Line Follower Work Sheet

This program is interesting because rather than running through a set of instructions, it actually makes decisions based on outside stimuli. This is the beginning of “intelligent” programs! This program will intelligently follow a line, making decisions of which motors to turn on and how much based on where the line is.

In order to make this application, we need to change the program code. You are not expected to write it from scratch, just click on the “import program” button for the line follower main.c. Again it should appear in your compiler, where you can compile, download and run it. The program should be able to follow a line, but it may not be very good yet. It is your job to improve it.

Compile the program, download it to your robot and run it as you did before.
If program fails, you must spell something wrong, then double check your syntax.
Once succeed, you can try the robot on the track and get it run.
Notice that the robot follows the line but not that fast or well. To improve this, we need to understand each piece of the code. Here we will work through the program simply.

```
1 #include "mbed.h"
2 #include "m3pi.h"
3
4 BusOut leds(LED1, LED2, LED3, LED4);
5 m3pi m3pi(p23,p9,p10);
```

The first part of program, as in most programs, includes libraries and gets everything ready for the main program. We are not supposed to do anything with it, so just leave it as it is.

```
6
7 int main () {
8
9     m3pi.locate(0,1);
10    m3pi.printf("Line Flw");
11
12    wait(2.0);
13
14    float position_of_line = 0.0;
15    m3pi.sensor_auto_calibrate();
16    float speed = 0.4;
```

The above bit of programming starts the main program. Lines 9 and 10 print "Line Flw" to the LCD display. Change this to your group's name, remembering you can only have 8 characters on each line.

After printing the characters, the robot will wait for 2 seconds before doing anything else.
In line 14, we defined a variable called “position_of_line” with an initial value of zero. The word “float” defines the type of the variable. For now you only need to know that we set a name to represent a numeric value.

“M3pi.sensor_auto_calibrate();” calibrates the sensor values. This is a set-up function which does not require you to do anything.

After that, we set another variable called “speed”, with an initial value of 0.4.

```
18      while (1) {
19
20          // -1.0 is far left, 1.0 is far right
21          position_of_line = m3pi.line_position();
22
23          //line is more than 25% to the left
24          if (position_of_line < -0.5) {
25              m3pi.left_motor(speed);
26              m3pi.right_motor(speed - 0.3);
27              leds = 0x4;
28          }
```

The above piece of code is repeated indefinitely (the while (1) {} repeats the programming inside the curly brackets forever). You can see that this is necessary for the line follower, as it needs to keep adjusting to the line.

In the next line, the function “m3pi.line_position()” will return to a value to indicate the position of the robot. If the robot is far over to the left, it will return to ‘-1’, and if it is far over to the right, it will return to ‘1’. Likewise, it will return to ‘0’ if it is in the middle, and numbers in between if it is elsewhere. Remember that “position_of_line” was declared as a variable earlier? Well this is it being set as a variable which “saves” the input value, so the program can use it.
The next part of the code is an “if” statement. This means that if the condition in the curly brackets is fulfilled, the next part of the code will be carried out. The condition for this is “if the position_of_line is less than -0.5”. Try to visualise where the robot would be in this case.
Now look at the commands contained in the “if” statement. The speed of each motor is controlled separately here. This tells the left motor to go at the speed defined earlier (speed is another variable). The right motor goes at speed – 0.3, which will be slower. Think which way this will make the robot turn. Notice how it will turn towards the line.

The command leds = 0x4; will turn the third LED on, (the LEDs will count in binary).

```
29          //line is more than 75% to the right
30          else if (position_of_line > 0.5) {
31              m3pi.right_motor(speed);
32              m3pi.left_motor(speed - 0.3);
33              leds = 0x2;
34          }
35          else {
36              m3pi.forward(speed);
37              leds = 0x0;
38          }
39      }
40  }
```

The above piece of code is very similar, see if you can figure out what it does. All you need to know is that the else if means that this condition will only be looked at if the previous condition is not met.

The three curly brackets at the end may confuse you. The first is the end of the else statement. The second is the end of the while (1) statement, and the third is the end of int main(). Remember each chunk of code beginning with curly brackets must end with a curly bracket.

Sometimes the robots fail to follow the path and go off the edge. Try writing in a piece of program that will stop the robot if it can't see the line. Remember that position_of_line returns -1 or +1 if no line is detected.

The group that makes the robot go round the track fastest is the winner. Here are some tips:

1. First try changing the speed. Increasing this may rush out of the line however.
2. Try changing other values in the program, for example how sharply it will turn or how close to the line it will turn.
3. Use your imagination! Perhaps you want it to turn faster the further away it is from the line? How would you do that?

Now, improve your robot and be ready to compete with your opponents.