**The sensors are connected just like in the tutorial forum Pololu, jumpers are not connected. The 1023 value is permanently displayed on the LCD, for each program below:**

PROGRAM 1

/\* Programa: paseo.c Jaume Brosa 11/2009

\* Descripción:

\* Programa para el 3pi con dos sensores Sharp 2YOA21 conectados

\* en ADC7 izq y ADC5 der y que le permite esquivar obstáculos

\* Quitar jumpers de PC5 y ADC7.

\*/

// Includes

#include <pololu/3pi.h> // especifico 3pi

#include <avr/pgmspace.h> // variables en memoria de programa

// Mensajes de introducción guardados en memoria de programa

const char linea1[] PROGMEM = " Pololu";

const char linea2[] PROGMEM = "3\xf7 Robot";

// Tonos musicales guardados en memoria de programa

const char hola[] PROGMEM = ">g32>>c32";

const char ir[] PROGMEM = "L16 cdegreg4";

// Variables generales

unsigned int sens\_der; // sensor right

unsigned int sens\_izq; // sensor left

int motor\_der; // motor right

int motor\_izq; // motor left

void initialize(){

// Set PC5 as an input with internal pull-up disabled

DDRC &= ~(1<< PORTC5);

PORTC &= ~(1<< PORTC5);

emitters\_off();

// Toca music y muestra mensaje de hola

print\_from\_program\_space(linea1);

lcd\_goto\_xy(0,1);

print\_from\_program\_space(linea2);

play\_from\_program\_space(hola);

delay\_ms(1000);

//Muestra el voltaje de la batería y espera botón

while(!button\_is\_pressed(BUTTON\_B))

{

clear();

print\_long(read\_battery\_millivolts()); // usa ADC6 para batería

print("mV");

lcd\_goto\_xy(0,1);

if (read\_battery\_millivolts()<4800){

print (" !Ahhh¡");

red\_led(1);

}

else

print("Pulsa B");

delay\_ms(100);

}

// Espera botón B para empezar a moverse

wait\_for\_button\_release(BUTTON\_B);

while(!button\_is\_pressed(BUTTON\_B))

{

clear();

lcd\_goto\_xy(0,0);

print ("I ");

print\_long(analog\_read(7)); // valor ADC7 sensor <

lcd\_goto\_xy(0,1);

print ("D ");

print\_long(analog\_read(5)); // valor ADC5 sensor >

delay\_ms(200);

}

wait\_for\_button\_release(BUTTON\_B);

clear();

// Toca música y espera a que termine para empezar.

play\_from\_program\_space(ir);

while(is\_playing());

}

void lee\_sensores(){

// Mira derecha si hay obstáculos

if (!analog\_is\_converting()) sens\_der = analog\_read(5);

// mira izquierda si hay obstáculos

if (!analog\_is\_converting()) sens\_izq = analog\_read(7);

}

void busca\_exit(){

clear();

lcd\_goto\_xy(0,0);

print("Buscando"); // Busca salida

red\_led(1); // Encienda leds

green\_led(1);

while (sens\_der>400 || sens\_izq>400){

play ("c32");

set\_motors(40,-40); // gira or -40,40

delay\_ms(200);

lee\_sensores();

play ("g32");

}

red\_led(0); // OK salida encontrada

green\_led(0); // apaga leds

}

int main(){

// inicializa 3pi

initialize();

// Bucle principal.

while(1){

lee\_sensores();

if (sens\_der>400 || sens\_izq>400){

set\_motors(0,0); // stop motors

delay\_ms(200);

busca\_exit();

}

if (sens\_izq > 200 && sens\_der < 200){

// obstaculo izq gira a derecha ------>

motor\_izq=100-sens\_der/10; // acelera

motor\_der=100-sens\_izq/10; // reduce

}

if (sens\_izq < 200 && sens\_der > 200){

// obstaculo a der gira izquierda <-----

motor\_izq=100-sens\_der/10; // reduce -

motor\_der=100-sens\_izq/10; // acelera +

}

if (sens\_izq < 200 && sens\_der < 200){

motor\_izq=127;

motor\_der=127;

}

if (sens\_izq < 100 && sens\_der < 100){

motor\_izq=150;

motor\_der=150;

}

set\_motors(motor\_izq, motor\_der); //izq^.^.^.^.^der

delay\_ms(100);

clear(); // Mostrar valores

lcd\_goto\_xy(0,0);

print\_long(sens\_izq);

lcd\_goto\_xy(5,0); // valor sensor

print\_long(sens\_der);

lcd\_goto\_xy(0,1);

print\_long(motor\_izq);

lcd\_goto\_xy(5,1); // valor motores

print\_long(motor\_der);

}

}

// end

PROGRAM 2

unsigned int read(unsigned char channel)

