Competencia de Robótica
R2-D2 2014

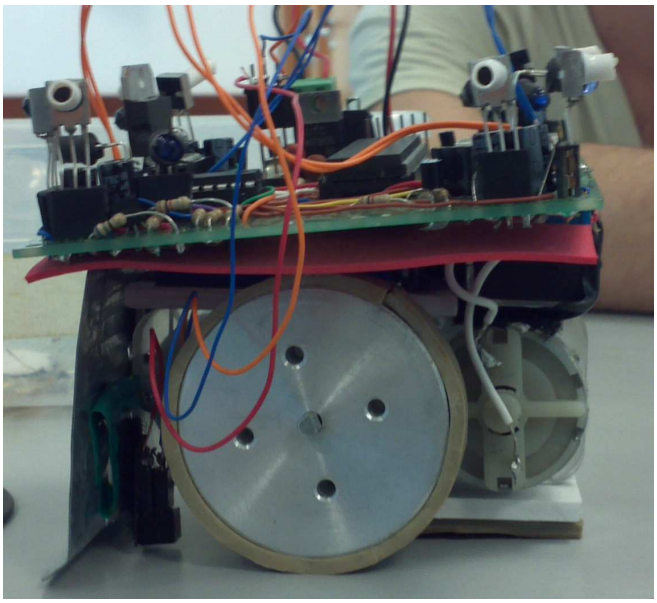
Categoría: minisumo

Nombre del Robot: USSOP

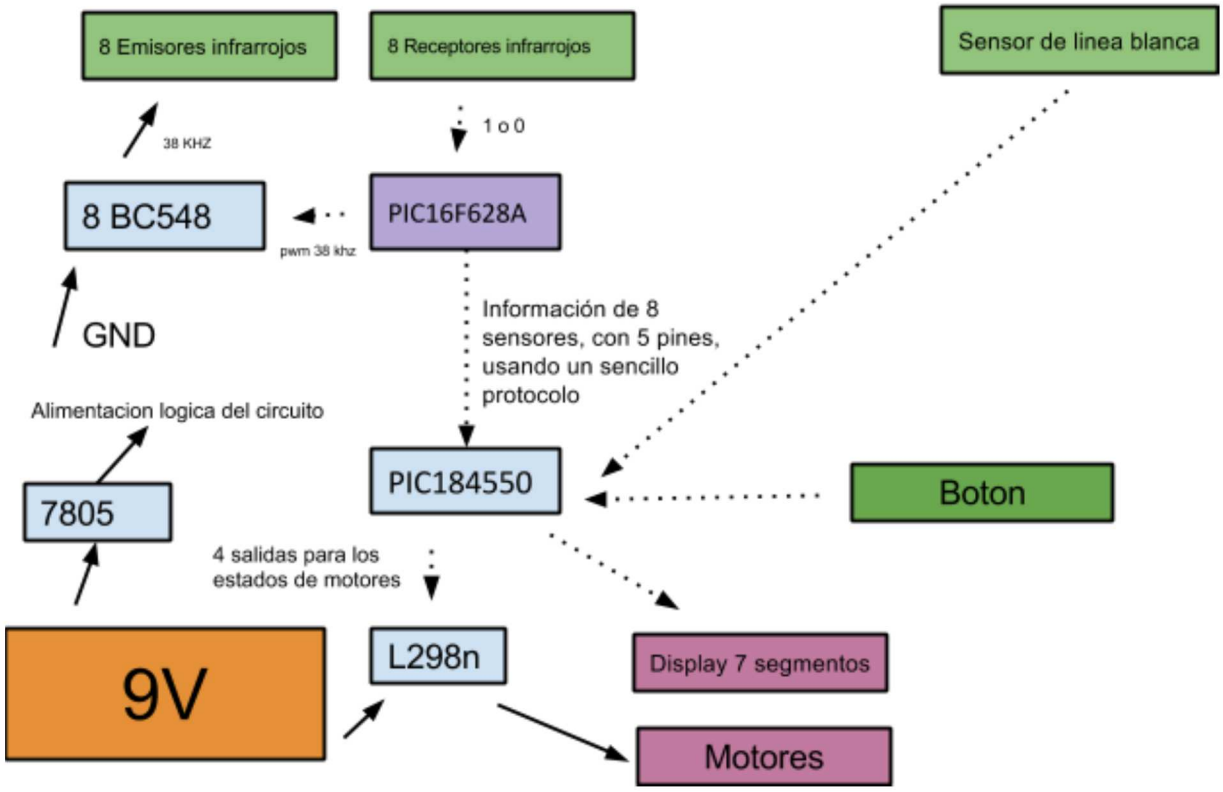
Institución: ORT ALMAGRO

Participantes:

- Ariel Nowik
- Sebastian Gonzalez



Funcionamiento



La electrónica de este robot se basó en las de muchos minisumos anteriores, pero con notables diferencias, debido a que el robot es un prototip, y el primero en poseer esta versión de plaqueta, que es la primera que he hecho, la misma se halla construida en una placa universal y no en un circuito impreso.

Los objetivos de este robot son:

- Aumentar la cantidad de sensores infrarrojos. Solían ser en menor cantidad en otros robots de la escuela, son 8 ahora. Se consigue tal cantidad de entradas multiplexando información a través de el micro chico, el 16f628a, que no trae programación variable, es fija y la cargue una sola vez.
- Aumentar la información mostrada y así facilitar el debug, en vez de colocar 2 o 3 leds, se ha colocado un display de 7 segmentos capaz de mostrar números del 0 al 9, y varias letras, ya que no utiliza 4511, sino que los segmentos son unidos directamente al microcontrolador
- Conseguir una programación más organizada que robots anteriores, dividiendo los códigos en 7 archivos y cabeceras de base, y 1 archivo de programación de funcionamiento, más, la programación del 16f628a para los sensores.

