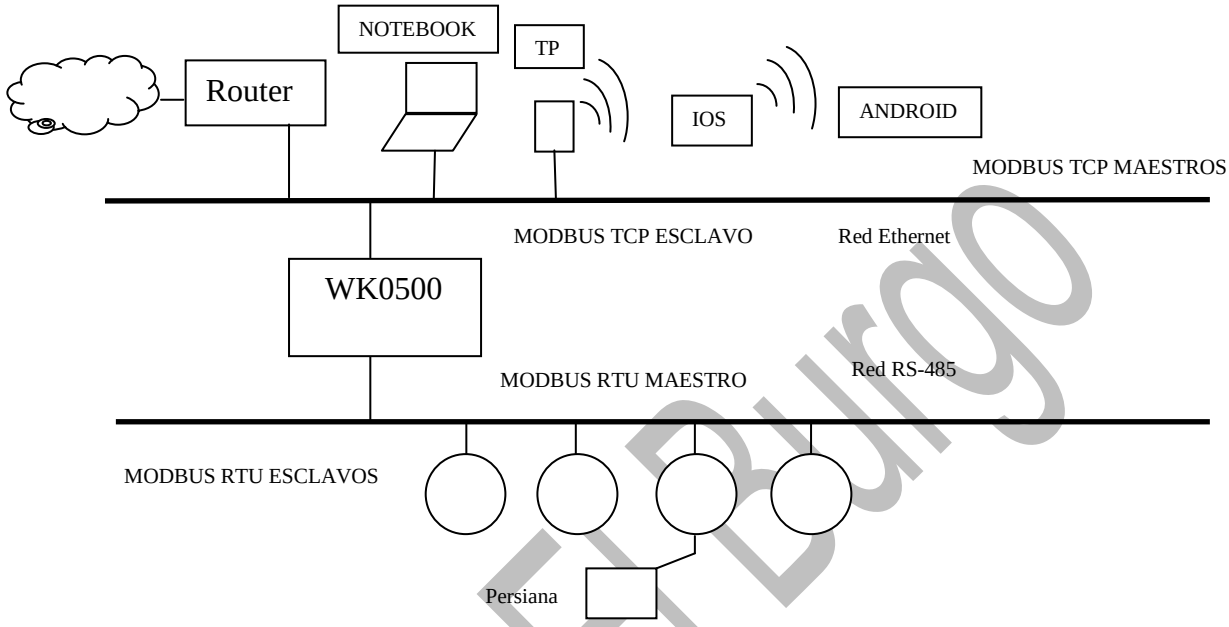
6.3.- Esquema del montaje final:

El montaje final nos permite controlar el movimiento de las persianas y toldos desde la red Ethernet (con un equipo informático,…) o de forma inalámbrica (desde una tableta, un teléfono móvil,…). También podremos subir y bajar las persianas y toldos de forma local, mediante unos pulsadores, y sin necesidad de tener conectado el máster ni ningún otro equipo (aparte de la alimentación correspondiente).



6.4.- Integración del sub-proyecto en el proyecto final.

Gracias a que todos los sub-proyectos se comunican con el máster (WK0500) de igual forma, con el mismo protocolo, basta con conectar cada uno de los esclavos a la red RS-485 para que el sistema domótico funcione correctamente.



7.- Desviaciones de lo previsto y soluciones aplicadas:

El proyecto no ha podido ser integrado plenamente en el curso 2011/12 en nuestro centro debido a que no se impartían asignaturas de domótica en los ciclos que impartíamos, pero se integrará ampliamente en los planes generales del centro a partir del curso 2012/2013 con el comienzo de los nuevos ciclos formativos en los que se imparten los módulos de hogar digital e instalaciones domóticas, tanto en el ciclo IT como en el nuevo STI.

Otro objetivo que se pretendía conseguir era la participación activa en las comunidades de Arduino (tanto en la española como en la internacional). Se intentará enlazar el portal web realizado con sitios de referencia de la plataforma (sobre todo en http://www.arduino.cc).

