/\*

\* 3pi-mazesolver - demo code for the Pololu 3pi Robot

\*

\* This code will solve a line maze constructed with a black line on a

\* white background, as long as there are no loops. It has two

\* phases: first, it learns the maze, with a "left hand on the wall"

\* strategy, and computes the most efficient path to the finish.

\* Second, it follows its most efficient solution.

\*

\* http://www.pololu.com/docs/0J21

\* http://www.pololu.com

\* http://forum.pololu.com

\*

\*/

// The 3pi include file must be at the beginning of any program that

// uses the Pololu AVR library and 3pi.

#include <pololu/3pi.h>

// This include file allows data to be stored in program space. The

// ATmega168 has 16k of program space compared to 1k of RAM, so large

// pieces of static data should be stored in program space.

#include <avr/pgmspace.h>

#include "bargraph.h"

#include "maze-solve.h"

// Introductory messages. The "PROGMEM" identifier causes the data to

// go into program space.

const char welcome\_line1[] PROGMEM = "Project2";

const char welcome\_line2[] PROGMEM = "3\xf7 Robot";

const char demo\_name\_line1[] PROGMEM = "STOP &";

const char demo\_name\_line2[] PROGMEM = "GO!";

// A couple of simple tunes, stored in program space.

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

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

// Initializes the 3pi, displays a welcome message, calibrates, and

// plays the initial music.

void initialize()

{

unsigned int counter; // used as a simple timer

unsigned int sensors[5]; // an array to hold sensor values

// This must be called at the beginning of 3pi code, to set up the

// sensors. We use a value of 2000 for the timeout, which

// corresponds to 2000\*0.4 us = 0.8 ms on our 20 MHz processor.

pololu\_3pi\_init(2000);

load\_custom\_characters(); // load the custom characters

// Play welcome music and display a message

print\_from\_program\_space(welcome\_line1);

lcd\_goto\_xy(0,1);

print\_from\_program\_space(welcome\_line2);

play\_from\_program\_space(welcome);

delay\_ms(1000);

clear();

print\_from\_program\_space(demo\_name\_line1);

lcd\_goto\_xy(0,1);

print\_from\_program\_space(demo\_name\_line2);

delay\_ms(1000);

// Display battery voltage and wait for button press

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

{

int bat = read\_battery\_millivolts();

clear();

print\_long(bat);

print("mV");

lcd\_goto\_xy(0,1);

print("Press B");

delay\_ms(100);

}

// Always wait for the button to be released so that 3pi doesn't

// start moving until your hand is away from it.

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

delay\_ms(1000);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

set\_motors(40,40);

delay\_ms(250);

// Auto-calibration: turn right and left while calibrating the

// sensors.

for(counter=0;counter<80;counter++)

{

if(counter < 20 || counter >= 60)

{

set\_motors(40,-40);

}

else

{

set\_motors(-40,40);

}

// This function records a set of sensor readings and keeps

// track of the minimum and maximum values encountered. The

// IR\_EMITTERS\_ON argument means that the IR LEDs will be

// turned on during the reading, which is usually what you

// want.

calibrate\_line\_sensors(IR\_EMITTERS\_ON);

// Since our counter runs to 80, the total delay will be

// 80\*20 = 1600 ms.

delay\_ms(20);

}

set\_motors(0,0);

/\*/ Display calibrated values as a bar graph.

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

{

// Read the sensor values and get the position measurement.

unsigned int position = read\_line(sensors,IR\_EMITTERS\_ON);

// Display the position measurement, which will go from 0

// (when the leftmost sensor is over the line) to 4000 (when

// the rightmost sensor is over the line) on the 3pi, along

// with a bar graph of the sensor readings. This allows you

// to make sure the robot is ready to go.

clear();

print\_long(position);

lcd\_goto\_xy(0,1);

display\_readings(sensors);

delay\_ms(100);

}

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

clear();

print("Go!");

// Play music and wait for it to finish before we start driving.

play\_from\_program\_space(go);

while(is\_playing());

}

// This is the main function, where the code starts. All C programs

// must have a main() function defined somewhere.

int main()

{

// set up the 3pi

initialize();

// Call our maze solving routine.

maze\_solve();

// This part of the code is never reached. A robot should

// never reach the end of its program, or unpredictable behavior

// will result as random code starts getting executed. If you

// really want to stop all actions at some point, set your motors

// to 0,0 and run the following command to loop forever:

while(1);

}

// Local Variables: \*\*

// mode: C \*\*

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

// tab-width: 4 \*\*

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

// end: \*\*