



**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
INFORMÁTICA**

GRADO EN INGENIERÍA INFORMÁTICA

INGENIERÍA DE COMPUTADORES

**Control remoto de un UGV
mediante STM32**

**Realizado por
Daniel García Bravo**

**Dirigido por
Ángel Jiménez Fernández**

**Co-tutor
Daniel Gutiérrez Galán**

**Departamento
Arquitectura y Tecnología de Computadores**

Sevilla septiembre de 2017



ESCUELA TÉCNICA SUPERIOR
DE INGENIERÍA INFORMÁTICA

Resumen

En este proyecto se plantea el diseño de un controlador inalámbrico para un vehículo terrestre no tripulado UGV. Para ello se hará uso de una plataforma basada en STM32, módulos de radio, controladores de motores DC y servos.

Para ser más concretos el controlador empleado es el STM32F4 Discovery para el que hemos utilizado la herramienta CoIde(Software libre) para su programación.

Motor de corriente continua en las ruedas traseras del vehículo, servo motor de 12kg(aplicados en palanca) MG995 para controlar el giro del vehículo.

Para la programación de la interface donde configuramos el puerto hemos empleado el entorno de desarrollo Visual Studio 2010 y lenguaje de programación C++.

This project proposes the design of a wireless controller for an unmanned UGV ground vehicle.

For this there is use of a platform based on STM32, radio modules, DC motor controllers and servos.

For more specific, the microcontroller used is the STM32F4 Discovery for which you have used the CoIde (Free Software) tool for programming.

DC motor on the rear wheels of the vehicle and a servo motor to control vehicle rotation. MG995 12kg

For programming the interface where we set the port is the Visual Studio 2010 development environment and the C ++ programming language.



Contenido

Introducción.....	4
Estudio del arte.....	5
Motivación.....	7
Objetivos generales.....	7
1- Dispositivo Hardware.....	9
1.1 Diagrama de bloques.....	9
2- Tecnologías empleadas.....	11
2.1 Placa de desarrollo.....	11
2.2 Entorno de desarrollo.....	13
2.3 Servomotor.....	17
2.4 Puente H + Motor DC.....	18
2.5 Módulo radiofrecuencia.....	19
2.6 Regulador de tensión.....	20
3- Desarrollo Hardware.....	21
4- Desarrollo Software.....	21
4.1 Firmware.....	22
4.2 Software de usuario.....	23
4.3 Comunicación entre Software y Firmware.....	27
5- Pruebas y resultados.....	29
5.1 Prueba del firmware.....	29
5.2 Prueba de comunicación.....	30
6- Planificación y costes.....	31
7- Hitos.....	35
8- Problemas sugeridos.....	35
9- Fabricación del producto.....	36
10- Generación del instalador.....	37
11- Instalación de la aplicación.....	43
12- Desinstalación de la aplicación.....	44
Conclusiones.....	45
Trabajos futuros.....	46
Bibliografía.....	47

Introducción

Para ser más concretos el micro-controlador empleado es el STM32F4 Discovery, aplicado en vehículos para lograr la autonomía del mismo, con el objetivo de conseguir que el vehículo obtenga la respuesta similar a la conducción de una persona en cualquier situación o medio. La autonomía del vehículo no es total ya que habrá alguien manejándolo desde fuera como un coche de radiocontrol.

Podemos emplear este tipo de vehículos para el reparto de paquetería en una empresa como *Amazon*, *Ebay* o como vehículo militar.

En la actualidad, las personas buscan la comodidad a la hora de realizar una compra rápida y no tener que salir de su casa y así evitarse el gasto de dinero extra en el trayecto a la compra. Y con esto me refiero a gastos de gasoil, comida y gastos imprevistos.

También está pensado para el uso personal.

Es decir, queremos un coche autónomo no tripulado, para perder el menor tiempo posible del trabajo a casa y viceversa y así poder aprovechar este tiempo. En nuestro caso hemos comenzado con un vehículo que usted podrá comprar y olvidarse totalmente de la conducción ya que este vehículo estará pilotado por un trabajador de nuestra empresa. Otro objetivo que buscamos con este proyecto consiste en vender estos coches a las empresas de compra/venta online como *Ebay* o *Amazon* las cuales dispondrían de una serie de coches eléctricos capaces de entregar pedidos sin que sus repartidores salgan de la sede. Lo que disminuiría la posibilidad de accidente laboral de los empleados de dichas empresas. La idea es conseguir coches autónomos pero para ello es necesario explotar primero esta versión y dar más adelante el salto a coche totalmente autónomo, pero para ello sería necesario la aceptación de la sociedad y lo más importante, mientras más coches autónomos pongamos en la calle, menor será el riesgo de accidentes de tráfico.

En cuanto al uso militar nos referimos a que:

El UGV se puede utilizar para múltiples funciones, entre ellas la detección, reconocimiento, observación, vigilancia, contra-IED (artefactos explosivos

improvisados), CBRNe (químicos, biológicos, radiológicos, nucleares y explosivos), limpieza de rutas, generador de humos, extinción de incendios, transporte, y otras misiones de seguridad civil.

Se puede equipar con ruedas o cadenas, y su diseño es de arquitectura ligera y abierta, que permite la integración de diferentes cargas útiles.

No es un vehículo destinado a sustituir a las unidades de infantería, más bien se trata de un vehículo con funciones de exploración, (como búsqueda antipersonas) para disminuir el tiempo de rastreo, que daría paso a las unidades convencionales del Ejército de Tierra.

Estudio del arte

Vamos a hacer una pasada por el mercado para ver unos vehículos terrestres no tripulados semejantes a nuestro proyecto.

En primer lugar vamos a ver un mini vehículo no tripulado llamado *Nerva LG*.



Foto 1: NERVA LG

El *NERVA LG* es un mini-vehículo polivalente terrestre no tripulado (UGV), diseñado y desarrollado por Nexter Robotics, para su uso por las fuerzas armadas.

Se dio a conocer durante la feria EuroSatory 2012 y presentado en DSEI 2013 Defence and Security Equipment International, Londres, en septiembre de 2013.

El *NERVA LG* lo ha adquirido el Centro de defensa holandés y la policía suiza entre otros.

Puede subir escaleras y atravesar terrenos accidentados. Es lanzable y puede transmitir información desde 1 Km. En terreno abierto o 300 m. en zonas urbanas.

Y por último y como característica no menos importante de este UGV es que se puede operar desde PCs, smart phones o teléfonos móviles.

En segundo lugar vamos a intentar asemejar los UGV aéreos que pretende sacar *Amazon* para repartos con nuestros vehículos terrestres ya que su función es la misma.



Foto 2: Vehículo aéreo no tripulado

El gigante del comercio electrónico ha obtenido permiso de la Administración Federal de Aviación (FAA) de Estados Unidos para probar el transporte de paquetes mediante aviones no tripulados.

Amazon deberá seguir reglas como volar a menos de 400 pies y a la luz del día. Debemos decir que aún no están operativos.

En 2013 la cadena Dominos Pizza mostraba al mundo su prototipo de dron de entrega a domicilio llamado Domicopter. Aunque esta misma empresa de comida ya ha estado trabajando con coches como los que nuestro proyecto pretende sacar adelante.



Foto 3: UGV aéreo



Foto 4: UGV terrestre

En nuestro caso nuestro vehículo terrestre tendrá la desventaja de que al ser terrestre el tiempo de llegada del pedido será algo mayor, pero en cuanto a coste de fabricación y de mantenimiento nuestro coche terrestre sigue siendo mejor que estos drones de reparto.

Motivación

La idea del proyecto, surge de la necesidad que tienen las personas mayores, o cualquier persona con discapacidad o problemas físicos, a la hora de salir de casa para realizar la compra. Esto no quiere decir que este medio no pueda ser utilizado por cualquier otra persona.

A raíz de esta idea inicial del proyecto nos surge la idea de vender estos coches como vehículos autónomos, aunque realmente no lo sean ya que están pilotados por nuestros trabajadores y el riesgo de sufrir un accidente es mucho menor que un coche totalmente autónomo.

En todos los casos, estas personas podrán recibir la compra sin necesidad de salir de casa. O ir a ver a la familia sin necesidad de tener el carnet de conducir, ya que nosotros los llevaremos.

Objetivos generales

El objetivo de este trabajo, es el diseño y la implementación de un vehículo terrestre no tripulado (UGV) que podemos manejar desde un pc. Nuestro proyecto se ha dividido en dos sub-objetivos que son los siguientes:

- Implementación del software para el micro-controlador, nos referimos a la configuración de los periféricos. Es decir, este objetivo pretende elaborar un sistema capaz de responder a una serie de órdenes enviadas por radiofrecuencia e interpretarlas como si se tratara de un vehículo tripulado.
- Implementación de un interfaz de usuario para poder controlar el vehículo de forma remota. Creada en Visual Studio 2010. Esta interfaz se encargará de recibir nuestras órdenes y enviárselas al vehículo.

También vamos a ver los objetivos a nivel profesional y educacional:

Primer acercamiento en primera persona con los sistemas empotrados, y manejo de plataformas y sus entornos de desarrollo, las cuales son usadas para ello.



ESCUELA TÉCNICA SUPERIOR
DE INGENIERÍA INFORMÁTICA

A nivel profesional:

- Trabajo individual, algo que es muy importante. Saber trabajar solo para luego poder trabajar en equipo.
- Soltura con el entorno de desarrollo Visual Studio 2010 el cual ya hemos usado en clase pero a menor nivel y orientado a programación Windows o de sistemas operativos. Por ello es un nuevo reto comenzar a trabajar con micro-controladores ya que es algo que me apasiona.
- Por otro lado destacamos la comprensión sobre un entorno de desarrollo para programar el micro-controlador el cual es CoIde y su compilador GCC.
- Estos dos pasos anteriores implican el conocimiento y manejo de dos lenguajes de programación como son el C y C ++ lo que puede abrirte muchas puertas en el mercado laboral.
- Destreza adquirida manejando circuitos, comprobando tensiones y el montaje de los mismos en protoboards.

A nivel educacional:

- Poner en marcha todos los conocimientos adquiridos a lo largo de estos 4 años de grado.
- Mejorar en la programación de sistemas empuotrados así como la interpretación del data-sheet de los micro-controladores y periféricos, los cuales hemos usados algunos. Los veremos más adelante.
- Tras esta experiencia hemos mejorado la forma de trabajar, así como la planificación del tiempo y de las tareas a la hora de afrontar un proyecto sin tener los conocimientos necesarios para realizarlo pero si las herramientas.

1 - Dispositivo Hardware

Diseño del sistema: Arquitectura.

El vehículo, estará basado en el μ C STM32, como elemento principal del mismo. Este μ C, se complementará con una serie de dispositivos auxiliares que permitirán efectuar todas las operaciones que se requieren. Estos dispositivos son:

- Dispositivo para dirección, (servo motor), con capacidad hasta 15Kg-cm de fuerza aplicada en palanca.
- Dispositivo de movimiento, (motor corriente continua), para poder mover el vehículo.
- Dispositivo de comunicaciones inalámbricas, (3DR Radio), para enviar a la placa las directrices que el piloto desea realizar.

1.1 – Diagrama de bloques

La siguiente figura, representa la estructura del sistema. Vamos a citar los elementos que aparecen en el diagrama y más importantes.

Estos elementos son:

- 1 Placa STM32F4 Discovery, que contiene el μ controlador del sistema, concretamente el STM32F407VGT6.
- 1 motor de corriente continua, el encargado de mover el vehículo.
- 1 servomotor TowerPro MG995 15Kg-cm.
- 2 módulos de radiofrecuencia
- 1 circuito integrado, L293D. Controlador del motor.
- 1 batería de 9V para alimentar la placa.
- 1 placa protoboard para montar toda la circuitería necesaria.

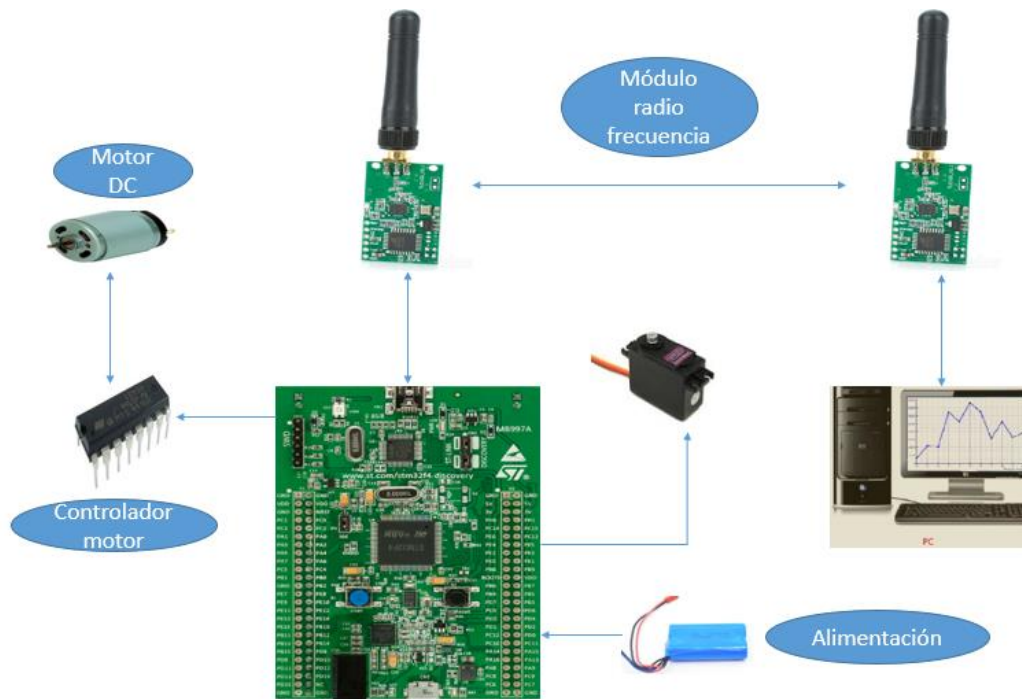


Figura 1: Diagrama de bloques de la arquitectura del sistema

Como puede observarse, a la placa de desarrollo tenemos conectado el módulo de radiofrecuencia como receptor, que se encarga de recoger la información que nosotros le enviamos desde el pc con el emisor (el otro módulo radiofrecuencia). El cual se encuentra conectado a la placa por los pines PC_6 (TX), PC_7 (RX).