Los elementos que he aprendido de otros robots y fueron usados aquí fueron:

- Utilizar puente H, L298
- Utilizar sensores infrarrojos con emisor y receptor por separado (he copiado de hecho parte de código incluso)
- Utilizar 7805, para conseguir 5V para el mantenimiento lógico
- Utilizar sensor de piso CNY70

Electronica

Componentes:

- PIC18F4550 (Controlador principal, con el programa principal) 130$
- PIC16F628A (Controlador de sensores infrarrojos, utilizado para aumentar entradas) 35$

- 7805 (Regulador de tensión, para suministrar tensión a la zona logica) 5$
- l298n (Conmutador para motores hecho a base de transistores) 50$

- CNY 70 (Sensor de luz) 32$
- 2 resistencias de 330 (en paralelo, para el emisor del CNY70)
- 1 resistencia de 10k (de la salida de información del CNY70 a +5v, para evitar ruidos eléctricos)

- 8 x IRM8601 (Receptor infrarrojo a 38khz) 10$ x 8 = 80$
- 8 x Led infrarrojo 8$ x 8 = 64$
- 8 x BC548 (Conmutador para el led al recibir PWM) 0.59$ x 8 = 4,72$
- 8 x resistencias de 68 (en serie con la alimentación de los irm8601)
- 8 x resistencias de 1k (en serie con la base de los BC548, es decir por donde entra el PWM)
- 8 x resistencias de 10k (de la salida de información del IRM8601 a +5v para evitar ruidos eléctricos)

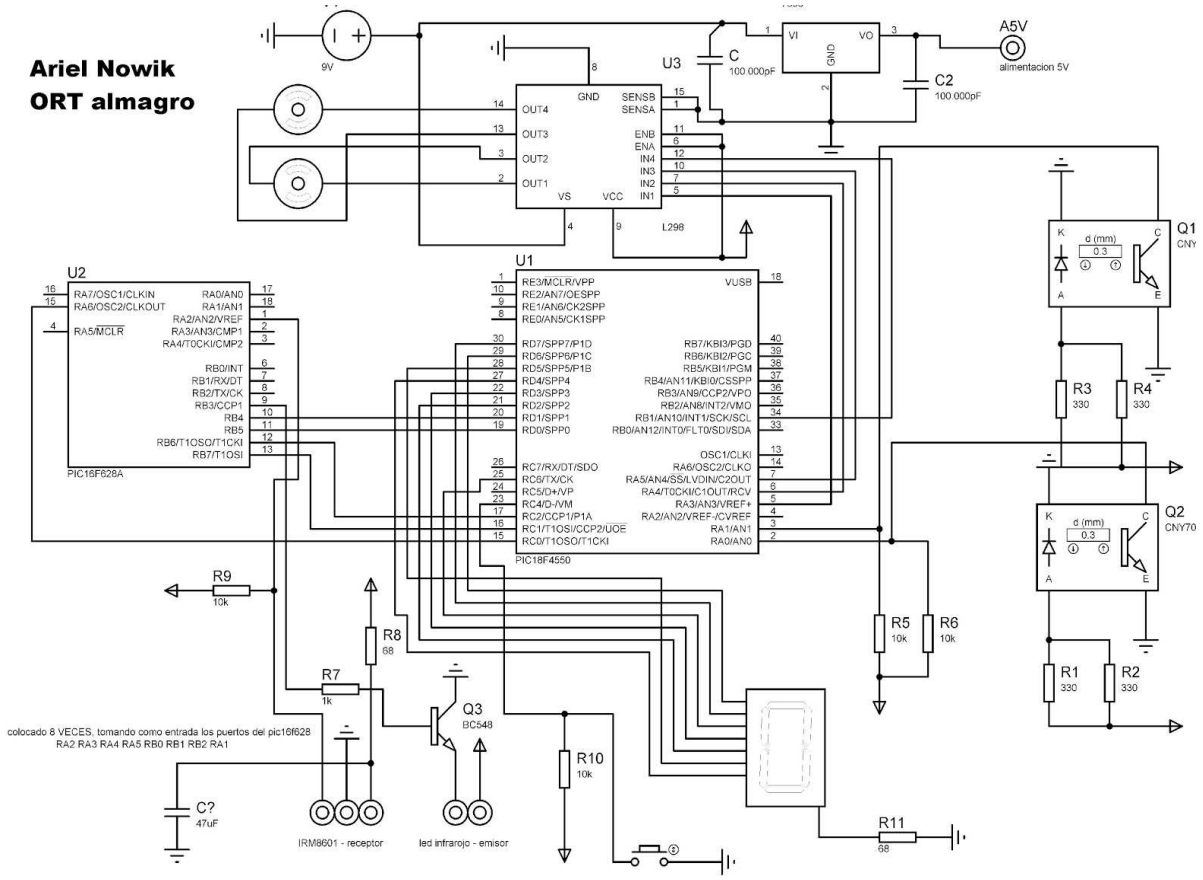
- Display de 7 segmentos (para mostrar información) 13$
- Boton (para mandar información al controlador) 1$

- Varios capacitores de 100 uF 50v, 47 uF 50v, 100.000 pF, para absorber rebotes (como una goma)
- Placa universal (precio no recordado)

- Se utilizó como programador para ambos micros, el pickit 3 de microchip

Subtotal: 414 $ (Sin la mecanica)

Esquematico simplificado:



Se tiene por un lado el pic16f628 al que le entran por 8 entradas los estados de los 8 sensores infrarojos, y además es el que genera el pulso PWM de 38khz que luego los 8 emisores de los 8 sensores utilizan para generar la luz infrarroja. Para transmitir la información de esos 8 sensores desde el pic16f628 al micro principal, el pic18f4550, se utilizan 5 salidas del pic16f628 y 5 entradas del pic18f4550 (en vez de 8) y se ganan 3 entradas ya que se hace una sencilla multiplexión para transmitir la información El cny70 manda la información directa , y después se tiene las salidas para el l298, que se encarga de conmutar ambos motores en el sentido de adelante y atrás. El display y el botón se conectan de manera clásica. Yy se utiliza un 7805 para bajar de los 9 volts que son para los motores también, a 5 volts que es lo que los micros, el display y los sensores necesitan.

