/\*

\* follow\_segment.c

\*

\* This file just contains the follow\_segment() function, which causes

\* 3pi to follow a segment of the maze until it detects an

\* intersection, a dead end, or the finish.

\*

\*/

#include <pololu/3pi.h>

void follow\_segment()

{

int last\_proportional = 0;

long integral=0;

while(1)

{

// Normally, we will be following a line. The code below is

// similar to the 3pi-linefollower-pid example, but the maximum

// speed is turned down to 60 for reliability.

// Get the position of the line.

unsigned int sensors[5];

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

// The "proportional" term should be 0 when we are on the line.

int proportional = ((int)position) - 2000;

// Compute the derivative (change) and integral (sum) of the

// position.

int derivative = proportional - last\_proportional;

integral += proportional;

// Remember the last position.

last\_proportional = proportional;

// Compute the difference between the two motor power settings,

// m1 - m2. If this is a positive number the robot will turn

// to the left. If it is a negative number, the robot will

// turn to the right, and the magnitude of the number determines

// the sharpness of the turn.

int power\_difference = proportional/20 + integral/10000 + derivative\*3/2;

// Compute the actual motor settings. We never set either motor

// to a negative value.

const int max = 60; // the maximum speed

if(power\_difference > max)

power\_difference = max;

if(power\_difference < -max)

power\_difference = -max;

if(power\_difference < 0)

set\_motors(max+power\_difference,max);

else

set\_motors(max,max-power\_difference);

// We use the inner three sensors (1, 2, and 3) for

// determining whether there is a line straight ahead, and the

// sensors 0 and 4 for detecting lines going to the left and

// right.

if(sensors[1] < 100 && sensors[2] < 100 && sensors[3] < 100)

{

// There is no line visible ahead, and we didn't see any

// intersection. Must be a dead end.

set\_motors(0,0);

clear();

print("STOP!");

delay\_ms(3000);

while (sensors[1] < 100 && sensors[2] < 100 && sensors[3] < 100)

{

set\_motors(100,100);

if (sensors[1] > 100 && sensors[2] > 100 && sensors[3] > 100)

{

if(power\_difference > max)

power\_difference = max;

if(power\_difference < -max)

power\_difference = -max;

if(power\_difference < 0)

set\_motors(max+power\_difference,max);

else

set\_motors(max,max-power\_difference);

}

}

if(sensors[1] < 100 && sensors[2] < 100 && sensors[3] < 100)

{

while (sensors[1] < 100 && sensors[2] < 100 && sensors[3] < 100)

{

set\_motors(100,100);

if (sensors[1] > 100 && sensors[2] > 100 && sensors[3] > 100)

{

const int max = 60; // the maximum speed

if(power\_difference > max)

power\_difference = max;

if(power\_difference < -max)

power\_difference = -max;

if(power\_difference < 0)

set\_motors(max+power\_difference,max);

else

set\_motors(max,max-power\_difference);

}

}

}

}

else if(sensors[0] > 200 || sensors[4] > 200)

{

// Found an intersection.

//\*\*\*\*\*\*\*\*\*\*\*\*\*stop and go\*\*\*\*\*\*\*\*\*\*\*

set\_motors(0,0);

clear();

print("Junction!");

delay\_ms(1000);

do

{

set\_motors(100,100);

} while (sensors[0] > 200 || sensors[4] > 200);

}

else if (sensors[1] >200 && sensors[2] >200 && sensors[3] > 200 && sensors[0] > 200 || sensors[4] > 200)

{

set\_motors(0,0);

clear();

print("Finish!");

delay\_ms(1000);

}

}

}

// Local Variables: \*\*

// mode: C \*\*

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

// tab-width: 4 \*\*

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

// end: \*\*