Como dispositivo de salida tenemos un servomotor para la dirección que se encuentra conectado al pin PB_6(TIM4).

También, se observa circuito integrado, concretamente el L293d, el cual está conectado a la placa por los pines PA_0 Y PA_1 (TIM2). Este circuito integrado es el controlador del motor.

Y como entrada tenemos una batería de 9V conectada a los pines de 5V y GND de la placa. No conectamos directamente la batería a la placa ya que si le conectamos 9V en el pin de 5V tendremos que comprar una placa nueva porque esta será historia.

Para evitar esto tenemos un pequeño elemento llamado regulador de tensión o BEC, que es el encargado de transformar estos 9V en los 5V que estamos buscando.

2 – Tecnologías empleadas

2.1 – Placa de desarrollo STM32F401

La placa de desarrollo STM32F401, es una placa de la compañía STMicroelectronics que proporciona una manera económica y flexible, para la realización de prototipos con cualquier línea del μC STM32, eligiendo entre las distintas combinaciones de rendimiento, consumo de energía y características. Dispone, además, un soporte de conectividad *ARDUINO*.

El μC STM32, es un potente μC , que dispone de una amplia conectividad con el exterior, gracias a una serie de pines y conectores, que, dispuestos en el PCB, podremos conectar una gran variedad de componentes y dispositivos e interactuar con ellos, bien recibiendo información de los mismos, bien realizando algún tipo de control sobre ellos, tanto de forma directa, (pulsadores, leds, etc.), como de forma indirecta, (comunicaciones I2C, serie, etc.).

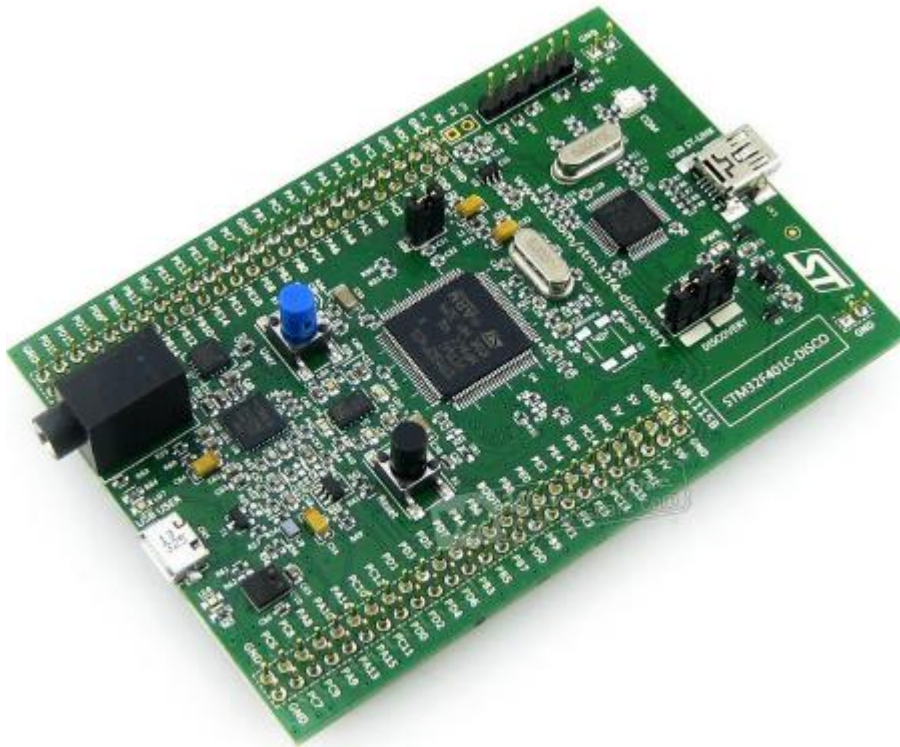


Figura 2: Placa de desarrollo STM32F401

La placa STM32F4DISCOVERY ofrece las siguientes características:

- 1 - Microcontrolador STM32F407VGT6 con 1 MB de memoria flash, 192 KB de RAM, encapsulado LQFP100.
- 2 - ST-LINK/V2 incorporado con selector usar el kit como un ST-LINK/V2 independiente (con conector SWD para programación y depuración).
- 3 - Fuente de alimentación: a través del bus USB o desde una fuente de alimentación externa de 5V.
- 4 - Sensor de movimiento ST MEMS LIS302DL, acelerómetro con salida digital de 3 ejes.
- 5 - Sensor de audio ST MEMS MP45DT02, micrófono digital omnidireccional.
- 6 - Audio DAC CS43L22 con controlador integrado de altavoz clase D.
- 7 - Ocho LEDs: - LD1 (rojo / verde) para la comunicación USB - LD2 (rojo) alimentación 3,3 V - Cuatro LEDs de usuario, LD3 (naranja), LD4 (verde), LD5 (rojo) y LD6 (azul) - 2 LEDs USB OTG LD7 (verde), VBus y LD8 (rojo).
- 8 - Dos pulsadores (usuario y reset).
- 9 - USB OTG con conector micro-AB.

La placa de desarrollo tiene incorporado un módulo de depuración (podemos verlo marcado en la siguiente imagen), que permite efectuar la depuración sobre la marcha, utilizando para ello un programa de desarrollo (En nuestro caso hemos elegido Colde).

Por ello antes de empezar a trabajar con la misma, debemos:

- 1 - Instalar el entorno de desarrollo que deseemos.
- 2 - Debemos instalar el controlador ST-LINK si no lo hace automáticamente. En caso de error el usuario puede proceder con la instalación manual.
- 3 - Establecer la conexión con la placa mediante el conector CN1.

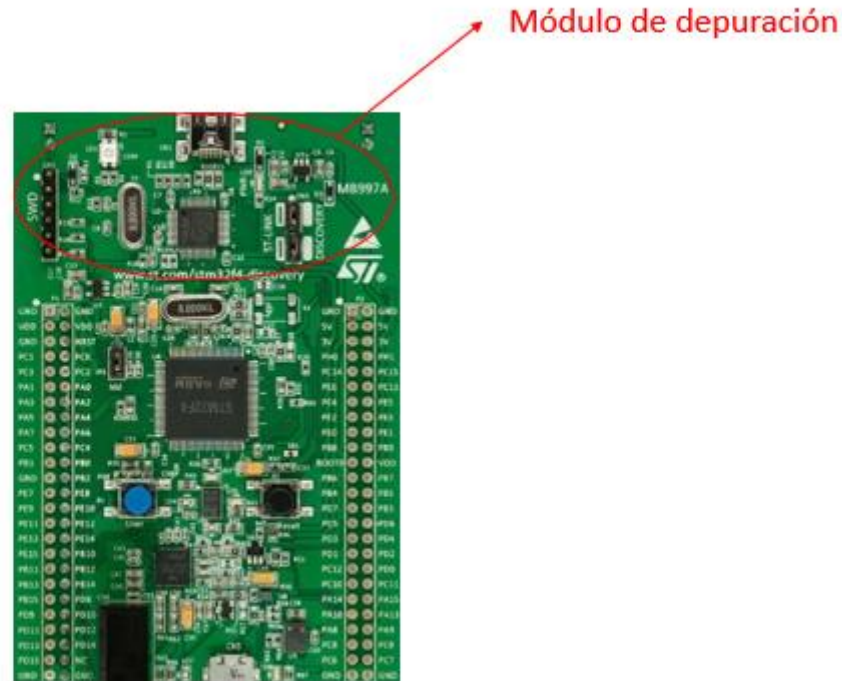


Figura 3: STM32F401

La placa STM32F401, tiene además la posibilidad de alimentación externa, que son de 3,3V a 5V, y otra con un rango de 7 a 12 voltios, administrados a través de una serie de puntos de accesos distintos y controlados por una serie de reguladores como son el LD 1117S50TR para transformar la entrada de 7 - 12 voltios, a 5V.

2.2 – Entorno de desarrollo

En nuestro caso hemos optado por una herramienta de software libre como es CoIde, (para la programación del hardware) que se centra en los microcontroladores ARM Cortex-M0 / M0 + / M3 / M4. Es una versión recortada de la cadena de herramientas Eclipse + GCC (GCC-ARM-Embedded) dedicada al desarrollo integrado, permitiendo a los usuarios experimentar tanto eficiencia y minimalismo en un IDE individual.

Con un poderoso componente de código compartido y una plataforma colaborativa en la nube Git integrada, más de 800 componentes de código están al alcance de la mano, permitiendo a los usuarios implementar programas simplemente apilando bloques de construcción.

Podemos descargarlo aquí: <http://www.coocox.org/software/coide.php>

Una vez descargado pasamos a la instalación:

- 1 - Haga doble clic en los archivos de instalación de CoIDE, tales como: CoIDE-V2Beta-20141205.exe
- 2 - Permitir que el programa se ejecute.
- 3 - Puede hacer clic en los "Términos de uso" para consultar los términos de uso.
- 4 - Haga clic en "Siguiente" para continuar.
- 5 - Elija la ruta de instalación, el menú Inicio y los iconos del escritorio.
- 6 - Haga clic en "Instalar" para comenzar a instalar.
- 7 - La secuencia muestra el progreso de la instalación y la introducción del producto.
- 8 - La instalación está completa.

Ahora solo nos quedaría instalar el compilador GCC, el cual solo tenemos que descargarlo e ejecutarlo. Si no lo hacemos, difícilmente podamos avanzar. Podemos descargarlo aquí: <https://launchpad.net/gcc-arm-embedded/+download>

Una vez lo tenemos instalado solo nos quedaría seleccionar la placa, la cual tenemos una gran variedad de marcas como se puede ver en la siguiente figura.

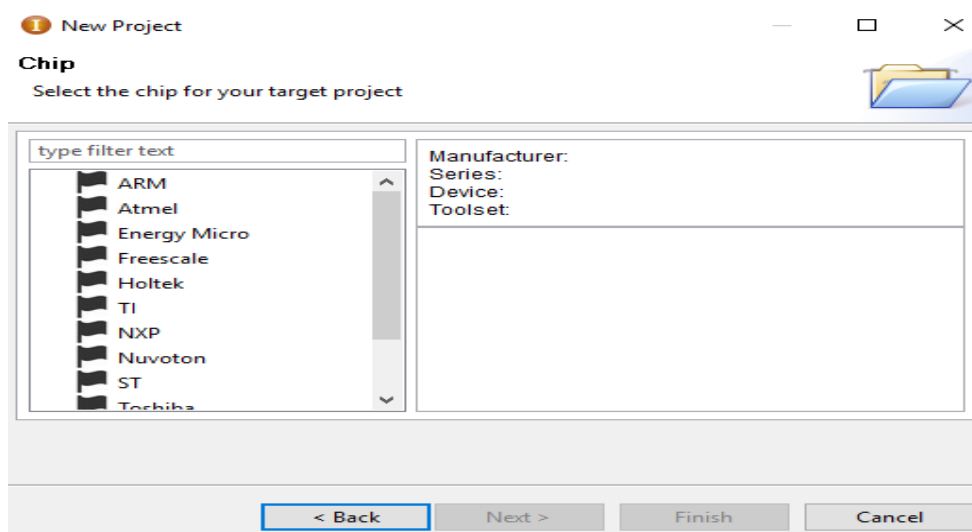


Figura 4: CoIDE, selección de placa

Dentro de cada firma tenemos los diferentes modelos con cada una de sus placas:

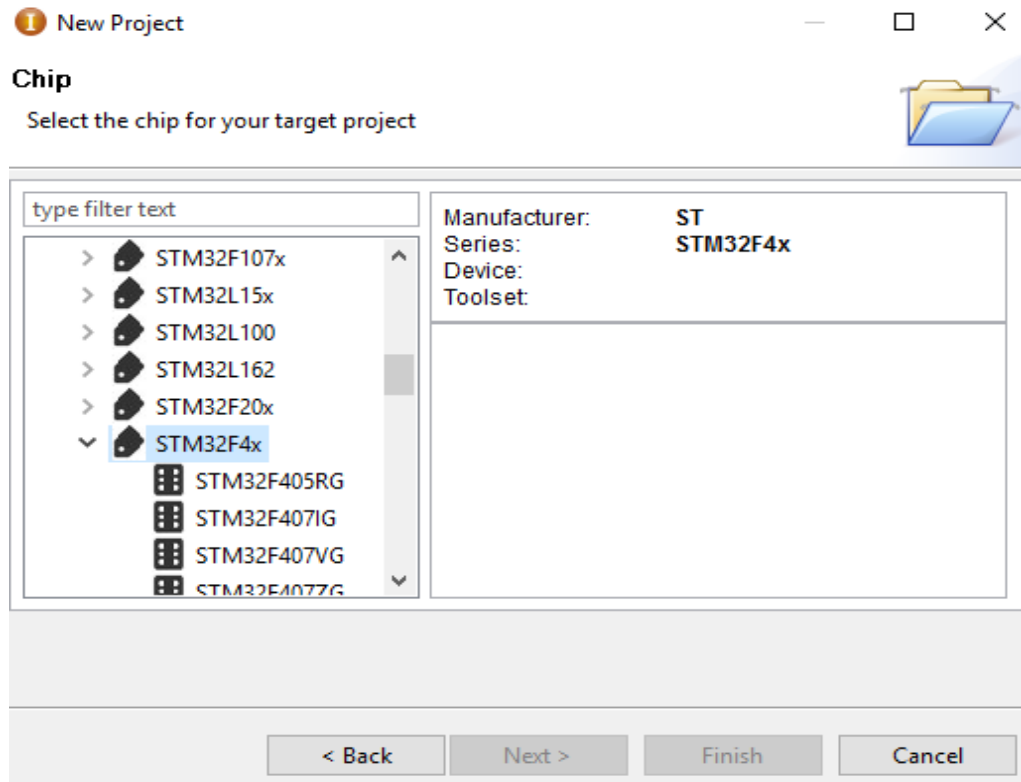


Figura 5: ColIDE, selección de placa

Cuando se selecciona la placa, se pasa a la ventana de proyectos, donde se efectuarán los proyectos con esa placa de desarrollo. Desde esta ventana, podremos crear nuevos proyectos, importar programas ejemplos directamente, y librerías necesarias para las distintas aplicaciones que necesitemos.