Mecanica

- Motores APYS, 130 rpm sp4l130 , 100$
- Pequeña base plastica, precio despreciable
- Rampa de metal reciclada de otro uso, precio despreciable
- Goma protectora plaqueta, , precio despreciable
- Goma en la parte inferior del robot para equilibrar y hacer muy difícil que lo arrastren, precio despreciable
- Ruedas, 50$

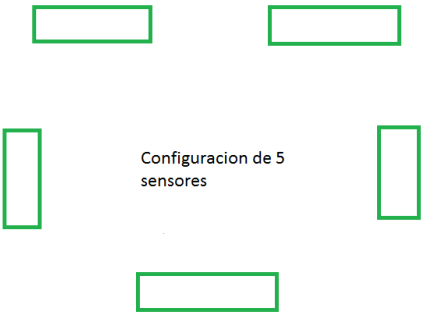
Ambos motores se encuentran unidos por un tornillo, y adheridos hacia la pequeña base plástica que hay abajo y arriba. La goma en la parte inferior del robot permite que sea difícil arrastrarlo. La rampa se encuentra fija al ras del piso atornillada hacia el motor.

Peso: 490 gr

Programación y conclusiones

Se utilizó para programar el robot el pickit 3 de microchip, ya que trae muchas funcionalidades como alimentar el circuito desde el programador, como también el debug muy sencillo de utilizar. El programa se creó de forma tal que la estrategia puede ser actualizada fácilmente ya que se separa el código de estrategia del código de administracion de sensores mediante diferentes archivos.

La estrategia clásica que se utiliza es avanzar hacia adelante, si ve el sensor de un costado girar hacia ese costado, si ve el del otro hacia el otro y, mientras se está girando, si ve algun sensor delantero cambia a un estado donde si ven ambos sensores delanteros o ninguno, se avanza hacia adelante, si ve el sensor delantero derecho se gira hacia la izquierda, si ve el sensor izquierdo se gira hacia la derecha, y así siempre el robot va a tender a dirigirse al enemigo con la parte delantera donde esta la rampa. Además, si ve el sensor de piso blanco, se procede a ir hacia atrás, a girar, y a continuar como antes. También se utiliza un sensor trasero, para ejecutar una secuencia de defensa en caso de que el otro rival se encuentre atrás cerca de tirar al robot.
Esta configuración utiliza 5 sensores, o al menos 5 sensores efectivos, los otros 3 se pueden usar para aumentar el rango de visión de esos 5 sensores.



La nueva configuración de 8 sensores (2 adelante, 2 atras, 2 derecha, 2 izquierda), que se quiere programar a diferencia de la de 5, tiene en cuenta la “cuarta dimensión”, es decir, si por ejemplo, ve primero un sensor de un lado, y luego el otro, se puede deducir que el otro robot se movió. Si por ejemplo sucede esta situación en el lado de atras, se puede decidir cómo girar para atacarlo, estimando donde estara ahora el otro robot. Lo mismo en los costados, si uno deduce que el robot contrario avanzó hacia adelante, se puede estimar que es más veloz el otro robot que el propio, por lo tanto, tener en cuenta tal condición en la programación (como ponerse en modo defensivo) . y si en cambio se ve al otro robot moverse en sentido contrario, uno puede atacar considerando la posicion futura del robot también. Esta idea de programación nunca ha sido probada, y es posible que en la competencia a modo experimental sea probada si no hay ninguna falla adicional que arreglar.

Anexo: Código fuente

Version 1.11 (estable y utilizada en competencias anteriores) la, versión inestable 1.12 se usará en la competencia y será finalizada de programar sobre la hora. A excepcion del ultimo codigo mostrado que corresponde a la programacion del pic16f628a,, todos son parte de la programación del pic18f4550,

