Escuela Universitaria Politécnica
La Almunia de Doña Godina
Zaragoza

ROBOT SEGUIDOR DE LÍNEAS COMO TRANSPORTE EN UN ALMACÉN

Nº:
106.10.70

MEMORIA

REALIZADO POR:
GARAIGORTA ZARZUELA, IÑAKI

Julio - 2011

Índice

1. Descripción del Proyecto

Aunque el campo de la robótica es ampliamente utilizado en multiples sectores, en el que destaca, sobre todo el industrial. El campo de la microbótica (una subrama de la anterior) , no es tan conocida.
Desde su aparicion, en la decada de los 90, gracias a los cada vez más modernos microcontroladores, aun no se a llegado a ver el gran potencial de estos dispositivos.

Cierto es, que desde finales de los 90, hasta la actualidad, miles de utilidades han salido a la luz. Diseños cada vez más ingeniosos de los que muchos consideraban meros "juguetes". Campos como la medicina, agricultura, ya empiezan a ver estos "pequeños hermanos" de los robots como posibles soluciones a muchos de sus problemas cotidianos. Al mismo tiempo ciertos servicios comienzan a ver viable el uso de autmotizaciones. Los ejemplos que podemos mencionar son abundantes, domótica, microrobot espias,reconcimiento aereo, salvamento,para herradicación de plagas, como automatización en el servicio de reparto de materiales..

Precisamente de este último ejemplo se basa la propuesta que les hago llegar en este proyecto. Una aplicación de microbótica en el mundo real. Concretamente automatizar un almacen mediante el uso de un microbot.
A diferencia de sus hermanos mayores , el desarrollo y mantenimiento de esta "herramienta" seria mucho menor.

Basandome en el modelo más típico de microbot (el seguidor de lineas), me prongo en este proyecto a mostrar un prototipo, funcional que nos serviria para tranportar mediante un carro motorizado, material desde un punto de recojida hasta diversos puntos de un almacen.

Mi idea es que desde ese punto inicial y con solo pulsar un simple boton en un programa monitor, el microbot se desplace sin tener que molestarse en vigilarlo ni manipularlo de ninguna otra forma. El propio microrobot, cuando termine su descarga, volvera al punto de inicio y esperará nuevas ordenes.

Además del modo autónomo, el programa monitor nos mostrará el estado de sus sensores, y nos permitirá un control manual del mismo por si necesitamos manipular el microrobot.

2. Actividades esenciales para el proyecto

El proyecto **deberá** contar con las siguientes características:

-***Un modelo funcional del microrobot***: El modelo contendrá diferentes sensores para poder desenvolverse en las tareas de desplazaminento y comunicación con el terminal.

-***Un software controlador para el microbot***: Se deberá desarrollar el software interno necesario, para que el robot pueda desenvolverse tanto en modo autónomo, como en modo manual. Además tendrá que poder comunicarse con el terminal. Para que se pueda ver que el modelo es viable, debemos mostrar que el microbot es capaz de moverse por un camino diseñado . Esto incluye ida hasta un destino, simular una descarga de algún modo, volver al punto de inicio y ponerse en disposición de recivir más ordenes.

-***Un software monitor*** : Crearemos ademas una aplicación para que sirva de puente entre el operario y el microbot. Este software monitor debe de incluir las siguientes características:

- Algún indicador de que se esta conectado al robot (ya sea mediante un indicador o mediante la lectura de parametros del robot).

-Un interfaz que nos permita seleccionar cada uno de los caminos que programemos.

-Un interfaz que nos permita controlar manualmente el microbot.

2.1 Actividades optativas para el proyecto

El desarroyo de este proyecto conlleva un proceso de mucho y duro trabajo. Por eso, estas caracteristicas solo representan un añadido extra al diseño del microbot.

Las siguientes características son optativas y solo se deberán plantear como un extra al ya de por si complicado proyecto:

-***Emulación del robot***: Crear un software, que se llame desde la aplicación del robot y simule el comportamiento del microbot en un circuito.

-***Programación de multiples caminos***:Aunque un camino de muestra sea indispensable para poder comprobar el correcto funcionamiento del microbot, sería interesante la programación de multiples caminos.

-***Añadir un sistema inalámbrico de comunicación***: La placa skypic contiene una comunicación serie. Este tipo de comunicación nos es suficiente para poder trabajar con el ordenador, pero no implica que sea ni mucho menos la más correcta. Diseñar otro método para la comunicación sería verdaderamente muy interesante, pero implicaría un costo de esfuerzo desorbitado.

3. Soluciones propuestas

3.1. Solución elegida para el desarrollo del proyecto

3.1.1 Selección del hardware

Para poder elegir una solución para el proyecto debemos pensar en las características del mismo. Este proyecto tiene una naturaleza didáctica y al mismo tiempo aporta un modelo práctico.

Por ello, lo primero en lo que debemos pensar es en la plataforma harware en la que tenemos que trabajar.

Durante ya algún tiempo la universidad Carlos III de Madrid (al igual que otras muchas otras universidades de España), ofrecen cursos de formación en el campo de robotica y microbótica usando como herramienta las placas **skypic** y **sky293**.
Estas placas fueron diseñadas por Andrés Prieto-Moreno, Juan González y Ricardo Gómez.

Además en la página de http://www.iearobotics.com (referente nacional en microbótica), aconsejan dichas placas como placas de iniciación. En dicha página encontraremos abundate información para que demos nuestros primeros pasos, tanto en el hardware como en software.Esto me hizo decantarme por este hardware frente a otras propuestas.

Respecto al PIC que usaré en la placa skypic, no tenía mucho problema. Esto se debe a que la placa está diseñada para trabajar con el **PIC 16F876A,** aunque se podría trabajar con cualquier otro modelo de la familia de los 16F8XX.
El escoger el modelo 16F876A y no su versión más usada y común, la 16F84, es por que, este último modelo ya está desfasado en características. Además, como indica la '**A'** del modelo 16F876A ,se incluye un convertidor analógico/digital-digital/analógico , que en un futuro nos puede ser de utilidad para incluir todo tipo de sensores.

Tenemos elegido el cerebro de nuestro microbot, pero nos falta el esqueleto,ojos y los músculos.

Para crear una estructura que se adapte a nuestras necesidades seleccioné las **piezas de Lego** (en concreto una caja con referencia 8048 y otra caja con referencia 8052) . Es cierto que frente a otras posibilidades puede resultar un poco más cara, pero nos ahorra el gasto extra que supondría el comprar herramientas para poder trabajar con otros materiales. Las ventajas de seleccionar piezas Lego como materia prima para nuestro chasis es que aportan, solidez, adaptabilidad y un fácil sistema para crear formas. Ademas las piezas son difíciles de romper y fácil de reparar en caso de que se desmonte algún elemento.

Como extra, si nos hacemos con una caja de piezas variada conseguiremos unas buenas ruedas para los motores.

A la hora de seleccionar un motor, me decanté por las opciones que mencionaban en http://www.iearobotics.com . Aquí explican que la mejor opción es el uso de servomotores. Son asequibles y además ofrecen una buena potencia. La marca mas típica es la Futaba. Pero esta elección tiene una pega y es que los servomotores no aportan un movimiento de giro completo, si no que su eje gira 180º . Esto se soluciona mediante una pequeña modificación.

Así que usaremos **2 servos Futaba 3003** como potencia motriz para nuestro proyecto.

Solo nos queda elegir que sensores usaremos en nuestro robot, para que pueda relacionarse con el entorno.

Al igual que en las otras opciones nos decantamos por los consejos de la página http://www.iearobotics.com . Así que como buen seguidor de linea montaremos dos sensores CN70 para que podamos seguir la linea. Además como quiero que podamos dejarle señales en el entorno, añado otros dos sensores más . Esto hace un total de **4 sensores CN70** para leer señales del suelo. Además añadiré **2 sensores de contacto,** por si quiero añadirle algún método de seguridad y **un sensor LDR** para poder comprobar las funciones del conversor Analógico/Digital-Digital/Analógico.

Con esto tenemos definida la estructura hardware, en el siguiente capítulo veremos la

selección del sofware con el que trabajaremos.

3.1.2 Selección del software

Cuando optamos por un lenguaje de programación debemos valorar todas las implicaciones que conlleva. Estas pueden ser desde, si somos nuevos al desarrollar en ese lenguaje, nos aporta mayor facilidad al desarrollar la aplicación o nos da opciones que en otros lenguajes nos costaría mucho más conseguir.

En este proyecto además nos encontramos con otro problema más. Esto se debe a que tengo que realizar dos aplicaciones que aunque en su naturaleza son diferentes tendrán que interactuar entre si.

Primero será en pensar en el programa controlador. Respecto a este tenemos muy pocas opciones realmente, ya que no estamos trabajando con un procesador al uso. Sino que trabajaremos con un PIC . Sopesando las opciones me decanto por usar el **lenguaje C** .
Lo primero es que no me resulta un lenguaje desconocido, y por tanto la curva de aprendizaje será menos pronunciada . Lo segundo es que programar tareas simples en este lenguaje resulta muy sencillo y debido a que el pic no soporta estructuras de programacion mas complicadas (los objetos), se nos antoja este lenguaje como todo un acierto. Por último en la página de http://www.iearobotics.com encontramos ejemplos sencillos, para poder dar nuestro primeros pasos en la realización de las funciones básicas.

Solo nos queda seleccionar en que lenguaje programaremos nuestra aplicación monitor. Tenemos multiples lenguajes posibles. Queremos poder hacer una ventana donde se muestren los estados de los sensores. Además se deberá incluir unos botones para las ordenes. Crear una ventana con estas características es sencillo. Anteriormente en la carrera ya hice programas en Java con estas características, además, encontré documentación y una librería que facilita enormemente el uso del puerto serie. Por lo que seleccionamos **Java** como nuestro lenguaje para nuestro programa monitor.

3.1.3 Tabla Resumen de las arquitecturás elegidas

Tabla Soluciones Hardware

Parte Hardware	Solución
Placa controladora	Skypic
Placa etapa potencia	Sky293
Pic	16F876A
Chasis	Piezas Lego
Motores	Futaba 3003
Sensores	CN70
Sensores	Sensor de contacto
Sensores	LDR

Tabla Soluciones Software

Aplicación	Lenguaje de programación
Programa Controlador	C
Programa Monitor	Java

3.1.4 Otras arquitecturas

Arquitecturas hardware descartadas:

-**Montar un PIC en una placa de pruebas** : Aunque esta opción es mas barata que la elegida el esfuerzo que supondría el montar el cableado en una placa de pruebas no compensaría el esfuerzo que tendríamos que realizar para obtener el mismo resultado que trabajar directamente con las placas Skypic.

-**Usar el PIC 16F84**: La cantidad de información que hay de este PIC y su contrastado uso sobre otros modelos lo hacen una buena opción. Sin embargo el elegir el modelo 16F876A se debe a que es un modelo superior al anteriormente citado, añadiendo ademas la posibilidad de usar dispositivos analógicos gracias a su conversor.

-**Chasis fabricado a medida**: Suele ser muy común el uso de chasis fabricados a medida para las placas que se vayan a usar. Esto hace que obviamente las piezas se adapten muy bien a la estructura . Pero para fabricar las piezas del chasis se necesitan herramientas de bricolaje, lo que aumentaría considerablemente el precio final del proyecto. Además los materiales pueden llegar a romperse, lo que significaría rehacer con piezas nuevas o parches de baja calidad para subsanar dichas averías.

-**Motores eléctricos** : Aunque existen motores eléctricos, conseguir un motor un poco potente exige un desenvolso extra. Por lo que la solución aportada es mucho mas asequible y viable.

-**Sensores de ultrasonidos**: Los sensores de ultrasonidos son un elemento muy interesante para un microbot de nuestras características. Desgraciadamente los más asequibles no tienen una gran precisión por lo que se vuelven una opción no muy recomendable.

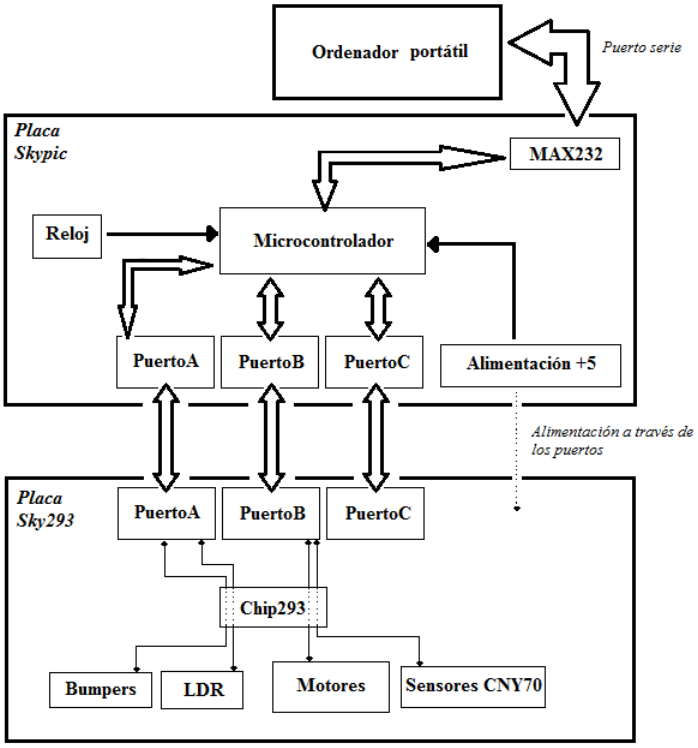
-**Arquitecturas Software descartadas**:

-**Programa controlador en Ensamblador**: Muchos programas para PICs los encuentras en este lenguaje. Aunque tengo nociones para desarrollar el programa en lenguaje ensamblador, no le veo ninguna ventaja. Es más cualquier función requiere de mucho mas esfuerzo que el programar en un lenguaje de mayor nivel como es C.

-**Programa Monitor en Visual C++**: En un principio estuve probando el realizar el programa monitor en VC++, ya que al igual que Java me ofrecía herramientas para desarrollar una aplicación simple como es la de una ventana y unos labels. Realicé bastantes pruebas e incluso hice un primer prototipo para hacer funcionar el robot de modo remoto en este lenguaje. Desgraciadamente me quedé atascado intentando realizar la recepción de datos con el puerto serie. Como también estaba probando la solución Java y encontré una librería de facil uso descarté seguir desarrollando esta aplicación en lenguaje VC++.

4. Esquema de la Arquitectura Hardware

Para poder comprender mejor el funcionamiento del robot mostramos a continuación el esquema de la arquitectura:



Esquema Hardware del funcionamiento del microbot

En el esquema podemos diferenciar los 3 diferentes niveles del hardware que componen nuestro microbot.

-**Placa Skypic**: El cerebro de nuestra máquina es la **placa Skypic**, que contiene el microcontrolador y que se encarga de procesar la información que le llega desde la placa de apoyo (la sky293) mediante los diferentes puertos (A,B y C) .

También se encarga tanto de interpretar las órdenes que nos llegan desde el ordenador, como de enviar la información del estado al mismo. Todo esto lo hace mediante el chip de comunicación (**el MAX232**).

-**Placa Sky293**: La tarjeta con el chip 293 nos servira como una placa de apoyo, un simple interfaz hardware, para conectar los diferentes componentes que usará el microbot para desenvolverse por el entorno. Estos componentes son desde los motores, hasta cada uno de los sensores que usemos.

Toda la información de estos dispositivos es recogida y enviada mediante los puertos de la tarjeta (A,B y C).

-**Ordenador Portatil**: Es una capa secundaria. Su única labor es la de monitorizar los datos que envíe el microbot. También tiene la posibilidad de mandar instrucciones al microbot (selección de caminos y en modo manual su control).