En la siguiente foto puede verse una instantánea de la ventana del compilador, de la aplicación donde aparece señalado el nombre del proyecto, (STM32_CoOS), y con selección rectangular, las partes que forman el programa.

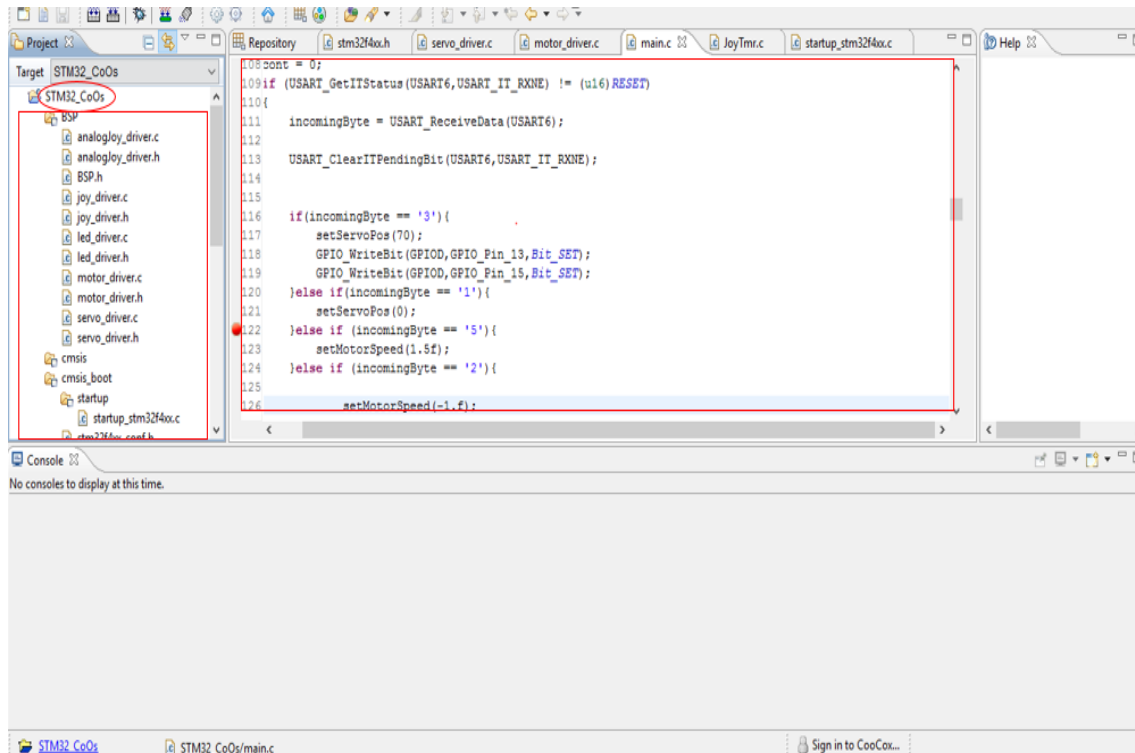


Figura 6: CoIDE, ventana principal

También permite, seleccionar componentes desde la página principal, y podemos obtener acceso a programas ejemplos y una serie de librerías que podemos añadirla desde la pestaña “Repository”.

La edición de los programas que forman parte del proyecto, se realizan en lenguaje C, por ejemplo el programa principal (main).

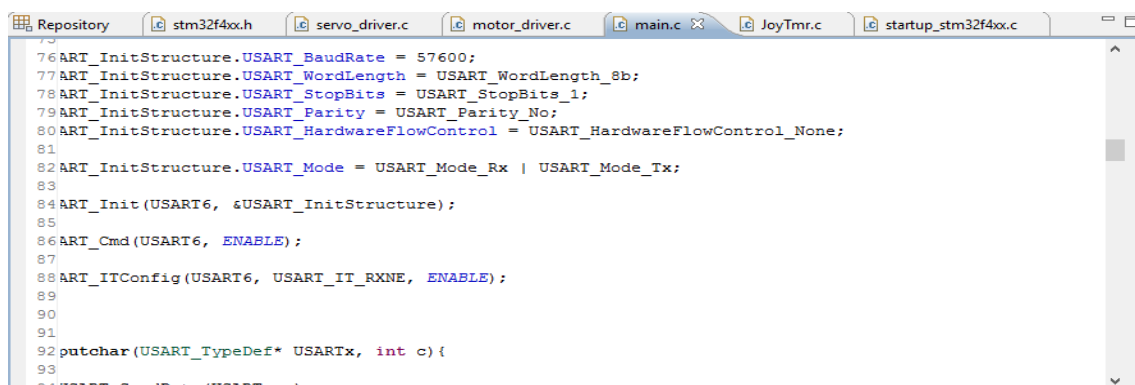


Figura 7: CoIDE, ventana principal

2.3 – Servomotor

Mecanismo que da movimiento a la dirección del vehículo.

Dentro de los servos económicos el servo MG995 Tower Pro destaca por su gran torque, engranajes metálicos y gran robustez. Funciona con la mayoría de tarjetas electrónicas con microcontroladores y además con la mayoría de los sistemas de radio control comerciales.

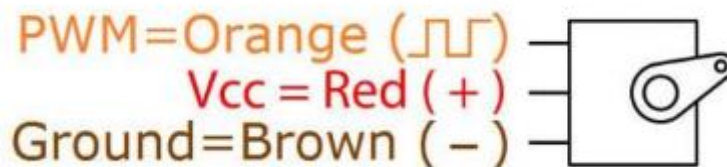
El servo MG995 Tower Pro tiene un conector universal tipo “S” que encaja perfectamente en la mayoría de los receptores de radio control incluyendo los Futaba, JR, GWS, Cirrus, Hitec y otros. Los cables en el conector están distribuidos de la siguiente forma: Rojo = Alimentación (+), Café = Alimentación (-) o tierra, Naranja = Señal PWM.

Recomendamos alimentar por separado el microcontrolador y los servos, ya que el ruido eléctrico puede dar lugar a errores en la ejecución del programa.

Veamos algunas de sus características de interés:

- Tipo de Interfaz: Analógica
- Peso: 55g
- Tamaño: 40.7 x 19.7 x 42.9 mm aprox.
- Torque a 4.8 volts: 8.9 oz/in (10.00 kg/cm)
- Voltaje de operación: 4.0 a 7.2 volts.
- Velocidad de giro a 4.8 volts: 0.2 sec / 60 °

La conexión de la alimentación, se realiza a través de la placa núcleo, en el suministro de +5V y GND, y la de las señales, a través de la entrada analógica PB_6.



Durante el proceso de pruebas y de investigación tuvimos que descartar un servo porque no cumplía los requisitos, que este si cumple.

Principalmente descartamos el servo sg90 ya que no tenía la suficiente fuerza para mover la dirección del vehículo. También influye que el sg90 tenga los mecanismos de plástico mientras que en el MG995 son metálicos.



Figura 8: Servomotores (el izquierdo es el empleado)

2.4 – Puente en H (L293D) + Motor DC

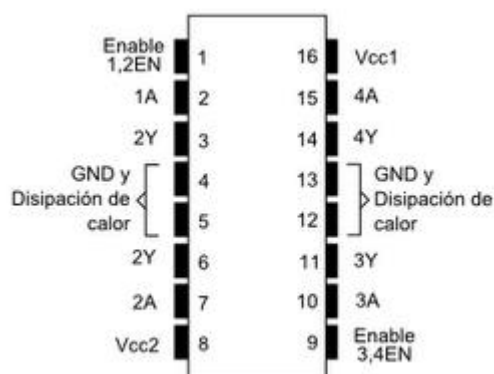
El L293D de Texas Instruments es sin lugar a dudas un circuito integrado de un gran valor cuando necesitamos controlar motores de corriente continua o de paso a paso.

Se trata de un puente en H , en este caso cuádruple, que sin bien podríamos crearlo con transistores, el hecho de que se encuentre integrado en un único chip es de agradecer.

Capaz de conducir corrientes hasta 600 mA en el modelo L293D y con tensiones que van desde los 4.5V hasta los 36V.

Las entradas son de tipo TTL y se activan por parejas, es decir, desde la pata Enable 1,2EN, activamos las entradas 1 y 2 y desde la pata Enable 3,4EN activamos la 3 y la 4. Cada par de entradas forma un puente en H completo (Full-H bridge).

A continuación podéis ver la distribución de su patillaje:



Y como no, un circuito a modo de ejemplo, donde vamos a la conexión que tiene nuestro vehículo:

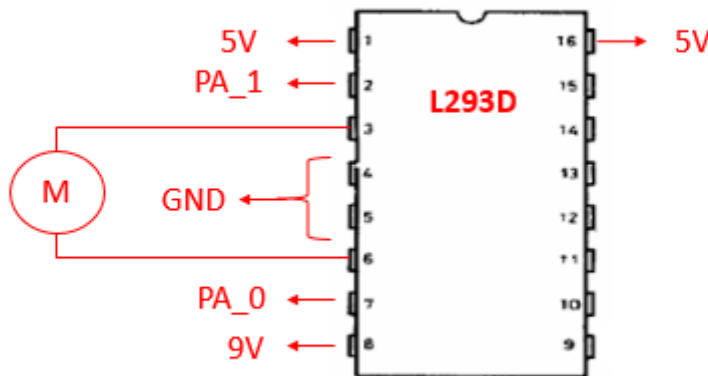


Figura 9: Conexiones puente h

Los 9V del pin 8 provienen de una pila cuadrada de dicho voltaje.

Por último, comentaros que debéis tener cuidado y no confundir las dos tensiones con las que puede trabajar este integrado. Una es la tensión de trabajo del propio chip Vcc1 ubicada en la patilla 16 y que debe ser superior a 4.5V y no debe superar los 7V y, otra es la tensión con la que es capaz de dirigir las cargas o motores que, en este caso como digo va de 4.5 a 36V (Patilla 8 Vcc2).

2.5 – Módulo Radiofrecuencia (TTL dual 3DR Radio)

Sistema de transmisión/recepción de datos sensores de su APM y GPS.

Este aparato de radio telemetría te permite enlazar a un controlador de un vehículo como piloto a través de un dispositivo equipado USB, como un ordenador, portátil o tableta que soporta una conexión USB. El sistema utiliza la banda de 433Mhz y proporciona un enlace dúplex.

El firmware SiK incluye un cargador de arranque que permite actualizaciones de firmware radio a través de la interfaz serie, y la configuración de parámetros del sistema. Actualizaciones y configuración del firmware son totalmente compatibles con

el sistema APM Mission Planner. La configuración también es posible a través del configurador 3DR Radio y comandos AT.

A continuación vamos a ver cómo va conectado este dispositivo en nuestro vehículo. Mientras que conectamos uno de ellos al PC el otro dispone de cuatro cables:



Figura 10: Módulos radiofrecuencia 3DR

RX – PC_6, TX – PC_7, 5V, GND

2.6 – Regulador de tensión (UBEC)

El UBEC es un regulador de conmutación avanzada con más de corriente, y la eficiencia máxima del sistema es de casi el 90%.

El tamaño pequeño y el peso ligero hacen que sea muy cómodo de usar.

No te preocupes por la polaridad de la batería. Si la polaridad no es correcta, el UBEC puede no trabajar, pero no será destruida. Lo que hay que hacer es cambiar la batería polaridad.

Un escudo cubre casi todos los componentes electrónicos en un CIRCUITO IMPRESO, y un anillo se adjunta con los cables de salida para reducir la interferencia electromagnética.

Además en la salida tenemos un switch que nos permite decidir si queremos a la salida 5V o 6V. En nuestro caso utilizaremos 5V.



Figura 11: Regulador de tensión UBEC

3 – Desarrollo del Hardware

Para poder realizar las conexiones de todos los elementos que componen el dispositivo Hardware con garantías de conexión eléctrica, se realizará en montaje del circuito en una protoboard.

Para el montaje hemos realizado una pequeña caja de madera de 20x15cm para la construcción y almacenamiento del cableado.

La conexión de cada uno de los periféricos ya la hemos visto previamente, pero aún no hemos comentado un factor muy importante como la alimentación de la placa y la tensión del motor que introducimos en el L293D.

Para alimentar la placa estamos utilizando una batería de 9,6V. Pero no la conectamos directamente ya que solo necesitamos 5 voltios. Para ello tenemos como bloque intermedio, un regulador de tensión o más conocido como BEC.