- Programa principal con la estrategia : **main.c** el más importante

```
/*
 * File:   main.c
 * Author: newtonis
 *
 * Created on August 16, 2014, 10:14 AM
 */

#include <xc.h>
#include <stdlib.h>

#include "display.h"
#include "timer.h"
#include "button.h"
#include "infrared.h"
#include "sensores_piso.h"
#include "engine.h"

#define TESTEO 0
#define ESPERA 1
#define INICIO 2
#define ADELANTE 3
#define GIRO_DERECHO 4
#define GIRO_IZQUIERDO 5
#define ATRAS 6
#define PERSEGUIR 7
#define LINEAATRAS 8

char number;
char sensor;
char estado;

unsigned long AUXILIAR;
```

```
void revisar_linea();

int main(int argc, char** argv) {
   configurations_init(); //inicio configuraciones
   timer_init();         //inicio timer
   display_init();       //inicio display
   button_init();        //inicio boton
   spiso_init();         //inicio sensores de piso
   engine_init();        //inicio motores
   engine_disable_speed();//desactivo velocidad
   infrared_init(8);     //inicio sensores infrarojos
   infrared_disable_sensor_2();
   infrared_disable_sensor_4();
   estado = TESTEO;


   while (1){
      /*** actualizaciones ***/
      engine_update();
      button_update();
      infrared_update_data();


      /*** maquina de estados ***/


      switch (estado){
         case TESTEO:
            if (spiso_get_sensor_1() == BLANCO){
               display_show_number(9);
            }else{
               display_show_number( infrared_get_amount_sensors_enabled() );
            }

            if (button_get_pressed() == button_PRESSED){
               estado = ESPERA;
               timer_reset();
               srand( timer_get_miliseconds() );
            }
         break;
         case ESPERA:
            display_show_number( 5 - timer_get_seconds() );
            if (timer_get_seconds() == 5){
               estado = INICIO;
               AUXILIAR = rand() % 300;
            }
         break;
         case INICIO:
```

```
    revisar_linea();
    display_show_number(1);
    engine_giro_antihorario();
    if (timer_get_seconds()*1000 + timer_get_miliseconds() > 300 + AUXILIAR ){
      estado = ADELANTE;
    }
  break;
  case ADELANTE:
    revisar_linea();
    display_show_number(2);
    engine_adelante();
    if (infrared_get_sensor_5() == infrared_DETECTA || infrared_get_sensor_6() == infrared_DETECTA){
      estado = PERSEGUIR;
    }else if (infrared_get_sensor_3() == infrared_DETECTA){
      estado = GIRO_DERECHO;
      timer_reset();
      AUXILIAR = rand() % 200;
    }else if (infrared_get_sensor_8() == infrared_DETECTA){
      estado = GIRO_IZQUIERDO;
      timer_reset();
      AUXILIAR = rand() % 200;
    }
  break;
  case GIRO_DERECHO:
    revisar_linea();
    display_show_number(3);
    engine_giro_horario();
    if (infrared_get_sensor_5() == infrared_DETECTA || infrared_get_sensor_6() == infrared_DETECTA){
      timer_reset();
      estado = PERSEGUIR;
    }
    if (timer_get_seconds()*1000 + timer_get_miliseconds() >= AUXILIAR+600){
      estado = ADELANTE;
    }
  break;
  case GIRO_IZQUIERDO:
    revisar_linea();
    display_show_number(4);
    engine_giro_antihorario();
    if (infrared_get_sensor_5() == infrared_DETECTA || infrared_get_sensor_6() == infrared_DETECTA){
      timer_reset();
      estado = PERSEGUIR;
    }
    if (timer_get_seconds()*1000 + timer_get_miliseconds() >= AUXILIAR+600){
      estado = ADELANTE;
    }
  break;
  case PERSEGUIR:
```

```
            revisar_linea();
            display_show_number(5);


            if (infrared_get_sensor_5() == infrared_DETECTA && infrared_get_sensor_6() == infrared_DETECTA){
               engine_adelante();
            }else if(infrared_get_sensor_5() == infrared_DETECTA){
               engine_giro_antihorario();
            }else if(infrared_get_sensor_6() == infrared_DETECTA){
               engine_giro_horario();
            }else{
               estado = ADELANTE;
            }

          break;
        }     case ATRAS:
            display_show_number(9);
            if (timer_get_miliseconds() <= 750 && timer_get_seconds() == 0){
               engine_atras();
            }else{
               timer_reset();
               estado = LINEAATRAS;
            }
          break;
          case LINEAATRAS:
            engine_giro_antihorario();
            if (timer_get_miliseconds() <= 500){

            }else{
               estado = ADELANTE;
            }
          break;

        }
      }
    }

    void revisar_linea(){
      if (spiso_get_sensor_1() == BLANCO){
        estado = ATRAS;
        AUXILIAR = rand() % 500;
        timer_reset();
      }
    }
```