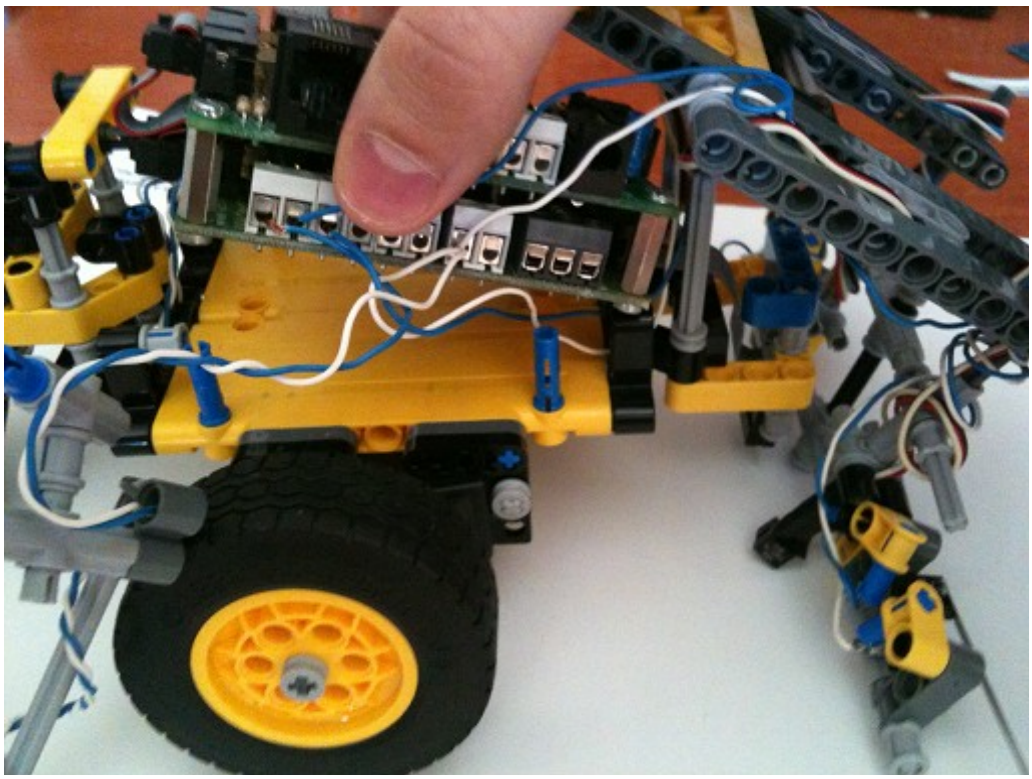
5. Montaje del Hardware

5.1. Montaje del chasis

Para el montaje del chasis, al tratarse de piezas Lego no hay ninguna reseña esencial en el montaje. El diseño por el que opté simplemente se basa en realizar una estructura que sea fácilmente ampliable y que se pueda desmontar sin demasiada dificultad.

En la estructura crearemos varios espacios esenciales para los componentes esenciales:

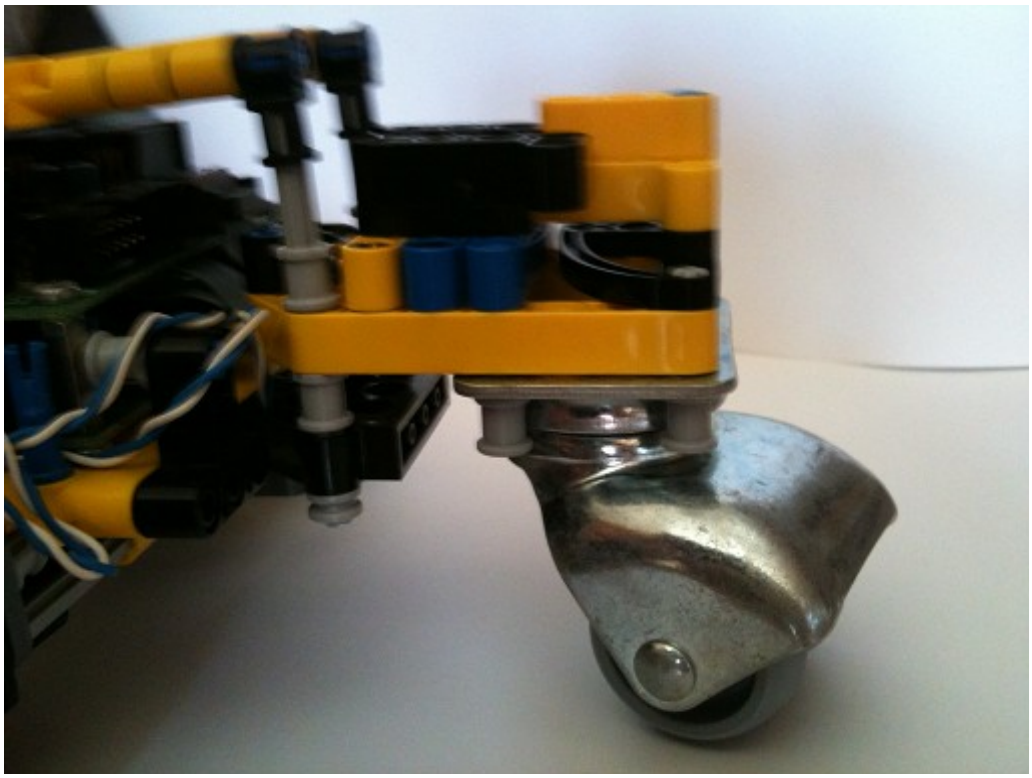
-Un lugar donde alojar la placa:



-Un lugar donde alojar la batería:

-Una forma que nos permita sostener los motores:

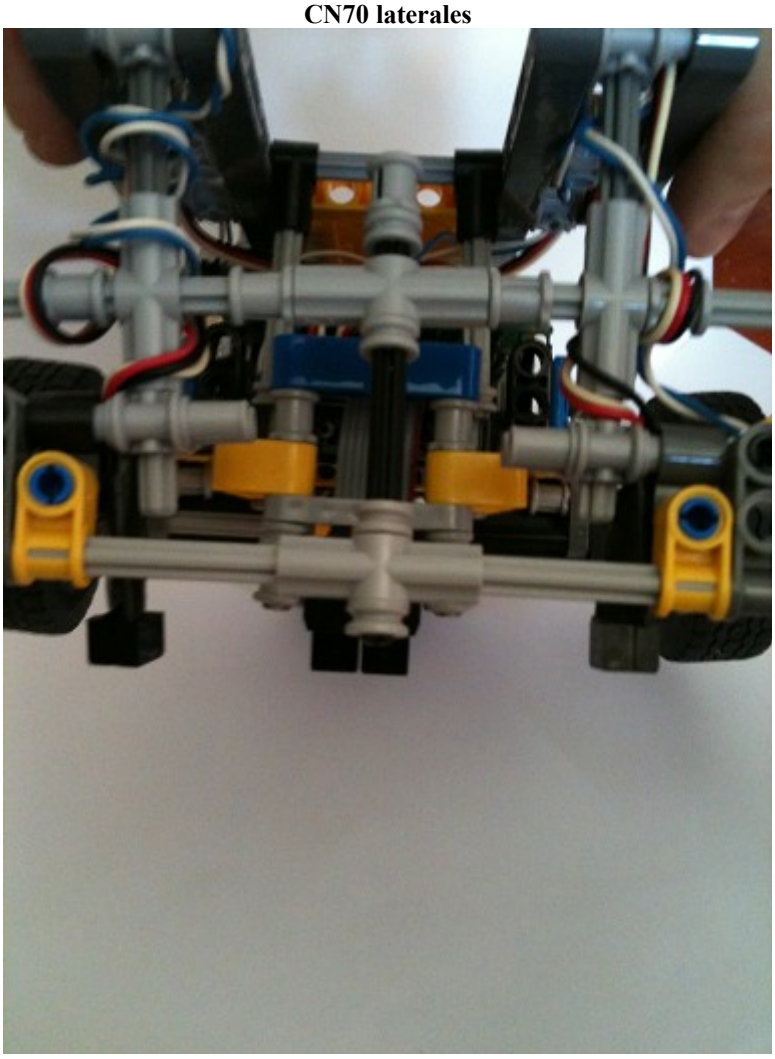
-Colocar la rueda loca:



-y distribuir los sensores:

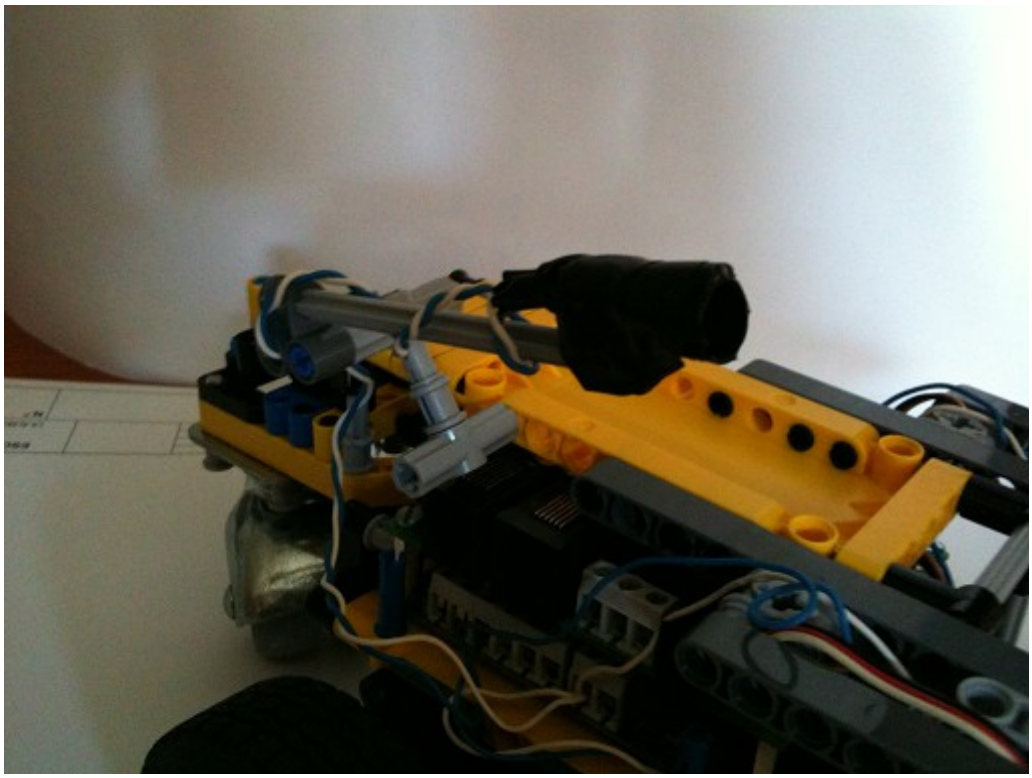
CN70 centrales

CN70 laterales



Bumpers

LDR



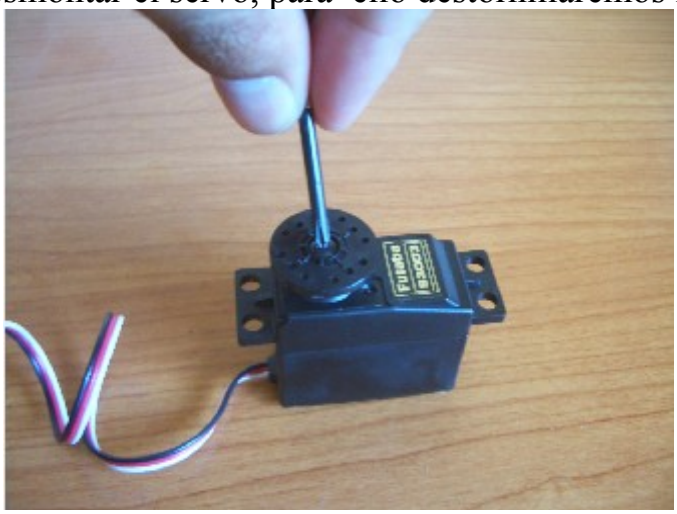
5.2. Modificaciónes

5.2.1. Modificar los servos

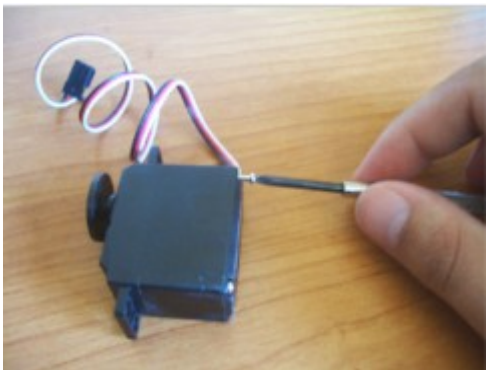
Para la modificación del servo necesitarmos las siguientes herramientas y materiales:

-VTD
-Destornilladores pequeños (tanto de estrella como planos)
-Soldador
-lima
-pinzas electrónica
-tenacillas
-estaño
-cable trenzado (azul y blanco)

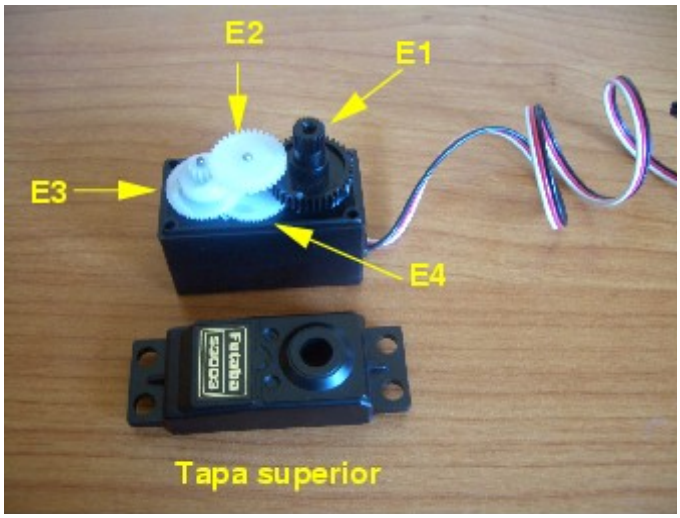
Primero debemos desmontar el servo, para ello destornillaremos la rueda del eje .



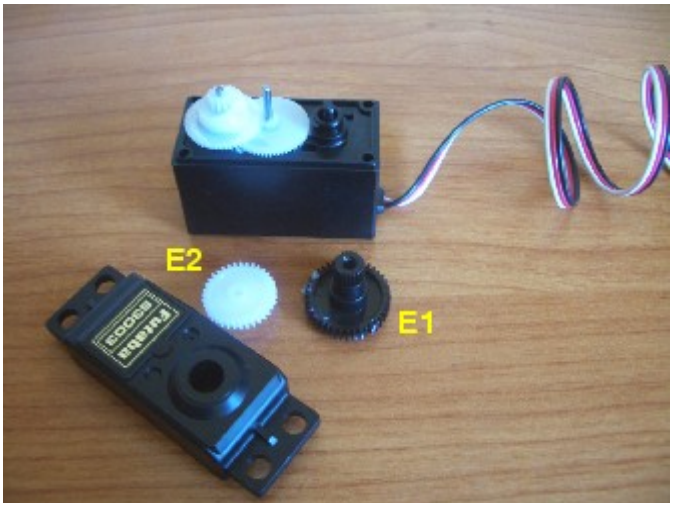
Acto seguido desmontaremos la tapa trasera del servo .



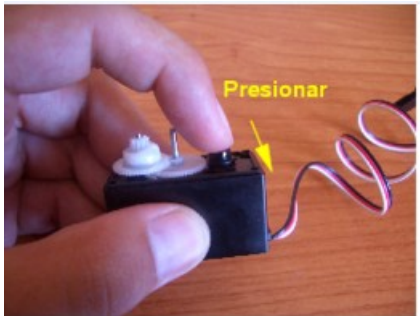
Una vez con el circuito al aire podemos sacar con cuidado todo el mecanismo.
Nos quedará el servo como podemos ver en la imagen.

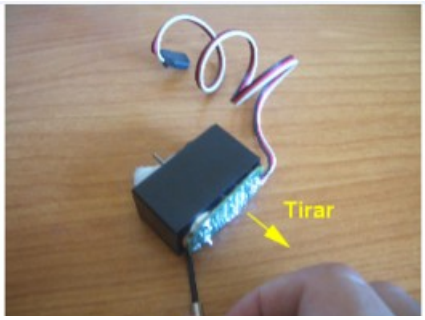


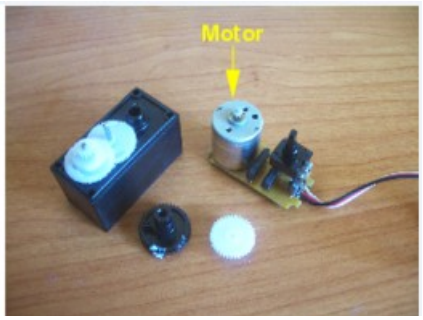
Si nos fijamos en los engranajes blancos(E1,E2,E3 y E4), veremos que estan engrasados. Es importante que cuando los vayamos a manipular, nos aseguremos de que no sustraemos esta grasa ya que es la que facilita el movimiento del motor.
Lo que haremos acontinuación es retirar el engranaje superior (E2) y acto seguido el engranaje negro (E1) como indicamos en la siguiente imagen.



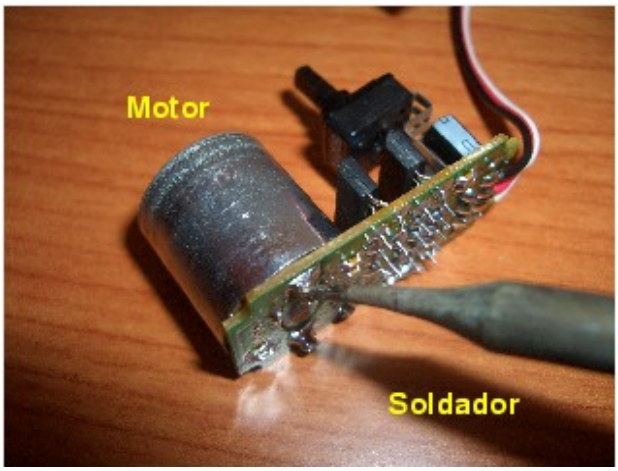
Tras hacer esto, el eje del motor eléctrico nos queda libre. Solo debemos presionar hacia abajo y el motor junto con la circuitería quedará desacoplado.