El BEC se encarga de estabilizar las tensiones de corriente continua usadas por el microcontrolador y los demás elementos como el servo.

La conexión es la siguiente, 9V de la batería pasan por el BEC (debe tener el jumper de salida a 5V si lo tiene a 6V podremos quemar el micro) y la salida de este la introducimos en los pines de 5V Y GND de la placa STM32.

4 – Desarrollo del Software

El dividido el desarrollo del software en dos partes:

- 1) Firmware.
- 2) Software de usuario (Aplicación de gestión y configuración).

4.1 Firmware

El desarrollo de éste software, se realiza en el entorno de compilación CoIDE el cual hemos detallado anteriormente y es una plataforma y CoOS el sistema operativo, para dispositivos conectados a internet, basado en microcontroladores ARM-Cortex, de 32 bits. También dispone de opciones de importar, exportar y compartir código. Se compone de las bibliotecas del núcleo que proporcionan los controladores de periféricos del microcontrolador, redes y entorno de ejecución, construcción de herramientas y scripts de prueba y depuración.

El software sigue el siguiente diagrama de bloques:

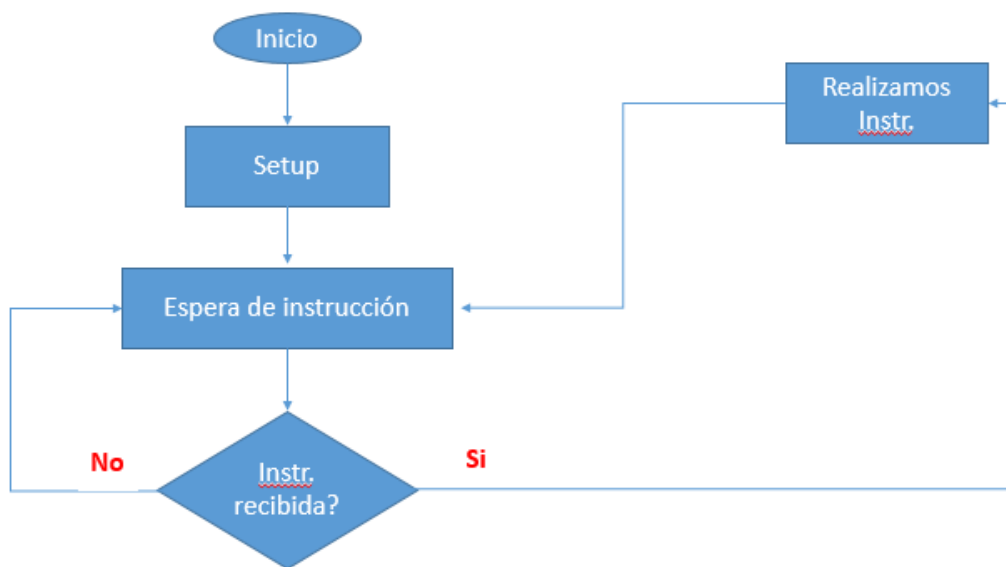


Figura 12: Diagrama bloques Firmware

La recepción y emisión de los datos (instrucción en el diagrama anterior) solicitados por el puerto serie, se controlarán por interrupción, llamando a las funciones `USART_ReceiveData ()` y `USART_SendData ()`.

En la función `SetUp ()`, se configuran las velocidades de transmisión de los módulos de radiofrecuencia (57600 bd), se configuran los periférico, es decir, el motor, el servo y los pines de la propia placa como por ejemplo el `GPIOD` para los leds, cuyas conexiones ya hemos visto anteriormente.

Las instrucciones nos llegan desde la interfaz de usuario a través de los módulos de radiofrecuencia.

La función `main()`, lo primero que hace es llamar a las subrutinas de configuración del puerto serie para la recepción de datos (`USART3`). Continúa con las subrutinas de inicialización del motor de corriente continua y el servo motor de dirección.

Una vez realizada la función Setup, no tenemos función Setup realmente escrita sino es una globalización de las subrutinas de inicialización para el diagrama de bloques, entramos en el bucle principal donde esperamos a que nos llegue alguna instrucción, nos pueden llegar las siguientes:

- Girar motor hacia adelante con velocidad progresiva.(Flecha arriba)
- Girar motor hacia atrás.(Flecha abajo)
- Servo a izquierda.(Flecha izquierda)
- Servo a derecha.(Flecha derecha)

Tanto motor como servo son independientes con lo cual podemos realizar varias instrucciones a las vez siempre cuando no se trate del mismo periférico que lo haremos de modo secuencial.

Si realizamos un giro del servomotor este volverá a la posición central tras terminar la instrucción, es decir, mientras tengamos la flecha del lado correspondiente presionada.

4.2 Software de usuario

El desarrollo de esté software, se realiza en Microsoft .NET, que es una plataforma de desarrollo y ejecución de aplicaciones. Esto quiere decir que no sólo nos brinda todas las herramientas y servicios que se necesitan para desarrollar modernas aplicaciones, sino que también nos provee de mecanismos seguros y eficientes para asegurar que la ejecución de las mismas sea óptima. Los componentes principales de la plataforma .NET son:

- Un entorno de ejecución de aplicaciones, también llamado “Runtime”, que es un componente de software cuya función es la de ejecutar las aplicaciones .NET e interactuar con el sistema operativo ofreciendo sus servicios y recursos.
- Un conjunto de bibliotecas de funcionalidades y controles reutilizables, con una enorme cantidad de componentes ya programados listos para ser consumidos por otras aplicaciones.
- Un conjunto de lenguajes de programación de alto nivel, junto con sus compiladores y linkers, que permitirán el desarrollo de aplicaciones sobre la plataforma .NET.
- Un conjunto de utilitarios y herramientas de desarrollo para simplificar las tareas más comunes del proceso de desarrollo de aplicaciones.

Bajo esta plataforma se usa el lenguaje de programación C++. En definitiva se trata de un lenguaje moderno, intuitivo y muy eficiente, que mejora la productividad en el desarrollo de software.

El software, sigue el siguiente diagrama de bloques:

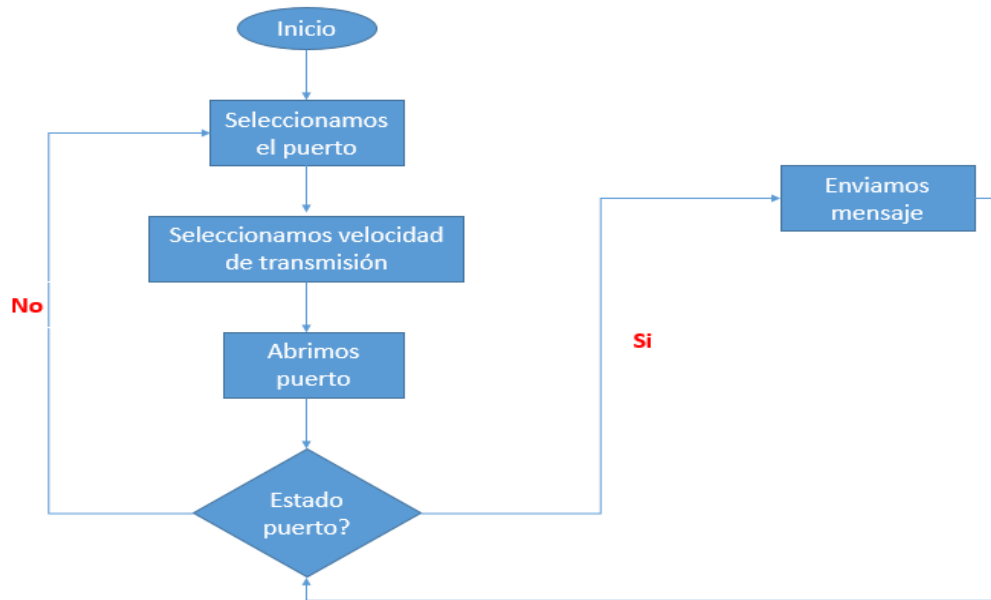


Figura 13: Diagrama bloques interfaz usuario

Para el desarrollo de la aplicación, haremos uso de las librerías necesarias.

En nuestro caso, las llamaremos espacios de nombre y no librerías aunque, al fin y al cabo son un conjunto de clases.

- *System::IO::Ports*, contiene clases para controlar puertos serie. La clase más importante, *SerialPort*, proporciona un marco de trabajo para la E/S sincrónica y orientada a eventos.
- *System.Windows.Forms*, contiene clases para crear aplicaciones basadas en Windows que aprovechen al máximo las características mejoradas de interfaz de usuario de las que dispone el sistema operativo Microsoft Windows.
- *System.Collections* contiene interfaces y clases que definen varias colecciones de objetos, como listas, colas, matrices de bits, tablas hash y diccionarios.
- *System.ComponentModel* espacio de nombres proporciona clases que se utilizan para implementar el comportamiento de tiempo de ejecución y tiempo de diseño de componentes y controles. Este espacio de nombres incluye las clases bases e interfaces para implementar atributos y convertidores de tipos, orígenes de enlace a datos y licencias de componentes.



ESCUELA TÉCNICA SUPERIOR
DE INGENIERÍA INFORMÁTICA

¿Cómo funciona este software?

Lo primero que hacemos es inicializar los componentes del formulario, seguidamente llamamos a la función *findPorts()* que como su nombre indica es la encargada de buscar puertos series abierto en el equipo y mostrarlos en el *comboBox* correspondiente.

Una vez que seleccionamos el puerto que deseamos indicamos la velocidad de transmisión que en nuestro caso es 57600 bd que es a la velocidad que trabaja nuestro módulo de radiofrecuencia. Seguidamente abrimos el puerto dando clic en el botón iniciar puerto mediante la función: *button1_Click()* que abre el puerto si es posible, sino nos mostrara un mensaje de error en el *textBox1*, en caso que pueda abrir el puerto correctamente la *progressbar* se rellenara de color verde y tendremos el canal abierto para comenzar la comunicación.

Cuando terminemos las comunicaciones debemos cerrar el puerto en el botón *cerrarPuerto*.

Para enviar instrucciones al vehículo solo tenemos que hacer clic en las flechas de nuestro teclado para enviar las instrucciones como antes detallamos y controlar el vehículo.

¿Qué tecnología hemos empleado para realizar esta interfaz de usuario?

Pues la respuesta a la pregunta es Microsoft Visual Studio 2010.

Microsoft Visual Studio es un entorno de desarrollo integrado para sistemas operativos Windows. Soporta múltiples lenguajes de programación tales como C++, C#, Visual Basic .NET, F#, Java, Python, Ruby, PHP. También puede ser empleado para el desarrollo web como ASP .NET MVC, Django...

Visual Studio permite crear aplicaciones web al igual que nos permite desarrollar sitios web. Así se pueden crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos, consolas...

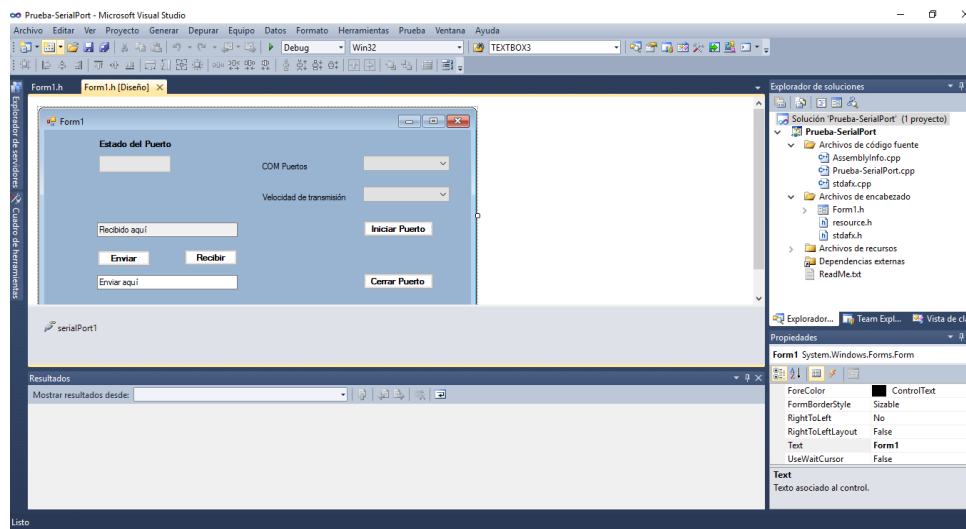


Figura 14: Pantalla principal Visual Studio 2010

4.3 Comunicación entre Software y Firmware

La comunicación entre el Firmware y el Software de la aplicación se realiza a través de un dispositivo de radio frecuencia 3DR mediante mensajes, estamos usando para esta comunicación el software HyperTerminal para pruebas.