Ha quedado por desarrollar la edición de un libro de texto enfocado a la educación con una visión didáctica de los diferentes aspectos del desarrollo del proyecto global; lo que se ha realizado es la documentación de los diferentes sub-proyectos realizados, así como la integración de todos ellos en un sistema único.

8.- Conclusiones y aplicaciones futuras:

Consideramos que éste tipo proyectos es muy adecuado para desarrollar y poner en común formación, investigación, innovación y transferencia tecnológica entre los diferentes participantes, tanto Centros Formativos como Empresas, así como poner en contacto docentes, con similares inquietudes e intereses, de diferentes Comunidades Autonómicas.

Por otro lado la introducción de importantes variaciones en los currículos de los nuevos ciclos formativos con respecto a los antiguos hace necesaria una formación específica del profesorado en nuevas materias, como en este proyecto en el que se estudian temas relacionados con el Hogar Digital y las instalaciones domóticas.

En este proyecto se han analizado las altas posibilidades de una plataforma libre (plataforma Arduino) en la enseñanza, para desarrollar un sistema domótico orientado a la formación de nuestros alumnos.

Los productos electrónicos desarrollados tienen una aplicación inmediata en nuestro actual Sistema Educativo además de alta disponibilidad por ser un sistema de desarrollo libre.

Los productos desarrollados son altamente competitivos en el mercado real, acercando así el mundo académico al comercial.

Trabajaremos para que el proyecto tenga continuidad en el aula y siga desarrollándose con la participación de nuestros alumnos en los diferentes módulos de los ciclos formativos que impartimos.

También sería deseable que estos programas sigan convocándose en sucesivos cursos escolares para poder seguir innovando y adquiriendo experiencias que nos ayuden a mejorar el aprendizaje de nuestros alumnos.

9.- Valoración final del proyecto.

Entre los objetivos conseguidos en la realización de este proyecto cabe destacar:

a) Introducción de la plataforma Arduino en la enseñanza.

b) Difusión de la importancia que tienen hoy en día los sistemas de inmótica y domótica.

c) Obtención de un sitio web de repositorio de proyectos docentes para llevar a cabo con los alumnos para el intercambio de conocimiento y experiencias.

d) Compartir conocimientos técnicos con empresas y otros centros de FP.

e) Potenciar la formación de los alumnos al trabajar en el desarrollo de unos proyectos reales de aplicación comercial.

f) Aplicación inmediata en el ámbito educativo, en distintos campos: Electrónica Analógica, Lógica Digital y Microprogramable, Instalaciones Domóticas, desarrollo de hardware y software,…

Los sistemas creados, la metodología e información están disponibles para toda la comunidad educativa en las plataformas de difusión creada integrando así las nuevas Tecnologías de la información y la comunicación (TIC).

10.- Anexos.