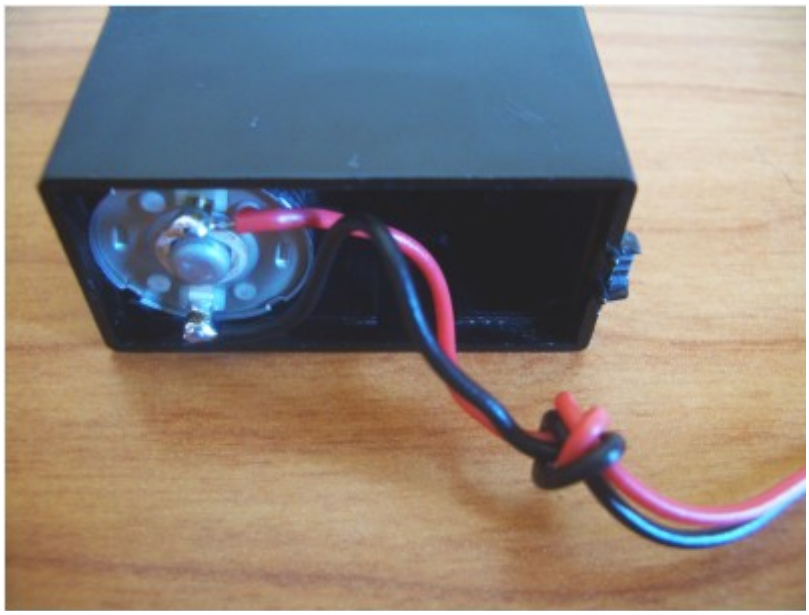


La parte siguiente es delicada, ya que tendremos que hacer uso del soldador.
Recordar que no debemos aplicar durante mucho tiempo el soldador en un circuito o componente electrónico ya que puede causar daños irreparables.
Procedemos a desoldar el motor mediante la bomba sucionadora (VTD) y el soldador. Seguramente nos haga falta repetir varias veces el proceso soldador/VTD antes de poder retirar el motor.

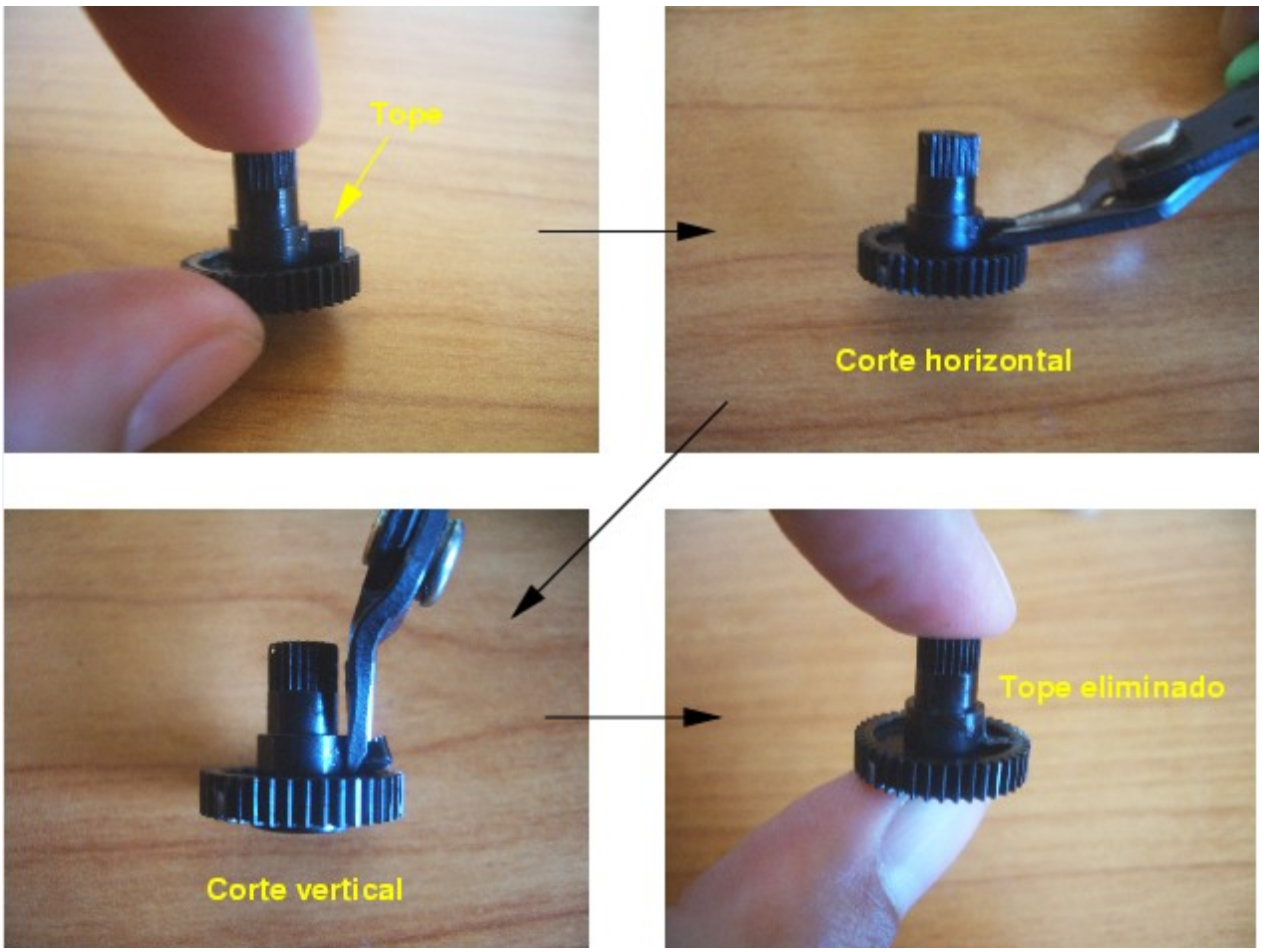


Con el motor libre del circuito tan solo debemos soldar el par de cables en el motor. El color/polaridad que usé en mi montaje fue azul (+) /blanco(-)). La polaridad del motor viene marcada en la parte inferior con una marca roja para la polaridad (+).

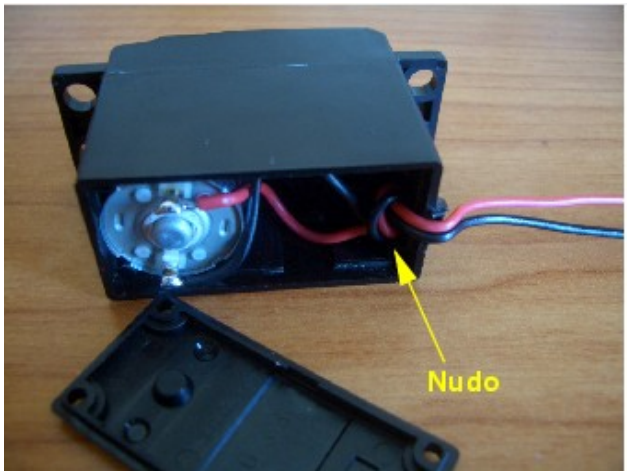
Con el motor cableado, solo nos queda introducirlo en la carcasa y atornillarlo (hacer los pasos inversos para montar los engranajes).Es interesante hacer un nudo que haga de tope para nuestros cables y así protegerlo de posibles tirones que puedan afectar a las soldaduras.

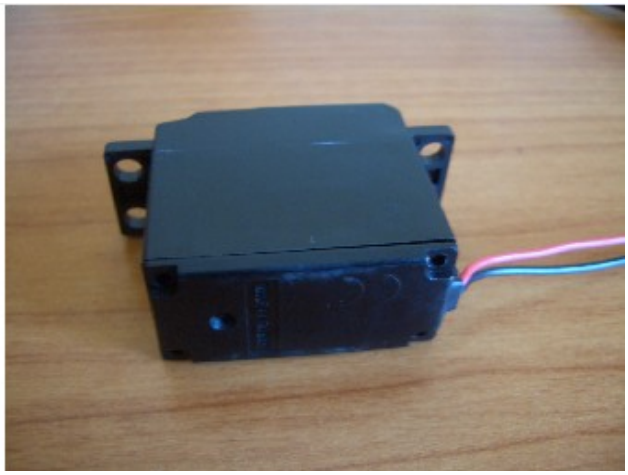


Antes de colocar la parte superior de la caja, se debe quitar el tope del eje negro. Para ello usaremos las tenacillas y luego pasaremos la lima para que no quede ninguna aspereza que pueda entorpecer el giro.



Ahora esta todo montado. Solo queda cerrar del todo el motor y ya tendremos listos los motores para su uso.



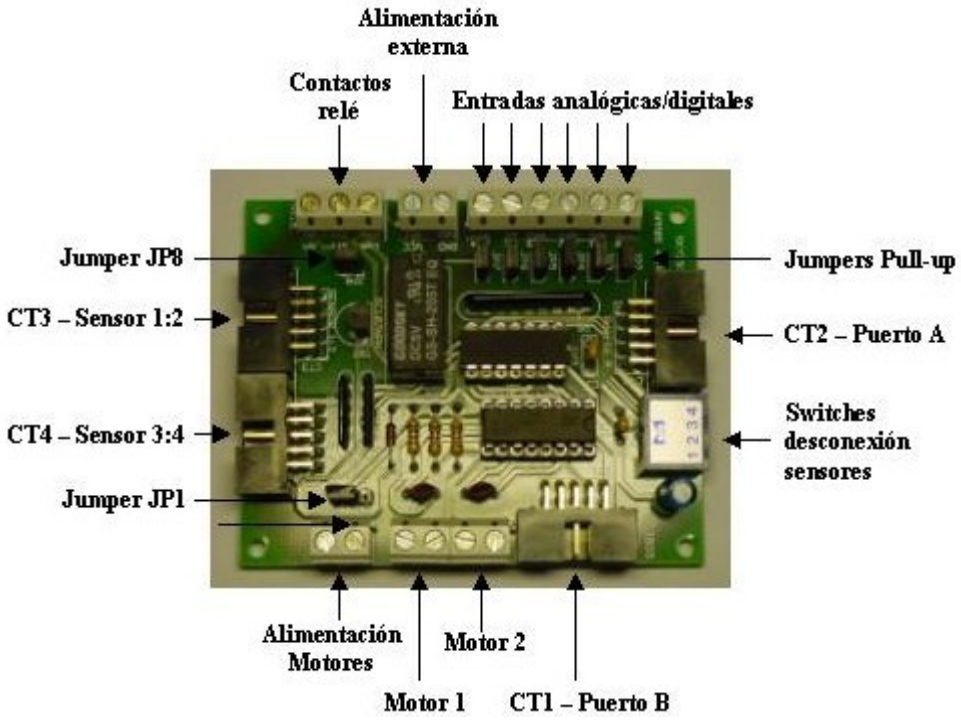


Con los servos trucados solo nos hace falta pegarle las ruedas de Lego. Cogeremos la pistola de silicona y pegaremos ambas ruedas a las plataformas circulares del eje del motor.

5.3. Montaje de los sensores en la placa Sky293

La placa Sky293 será la encargada de conectar todos los sensores y motores que queramos usar con nuestra placa controladora (la Skypic). Por lo que procederemos a montar todos los sensores y los servos que previamente hemos modificado.

La conexión de los sensores y servos irán distribuidos en los puertos como mostramos a continuación.

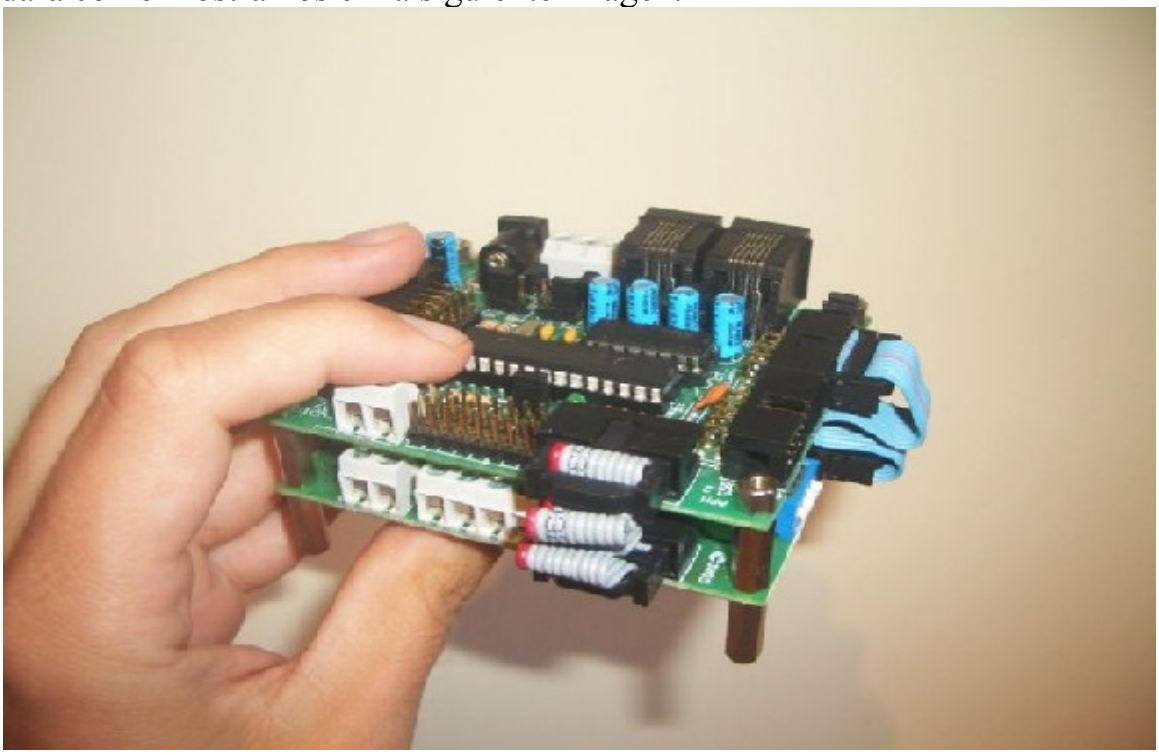


5.4. Ensamblaje de las dos placas

La placa Sky293 ya está conectada a los sensores y los servos. Pero nos falta el "cerebro" del microbot. Este es el PIC16F876A que viene montado en la placa Skypic.
Aunque para trabajar con el PIC, necesitemos quitarlo de la placa, podemos ir acoplando las dos placas.
Para realizar esto necesitaremos:

-Separadores
-Tornillos
-Destornillador plano
-Cables de bus plano

Este proceso es muy sencillo. Solo debemos atornillar los separadores para que quede por la parte superiror de la placa Sky293 y que así pueda sostener la placa Skybot. Dejamos esta última por encima, ya que si trabajamos quitando el PIC nos ahorramos el tener que desmontar las placas. Ademas como la Skypic lleva la comunicación (el puerto serie) es mejor cuanto más alto esté (entorpece menos el cable).
Quedará como mostramos en la siguiente imagen.



En la imagen anterior se puede ver también como acoplamos todos los buses (quedando comunicados los sensores y motores con la placa Skypic).