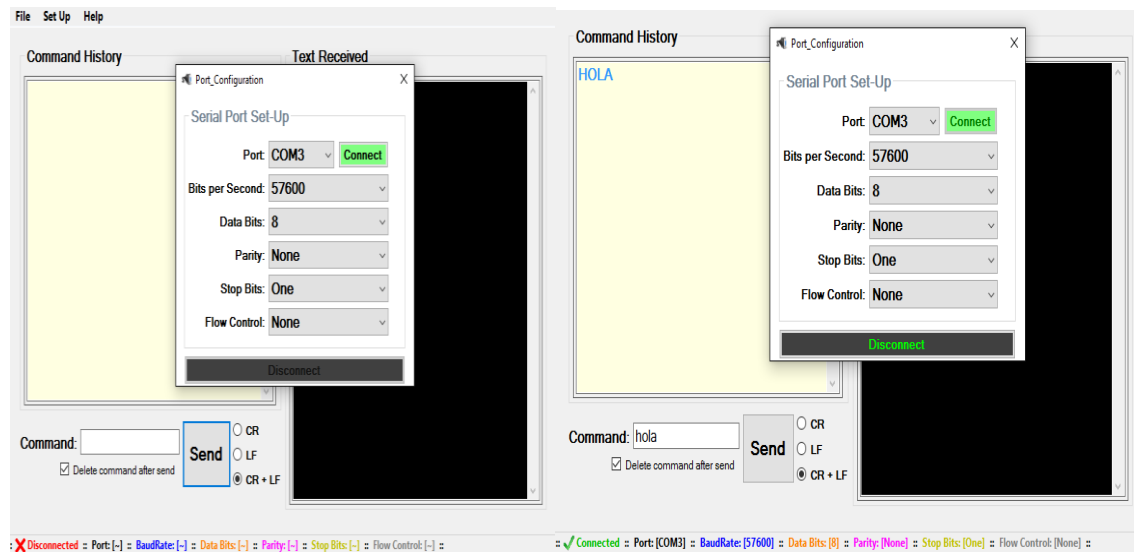


Figura 15: Interfaz HyperTerminal

En la imagen de la izquierda podemos ver el programa recién arrancado. En el estamos intentando iniciar el puerto (Pestaña Set Up/ Port Configuration).

En la segunda imagen tenemos el puerto abierto, un mensaje enviado y otro preparado para enviar.

Pero en nuestro caso hemos usado una comunicación Firmware-Software que hacemos nosotros mismo con una aplicación .NET como antes hemos comentado.

Se trata de una interfaz de usuario muy similar a Hyperterminal pero desarrollada por nosotros y mucho más cómoda a la hora de enviar los paquetes.

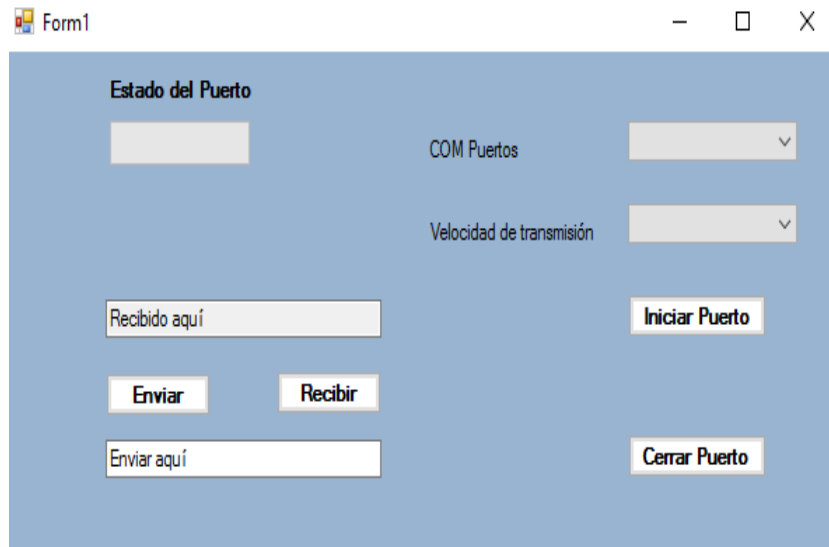
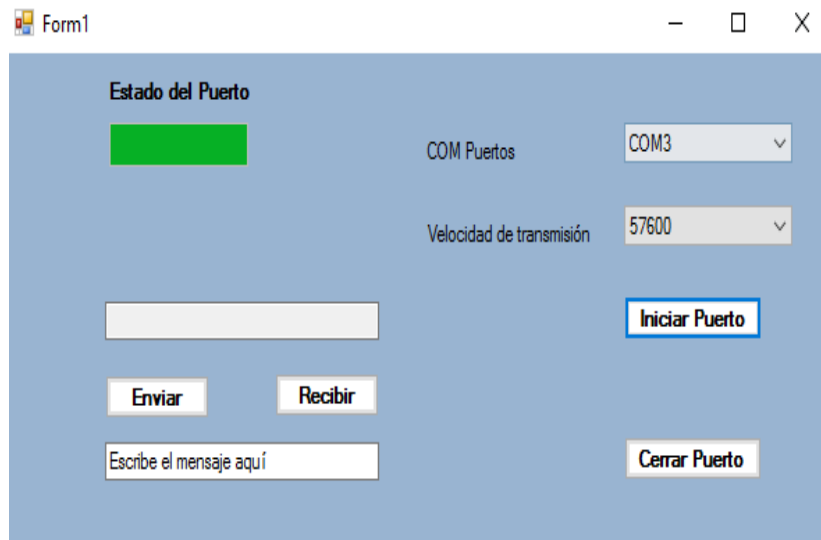


Figura 16: Nuestro software de usuario

Esta imagen muestra el programa tras ejecutarse.



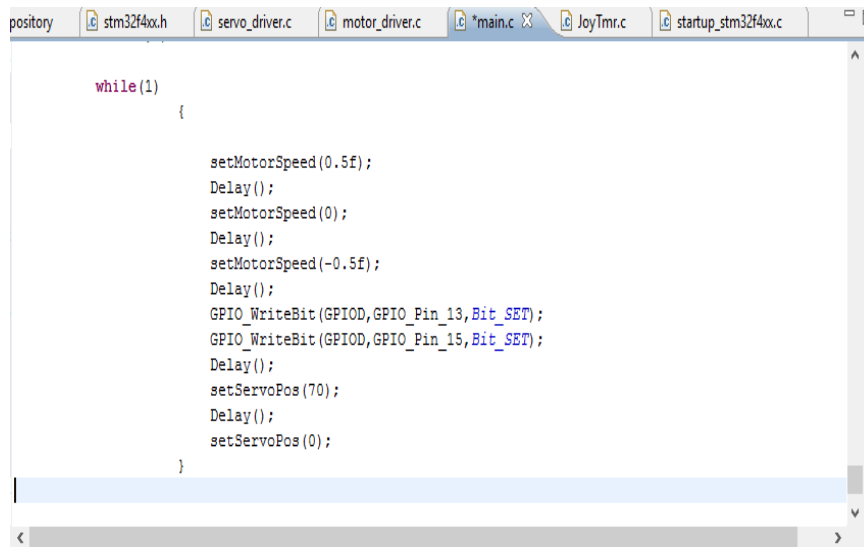
La imagen anterior muestra el módulo emisor de radio conectado y listo para usarse.

5. Pruebas y resultados

5.1 – Pruebas del software de STM32

Estas pruebas de software son realmente rápidas y sencillas. Principalmente nos encargamos de comprobar que el funcionamiento de cada uno de los periféricos sea correcto, es decir, para cada periférico podemos realizar un *main.c* independiente con alguna de las funciones individuales de cada periférico o podemos tener un proyecto de prueba (este es nuestro caso) en el que comprobamos todos los periféricos y la propia placa principal (leds) de manera secuencial.

A continuación vamos a ver una imagen donde vemos la secuencia principal anteriormente descrita (programa de prueba número 1):



```
while(1)
{
    setMotorSpeed(0.5f);
    Delay();
    setMotorSpeed(0);
    Delay();
    setMotorSpeed(-0.5f);
    Delay();
    GPIO_WriteBit(GPIOD,GPIO_Pin_13,Bit_SET);
    GPIO_WriteBit(GPIOD,GPIO_Pin_15,Bit_SET);
    Delay();
    setServoPos(70);
    Delay();
    setServoPos(0);
}
```

Figura 17: Software CoIDE (*main.c*)

Puede ocurrir que los periféricos no respondan. Debemos tener en cuenta también que podríamos tener alguna conexión mal (Importante este punto).

Si sigue algún periférico sin respondernos puede la última opción es que la función de configuración este mal. Como nosotros tenemos estas funciones predefinidas y siempre son iguales para nuestros productos este fallo debería ser el menos ocurente.

En el hipotético caso de que esto ocurra debemos saber que estas funciones se encuentran en la clase llamada como el periférico en cuestión, por ejemplo: Para el motor DC deberíamos ir a configurarlo a *motor_driver.c*

Volvemos a repetir esto no debería ocurrir.

5.2 – Pruebas de la comunicación software – hardware

Esta prueba se basa en comprobar el funcionamiento del módulo de radiofrecuencia y a su vez el correcto funcionamiento de la interfaz de usuario.

Para ello ejecutamos el software de usuario a su vez cargamos en la placa el software de prueba número 2 que es el mismo del anterior pero con el siguiente bucle principal:

```
char a;

while(1)
{
    a=getchar(USART3);
    putchar(USART6,a);
    Delay();
}
```

Este bucle se encarga de la comunicación entre ambos software.

```
void USART6_IRQHandler(void)
{
    int cont = 0;
    if (USART_GetITStatus(USART6,USART_IT_RXNE) != (u16)RESET)
    {
        incomingByte = USART_ReceiveData(USART6);

        USART_ClearITPendingBit(USART6,USART_IT_RXNE);

        if(incomingByte == '3'){
            GPIO_WriteBit(GPIOD,GPIO_Pin_13,Bit_SET);
            GPIO_WriteBit(GPIOD,GPIO_Pin_15,Bit_SET);
        }
    }
}
```

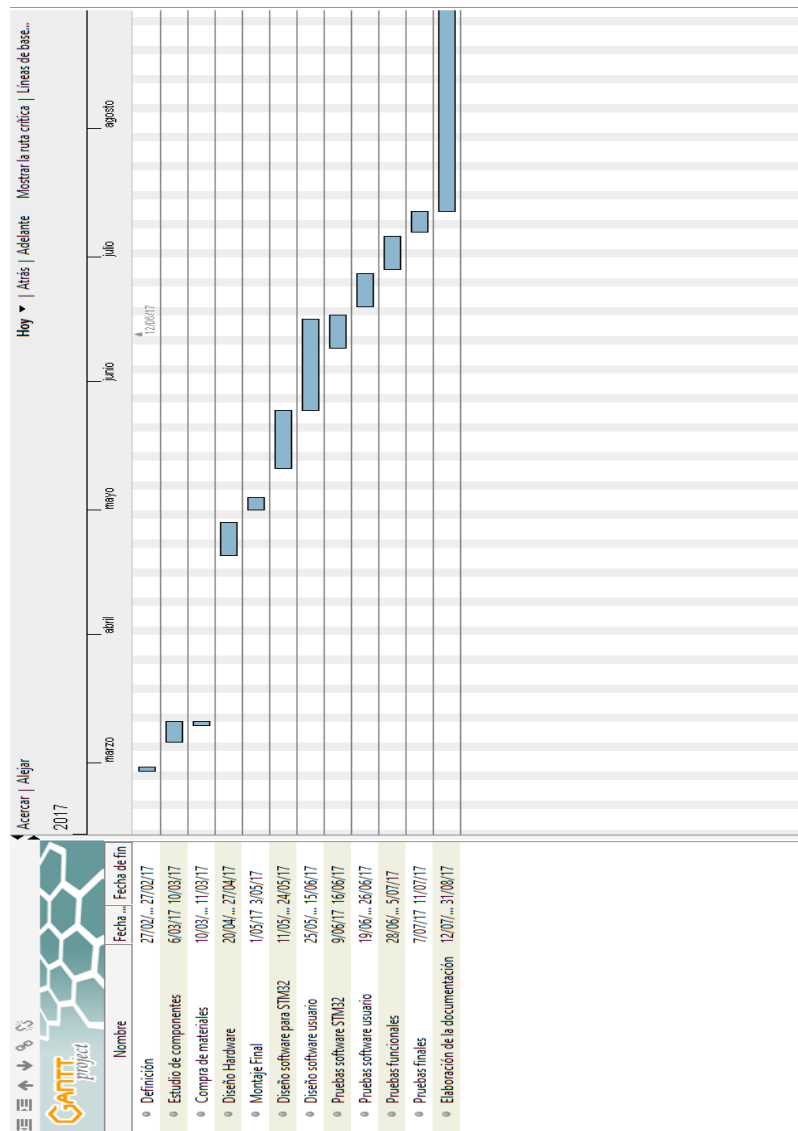
Y con esta pequeña función encendemos los dos leds cuando pulsemos un 3 en la interfaz de usuario.

6. Planificación y Costes

Vamos a detallar en una pequeña tabla los tiempos de ejecución.

Tarea	Duración (Horas)
Definición	6
Estudio de componentes	8
Compra de materiales	18
Diseño Hardware	20
Montaje final	13
Diseño software para STM32	35
Diseño software usuario	25
Pruebas software STM32	15
Pruebas software usuario + comunicación	16
Pruebas finales	20
Pruebas funcionales	16
Elaboración de la documentación	101
Total	302

En la siguiente imagen, se muestra un diagrama de Gantt de las operaciones y de los recursos. La temporización no es por horas, sino por días completos de 8 horas cada uno.



6-1. Costes del desarrollo

El coste de los componentes del Dispositivo Hardware, después de realizar una búsqueda por varias tiendas on-line, para comparar precios, RS-online u otras páginas webs que venden de forma general, como eBay, Amazon o Dealextreme. Los precios que vamos a ver a continuación son precios por unidad, esto quiere decir que si vamos a producir nuestro producto en cantidad podemos reducir los costes entre un 20% y un 30%.