ANEXO I

```
/*********  CONTROL PERSIANAS  WK 300 ***************************/

#include <Flanco.h>
#include <Temporizador.h>

#define  PulsadorSubir  2
#define  PulsadorBajar  3
#define  ReleSubir  16
#define  ReleBajar  17

boolean EstadoSubirAnt=false;
boolean EstadoBajarAnt=false;
boolean EstadosSubir=false;
boolean EstadoBajar=false;
boolean pulsadorSubir=false;
boolean pulsadorBajar=false;
boolean InicioRecepcionDatos=false;

 int NumDatos=0;
 char buffer[3];
 boolean DatosRecibidos=false;

Temporizador Temp1=Temporizador(1,false,120000);
Temporizador Temp2=Temporizador(3,false,1000);
Temporizador Temp3=Temporizador(1,false,120000);
Temporizador Temp4=Temporizador(3,false,1000);
Temporizador Temp5=Temporizador(1,false,1500);//Temporizador para detectar el doble pulso
de subida;
Temporizador Temp6=Temporizador(1,false,1500);//Temporizador para detectar el doble pulso
de bajada;
Temporizador Temp7=Temporizador(1,false,10);//Temporizador para recibir los datos por el
puerto serie;

Flanco FPulsoSubida=Flanco(1,false);
Flanco FPulsoBajada=Flanco(1,false);
Flanco FTemporizacion2=Flanco(0,true);
Flanco FTemporizacion4=Flanco(0,true);
Flanco FTemporizacion7=Flanco(0,true);

unsigned char T1=false;
unsigned char T2=false;
unsigned char T3=false;
unsigned char T4=false;
unsigned char T5=false;
unsigned char T6=false;
unsigned char T7=false;

unsigned char P1=false;
unsigned char P2=false;
unsigned char P1Ant=false;
unsigned char P7=false;
unsigned char P2Ant=false;
byte Mascara=0xFF;
boolean subiendo=false;
boolean bajando=false;
```

```
boolean paradaSubir=false;
boolean paradaBajar=false;
unsigned long RetardoRebote=50;
unsigned long UltimoTiempoRebote=0;
int NumPulsosSubida=0;
int NumPulsosBajada=0;

String t;

void setup()
{
  pinMode(PulsadorSubir,INPUT);
  pinMode(PulsadorBajar,INPUT);
  pinMode(4,INPUT);
  pinMode(5,INPUT);
  pinMode(6,INPUT);
  pinMode(7,INPUT);
  pinMode(8,INPUT);
  pinMode(9,INPUT);
  pinMode(10,INPUT);
  pinMode(15,INPUT);
  pinMode(ReleSubir,OUTPUT);
  pinMode(ReleBajar,OUTPUT);
  digitalWrite(ReleSubir,LOW);
  digitalWrite(ReleBajar,LOW);
  ComprobarMascara();
  Serial.begin(9600);
}

void loop()
{
  boolean pulsadorSubirAnt=pulsadorSubir;
  boolean pulsadorBajarAnt=pulsadorBajar;
  pulsadorSubir=digitalRead(PulsadorSubir);
  pulsadorBajar=digitalRead(PulsadorBajar);


  if(pulsadorSubir !=pulsadorSubirAnt || pulsadorBajar!=pulsadorBajarAnt)
  {
    UltimoTiempoRebote=millis();
  }

  if(millis()-UltimoTiempoRebote>RetardoRebote)
  {
    P1=FPulsoSubida.comprobar(!pulsadorSubir);
    P2=FPulsoBajada.comprobar(!pulsadorBajar);
    if(P1 && EstadosSubir)
    {
      paradaSubir=true;
    }
    if(P1 && EstadoBajar)
    {
      bajando=true;
    }
    if(P2 && EstadosSubir)
    {
      subiendo=true;
    }
    if(P2 && EstadoBajar)
    {
```

```
      paradaBajar=true;
    }

    T1=Temp1.comprobar(P1);
    T5=Temp5.comprobar(P1);
    T2=Temp2.comprobar(!pulsadorSubir);
    T3=Temp3.comprobar(P2);
    T6=Temp6.comprobar(P2);
    T4=Temp4.comprobar(!pulsadorBajar);
    DoblePulsacion();
    PulsacionConsecutiva();
    PulsacionProlongada();
    ComprobarPuertoSerie();
    ActuarMotores();
  }
}

void ComprobarPuertoSerie()