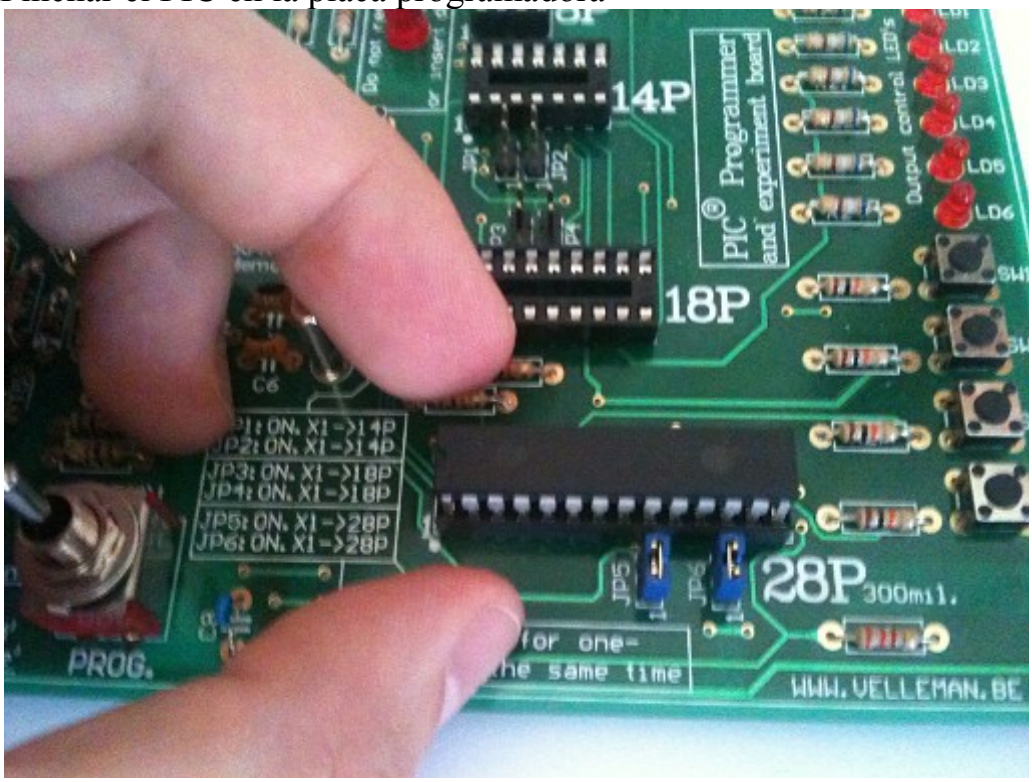
Solo nos queda inserta las placas en nuestro chasis en el siguiente habitáculo y habremos finalizado.

5.5. Preparación del PIC

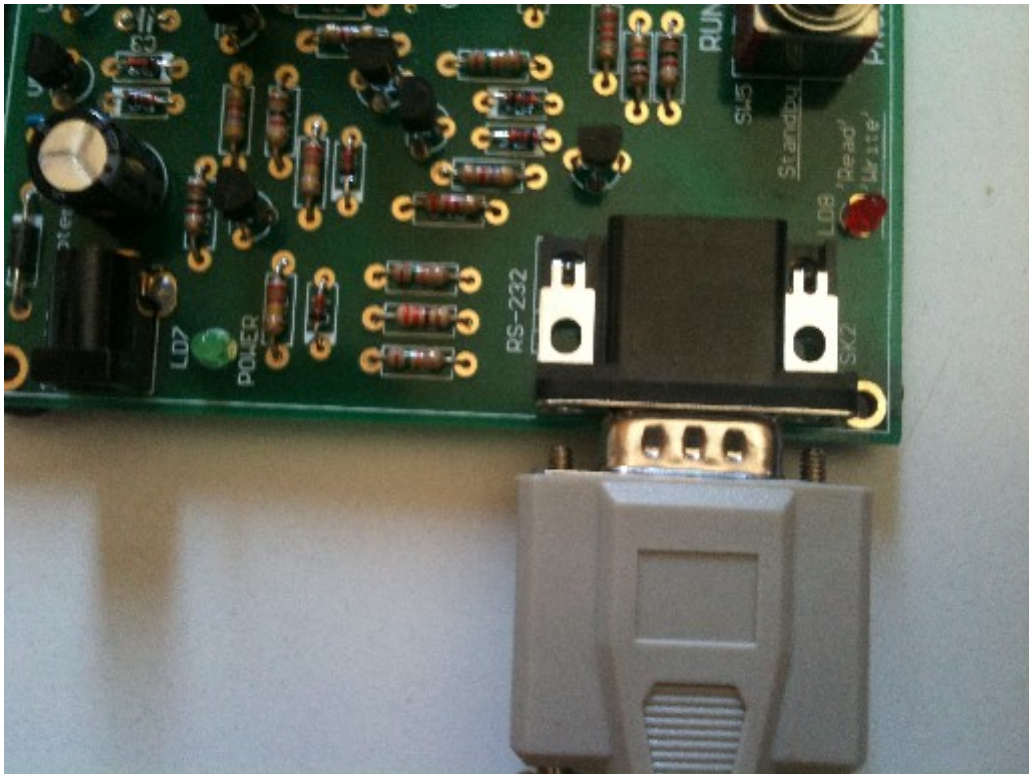
Para poder trabajar con el PIC sin necesidad de la placa programadora, necesitaremos primero introducirle el fichero Bootloader. Este fichero nos permite usar la propia Skypic como programadora , lo que nos ahorra el tener que estar sacando y pinchando el PIC cada vez que hagamos una modificación al programa controlador.

Para introducir el fichero Bootloader realizaremos los siguientes pasos :

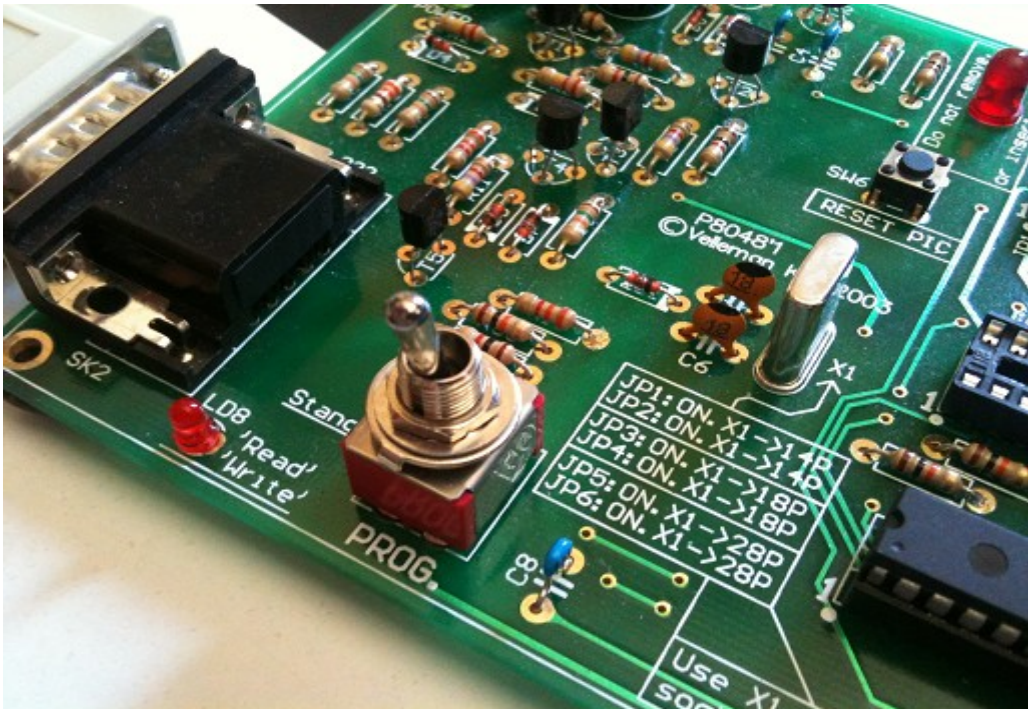
1-Pinchar el PIC en la placa programadora



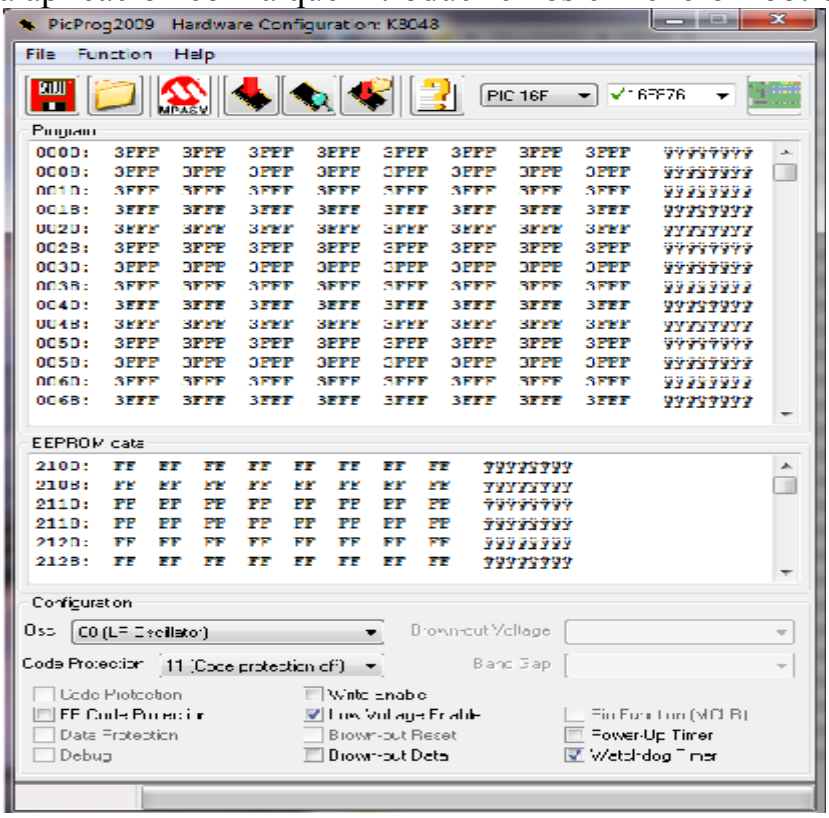
2-Conectar la placa programadora en con al puerto serie ***(altamente recomendable que sea un puerto serie real y no un conversor USB-SERIE)***.



3- Enchufar a la alimentación la placa y ponerla en modo de programación

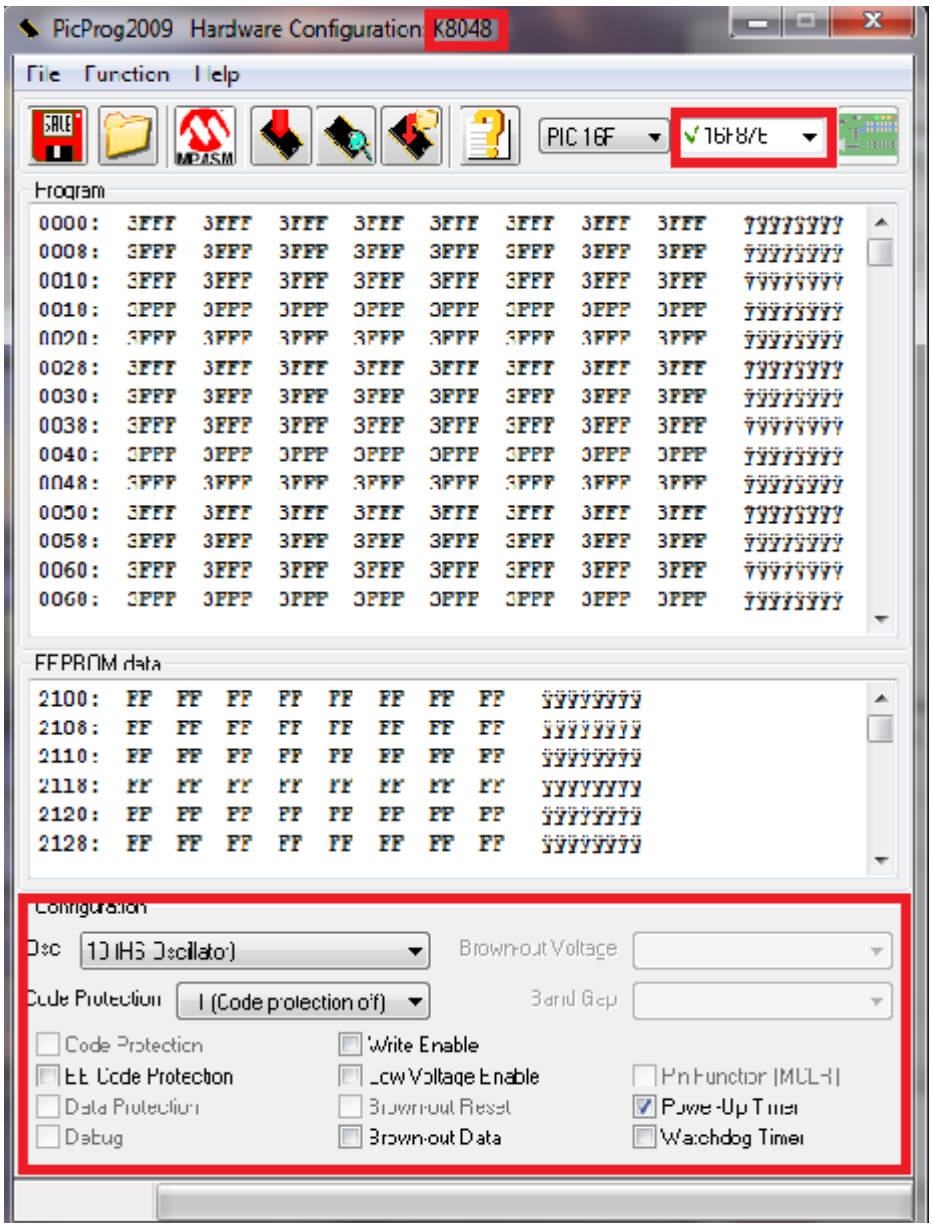


4-Abrir la aplicación con la que introduciremos el fichero Bootloader

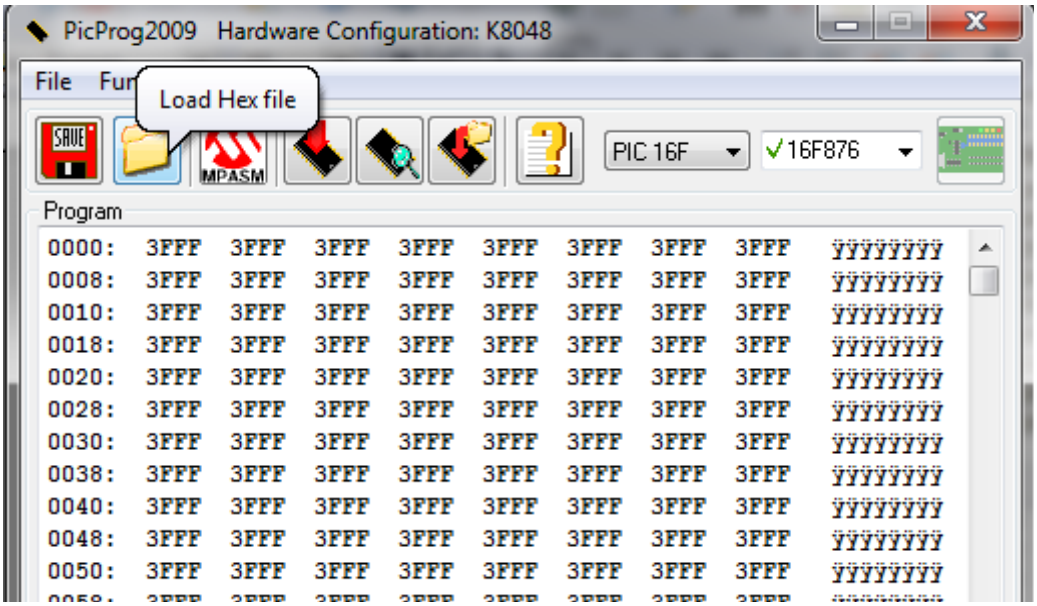


5-Configurar el programa para nuestra placa y nuestropic

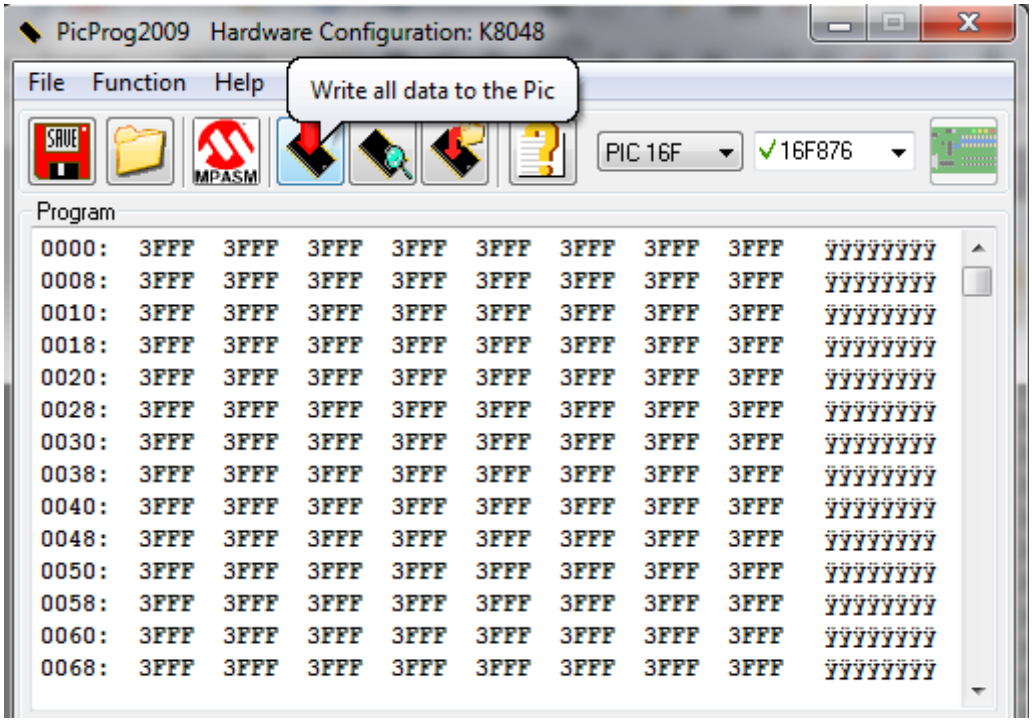
(MUY IMPORTANTE LA PALABRA DE CONFIGURACIÓN(3F32)!!)