- Programa para los sensores infrarojos, *infrared.h*, seguido de *infrared.c*

```
/*
 * File:   infrared.h
 * Author: newtonis
 *
 * Created on August 16, 2014, 8:28 PM
 */

#ifndef INFRARED_H
#define  INFRARED_H

#ifdef    __cplusplus
extern "C" {
#endif

#include <xc.h>

#define OUTPUT 0
#define INPUT  1

#define infrared_DETECTA 0
#define infrared_NODETECTA 1
#define infrared_DISABLED 2

#define DEFINE_BIT PORTDbits.RD1
#define IN_BIT_0   PORTDbits.RD0
#define IN_BIT_1   PORTCbits.RC2
#define IN_BIT_2   PORTCbits.RC1
#define IN_BIT_3   PORTCbits.RC0

void infrared_init(char enabled);
void infrared_update_data();
void infrared_disable_sensor_2();
void infrared_disable_sensor_4();

char infrared_get_sensor_1();
char infrared_get_sensor_2();
char infrared_get_sensor_3();
char infrared_get_sensor_4();
char infrared_get_sensor_5();
char infrared_get_sensor_6();
char infrared_get_sensor_7();
char infrared_get_sensor_8();
char infrared_get_amount_sensors_enabled();
```

```
void infrared_set_sensibilidad(char s);

#ifdef    __cplusplus
}
#endif

#endif    /* INFRARED_H */


----------------------

#include "infrared.h"
#include <xc.h>

char estado_sensor_1;
char estado_sensor_2;
char estado_sensor_3;
char estado_sensor_4;
char estado_sensor_5;
char estado_sensor_6;
char estado_sensor_7;
char estado_sensor_8;


void infrared_init(char enabled){
   TRISDbits.TRISD1 = INPUT; //DEFINE BIT
   TRISDbits.TRISD0 = INPUT; //bit 0
   TRISCbits.TRISC2 = INPUT; //bit 1
   TRISCbits.TRISC1 = INPUT; //bit 2
   TRISCbits.TRISC0 = INPUT; //bit 3

   if (enabled >= 1) {estado_sensor_1 = infrared_NODETECTA; }else{ estado_sensor_1 = infrared_DISABLED; }
   if (enabled >= 2) {estado_sensor_2 = infrared_NODETECTA; }else{ estado_sensor_2 = infrared_DISABLED; }
   if (enabled >= 3) {estado_sensor_3 = infrared_NODETECTA; }else{ estado_sensor_3 = infrared_DISABLED; }
   if (enabled >= 4) {estado_sensor_4 = infrared_NODETECTA; }else{ estado_sensor_4 = infrared_DISABLED; }
   if (enabled >= 5) {estado_sensor_5 = infrared_NODETECTA; }else{ estado_sensor_5 = infrared_DISABLED; }
   if (enabled >= 6) {estado_sensor_6 = infrared_NODETECTA; }else{ estado_sensor_6 = infrared_DISABLED; }
   if (enabled >= 7) {estado_sensor_7 = infrared_NODETECTA; }else{ estado_sensor_7 = infrared_DISABLED; }
   if (enabled >= 8) {estado_sensor_8 = infrared_NODETECTA; }else{ estado_sensor_8 = infrared_DISABLED; }
}
void infrared_disable_sensor_2(){
   estado_sensor_2 = infrared_DISABLED;
}
void infrared_disable_sensor_4(){
   estado_sensor_4 = infrared_DISABLED;
}
void infrared_update_data(){
   if (DEFINE_BIT == 0){
```

```
      if (estado_sensor_1 != infrared_DISABLED) { estado_sensor_1 = IN_BIT_0; }
      if (estado_sensor_2 != infrared_DISABLED) { estado_sensor_2 = IN_BIT_1; }
      if (estado_sensor_3 != infrared_DISABLED) { estado_sensor_3 = IN_BIT_2; }
      if (estado_sensor_4 != infrared_DISABLED) { estado_sensor_4 = IN_BIT_3; }
   }else if (DEFINE_BIT == 1){
      if (estado_sensor_5 != infrared_DISABLED) { estado_sensor_5 = IN_BIT_0; }
      if (estado_sensor_6 != infrared_DISABLED) { estado_sensor_6 = IN_BIT_1; }
      if (estado_sensor_7 != infrared_DISABLED) { estado_sensor_7 = IN_BIT_2; }
      if (estado_sensor_8 != infrared_DISABLED) { estado_sensor_8 = IN_BIT_3; }
   }
}
char infrared_get_sensor_1(){
   return estado_sensor_1;
}
char infrared_get_sensor_2(){
   return estado_sensor_2;
}
char infrared_get_sensor_3(){
   return estado_sensor_3;
}
char infrared_get_sensor_4(){
   return estado_sensor_4;
}
char infrared_get_sensor_5(){
   return estado_sensor_5;
}
char infrared_get_sensor_6(){
   return estado_sensor_6;
}
char infrared_get_sensor_7(){
   return estado_sensor_7;
}
char infrared_get_sensor_8(){
   return estado_sensor_8;
}
char infrared_get_amount_sensors_enabled(){
   char amount;
   amount = 0;
   if (infrared_get_sensor_1() == infrared_DETECTA){ amount ++; }
   if (infrared_get_sensor_2() == infrared_DETECTA){ amount ++; }
   if (infrared_get_sensor_3() == infrared_DETECTA){ amount ++; }
   if (infrared_get_sensor_4() == infrared_DETECTA){ amount ++; }
   if (infrared_get_sensor_5() == infrared_DETECTA){ amount ++; }
   if (infrared_get_sensor_6() == infrared_DETECTA){ amount ++; }
   if (infrared_get_sensor_7() == infrared_DETECTA){ amount ++; }
   if (infrared_get_sensor_8() == infrared_DETECTA){ amount ++; }
   return amount;
}
```

- Programa que configura los segmentos para el display de 7 segmentos, display.h, seguido de display.c

```
/*
 * File:   display.h
 * Author: newtonis
 *
 * Created on August 16, 2014, 10:15 AM
 */

#ifndef DISPLAY_H
#define  DISPLAY_H

#ifdef    __cplusplus
extern "C" {
#endif

#include <xc.h>
#include "configurations.h"


/*** outputs ***/
#define A PORTDbits.RD6
#define B PORTDbits.RD7
#define C PORTCbits.RC6
#define D PORTDbits.RD3
#define E PORTDbits.RD2
#define F PORTDbits.RD5
#define G PORTDbits.RD4

#define DP PORTCbits.RC7

void display_init();  // iniciar el modulo
void display_set_IO();// configurar entradas y salidas
void display_show_number(char number); // mostrar tal numero
void display_shut_off(); //apagar
void display_shut_on();  //prender
void display_blank(); //blanquear

#ifdef    __cplusplus
}
#endif

#endif    /* DISPLAY_H */
```

```
#include <xc.h>
#include "display.h"

#define OUTPUT 0
#define INPUT  1

#define ACTIVADO 1
#define DESACTIVADO 0

char segmentos[10][7] = {
   {1,1,1,1,1,1,0},
   {0,1,1,0,0,0,0},
   {1,1,0,1,1,0,1},
   {1,1,1,1,0,0,1},
   {0,1,1,0,0,1,1},
   {1,0,1,1,0,1,1},
   {1,0,1,1,1,1,1},
   {1,1,1,0,0,0,0},
   {1,1,1,1,1,1,1},
   {1,1,1,1,0,1,1}
};
char show;

void display_init(){
   show = ACTIVADO;
   display_set_IO();
}
void display_set_IO(){
   TRISDbits.TRISD7 = OUTPUT;
   TRISDbits.TRISD6 = OUTPUT;
   TRISDbits.TRISD5 = OUTPUT;
   TRISDbits.TRISD4 = OUTPUT;
   TRISCbits.TRISC7 = OUTPUT;
   TRISCbits.TRISC6 = OUTPUT;
   TRISDbits.TRISD3 = OUTPUT;
   TRISDbits.TRISD2 = OUTPUT;
}
void display_show_number(char number){
   if (number > 10){
      number = number % 10;
   }
   if (show == ACTIVADO){
      A = segmentos[number][0];
      B = segmentos[number][1];
      C = segmentos[number][2];
      D = segmentos[number][3];
      E = segmentos[number][4];
```

```
      F = segmentos[number][5];
      G = segmentos[number][6];
      DP= 0;

   }else if (show == DESACTIVADO){
      display_blank();
   }
}
void display_shut_on(){
   show = ACTIVADO;
}
void display_shut_off(){
   show = DESACTIVADO;
   display_blank();
}
void display_blank(){
   A = 0;
   B = 0;
   C = 0;
   D = 0;
   E = 0;
   F = 0;
   G = 0;
   DP= 0;
}
```