Lista de componentes con enlace de proveedor y precio

Producto	Enlace	Precio
STM32F4 Discovery	http://es.rs-online.com/web/p/kits-de-desarrollo-de-procesador-y-microcontrolador/7892819/	25,50 €
Servomotor	http://www.dx.com/es/p/mg995-tower-pro-copper-servo-gear-for-r-c-car-plane-helicopter-black-173907?tc=EUR&gclid=CjwKCAjw2NvLBRAjEiwAF98GMUz43nI98s5ybtY8ZBEstsMbdWULhKZvuzHgJtSNlwBy9UFIHMEJ6hoCSJsQAvD_BwE#.WXc1_ojyi00	4,97 €
Módem 3DR	http://www.dx.com/es/p/433mhz-single-ttl-3d-robotics-3dr-radio-telemetry-kit-for-apm-apm2-blue-green-235604#.WXc08ojyi01	23,61 €
Puente H	http://www.ebay.es/itm/like/172576845674?chn=ps	2,99 €
UBEC	http://www.dx.com/es/p/hobbywing-5v-6v-3a-switch-mode-ultimate-bec-ubec-15149#.WXh_nYjyiIU	3,37 €
Varios		10,00 €
Total		70,44 €

En gastos varios nos referimos a la protoboard, cables, leds, etc...

Falta contabilizar el precio de la mano de obra de la persona que lo implemente, que con un sueldo de 1.800€ brutos al mes, y en un periodo de tiempo de 1 mes y 1 semana, tenemos un gasto de 2.320,44 € por todo. Ni que decir tiene que las primeras fases (montaje) del proceso podrían automatizarse y mandar el mayor tiempo posible a las pruebas.

Si le añadimos nuestro beneficio, que será de un 15%, y el impuesto del IVA, (21%), el precio final de venta, será de 3.155,80€.

6-2. Costes de producción

El coste calculado, es el de UNA unidad, por lo que como pensamos en la producción en masa, (en principio de 300 unidades), pasamos a calcular la amortización del prototipo. Calcularemos el precio por unidad.

$$\text{Amortización} = \frac{\text{Coste total}}{\text{nº unidades}} = \frac{3.155,80 \text{ €}}{250} = 12,6 \text{ €}$$

Si calculamos el descuento por cantidad que nos harían los proveedores de los elementos, en un 15% aproximadamente, el nuevo valor del prototipo, pasa a ser:

$$\text{Coste prototipo} = \text{Coste prototipo} \times 0,85 = 67,07 \times 0,85 = 57,00 \text{ €}$$

Ahora calcularemos el coste del sistema completo sumando al coste, la amortización calculada, y el coste del montaje en serie, que suponemos será de 10,00 € por unidad.

$$\text{SubCosteProducto} = \text{Coste prototipo} + \text{Amortización} + \text{montaje} = 57,00 + 12,60 + 10,00 = 79,6 \text{ €}$$

Ya solo queda calcular el precio final al público, incrementando el beneficio, (15%), y el IVA, (21%), con lo que queda:

$$\text{Coste Total} = \text{SubCosteProducto} + \text{Beneficio } 20\% + \text{IVA } 21\% = 108,26 \text{ €}$$

Es un precio realmente barato, lo que daría pie a que se distribuyera muy fácilmente.

7. Hitos

7.1 – Contratación de servicios externos

La idea principal es montar el circuito en una PCB para ello se contratará a una empresa externa, por lo que se nos permitirá realizar paralelamente, la consecución de los siguientes hitos.

7.2 – Generación de equipo de pruebas

Se realizará la fabricación de un equipo de pruebas compuesto por 4 personas y un pequeño circuito con obstáculos que se asemejen a las situaciones cotidianas.

Además abriremos un programa de formación para personas cualificadas que se vayan incorporando a esta fase tan importante del negocio.

7.3 – Resultado esperado

Una vez finalizado el proyecto se espera haber conseguido los resultados siguientes:

- Correcto funcionamiento del vehículo, tanto hardware como software y la comunicación entre ambos.

7.4 – Análisis de riesgo

Pueden ocurrir varias cosas que nos afecten seriamente a nivel económico y de prestigio.

- Un retraso en una de las fases del proyecto, lo cual conllevaría un fuerte impacto económico. La probabilidad de que esto ocurra es bastante elevada del 50%, pero es algo que irá aminorando con el paso del tiempo.
- Errores de comunicación software-hardware, esto es algo de menor probabilidad ya que partimos de la idea del trabajo en cadena.

8. Problemas surgidos

Durante la fabricación del producto, el tema de potencia necesaria para el motor ha sido algo bastante conflictivo, ya que a medida que incorporábamos piezas y el peso aumentaba.

- El problema venia que al dividir la tensión de la batería entre la alimentación de la placa principal y toda la circuitería no dejaba suficiente potencia al motor. Con lo cual hemos solucionado este problema alimentando directamente el motor con una pila de 9V.

- El segundo problema que nos ocurrió fue la elección de un servo ya que la primera compra fue de un servomotor Mini SG-90 de 9g el cuál no tenía la fuerza suficiente para mover la dirección del vehículo cuando se le colocaban todos los elementos encima. Una de los principales problemas es que este servo tiene los engranajes de plástico con lo cual la fuerza que puede ejercer está bastante limitada. Por ello nos decidimos por un servo de 12 kg de fuerza en palanca con los engranajes de cobre y con el mismo tamaño del anterior el cual funcionaba a la perfección. Con lo cual este fue el elegido.

9. Fabricación del producto

Este producto debe tener unas características específicas de tamaño y peso para que el motor sea capaz de tirar con todo.

En el diseño buscamos minimizar el número de elementos que el vehículo debe soportar, se basa de un soporte que contenga adecuadamente solamente los elementos imprescindibles de la circuitería. Para ello tenemos un pequeño recipiente de madera (panel poco pesado).

Dicha caja va sujeta al chasis del vehículo con dos agujeros para dos tornillos de forma que no se mueva lo más mínimo en los giros. Para reducir espacio la batería de 9V la hemos sujetado en la parte inferior del vehículo mientras que hemos dejado esta caja para la placa, protoboard, pila 9V, UBEC y todos los cables necesarios para las conexiones. El motor está situado en el eje trasero y el servomotor va atornillado a 3 cm de distancia del eje.



Figura 18: Servomotor en chasis

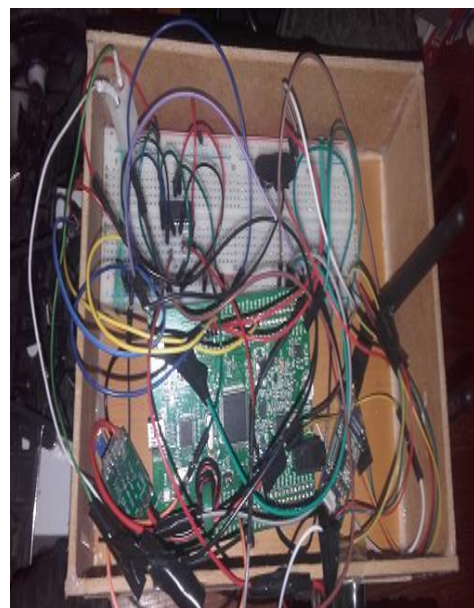


Figura 19: Circuitería completa

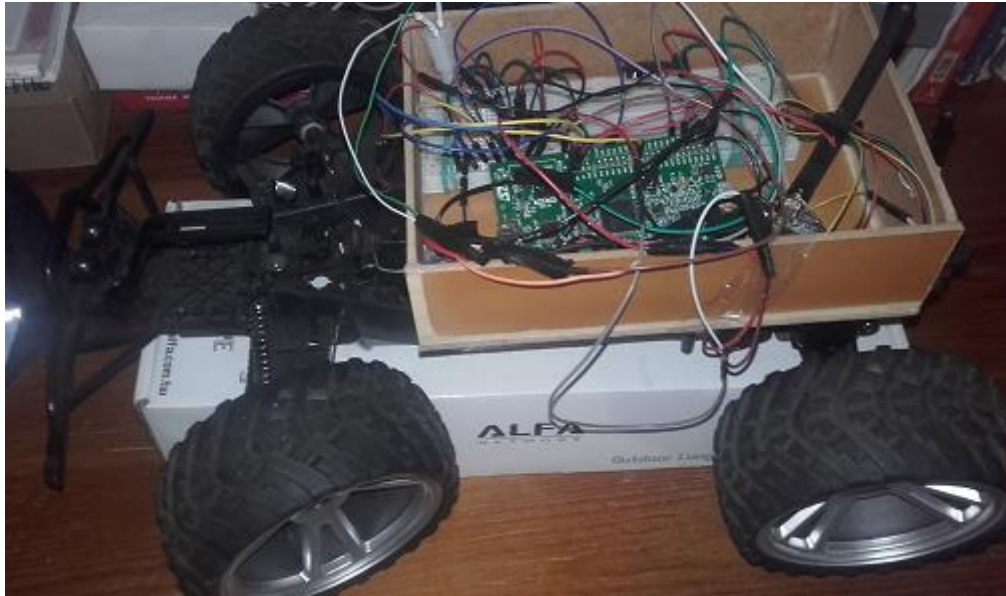
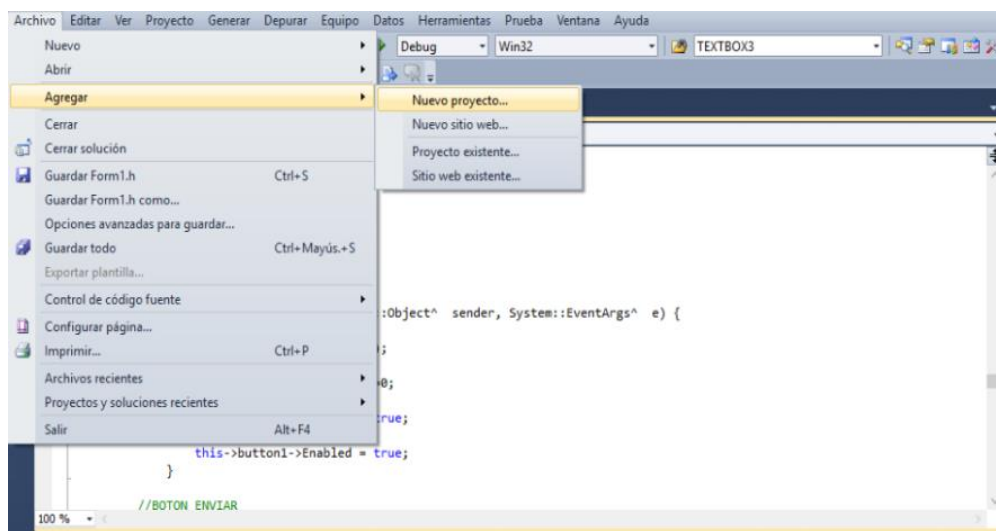


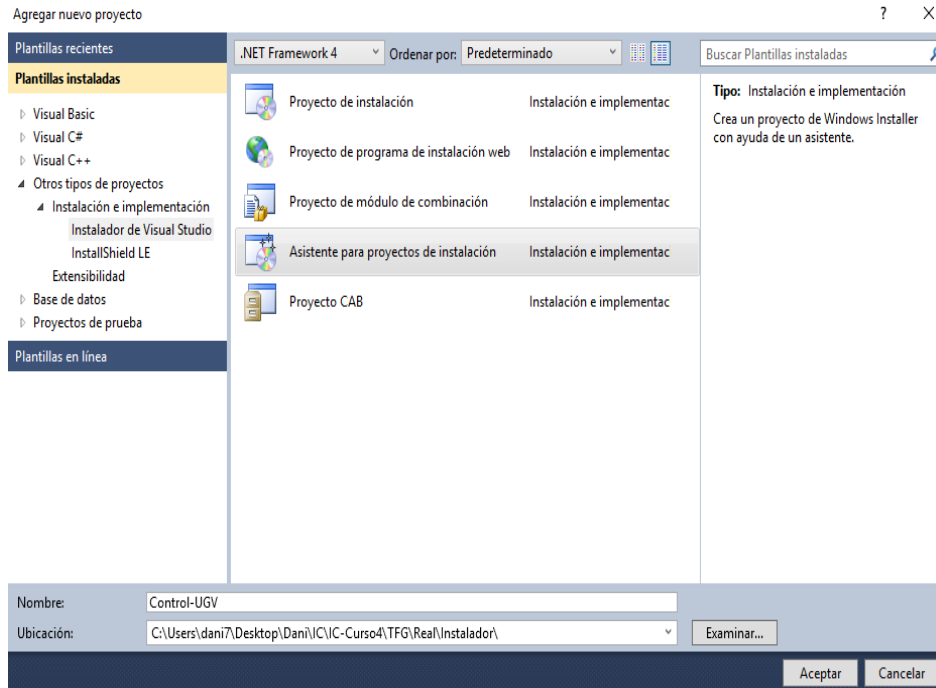
Figura 20: Vehículo completamente equipado

10. Generación del instalador

Una vez desarrollada la interfaz de usuario el último paso es generar un instalador que nos permita instalar la aplicación en el equipo y poder ejecutarla. El primer paso será abrir nuestro proyecto en Visual Studio 2010, una vez lo tengamos abierto nos vamos a *Archivo -> Agregar -> Nuevo proyecto...* como se muestra en la siguiente imagen.



Se nos abrirá la siguiente ventana:



Debemos seleccionar tal y como se ve en la imagen *Asistente para proyectos de instalación*, le damos un nombre, una ruta y clic a *Aceptar*. En la ventana que se nos abre hacemos clic en *Siguiente*.