6-Seleccionar el Bootloader como fichero a grabar.

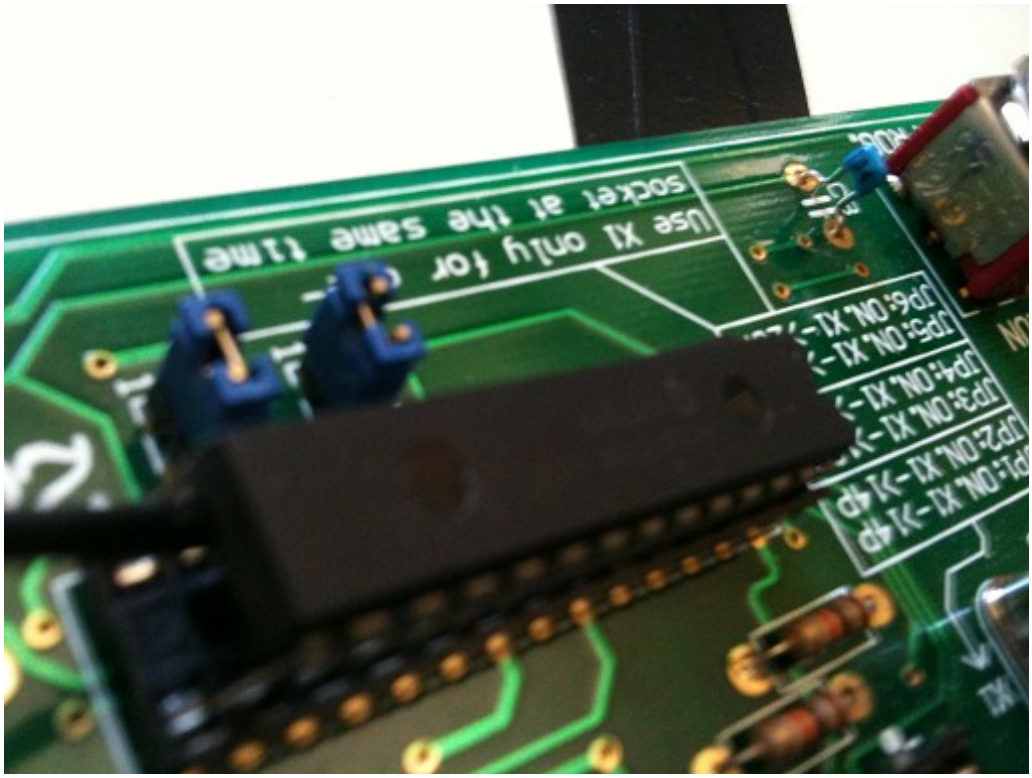


7-Realizar la grabación



8-Quitar el modo de trabajo de la placa y desenchufar la alimentación

9-Retirar el PIC de la placa entrenadora y colocarlo en la Skypic de nuevo



Ahora la Skypic ya esta lista para trabajar directamente con el ordenador.

6. Pruebas del hardware

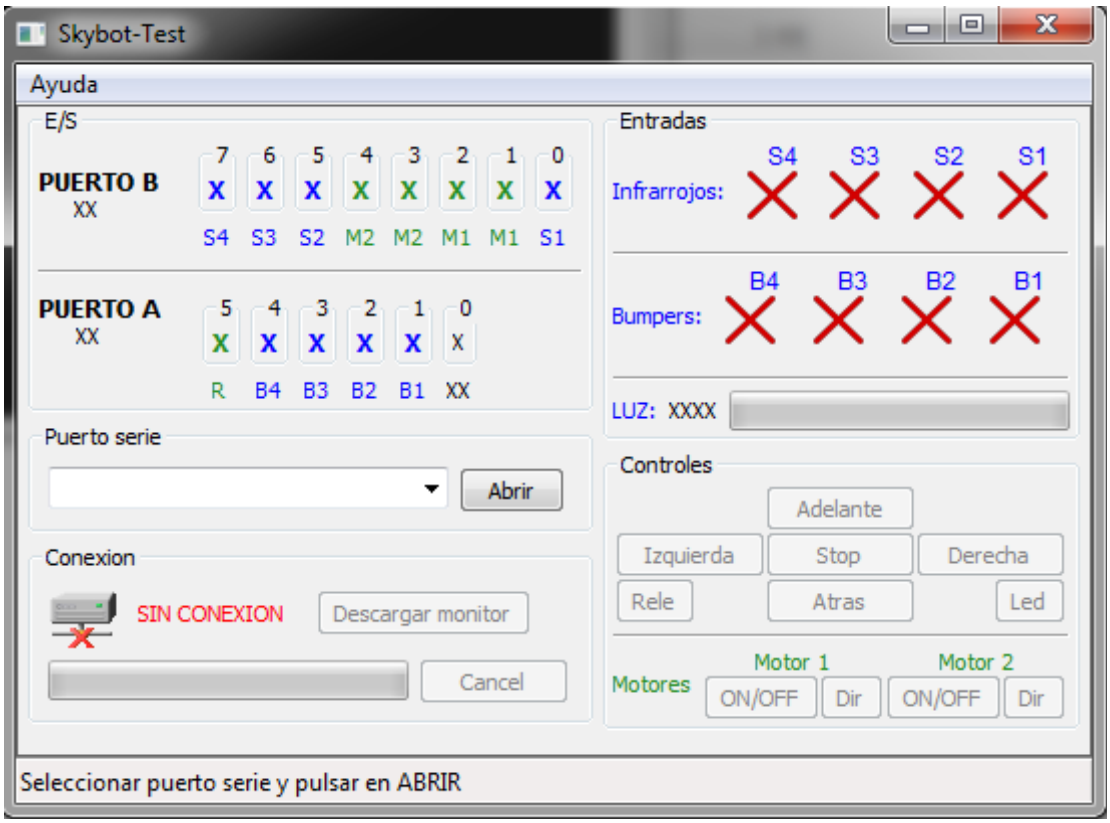
Ahora que tenemos el pic montado en la placa y todos los elementos conecados, debemos comprobar su buen funcionamiento. Para esto contaremos con la aplicación Skybot-test.

La aplicación Skybot-test esta desarrollada en lenguaje Python especificamente para la placa skybot.

Con esta aplicación no solo monitoriza los sensores conectados a la placa si no que también muestra el valor binario de los puertos A y B .

Nosotros usaremos la versión 1.0.1 para realizar las pruebas.

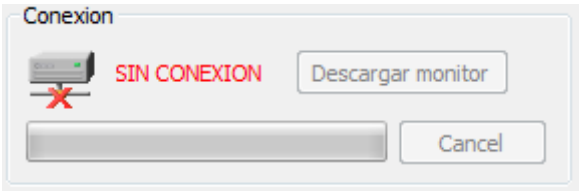
Cuando ejecutemos el programa veremos la siguiente ventana :



Como podemos comprobar en la imagen todo los valores estan marcados con una 'x'. Esto se debe a que no tenemos conexión con el dispositivo, tal como informa el apartado de conexión.

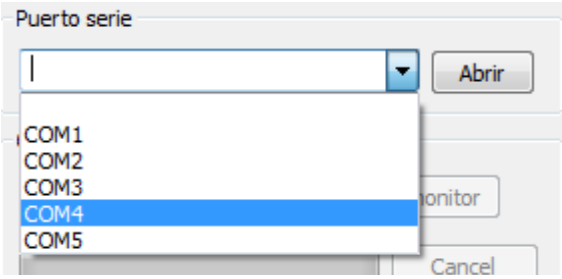
Detalle de los sensores en modo desconocido (por fallo de conexión)

Detalle del apartado de conexión



Para comenzar las pruebas con la placa montada conectaremos el cable serie a la placa y al ordenador.

Luego seleccionaremos el puerto serie configurado para el uso de la placa (en nuestro caso es el puerto COM 4)

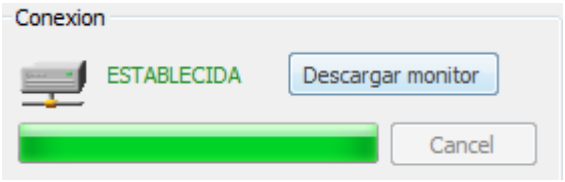


Detalle de la selección del puerto serie

Con el puerto serie seleccionado procedemos a descargar la aplicación monitor en el pic (**asegurarse de que la placa está alimentada**).

Ahora pulsaremos el botón abrir, que se encuentra situado al lado de la pestaña de selección de puerto. A continuación pulsamos el botón de "descarga monitor". Aparecerá una barra de progreso que nos indica la grabación del programa en el pic.

veremos como cambian el estado de conexión y los valores de los sensores se reflejan en la aplicación



Detalle de la conexión y la barra de progreso

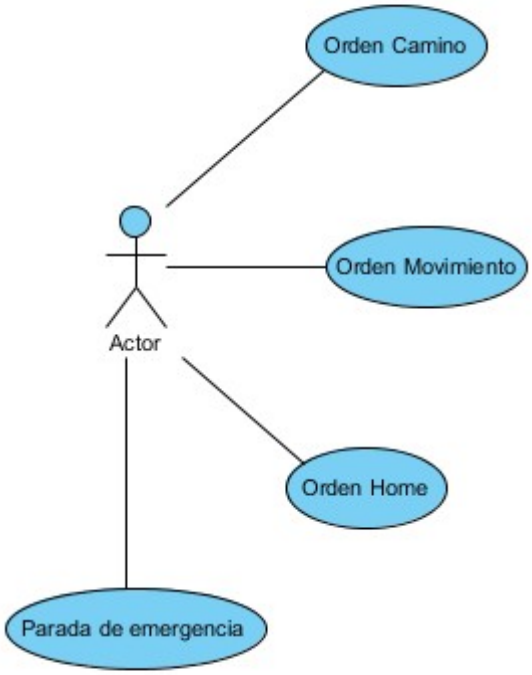
Detalle de los diferentes valores de los puertos y sensores

Con esto habremos comprobado el funcionamiento tanto de la conexión como de los diferentes componentes.

7. Casos de uso

Dada la naturaleza del proyecto, los casos de uso són muy limitados. Las acciones que puede realizar el operario se reducen a unas cuantas órdenes.

Diagrama de casos de uso para la aplicación monitor



Orden camino: Con estas órdenes mandamos al robot que ejecute un recorrido según la elección del operario.

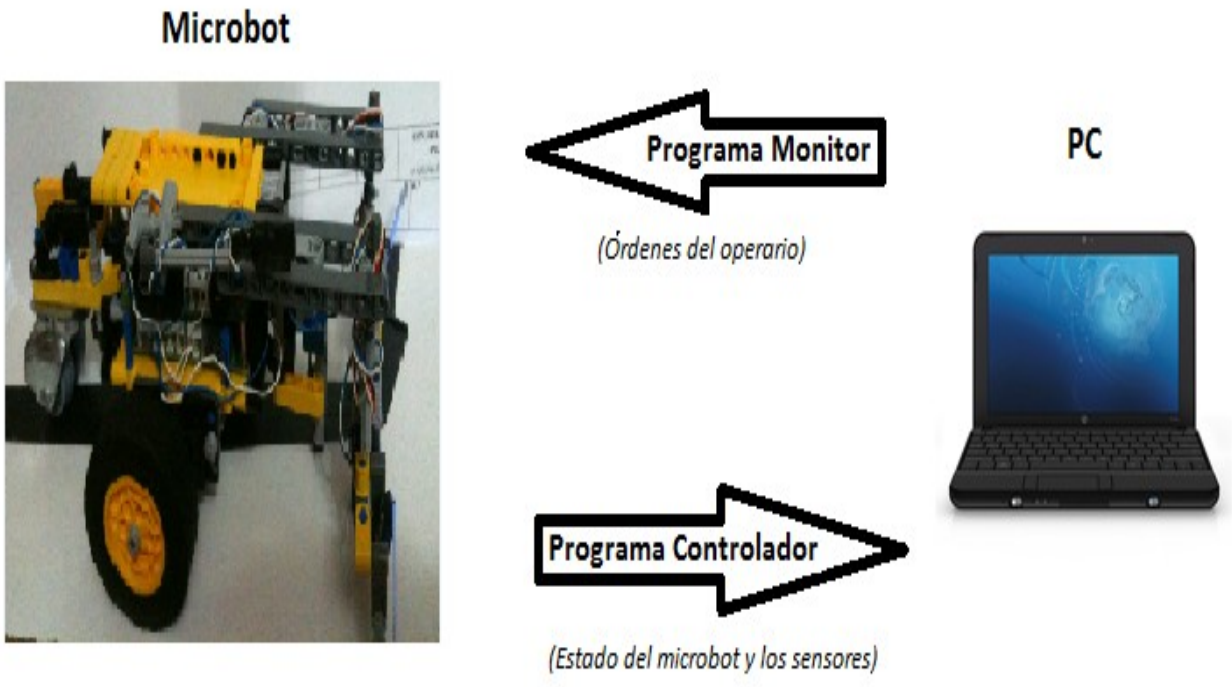
Orden Movimiento: Mediante la interfaz gráfica el usuario puede manejar el robot a su antojo.

Orden Home: Para que el microbot sea capaz de tener un punto de referencia tras moverse en modo manual el operario debe señalizarselo al robot.

Parada de emergencia: Al tratarse de una máquina que debe trabaja de forma autónoma es aconsejable tener una orden de parada de emergencia. Al entrar en modo parada de emergencia el microbot congelará sus acciones a la espera de que se le permita continuar.

8. Diagrama de despliegue

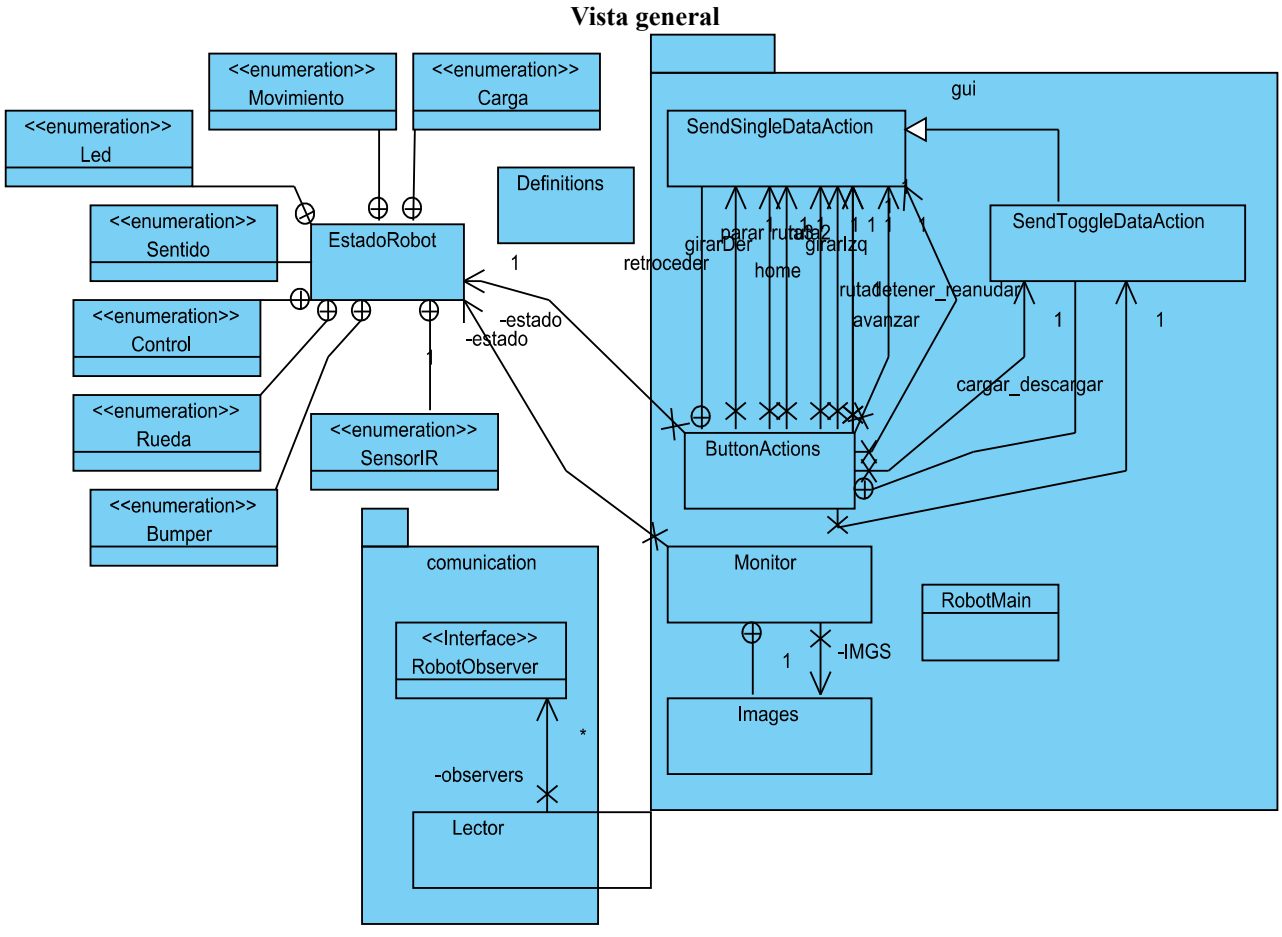
Diagrama de despliegue del Microbot y el pc