{

// Channel numbers greater than 31 are invalid.

if (channel > 31)

{

return 0;

}

ADCSRA = 0x87; // bit 7 set: ADC enabled

// bit 6 clear: don't start conversion

// bit 5 clear: disable autotrigger

// bit 4: ADC interrupt flag

// bit 3 clear: disable ADC interrupt

// bits 0-2 set: ADC clock prescaler is 128

// 128 prescaler required for 10-bit resolution when FCPU = 20 MHz

ADMUX &= ~(1 << 7);

ADMUX |= 1 << 6; // use AVCC as voltage reference

ADMUX &= ~0x1F; // clear channel selection bits of ADMUX

ADMUX |= channel; // we only get this far if channel is less than 32

ADCSRA |= 1 << ADSC; //

PROGRAM 3

#include <pololu/3pi.h>

int main()

{

while(1)

{

print\_long(read\_trimpot());

print(" p "); // to clear the display

lcd\_goto\_xy(0,0);

}

while(1);

}

PROGRAM 4

/\*

OrangutanAnalog.h - Library for using the handling analog inputs on the

Orangutan LV-168, Baby Orangutan B, or 3pi robot. This library also

provides a method for reading the temperature sensor on the LV-168.

\*/

/\*

\* Written by Ben Schmidel, May 27, 2008.

\* Copyright (c) 2008 Pololu Corporation. For more information, see

\*

\* http://www.pololu.com

\* http://forum.pololu.com

\* http://www.pololu.com/docs/0J18/3

\*

\* You may freely modify and share this code, as long as you keep this

\* notice intact (including the two links above). Licensed under the

\* Creative Commons BY-SA 3.0 license:

\*

\* http://creativecommons.org/licenses/by-sa/3.0/

\*

\* Disclaimer: To the extent permitted by law, Pololu provides this work

\* without any warranty. It might be defective, in which case you agree

\* to be responsible for all resulting costs and damages.

\*/

#ifndef OrangutanAnalog\_h

#define OrangutanAnalog\_h

#define MODE\_8\_BIT 1

#define MODE\_10\_BIT 0

#define TRIMPOT 7

#define TEMP\_SENSOR 6

class OrangutanAnalog

{

public:

// constructor (doesn't do anything)

OrangutanAnalog();

// set the ADC to run in either 8-bit mode (MODE\_8\_BIT) or

// 10-bit mode (MODE\_10\_BIT)

static void setMode(unsigned char mode);

// returns 0 if in 10-bit mode, otherwise returns non-zero. The return

// value of this method can be directly compared against the macros

// MODE\_8\_BIT and MODE\_10\_BIT:

// For example: if (getMode() == MODE\_8\_BIT) ...

static unsigned char getMode();

// take a single analog reading of the specified channel

static unsigned int read(unsigned char channel);

// take 'sample' readings of the specified channel and return the average

static unsigned int readAverage(unsigned char channel,

unsigned int samples);

// returns the position of the trimpot (20 readings averaged together).

// The trimpot is on ADC channel 7

static unsigned int readTrimpot();

// returns the output of the LV-168's temperature sensor in tenths of a

// degree F or C (20 readings averaged together). The temperature sensor

// is on ADC channel 6.

static int readTemperatureF();

static int readTemperatureC();

// the following methods can be used to initiate an ADC conversion

// that runs in the background, allowing the CPU to perform other tasks

// while the conversion is in progress. The procedure is to start a

// conversion on a channel with startConversion(channel), and then

// poll isConverting in your main loop. Once isConverting() returns

// a zero, the result can be obtained through a call to conversionResult().

static void startConversion(unsigned char channel);

// returns 1 if the ADC is in the middle of an conversion, otherwise

// returns 0

static unsigned char isConverting();

// returns the result of the previous ADC conversion.

static unsigned int conversionResult();

// converts the specified ADC result to millivolts

static unsigned int toMillivolts(unsigned int adcResult);

// 3pi: returns the voltage of the battery in millivolts,

// using 10 averaged samples.

static unsigned int readBatteryMillivolts\_3pi();

// SV-168: returns the voltage of the battery in millivolts,

// using 10 averaged samples.

static unsigned int readBatteryMillivolts\_SV168();

// This version of the function is included because the 3pi was

// originally the only supported board with battery voltage

// sensing. Instead of using this one, reading the battery

// voltage should be done with the board-specific functions above.

static inline unsigned int readBatteryMillivolts()

{

return readBatteryMillivolts\_3pi();

}

};

#endif

// Local Variables: \*\*

// mode: C++ \*\*

// c-basic-offset: 4 \*\*

// tab-width: 4 \*\*

// indent-tabs-mode: t \*\*

// end: \*\*