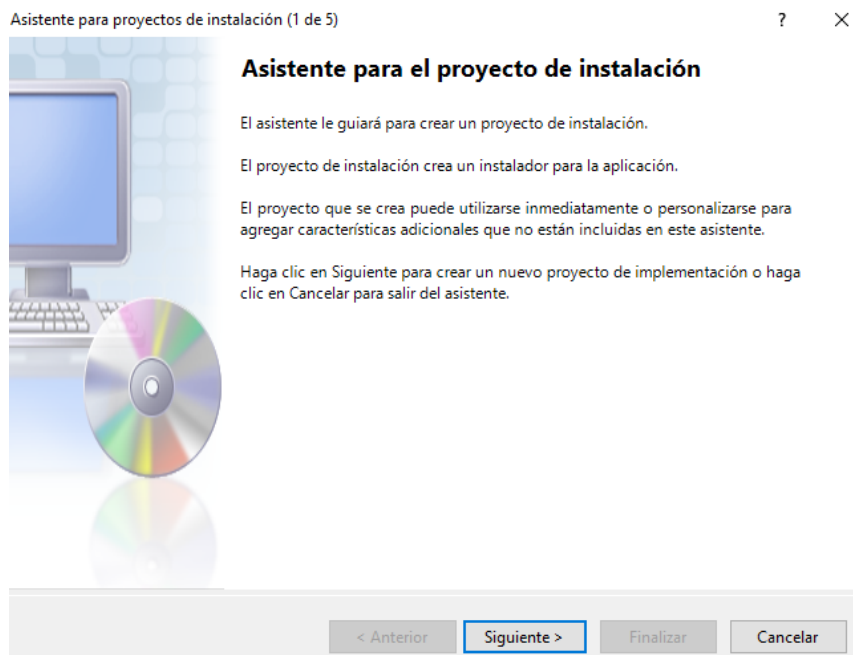


Figura 21: Iniciamos el instalador

En la siguiente ventana, seleccionamos la primera opción y hacemos clic en *Siguiente*.

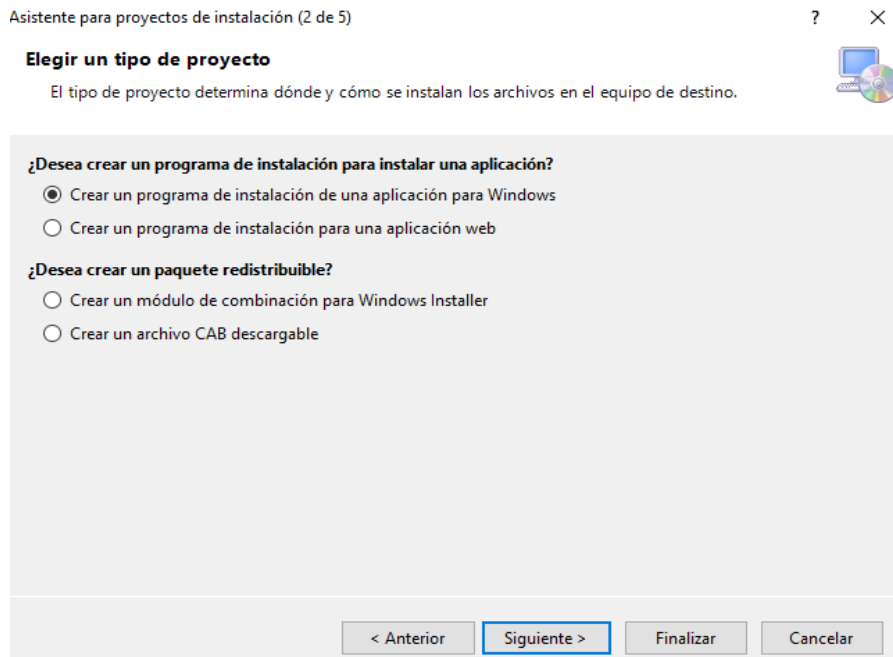


Figura 22: Crear un programa de instalación de una aplicación para Windows

En la siguiente ventana debemos hacer exactamente lo mismo.

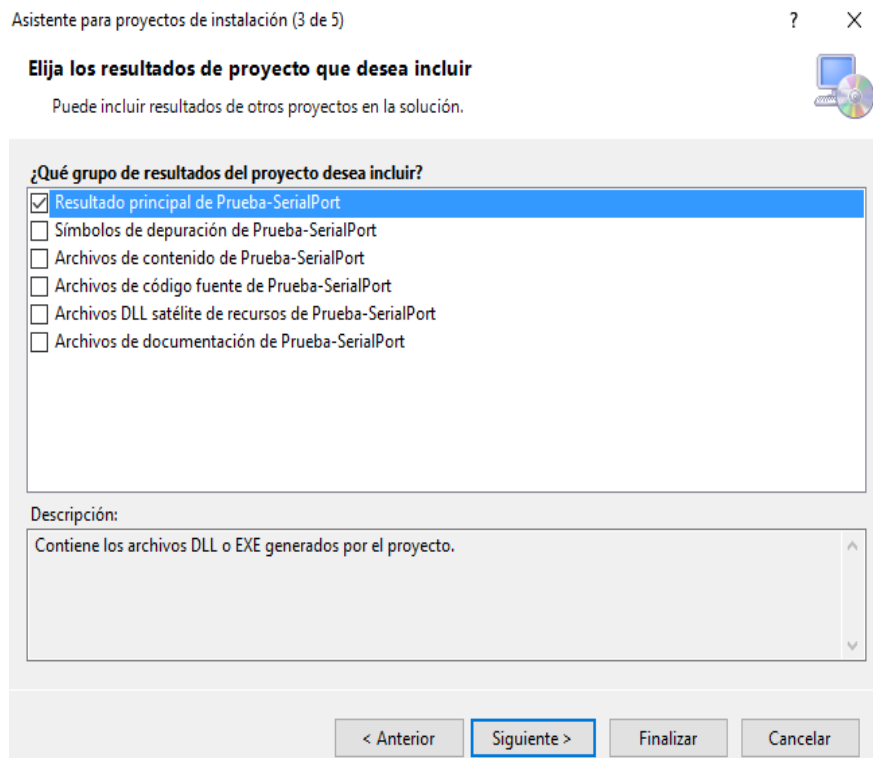
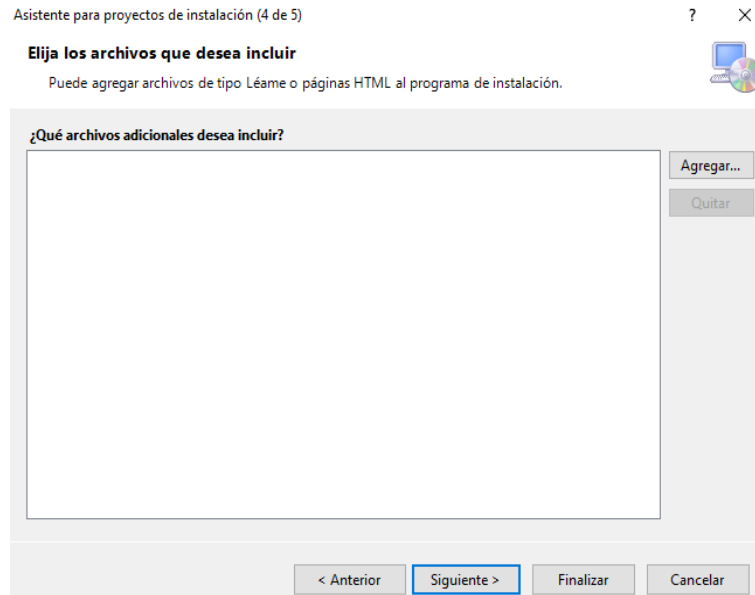


Figura 23: Resultado principal

En la siguiente ventana que se nos abre no debemos hacer nada simplemente clic en *Siguiente*.



Tras esto se nos abrirá otra ventana en la que damos a *Finalizar* y se nos abrirá el siguiente proyecto de Visual Studio:

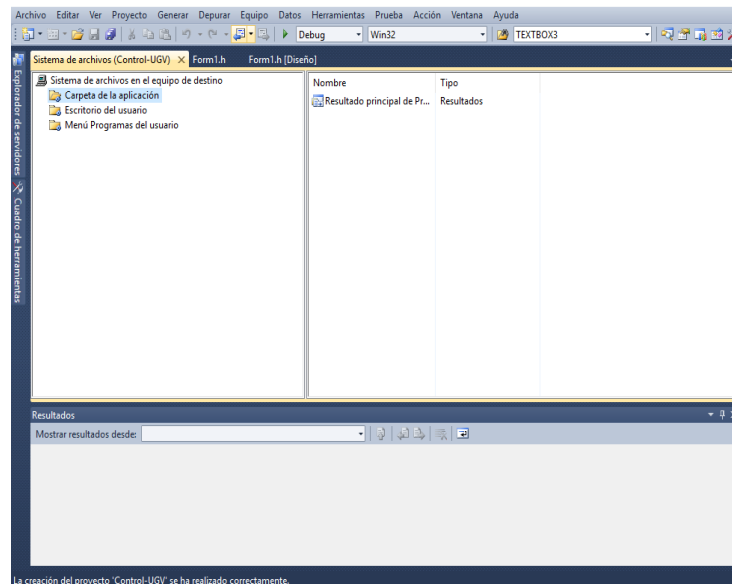
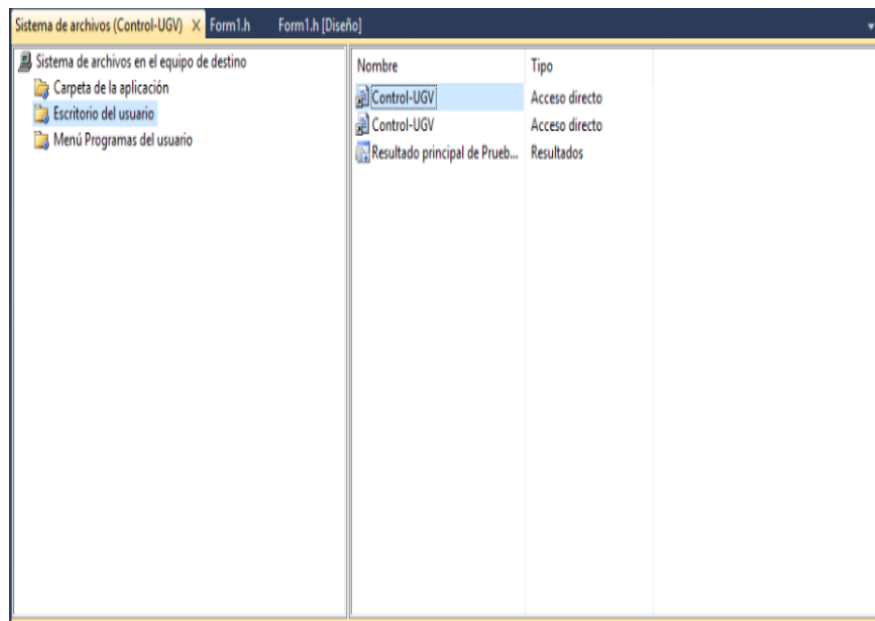
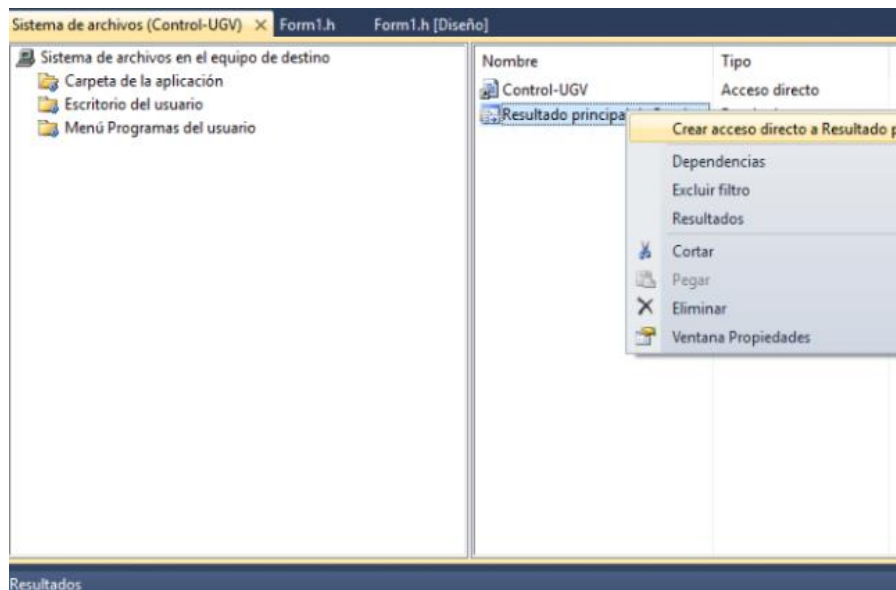


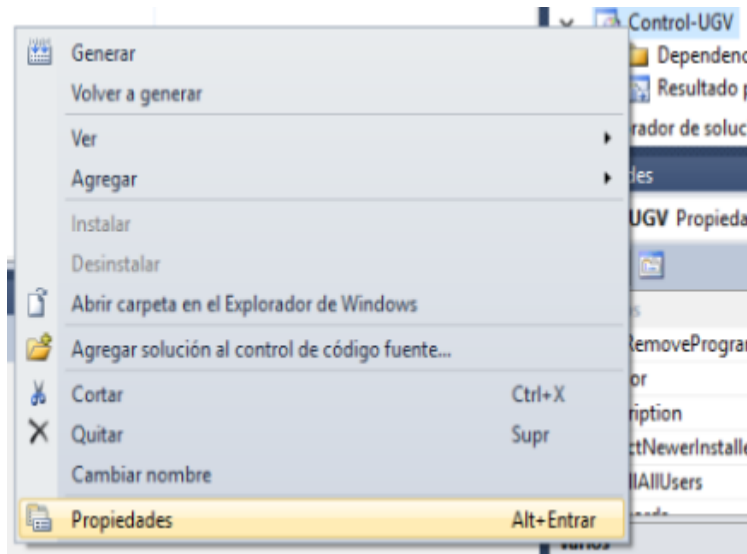
Figura 24: Proyecto del instalador en Visual Studio

En el explorador de la parte izquierda seleccionamos *Carpeta de la Aplicación* y en el explorador de la derecha veremos un archivo de tipo *Resultado* con nombre *Resultado principal de...*

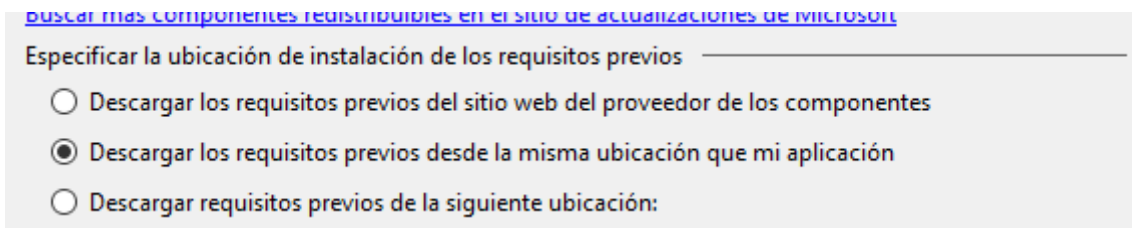
Hacemos clic derecho sobre el archivo y le damos a crear acceso directo. Esto lo debemos hacer dos veces como se muestra en la siguiente imagen. Yo a cada acceso directo le he dado el nombre de *Control-UGV* que será el nombre que llevará el instalador.



