```
#include "mbed.h"

#include "m3pimaze.h"


BusOut leds(LED1,LED2,LED3,LED4);

m3pi m3pi(p23,p9,p10);


#define MAX 0.5

#define MIN 0




#define P_TERM 1

#define I_TERM 0

#define D_TERM 20


// Global variables

// The path array stores the route info. Each element shows what we did at an intersection


//  'L' for left

//  'R' for right

//  'F' for forward

//  'B' for back

//

 int foundjunction=0;

char path[100] = "";

unsigned char path_length = 0; // the length of the path so far


void doturn(unsigned char dir)
```

```
{
   if (dir=='L')
      {m3pi.left(0.25);wait(0.28);}
    else if(dir=='R')
     {m3pi.right(0.25);wait(0.28);}
    else if(dir=='F')
     {m3pi.forward(0.3);wait(0.15);}
    else if(dir=='B')
     {m3pi.right(0.25);wait(0.6);}

      m3pi.forward(0.1);wait(0.1);m3pi.forward(0);
       return;
}



void follow_segment()
{

    float right;
   float left;
   float position_of_line = 0.0;
   float prev_pos_of_line = 0.0;
   float derivative,proportional;
   float integral = 0;
   float power;
   float speed = MAX;
   int sensors[5];
```

```
while (1) {

    // Get the position of the line.
    position_of_line = m3pi.line_position();
    proportional = position_of_line;
    // Compute the derivative
    derivative = position_of_line - prev_pos_of_line;
    // Compute the integral
    integral += proportional;
     // Remember the last position.
    prev_pos_of_line = position_of_line;
    // Compute
    power = (proportional * (P_TERM) ) + (integral*(I_TERM)) + (derivative*(D_TERM)) ;

    //    Compute new speeds
    right = speed+power;
    left  = speed-power;
    // limit checks
    if (right < MIN)
        right = MIN;
    else if (right > MAX)
        right = MAX;

    if (left < MIN)
        left = MIN;
```

```
        else if (left > MAX)
            left = MAX;

       // set speed
        m3pi.left_motor(left);
        m3pi.right_motor(right);

    // Next, we are going to use the sensors to look for whether there is still a line ahead
       // and try to detect dead ends and possible left or right turns.
       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.
            foundjunction=1;
        }
        else if(sensors[0] > 200 || sensors[4] > 200)
        {
            // Found an intersection.
            foundjunction=1;
        }
       }
}

// This function decides which way to turn during the learning phase of
// maze solving.  It uses the variables found_left, found_straight, and
// found_right, which indicate whether there is an exit in each of the
// three directions, applying the "left hand on the wall" strategy.
```

```
char turn(unsigned char found_left, unsigned char found_forward, unsigned char found_right)

{

   // The order of the statements in this "if" is sufficient to implement a follow left-hand wall algorithm

   if(found_left)

      return 'L';

   else if(found_forward)

      return 'F';

   else if(found_right)

      return 'R';

   else

      return 'B';

}


void maze_solve()

{

    unsigned char found_left=0;

      unsigned char found_forward=0;

      unsigned char found_right=0;

      int sensors[5];


      while(1)

      {

      follow_segment();

       // Inch forward a bit.  This helps us in case we entered the

      // intersection at an angle.

      m3pi.forward(0.5);
```

```
wait(0.5);
 // These variables record whether the robot has seen a line to the
// left, straight ahead, and right, while examining the current
// intersection.



  m3pi.readsensor(sensors);


   if(sensors[0] > 100)
   found_left = 1;


else if(sensors[4] > 100)
   found_right = 1;


   m3pi.forward(0.4);
wait(0.02);


 m3pi.readsensor(sensors);


 if(sensors[1] > 200 || sensors[2] > 200 || sensors[3] > 200)
   found_forward = 1;


   if(sensors[0]>600 && sensors[1] > 600 && sensors[2] > 600 && sensors[3] > 600 &&
sensors[4]>600)
   break;



unsigned char dir = turn(found_left, found_forward, found_right);
```

```
// Make the turn indicated by the path.
doturn(dir);

// Store the intersection in the path variable.
path[path_length] = dir;
path_length ++;


 // Simplify the learned path.
simplify_path();


}

while(1)

{

int i;
for(i=0;i<path_length;i++)
{
 follow_segment();

m3pi.forward(0.5);
wait(0.5);
```

```
        m3pi.forward(0.4);
        wait(0.2);

        doturn(path[i]);
        }

          }
         }

void simplify_path()

{

 // only simplify the path if the second-to-last turn was a 'B'
    if(path_length < 3 || path[path_length-2] != 'B')
        return;

         int total_angle = 0;
    int i;
    for(i=1;i<=3;i++)


     {
        switch(path[path_length-i])
        {
        case 'R':
```

```
        total_angle += 90;
        break;
    case 'L':
        total_angle += 270;
        break;
    case 'B':
        total_angle += 180;
        break;
    }
}


 // Get the angle as a number between 0 and 360 degrees.
total_angle = total_angle % 360;



// Replace all of those turns with a single one.
switch(total_angle)
{
case 0:
    path[path_length - 3] = 'F';
    break;
case 90:
    path[path_length - 3] = 'R';
    break;
case 180:
```

```
        path[path_length - 3] = 'B';
        break;
    case 270:
        path[path_length - 3] = 'L';
        break;
    }

    // The path is now two steps shorter.
    path_length -= 2;
}



int main() {
 //  int sensors[5];

    m3pi.locate(0,1);
    wait(2.0);
    m3pi.sensor_auto_calibrate();
    m3pi.printf("MazeSolve");
 maze_solve();
}
```