{
  T7=Temp7.comprobar(InicioRecepcionDatos);
  P7=FTemporizacion7.comprobar(T7);
  if(Serial.available() && NumDatos<4)
  {
    if(NumDatos==0)
    {
       InicioRecepcionDatos=true;
    }
    byte b=Serial.read();
    if(NumDatos==3)
    {
      DatosRecibidos=true;
      InicioRecepcionDatos=false;
    }
    buffer[NumDatos]=b;
    NumDatos++;
  }
  if(P7 && NumDatos<4)
  {
    Serial.flush();
    InicioRecepcionDatos=false;
    NumDatos=0;
  }
  if(DatosRecibidos)
  {
    byte ModoFuncionamiento=buffer[0];
    byte MascaraMaestro=buffer[1];
    byte aux;
    ComprobarMascara();
    aux=MascaraMaestro & Mascara;
    if(aux!=0)
    {
      if(ModoFuncionamiento==0x53)
      {
        Temp3.parar();
        Temp1.comprobar(1);
      }
      if(ModoFuncionamiento==0x42)
      {
        Temp1.parar();
        Temp3.comprobar(1);
```

```
      }
    }
    NumDatos=0;
    DatosRecibidos=false;
  }
}
void PulsacionProlongada()
{
    if(FTemporizacion2.comprobar(T2))
    {
      Temp1.parar();
    }
    if(FTemporizacion4.comprobar(T4))
    {
      Temp3.parar();
    }
}
void PulsacionConsecutiva()
{
 if(paradaSubir)//Pulso subir y estaba subiendo
  {
     Temp1.parar();
     Temp2.parar();
     Temp3.parar();
     Temp4.parar();
     paradaSubir=false;
  }
  if(bajando)//Pulso subir y estaba bajando
  {
    Temp2.parar();
    Temp3.parar();
    Temp4.parar();
    bajando=false;
  }
    if(paradaBajar)//Pulso bajar y estaba bajando
    {
      Temp1.parar();
      Temp2.parar();
      Temp3.parar();
      Temp4.parar();
      paradaBajar=false;
    }
    if(subiendo)//Pulso bajar y estaba subiendo
    {
      Temp1.parar();
      Temp2.parar();
      Temp4.parar();
      subiendo=false;
    }
}
void ActuarMotores()
{
  EstadosSubir=!EstadoBajar&&(!pulsadorSubir||(T1&&!T2));
  EstadoBajar=!EstadosSubir&&(!pulsadorBajar||(T3&&!T4));

  digitalWrite(ReleSubir,EstadosSubir);
  digitalWrite(ReleBajar,EstadoBajar);
}
void DoblePulsacion()
{
```

```
  if(P1)
  {
    if(NumPulsosSubida<2)
    {
      NumPulsosSubida++;
      NumPulsosBajada=0;
    }
    else
    {
      if(T5)
      {
        OrdenSubida();
        NumPulsosSubida=0;
      }
    }
  }
  if(P2)
  {
    if(NumPulsosBajada<2)
    {
      NumPulsosBajada++;
      NumPulsosSubida=0;
    }
    else
    {
      if(T6)
      {
        OrdenBajada();
        NumPulsosBajada=0;
      }
    }
  }
  if(T5==0)
  {
    NumPulsosSubida=0;
  }
  if(T6==0)
  {
    NumPulsosBajada=0;
  }
}

void OrdenSubida()
{
  byte b=0x53;
  ComprobarMascara();
  byte b1=Mascara;
  //int iMascara= (b<<8) | Mascara;
  //Serial.println(iMascara);
  Serial.print(b);
  Serial.println(b1);
}
void OrdenBajada()
{
  byte b1=0x42;
  ComprobarMascara();
  byte b2=Mascara;
```

```
  //int iMascara= (b1<<8) | Mascara;
  //Serial.println(iMascara,BIN);
  Serial.print(b1);
  Serial.println(b2);
}
void ComprobarMascara()
{
  bitWrite(Mascara,0,!digitalRead(4));
  bitWrite(Mascara,1,!digitalRead(5));
  bitWrite(Mascara,2,!digitalRead(6));
  bitWrite(Mascara,3,!digitalRead(7));
  bitWrite(Mascara,4,!digitalRead(8));
  bitWrite(Mascara,5,!digitalRead(9));
  bitWrite(Mascara,6,!digitalRead(10));
  bitWrite(Mascara,7,!digitalRead(15));
}
```