- Programa para el sensor de piso, sensores_piso.h, seguido de sensores_piso.c

```
/*
 * File:   sensores_piso.h
 * Author: newtonis
 *
 * Created on August 21, 2014, 11:46 PM
 */

#ifndef SENSORES_PISO_H
#define  SENSORES_PISO_H

#ifdef    __cplusplus
extern "C" {
#endif

#define BLANCO 0
#define NEGRO  1
```

```
#include <xc.h>

#define spiso_SENSOR_1 PORTAbits.RA0


void spiso_init();
char spiso_get_sensor_1();


#ifdef __cplusplus
}
#endif

#endif /* SENSORES_PISO_H */
```

```
#include "sensores_piso.h"


#include <xc.h>

#define OUTPUT 0
#define INPUT 1

void spiso_init(){
  TRISAbits.TRISA0 = INPUT; //sensor de piso
}

char spiso_get_sensor_1(){
  if (spiso_SENSOR_1 == 0){
    return BLANCO;
  }else{
    return NEGRO;
  }
}
```

- Programa para hacer la maquina de estados del boton, button.h, seguido de button.c

```
/*
 * File:   button.h
 * Author: newtonis
 *
 * Created on August 16, 2014, 10:54 AM
 */
```

```
#ifndef BUTTON_H
#define  BUTTON_H

#ifdef     __cplusplus
extern "C" {
#endif

#include <xc.h>

#define BUTTON PORTCbits.RC4

#define DONOW 0
#define ALREADYDONE 1
#define RELEASED 2

#define button_PRESSED 3
#define button_NOPRESSED 4

void button_init();
void button_update();
char button_get_pressed();


#ifdef     __cplusplus
}
#endif

#endif    /* BUTTON_H */


-----------------------

#include "button.h"
#include <xc.h>

#define OUTPUT 0
#define INPUT 1
char state;

void button_init(){
   state = RELEASED;

}
void button_update(){
   switch (state){
      case RELEASED:
         if (BUTTON == 0){
            state = DONOW;
```

```
        }
      break;
      case ALREADYDONE:
        if (BUTTON == 1){
          state = RELEASED;
        }
      break;

  }
}
char button_get_pressed(){
  if (state == DONOW){
    state = ALREADYDONE;
    return button_PRESSED;
  }
  return button_NOPRESSED;
}
```

- Programa con las funciones para los motores, adelante, atras, derecha, izquierda. engine.h, seguido de engine.c

```
/*
 * File:   engine.h
 * Author: newtonis
 *
 * Created on August 17, 2014, 8:27 PM
 */

#ifndef ENGINE_H
#define  ENGINE_H

#ifdef     __cplusplus
extern "C" {
#endif

#include <xc.h>

#define OUTPUT 0
#define INPUT  1

#define OUT_1 PORTAbits.RA6
#define OUT_2 PORTAbits.RA5
#define OUT_3 PORTAbits.RA4
#define OUT_4 PORTAbits.RA3

#define NORMAL 1
```

```
#define APAGADO 0

#define engine_giro_derecha engine_giro_horario
#define engine_giro_izquierda engine_giro_antihorario

void engine_set_velocidad(char velocidad);
void engine_enable_speed();
void engine_disable_speed();

void engine_init();

void engine_frenar();
void engine_adelante();
void engine_atras();

void engine_giro_horario();
void engine_giro_antihorario();

#ifdef    __cplusplus|
}
#endif

#endif    /* ENGINE_H */
```

```
#include "engine.h"
#include <xc.h>

unsigned char speed;
unsigned char ciclos;

char OUT1STATE;
char OUT2STATE;
char OUT3STATE;
char OUT4STATE;

void engine_set_velocidad(char velocidad){
   speed = velocidad;
}

void engine_init(){
   TRISAbits.RA6 = OUTPUT;
   TRISAbits.RA5 = OUTPUT;
   TRISAbits.RA4 = OUTPUT;
   TRISAbits.RA3 = OUTPUT;
```

```
  ciclos = 0;
  engine_set_velocidad(255); //ANTENCION, MUY EXPERIMENTAL
  engine_frenar();
}

void engine_frenar(){
  OUT1STATE = APAGADO;
  OUT2STATE = APAGADO;
  OUT3STATE = APAGADO;
  OUT4STATE = APAGADO;
}
void engine_adelante(){
  OUT1STATE = NORMAL;
  OUT2STATE = APAGADO;
  OUT3STATE = NORMAL;
  OUT4STATE = APAGADO;
}
void engine_atras(){
  OUT1STATE = APAGADO;
  OUT2STATE = NORMAL;
  OUT3STATE = APAGADO;
  OUT4STATE = NORMAL;
}
void engine_giro_horario(){
  OUT1STATE = APAGADO;
  OUT2STATE = NORMAL;
  OUT3STATE = NORMAL;
  OUT4STATE = APAGADO;
}
void engine_giro_antihorario(){
  OUT1STATE = NORMAL;
  OUT2STATE = APAGADO;
  OUT3STATE = APAGADO;
  OUT4STATE = NORMAL;
}

void engine_update(){
  if (ciclos >= speed){
    OUT_1 = 0;
    OUT_2 = 0;
    OUT_3 = 0;
    OUT_4 = 0;
  }else{
    if (OUT1STATE == NORMAL){ OUT_1 = 1; }else{ OUT_1 = 0; }
    if (OUT2STATE == NORMAL){ OUT_2 = 1; }else{ OUT_2 = 0; }
    if (OUT3STATE == NORMAL){ OUT_3 = 1; }else{ OUT_3 = 0; }
    if (OUT4STATE == NORMAL){ OUT_4 = 1; }else{ OUT_4 = 0; }
  }
```

```
   ciclos ++;
}
```