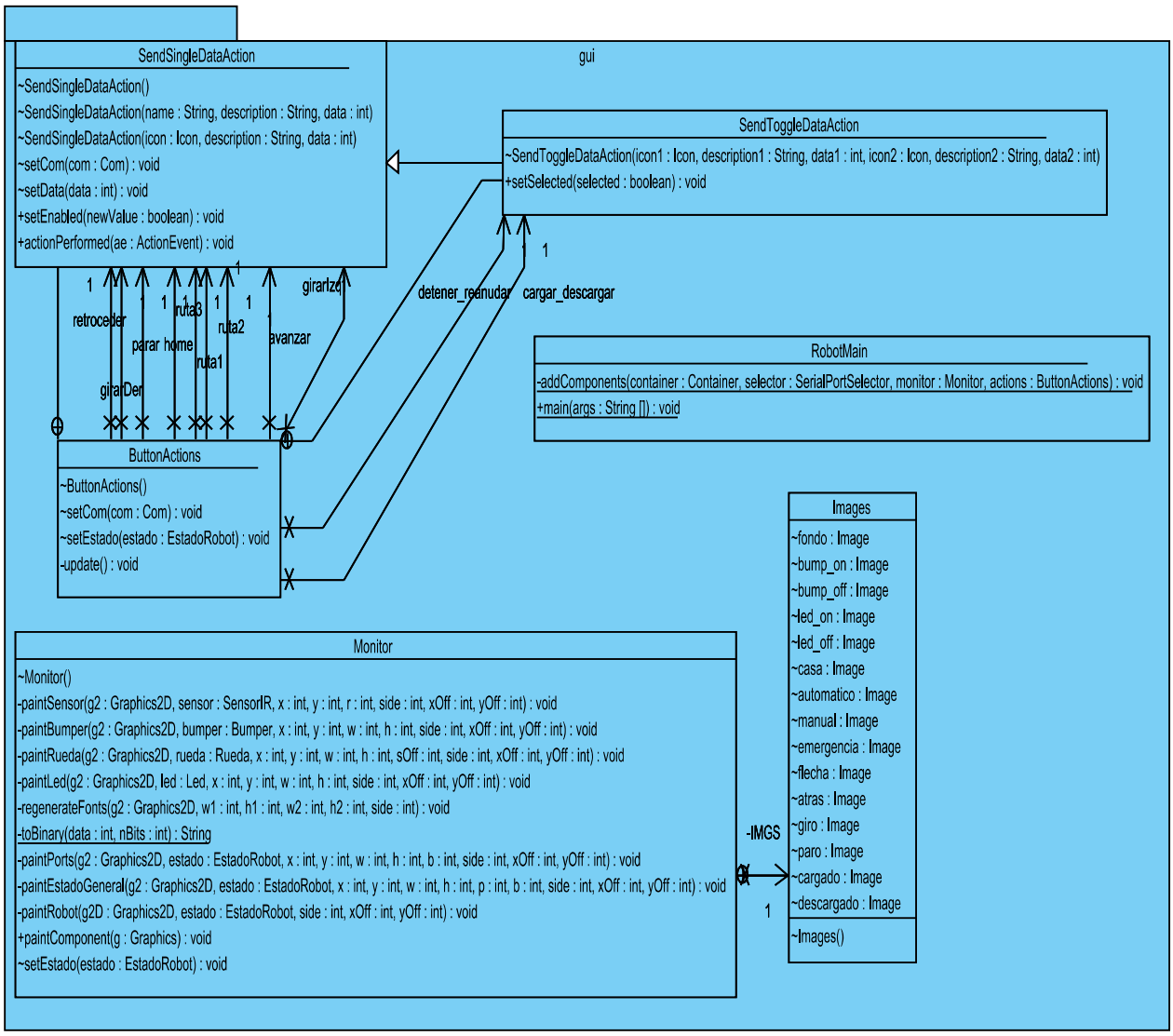
9. Diagrama de clase, descripción de métodos y funciones

9.1. Programa de monitorización

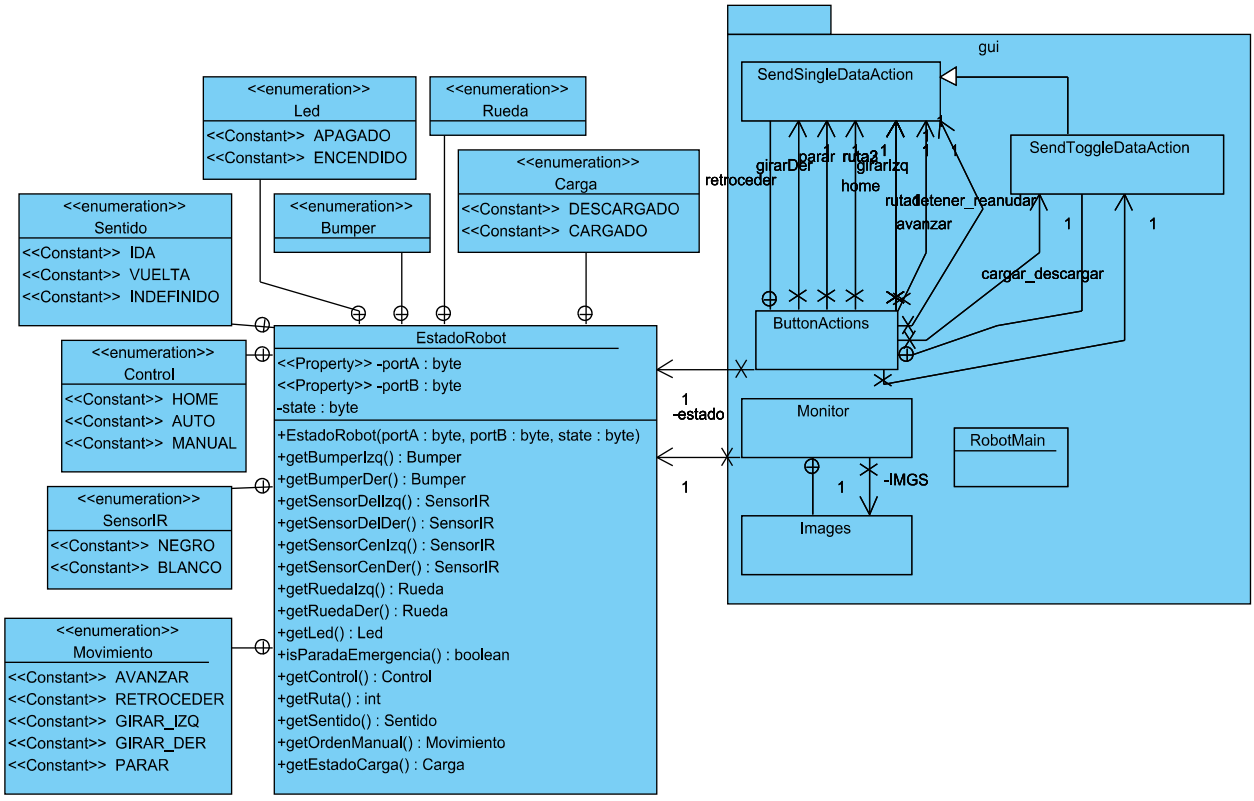
9.1.2 Diagrama de clases del programa monitor



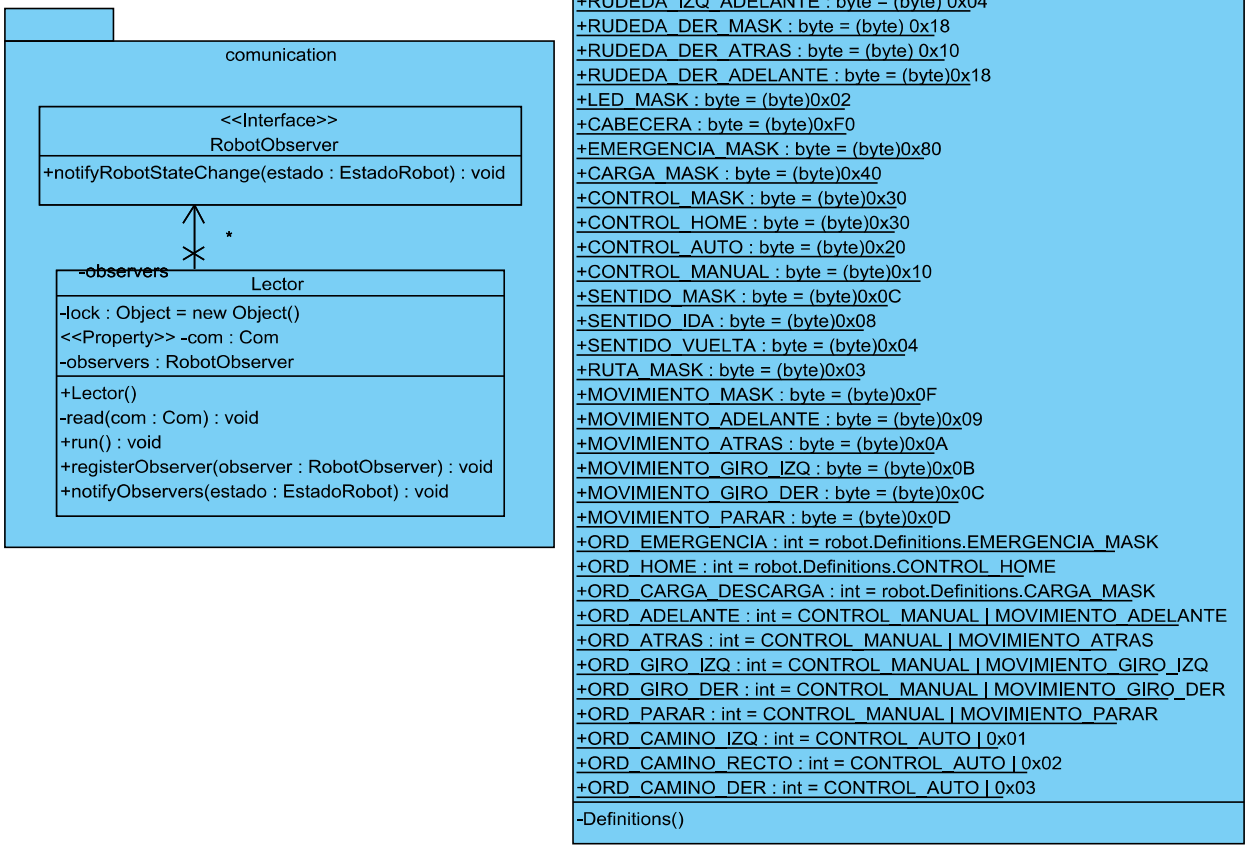
Ampliación diagrama de clases del programa monitor



Ampliación diagrama de clases del programa monitor

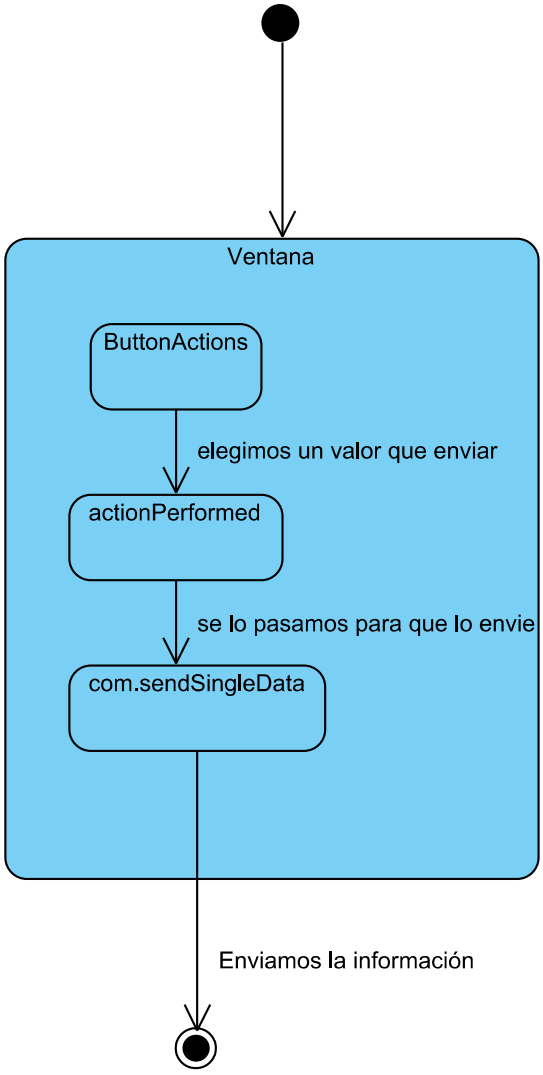


Ampliación diagrama de clases del programa monitor

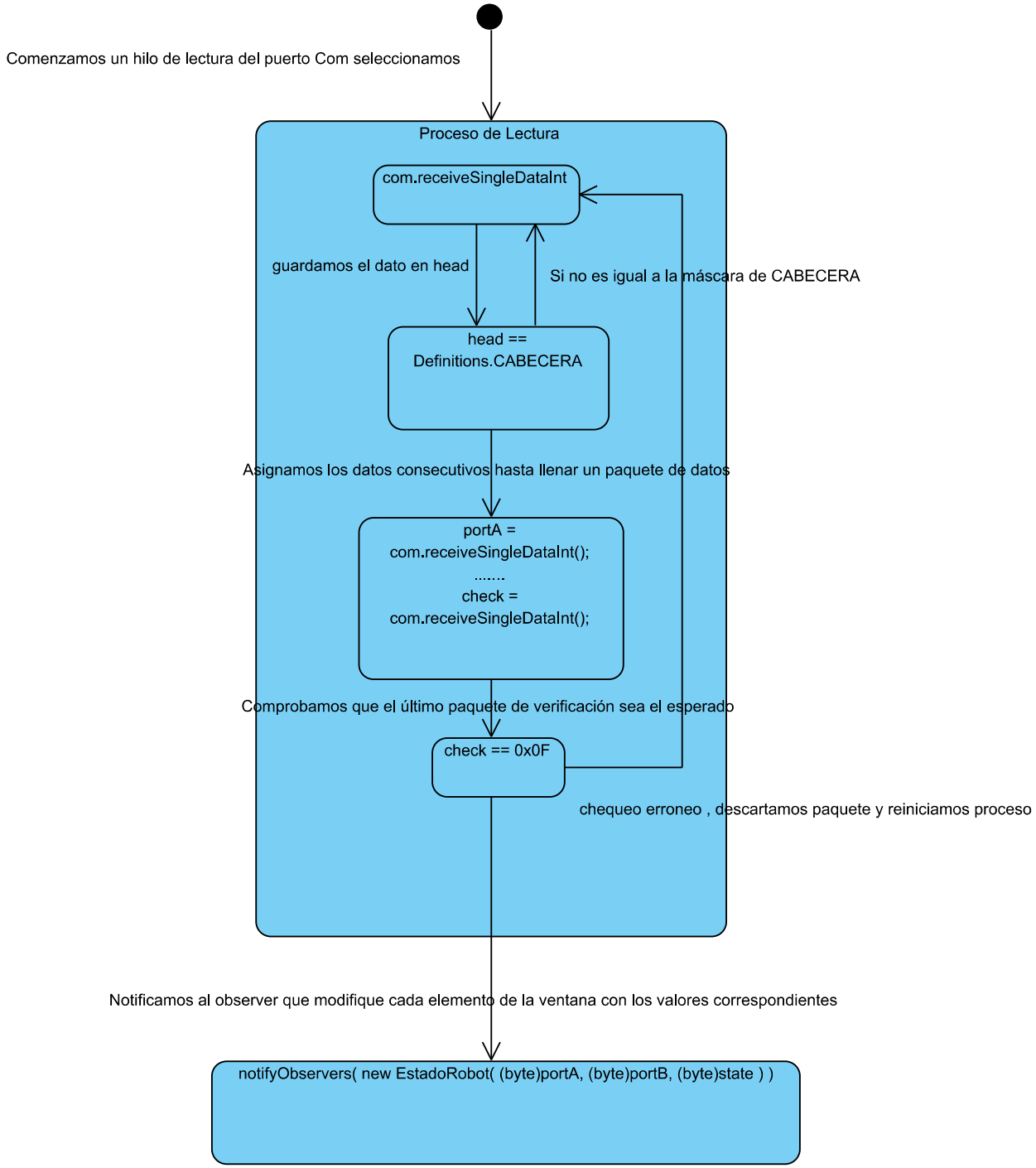


9.1.3 Diagramas de estados

Envío de datos



Lectura de datos



9.1.4 Descripción de las clases

class RobotMain

RobotMain
-addComponents(container : Container, selector : SerialPortSelector, monitor : Monitor, actions : ButtonActions) : void +main(args : String []) : void