A continuación nos vamos a las propiedades del proyecto *Control-UGV*.



En propiedades nos vamos a *Requisitos* y seleccionamos la segunda opción y *Aceptar*.



Por ultimo clic derecho de nuevo en el nombre del proyecto, en nuestro caso *Control-UGV* y generamos el instalador.



Una vez hecho todo esto tendremos el instalador en la carpeta que pusimos como ubicación del proyecto de instalación.

En nuestro caso en *Control-UGV/Debug* encontraremos el archivo *setup*.

11. Instalación de la aplicación

Una vez generado el instalador la aplicación se instala en unos simples pasos. Primeros ejecutamos el *setup* anterior mente creado y hacemos clic sobre *Siguiente*.

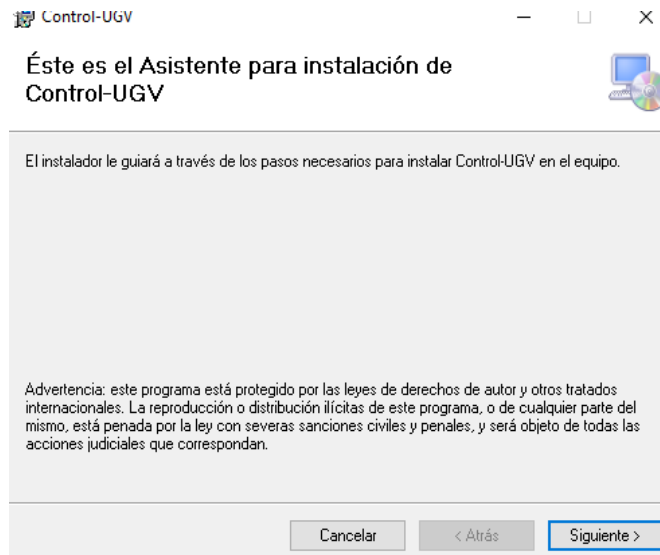


Figura 25: Ventana principal del instalador

Posteriormente se nos abre un formulario donde podemos indicar el directorio donde se instalará e indicar sobre que usuarios, sino se modifica se instala en el directorio por defecto donde Windows instala los programas para el usuario activo;

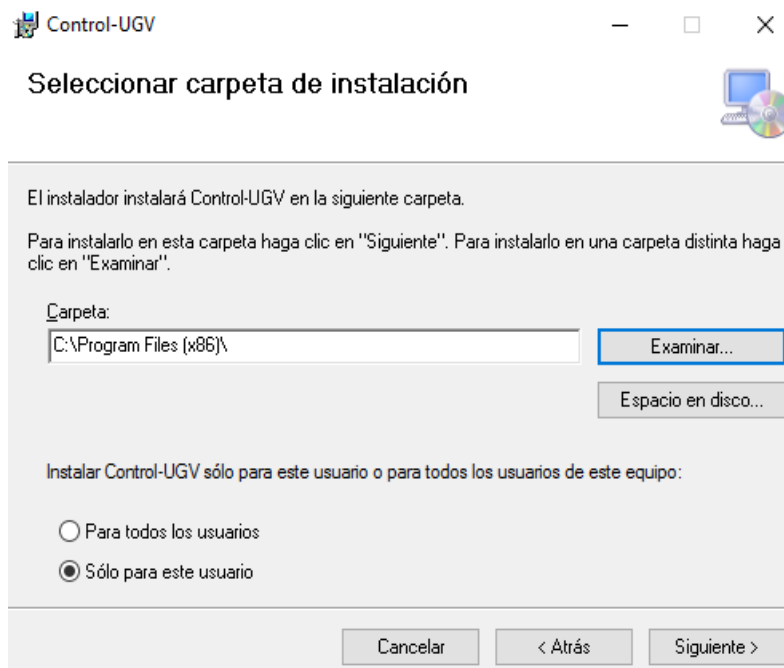
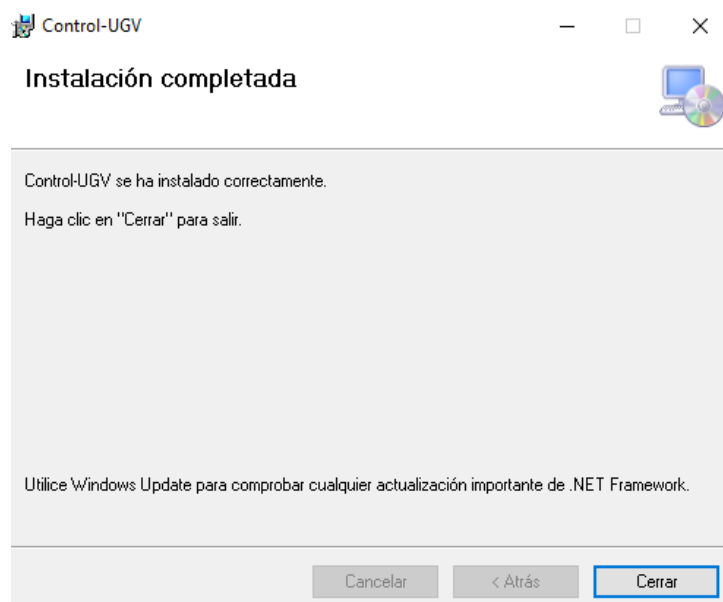


Figura 26: Configuración del instalador

Hacemos clic en siguiente en la ventana anterior y en la siguiente que nos aparece. Y una vez se instale la aplicación hacemos cerramos el instalador.



12. Desinstalación de la aplicación

El mismo instalador que hemos empleado para instalar el software nos sirve ahora para desinstalarla, si la aplicación no está instalada nos dará como opción su instalación, sin embargo, si está instalada tendremos dos opciones:

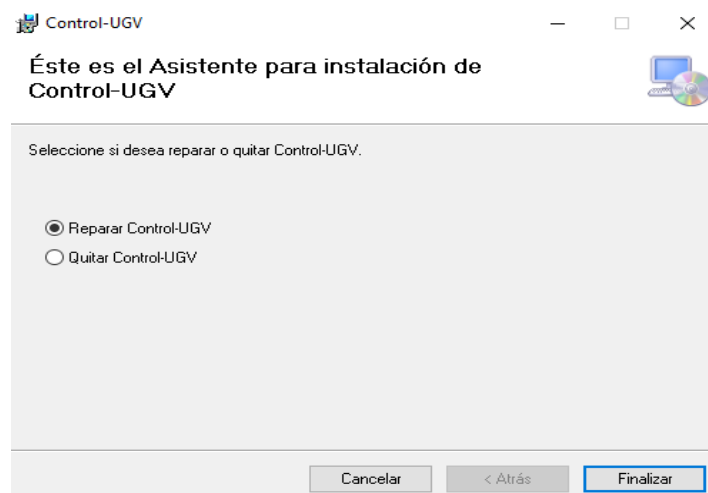
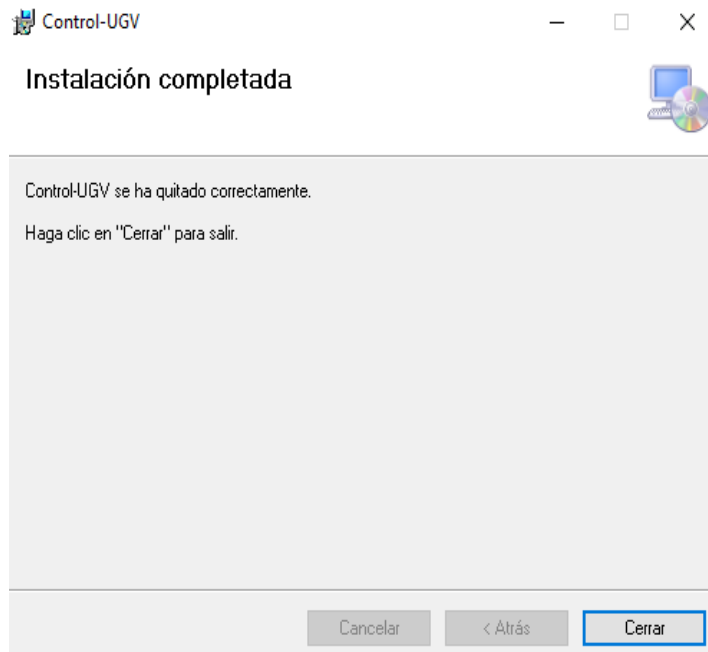


Figura 27: Ventana de reparación o eliminación

Si por cualquier razón, la aplicación no funcionara correctamente, podremos elegir *Reparar Control-UGV* y volverá a reinstalarla corrigiendo los posibles errores de la instalación. Elegimos la opción *Quitar Control-UGV* y hacemos clic en *Finalizar* y la aplicación se desinstalará por completo del equipo.



Conclusiones

Las conclusiones del proyecto las hemos dividido en dos vertientes, una a nivel docente y otra a nivel de utilidad.

En cuanto a nivel docente, pienso que es bastante recomendable, ya que pasamos por todas las fases aprendidas durante el Grado en Ingeniería Informática, empezando por la investigación para elegir las mejores tecnologías, pasando por el estudio de librerías, datasheets, compiladores y finalizando con la creación tanto en hardware como en software de un sistema funcional completo.

A nivel de utilidad es algo que realmente ya existe y está en “producción”, pero también es cierto que está en plena iniciación con lo cual podríamos abrirnos un hueco en el mercado. La principal utilidad que pensamos que más beneficio podríamos obtener sería como vehículo para desactivar artefactos explosivos o para abrir caminos en operaciones de guerra en territorios hostiles y desconocidos. Pero las utilidades detalladas en la introducción nos darían mayor margen de beneficios con lo cual trabajaremos en todas las ramas haciendo varios departamentos especializados en un área específica

En definitiva, este proyecto es muy interesante, ya que nos ha permitido poder investigar y aprender muchas cosas, que podrían aplicarse en un futuro en otro tipo de proyecto.

Trabajos futuros

Este es un proyecto, que permite realizar una serie de modificaciones y mejoras que conseguirían completar el sistema y llevarlos a otras superficies. De haber dispuesto de tiempo y de financiación, los trabajos que realizaría sobre el proyecto son:

1. Añadir un detector de presencia a cada lado del vehículo para añadir al software de usuario un pequeño mapa y poder saber a qué distancia tenemos los objetos y mejorar así la conducción y alargar la vida del producto.
2. El siguiente paso sería adaptarle una cámara en la parte superior y ver la imagen en tiempo real desde nuestro monitor. Esto es algo que se salía claramente del presupuesto con lo que no hemos podido adaptarla. Además se le añadiría una tarjeta SD para poder almacenar las imágenes captadas por la cámara y el sonido en el caso que fuera posible. Con esto se elevaría bastante el precio pero el vehículo sería muy superior al anterior. Hay que tener en cuenta que sería la principal puesta a punto en el caso de tener financiación.
3. Automatizar totalmente al vehículo y dotarlo de inteligencia artificial. Con esto nos referimos a convertir el vehículo en un agente inteligente capaz de aprender por si solo y poder llegar a lugares sin necesidad de tener un operador detrás. Un ejemplo sería para una empresa de reparto de paquetería donde le indicaríamos al vehículo las coordenadas correctas y él se presentaría allí, lo dotaríamos de inteligencia artificial en la medida de lo posible por ejemplo para actualizar sus mapas en el caso de calle encontrarse una calle nueva.
4. Y por último nos gustaría ampliar la flota al espacio aéreo. La idea sería un vehículo exactamente igual en cuanto a funcionalidad pero que se desplazara por el aire, como un dron, para reducir principalmente el tiempo por ejemplo en el caso de la empresa de reparto, la ganancia de tiempo sería brutal.



ESCUELA TÉCNICA SUPERIOR
DE INGENIERÍA INFORMÁTICA

Bibliografía

[1] Wikipedia

[2] Guía programación c# / c++

http://www.lcc.uma.es/~vicente/docencia/cppdoc/programacion_cxx.pdf

[3] Datasheet STM32

http://www.disca.upv.es/aperles/arm_cortex_m3/curset/guia_iniciacion_STM32_F4_discovery.pdf

[4] Introducción a los UGV's y extracción de ideas para las utilidades de nuestro proyecto

<http://ixion.es/ugv-robotica-terrestre/>

<https://prezi.com/opnejgt2mof/desarrollo-de-un-vehiculo-terrestre-ugv-para-la-deteccion/>

[5] Stackoverflow

<http://stackoverflow.com>