- Programa para las funciones del timer que se utiliza para contar tiempo en segundos. timer.h, seguido de timer.c

```
/*
 * File:   timer.h
 * Author: newtonis
 *
 * Created on August 16, 2014, 10:34 AM
 */

#ifndef TIMER_H
#define  TIMER_H

#include <xc.h>


#ifdef    __cplusplus
extern "C" {
#endif

typedef struct megaclock{
   unsigned int miliseconds;
   unsigned char seconds;
   unsigned char minutes;
}megaclock;

void timer_init();
void timer_reset();
int timer_get_miliseconds();
int timer_get_seconds();
int timer_get_minutes();

#ifdef    __cplusplus
}
#endif

#endif    /* TIMER_H */

--------------------


#include "timer.h"
#include <xc.h>
```

```
megaclock clock;

void timer_init(void){
   T0CONbits.TMR0ON = 0; //timer0 OFF
   T0CONbits.T08BIT = 0; //timer 0 a 16 bit
   T0CONbits.T0CS   = 0; //timer0 en modo interno
   T0CONbits.PSA    = 1; //no activo prescaler
   TMR0H = 0xF8;
   TMR0L = 0x2F; //f82f = 63535 = 65535 - 1000 -> a interrumpir cada 1000 microsegundos

   //ALTA PRIORIDAD, BAJA PRIORIDAD

   INTCON2bits.TMR0IP = 1; //ALTA PRIORIDAD
   RCONbits.IPEN      = 0; //que no haya prioridades
   INTCONbits.TMR0IE  = 1; //habilitar interrupcion timer 0


   INTCONbits.PEIE_GIEL = 1; //habilitar interrupciones de perifericos
   INTCONbits.GIE = 1; //GLOBAL INTERRUPT ENABLE
   T0CONbits.TMR0ON = 1; //activo timer 1
}

void timer_reset(){
   clock.seconds = 0;
   clock.minutes = 0;
   clock.miliseconds = 0;
}

int timer_get_miliseconds(){
   return clock.miliseconds;
}
int timer_get_seconds(){
   return clock.seconds;
}
int timer_get_minutes(){
   return clock.minutes;
}

void interrupt timer0(void){
   //T0CONbits.TMR0ON = 0;

   if (TMR0IF){
      TMR0H = 0xF8;
      TMR0L = 0x2F; //f82f = 63535 = 65535 - 1000 -> a interrumpir cada 1000 microsegundos

     clock.miliseconds ++;
     if (clock.miliseconds >= 1000){
         clock.miliseconds = 0;
```

```
        clock.seconds ++;
    }
    /* fin contenido */

    TMR0IF = 0;
     //T0CONbits.TMR0ON = 1;
  }
}
```

- Programa con las configuraciones de los bits de configuracion del pic18f4550. Se separan en otro archivo para no ensuciar el archivo main.c. configurations.h seguido de configurations.cpp

```
/*
 * File:   configurations.h
 * Author: newtonis
 *
 * Created on August 16, 2014, 10:14 AM
 */

#ifndef CONFIGURATIONS_H
#define  CONFIGURATIONS_H

#ifdef    __cplusplus
extern "C" {
#endif

#include <xc.h>


#pragma config VREGEN = OFF            // Voltage regulat USB , is  Suspended
#pragma config WDT    = OFF            // Watchdog timer is suspended
#pragma config PLLDIV = 5           // Internal Oscillator engaged
#pragma config MCLRE  = OFF
#pragma config WDTPS  = 32768
#pragma config CCP2MX = ON
#pragma config PBADEN = OFF
#pragma config CPUDIV = OSC1_PLL2
#pragma config USBDIV = 2
#pragma config FOSC   = INTOSCIO_EC
#pragma config FCMEN  = OFF
#pragma config IESO   = OFF
#pragma config PWRT   = OFF
#pragma config BOR    = OFF
#pragma config BORV   = 3
#pragma config LPT1OSC= OFF
#pragma config STVREN = ON
#pragma config LVP    = OFF
```

```
#pragma config ICPRT  = OFF
#pragma config XINST  = OFF
#pragma config DEBUG  = OFF
#pragma config CP0    = OFF, CP1 = OFF, CP2 = OFF, CP3 = OFF
#pragma config CPB    = OFF          // CPB off
#pragma config CPD    = OFF
#pragma config WRT0   = OFF, WRT1 = OFF, WRT2 = OFF, WRT3 = OFF
#pragma config WRTC   = OFF
#pragma config WRTB   = OFF
#pragma config WRTD   = OFF
#pragma config EBTR0  = OFF, EBTR1 = OFF, EBTR2 = OFF, EBTR3 = OFF
#pragma config EBTRB  = OFF

void configurations_init();


#ifdef    __cplusplus
}
#endif

#endif    /* CONFIGURATIONS_H */

-------------------------
#include "configurations.h"

void configurations_init(){
   OSCCONbits.IRCF = 7;    /* OSCILLATOR CONTROL REGISTER -> 8MHz */
   ADCON1=0x0F;           // Set all pins as digital I/O
   CMCON=0x07;           // Set all comparators as digital I/O

   //disable usb
   UCONbits.USBEN = 0;
   UCFGbits.UTRDIS = 1;
}
```