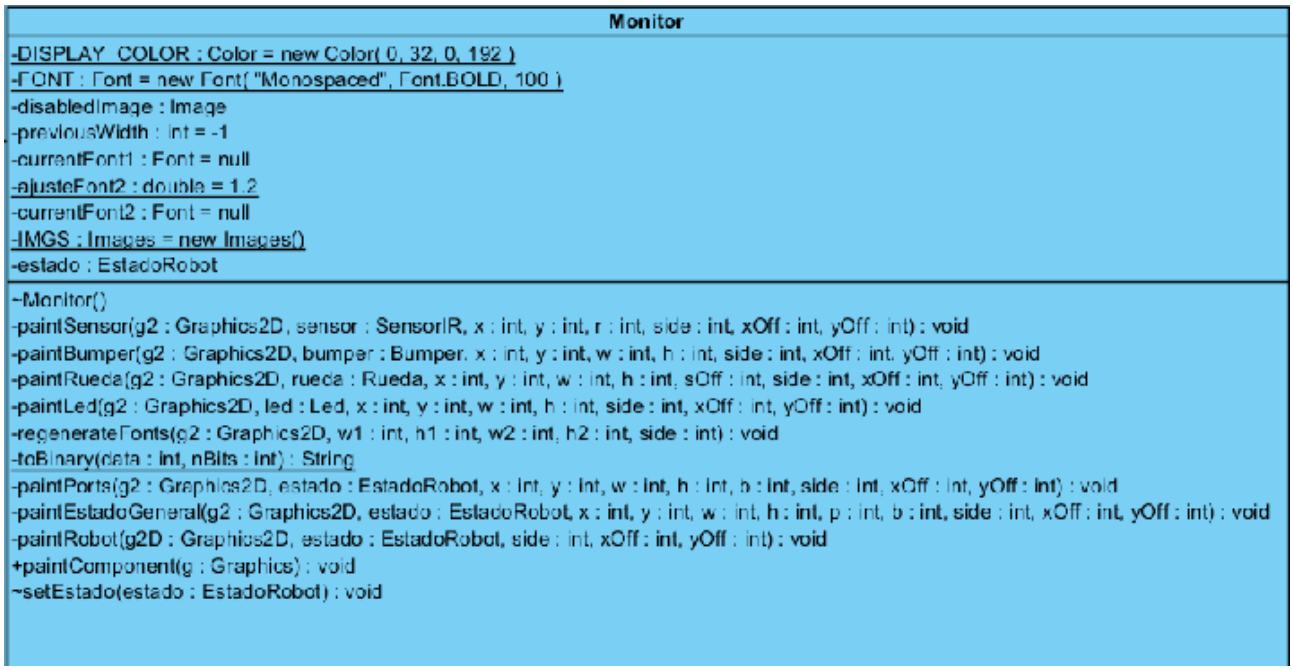
Esta es la clase principal del programa monitor.

class Images

Esta es la clase que contiene cada una de las labels que se utilizan en la aplicación para representar los datos.

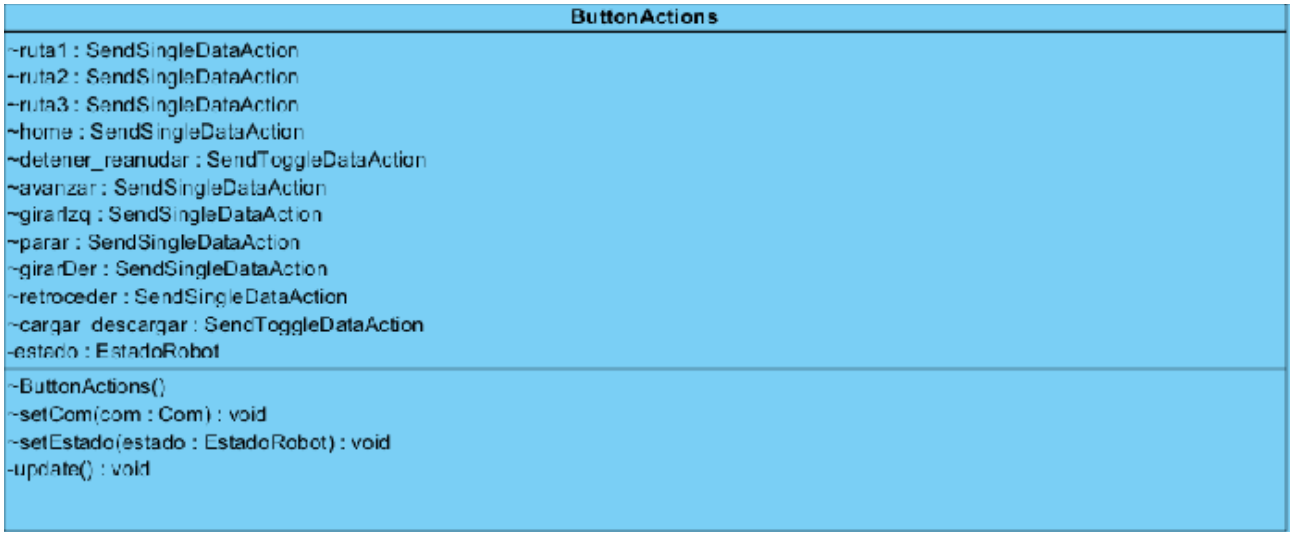
Images
~fondo : Image ~bump_on : Image ~bump_off : Image ~led_on : Image ~led_off : Image ~casa : Image ~automatico : Image ~manual : Image ~emergencia : Image ~flecha : Image ~atras : Image ~giro : Image ~paro : Image ~cargado : Image ~descargado : Image
~Images()

class Monitor



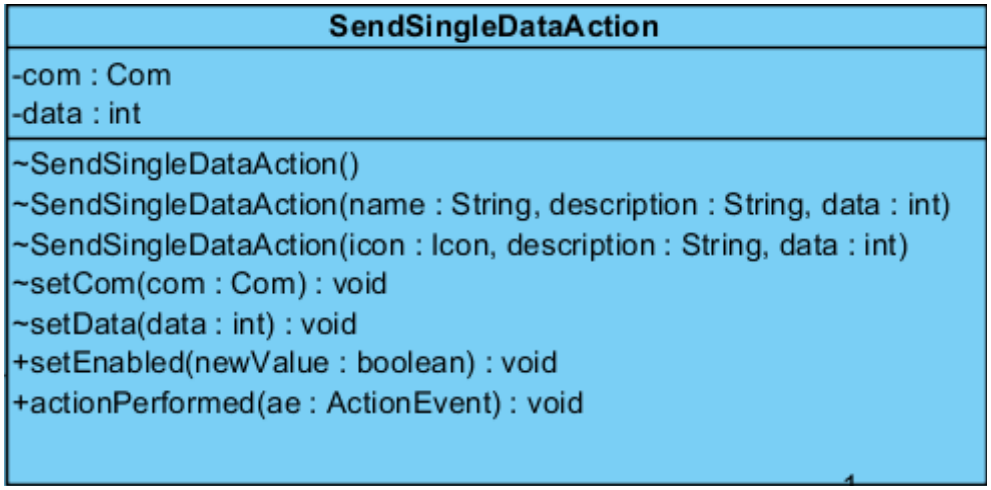
Clase que pinta cada uno de los elementos de la ventana de la aplicación .

class ButtonActions



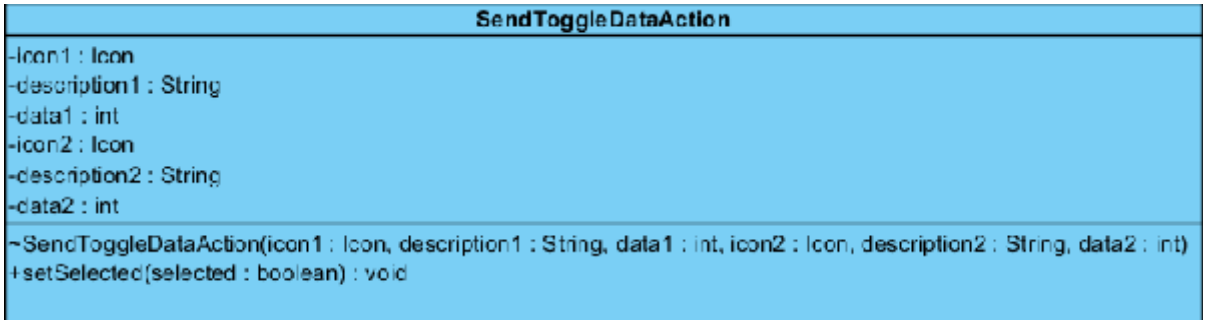
Clase que contiene cada una de las acciones de los botones de la aplicación.

class SendSingleDataAction



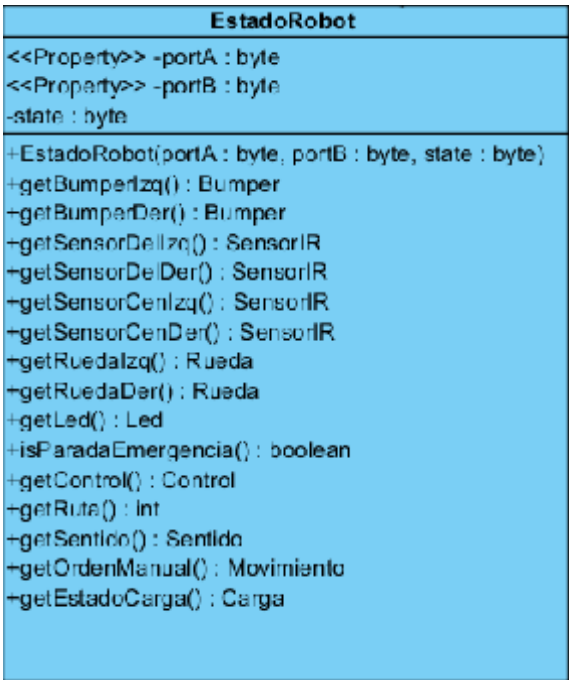
Clase que se encarga de enviar un caracter por el puerto serie.

class SendToggleAction



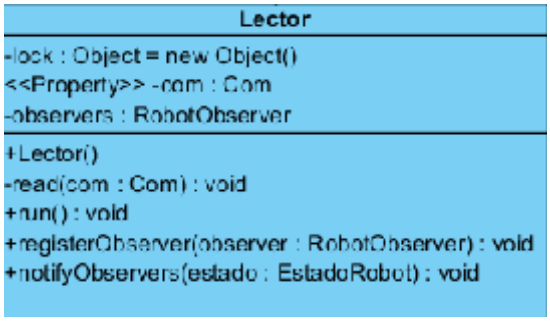
class EstadoRobot

La función de esta clase es contener cada uno de los valores que componen el estado del robot y los registros del puerto A y el puerto B.



Class Lector

La clase Lector es la encargada de leer el puerto serie .

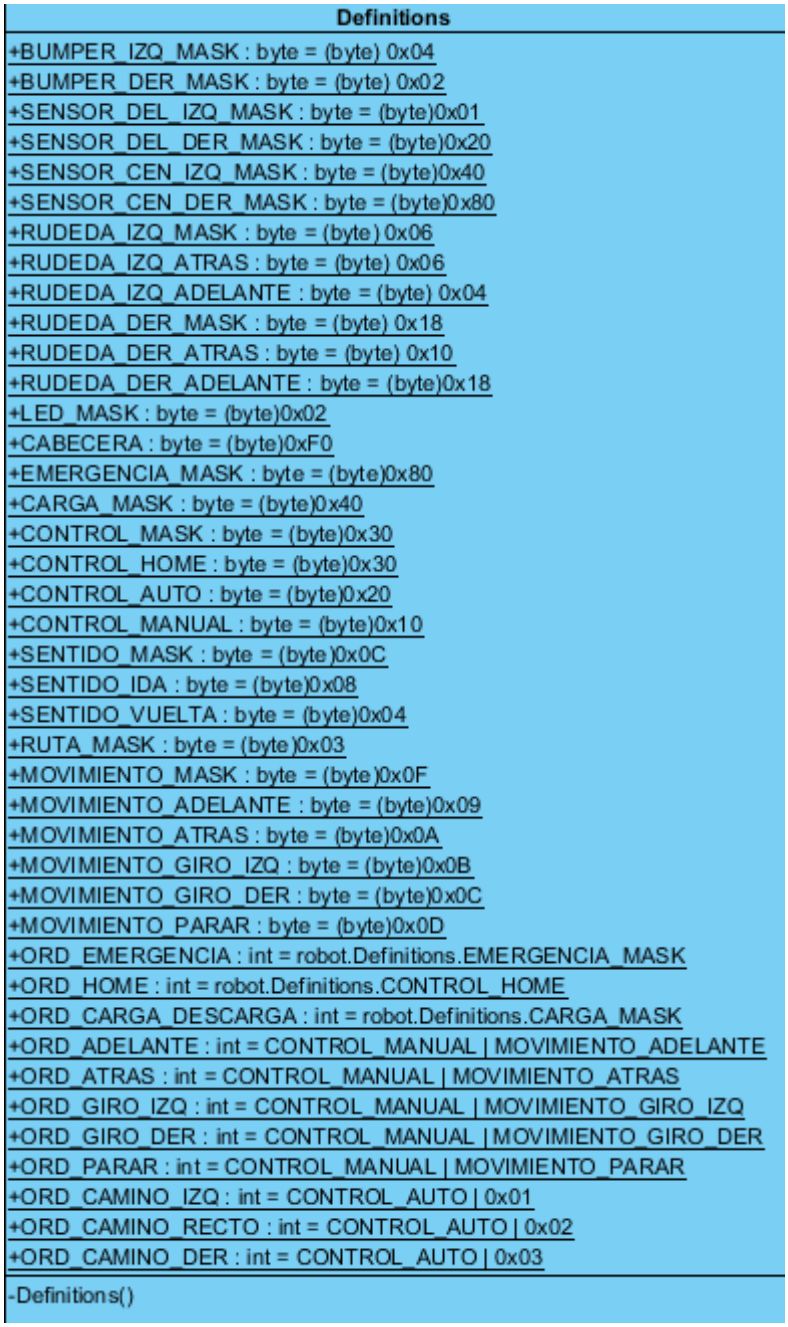


class RobotObserver

Esta clase se en carga de actualizar la ventana cuando recive los datos que le pasa la clase Lector.

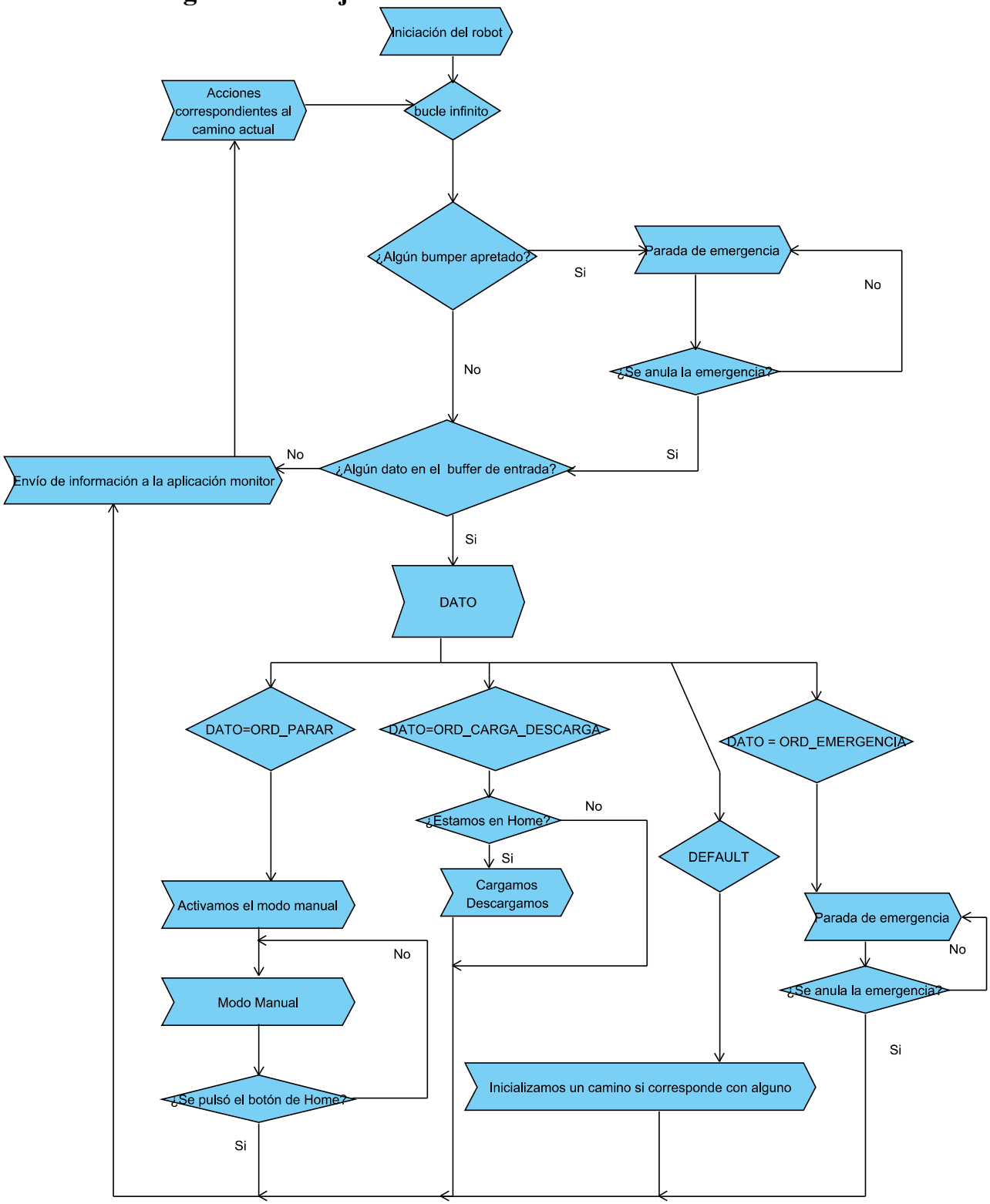
class Definitions

La clase Definitions contiene los valores de las máscaras que usamos para interpretar los datos que nos llegan desde el microbot. Por legibilidad los valores de las máscarás son los mismos que en el código C.



9.2. Programa controlador

9.2.1 Diagrama de flujo



9.2.2 Librerias

El programa en C se divide en las siguientes librerias:

-funciones_skybot.h: En esta libreria se incluyen las funciones de uso genérico.

***void Seguir_Linea(void)*:** El robot realiza un movimiento dependiendo de lo que capten los sensores 3 y 4:

Movimiento	**Led(3)**	**Led(4)**
Recto	*NEGRO*	*NEGRO*
Giro izquierdo	*NEGRO*	*BLANCO*
Giro Derecho	*BLANCO*	*NEGRO*
Parar	*BLANCO*	*BLANCO*

***void Cambiar_Orientación(unsigned char giro)*:** El robot realiza un giro de 180º tomando como eje central la linea negra donde esté situado. El sentido del giro se le pasa como variable.

u***nsigned char giro***= El sentido del giro. Si se pasa 'I' girará hacia la izquierda. Si se le pasa el valor 'D', girará a la derecha

unsigned char Estado_Sensores(void): Comprueba el valor de los sensores laterales 1 y 2. Dependiendo de la combinación del estado de los sensores devuleve un valor:

Led (1)	Led (2)	Valor devuelto
BLANCO	*BLANCO*	*0*
NEGRO	*BLANCO*	*1*
BLANCO	*NEGRO*	*2*
NEGRO	*NEGRO*	*3*

-comunicacion_skybot.h: En esta función vienen definidas las constantes para la configuración del puerto serie del robot, así como las funciones de envio y recepción de datos.

***void sci_conf(void)*:** Inicializa los registros del PIC para que se pueda usar el puerto serie.

***unsigned char sci_read(unsigned char* dato)*:** Comprueba si el buffer del puerto serie no esta vacio. Si no lo está guarda en la dirección del puntero el dato recivido. Luego si ha guardado algún dato devuelve un 1, en caso contrario devuelve un 0.

unsigned char* dato: Puntero que nos guarda los datos que envía el ordenador.

***void sci_write(unsigned char car)*:** Comprueba que el puerto serie le permita transmitir. En caso afirmativo manda al ordenador el caracter que le pasemos.

unsigned char car: variable donde se encuentra la variable que queremos enviar por el puerto serie .

-caminos.h: En esta libreria se definen cada una de las funciones para recorrer los caminos. También se incluye las funciones de inicialización de estado para cada camino.

void Inicializa(unsigned char camino, Estado* estado): Inicializa el registro de estado del robot según el camino seleccionado.

unsigned char camino: Variable donde se guarda la selección del camino elegido por el usuario.

Estado* estado: Puntero al registro del estado del robot para poder actualizar los valores del mismo.

void Camino_Izq(Estado* estado): Función que realiza los pasos para recorrer el camino izquierdo.

***Estado* estado*:** Puntero al registro del estado del robot. Necesario para ir actualizando al microbot sobre su situación en el camino.

void Camino_Rec(Estado* estado): Función que realiza los pasos para recorrer el camino central.

Estado* estado: Puntero al registro del estado del robot. Necesario para ir actualizando al microbot sobre su situación en el camino.

void Camino_Der(Estado* estado): Función que realiza los pasos para recorrer el camino derecho.

***Estado* estado*:** Puntero al registro del estado del robot. Necesario para ir actualizando al microbot sobre su situación en el camino.

-chequeo.h: En esta libreria vienen definidas las estructuras que usaremos para guardar el estado del robot. Además incluye las funciones que actualizan estos registros y la funcion que se encarga del envio de paquete a la aplicación monitor.

Los registros :

```
typedef struct
{
  unsigned char saltosIda;
  unsigned char saltosVuelta;
  unsigned char saltosRestantes;
  unsigned char realizandoSalto;


  unsigned char camino;
  unsigned char volviendo;
  unsigned char emergencia;
  unsigned char carga;
  unsigned char home;
  unsigned char automatico;
  unsigned char sentido;

  unsigned char orden_manual;

} Estado;
```

Registro donde se guarda el estado del robot, donde:

- ***unsigned char saltosIda:*** Número de saltos que se dan en el recorrido a la ida

-***unsigned char saltosVuelta:*** Número de saltos que se dan en el recorrido a la vuelta.

-***unsigned char saltosRestantes:*** Contador de saltos que quedan en el camino en curso.

-***unsigned char realizandoSalto:*** Marca que nos indica si estamos en medio

de una operación de salto.

-***unsigned char camino***:Variable donde guardamos el camino seleccionado.

-***unsigned char volviendo***: Marca que nos indica si estamos regresando de un camino.

-***unsigned char emergencia***: Marca que nos indica si estamos en una parada de emergencia.

-***unsigned char carga***: Marca que nos indica si hemos cargado el robot o si va vacio.

-***unsigned char home***: Marca que nos indica si nos encontramos en el punto de inicio.

-***unsigned char automatico***: Marca que nos indica si estamos en modo automático.

-***unsigned char sentido***: variable donde guardamos el sentido del giro.

-***unsigned char orden_manual***: variable donde guardamos la orden realizada manualmente.

```
typedef struct
{
  int bytes_enviados;
  unsigned char portA;
  unsigned char portB;
  unsigned char state;
} Paquete;
```

Registro donde guardamos la trama que vamos a enviar desde el microbot a la aplicación monitor , donde:

- ***int bytes_enviados*:** Variable donde realizamos la cuenta de los bytes enviados.

***-unsigned char portA*:** Variable donde guardamos los valores de los bits del puerto A.

***-unsigned char portB*:** Variable donde guardamos los valores de los bits del puerto B.

***-unsigned char state*:** Variable donde guardamos los valores de los bits del estado del robot.

unsigned char Chequea_Estado (Estado* estado): Función que aplica las mácaras del robot para traducir a un grupo de 8 bits (unsigned char) el valor del registro de estado del robot.

***Estado* estado*:**Puntero al registro del estado del robot. Los valores de este registro será los que traduzcamos a un grupo de 8 bits.

void enviar(Estado* estado, Paquete* paquete): Función que se encarga del envio en paquete de cada uno de los registros del microbot.

Estado* estado: Puntero al registro de estado para poder leer los datos de este registro.

Paquete* paquete: Puntero al registro donde empaquetamos la información que enviaremos a la aplicación monitor.

-definiciones.h: Aquí vienen definidas cada una de las máscaras que usaremos para interpretar los bits de los registros del robot.

9.2.3 Función principal

Funciones:

***void Parada_Emergencia(void)*:**Función que guarda el estado del robot y lo mantiene parado hasta que no recive la de nuevo la orden de parada de emergencia. En el reinicio se devuelve al robot el estado anterior a la parada.

***void Interpreta_Orden(unsigned char orden)*:**Función que realiza la selección del movimiento que se va a realizar.

unsigned char orden: Variable donde guardamos la orden que vamos a realizar.

void Control_manual(void)*:** Función que pone al robot en control manual. Para realizar las ordenes escucha el puerto serie y le pasa a la función ***Interpreta_Orden( ) las instrucciones. Cuando recive la instrucción Home el modo manual termina y el microbot pasa a modo escucha.

void Caminos(Estado* estado): Función que selecciona la función del camino seleccionado.

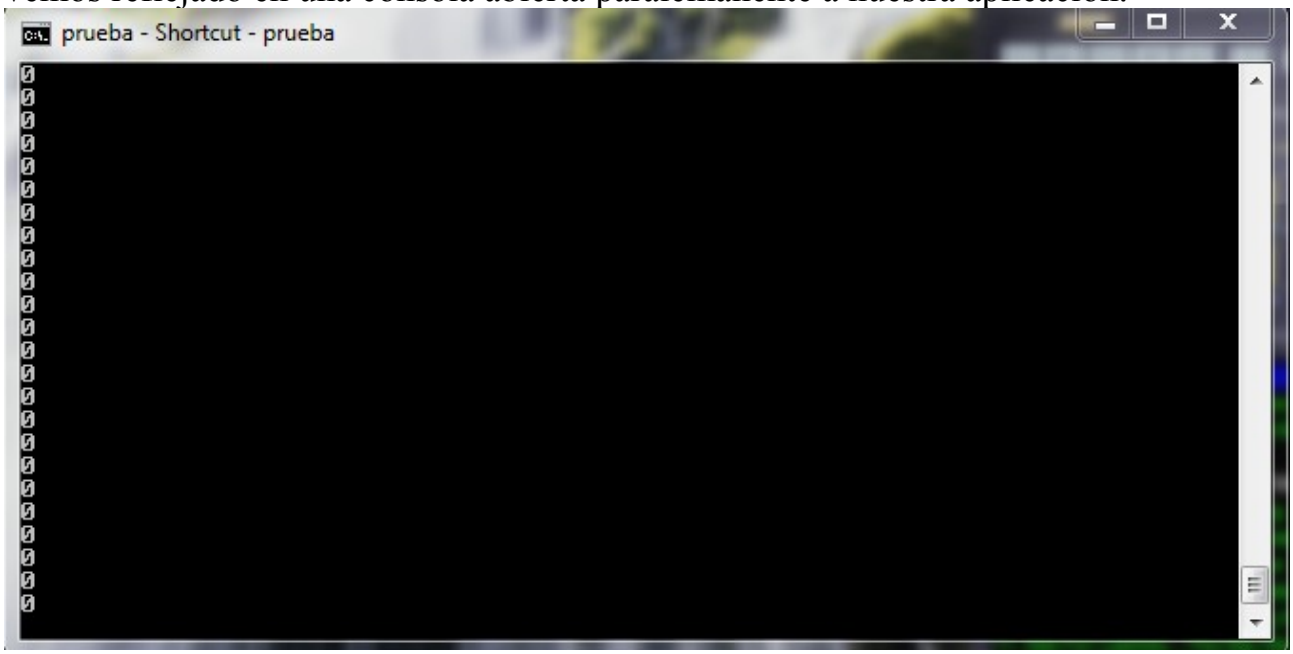
Estado* estado:puntero al registro del robot. Lo necesitamos para consultar el camino que tiene seleccionado.

Para el desarrollo de estas librerías usamos las siguientes librerias propietarias :

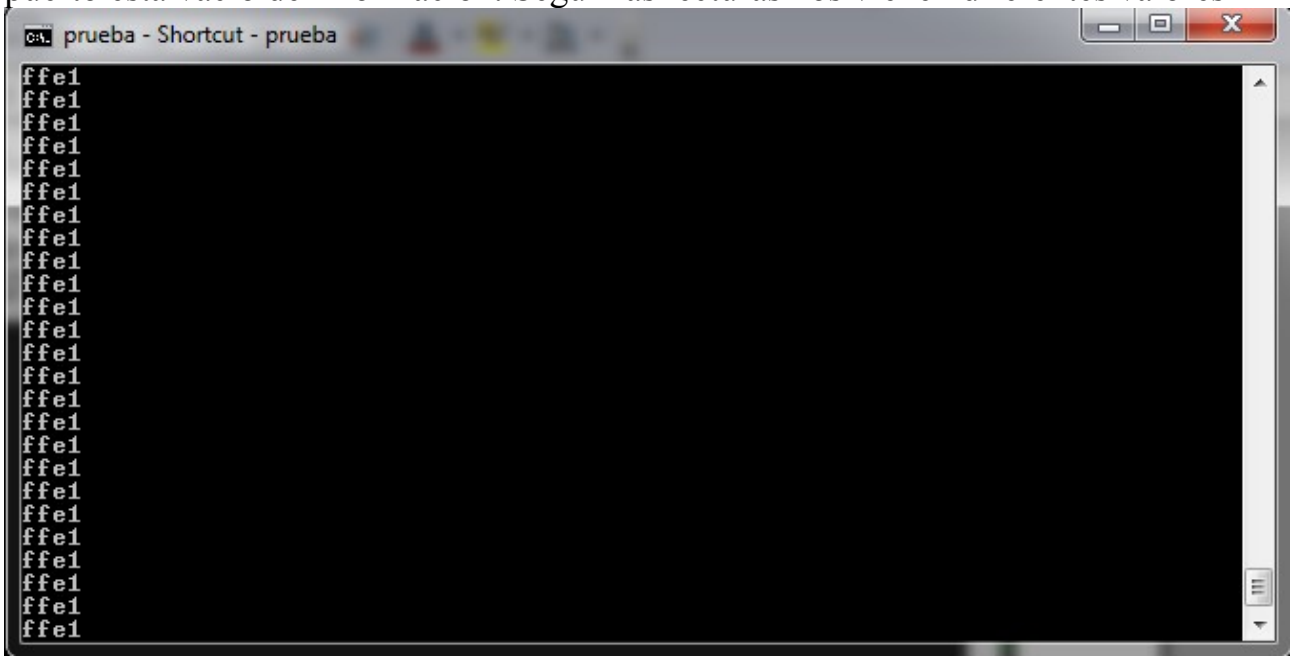
-libreria_skybot.h (desarrollada por Juan Gónzalez Gómez)
-delay0.h (desarrollada por Juan Gónzalez Gómez)
-p16f876a.h (desarrollada por Microchip Technology Inc.)

10.Pruebas del software

Para poder comprobar el correcto funcionamiento del programa monitor, cuando ejecutamos el programa toda información que llega al puerto seleccionado lo vemos reflejado en una consola abierta paralemanente a nuestra aplicación.



En la anterior imagen podemos ver como llegan 0 al puerto, esto quiere decir que el puerto esta vacio de información. Según las lecturas nos vienen diferentes valores



En la pantalla anterior vemos la consola reciviendo información desde el microbot. Si ocurriese cualquier fallo vendría reflejado en la consola.

11. Blibliografía y Webgrafía

Para el desarrollo del proyecto se consultaron los siguientes libros:

-**Microcontroladores PIC (Diseño práctico de aplicaciones 2º parte) PIC16F87X y PIC18FXXX**; *Autores: José María Angulo Usategui, Susana Romero Yesa e Ignacio Angulo Martínez.(publ Mc Graw Hill)*

Además se consultaron las siguientes webs:

- http://www.iearobotics.com

-https://sites.google.com/site/giovynetdesarrollos/manejo-de-puertos-seriales-rs-232-con-java--para-windows

12.Relación de documentos

RELACIÓN DE DOCUMENTOS

Documento	Nº de páginas
Pliego de condiciones	22
Planificación	45
Presupuesto	9
Memoria	66
Manual de usuario	18

13.Equipo de desarrollo

Iñaki Garaigorta Zarzuela