● Programación del pic16f628a

Este chip administra los sensores y transmite informacion al otro micro mediante mutiplexión.

```
/*
 * File:   main.c
 * Author: newtonis
 *
 * Created on August 16, 2014, 11:51 AM
 */

#include <xc.h>

#pragma config BOREN = OFF, CPD = OFF, FOSC = INTOSCIO, MCLRE = OFF, WDTE = OFF, CP = OFF, LVP = OFF,
PWRTE = OFF
#define _XTAL_FREQ 4000000

#define INICIO  0
#define PAUSA   1
#define LEER    2
#define APAGADO 3

#define OUTPUT 0
#define INPUT  1

#define SENSOR_1 PORTAbits.RA2
#define SENSOR_2 PORTAbits.RA3
#define SENSOR_3 PORTAbits.RA4
#define SENSOR_4 PORTAbits.RA5
#define SENSOR_5 PORTBbits.RB0
#define SENSOR_6 PORTBbits.RB1
#define SENSOR_7 PORTBbits.RB2
#define SENSOR_8 PORTAbits.RA1

#define DEFINE_BIT PORTBbits.RB4
#define OUT_BIT_0 PORTBbits.RB5
#define OUT_BIT_1 PORTBbits.RB6
#define OUT_BIT_2 PORTBbits.RB7
#define OUT_BIT_3 PORTAbits.RA6

char estado;
char contador;
char ciclos;

char sensor_1;
char sensor_2;
char sensor_3;
char sensor_4;
char sensor_5;
char sensor_6;
char sensor_7;
char sensor_8;

void init(void);
```

```
void configurar_pwm(void);
void configurar_IO(void);
void configurar_timer0(void);
void configurar_timer1(void);
void sensores(void);
void read();
void actualizar_salidas(void);

int main(int argc, char** argv) {

   init();
   configurar_IO();
   configurar_pwm();
   configurar_timer1();
   configurar_timer0();
   while (1){
      sensores();
      actualizar_salidas();
      ciclos ++;
   }
}


void init(void){
   CMCON = 0x07;
   contador = 0;
   estado = INICIO;
   ciclos = 0;
}
void configurar_pwm(void){
   T2CONbits.TMR2ON=1; // timer2 prendido
   T2CONbits.TOUTPS=0; // hasta 15
   T2CONbits.T2CKPS=0; // 00=1:1 01=1:4 1x=1:16
   TRISBbits.TRISB3=0; // pwm salida
   PR2=26; // periodo 38Khz
   CCPR1L=0; //apagado
   CCP1CON=12;
}
void configurar_timer1(void){
   TMR1H=0xFC;
   TMR1L=0x18;

   PIE1bits.TMR1IE    = 0; //desabilita interrupcion
   PIR1bits.TMR1IF    = 0; //desborde

   T1CONbits.T1CKPS   = 0; //prescaler

   T1CONbits.nT1SYNC  = 1;
   T1CONbits.T1OSCEN  = 0;
   T1CONbits.TMR1CS   = 0;
```

```
   T1CONbits.TMR1ON    = 0; //on-off
   T1CONbits.TMR1ON    = 1; //prender timer
}
void configurar_timer0(void){
   OPTION_REGbits.T0CS = 0;
   OPTION_REGbits.PSA  = 0;
   OPTION_REGbits.PS   = 1;

   INTCONbits.T0IF     = 0;
   INTCONbits.T0IE     = 1;
   INTCONbits.GIE      = 1;
}
void configurar_IO(void){
   TRISB3 = OUTPUT;  //pwm

   TRISA2 = INPUT;  //informacion sensor 1
   TRISA3 = INPUT;  //informacion sensor 2
   TRISA4 = INPUT;  //informacion sensor 3
   TRISA5 = INPUT;  //informacion sensor 4
   TRISB0 = INPUT;  //informacino sensor 5
   TRISB1 = INPUT;  //informacion sensor 6
   TRISB2 = INPUT;  //informacion sensor 7
   TRISA1 = INPUT;  //informacion sensor 8

   TRISB4 = OUTPUT; //bit que define sensores 1-4 o 5-8
   TRISB5 = OUTPUT; //bit sensor 1 o 5
   TRISB6 = OUTPUT; //bit sensor 2 o 6
   TRISB7 = OUTPUT; //bit sensor 3 o 7
   TRISA6 = OUTPUT; //bit sensor 4 o 8
}
void interrupt t0_int(void){
   INTCONbits.T0IF=0;
   contador ++;
}
void sensores(){
   switch (estado){
      case INICIO:
         CCPR1L=15; //prender pwm
         T1CONbits.TMR1ON=1; //prender timer
         estado = PAUSA;
      break;
      case PAUSA:
         if(PIR1bits.TMR1IF==1){  //desborde
            PIR1bits.TMR1IF=0; //borro desborde
            T1CONbits.TMR1ON=0; //apagar timer
            TMR1H=0xFC;
            TMR1L=0x17;
            estado = LEER;
         }
      break;
```

```
      case LEER:
        read();
        contador = 0;
        CCPR1L=0; //apagado pwm
        estado = APAGADO;
      break;
      case APAGADO:
        if(contador == 4){
          estado = INICIO;
        }/*
              }else if(contador_maquina < 1000){

              }else{
                    contador_maquina = 0;
              }
        */
      break;
    }
}
void read(){
   sensor_1 = SENSOR_1;
   sensor_2 = SENSOR_2;
   sensor_3 = SENSOR_3;
   sensor_4 = SENSOR_4;
   sensor_5 = SENSOR_5;
   sensor_6 = SENSOR_6;
   sensor_7 = SENSOR_7;
   sensor_8 = SENSOR_8;
}
void actualizar_salidas(){
   if (ciclos % 10 > 4){ //ciclos 0,1,2,3,4
      DEFINE_BIT = 0;
      OUT_BIT_0 = sensor_1;
      OUT_BIT_1 = sensor_2;
      OUT_BIT_2 = sensor_3;
      OUT_BIT_3 = sensor_4;
   }else{
      DEFINE_BIT = 1; //ciclos 5,6,7,8,9

      OUT_BIT_0 = sensor_5;
      OUT_BIT_1 = sensor_6;
      OUT_BIT_2 = sensor_7;
      OUT_BIT_3 = sensor_8;
   }
}
```