Line Following with ROBOTC

By Michael David Lawton

Introduction

Line following is perhaps the best way available to VEX Robotics teams to quickly and reliably get to a certain point usually situated at the end of the line.
This booklet will cover on the hardware side sensor positioning, patterns and height and on the software side PID loops, grey level sensing, finding the line, + junctions, and of course programming techniques using different numbers of sensors.
Line following is not hard, but it helps to have a good understanding of the underlying concepts of how it works. The aim of this booklet is to describe the basic concepts of line following and give examples of problems and their corresponding solutions in real world applications.

My Experience

The first line following I ever did was for 2921's 20 second autonomous and the programming challenge in the 2009 Vex Elevation NZ Regionals. I could not find any suitable example programmes so I made my own up. I used three sensors in the traditional format. Using black tape on the carpet of my room, I tested the programme. When I was able to use a field some simple reversing of the values (because I built the programme using black tape instead of white) made it work. The programme worked most the time during the tournament, but had a few troubles with the lighting on the fields and on one field, on the red side it wouldn't work at all. Searching on the Vex forum for answers, I found a post with a link to a page on line following for a micromouse. The concept was based on a PID routine, so if the robot was only slightly to the right of the line it would slightly slow down the left motor therefore following the line very smoothly and as it calculated the difference between the outputs of the sensors, it would work in any light including a direct spot lamp, completely dark or even when a light was switched on or off when the robot was half way down the line. With some adjustment, this concept worked very well, and 2921 went on to win the world title for the programming challenge as well as being one of better robots in regard to the 20 second autonomous. This booklet shares the results of hundreds of hours of my research and experimentation.

Section 1 – Hardware

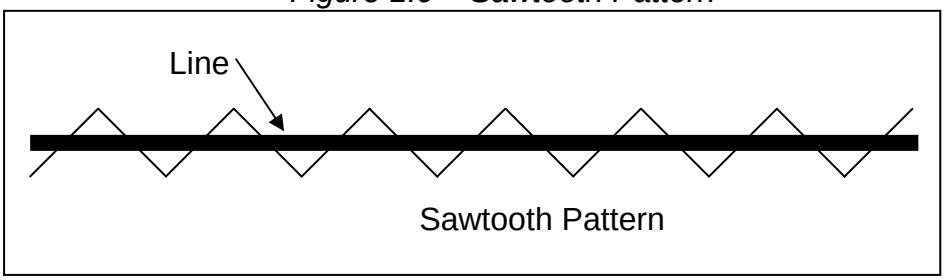
1.0 – Sensor Overview

The line following sensor consists of an infrared led and phototransistor mounted behind an infrared filter and encased in plastic, giving an analogue output of 0 to 1023. The sensor will give a low output when the surface is very reflective (e.g. white electrical tape) and a high output when the surface is non-reflecting (e.g. the grey field tiles). The coloured tiles output much the same value as the white tape.

1.1 – Positioning the Sensors

The position of the sensors in relation to the centre of the turning circle can make a big difference to the performance of the robot. If you imagine a string attached to a robot behind or up to the centre of the turning circle, when you pulled the string along a straight line the robot would veer off the line and would not recover. Attaching the string a little way forward of the centre, the robot would still veer away from the line, but would correct itself to a certain extent. As a last example, if you attached the string to the front of the robot it would travel straight without veering to either side. Putting the string further forward would increase this effect. The line following sensors will follow this pattern, being a lot more accurate without the sawtooth movement (see figure 1.0) exhibited when the sensors are mounted further back.

Figure 1.0 – Sawtooth Pattern



Mounting the sensors too far forward, however, will cause the robot to overshoot the line when it corrects itself. The sensors should always be mounted in the centre with respect to the x axis.
The optimal position is best found by experimentation as it will be affected by the size, weight, speed, power, turning circle etc of the robot. Grey level sensors (discussed later) should be mounted behind the main sensors, or if you have two, behind the main sensors to the left and right.

1.2 – Sensor Patterns

There are many sensor patterns that you can use, ranging from a single sensor, to large multisensor arrays with 9 or more sensors.
The most basic of these is a single sensor mounted on the robot. Using this design the robot will exhibit a large sawtooth pattern of movement, and accordingly will be very slow.
With this design the robot will also overshoot a lot.
The next level is to use two sensors side by side. This will reduce the sawtooth pattern to a certain extent, but the robot will still be prone to overshooting.
The usual design is to use three sensors. This provides a reasonable level of sawtooth and does not usually overshoot on a 1:1 geared robot with 4 in. wheels

turning at approximately 60% speed on the slower side. Any faster then this and the robot is likely to overshoot. To make it more accurate, you can put one sensor in front and put the two back sensors side by side as shown in figure 1.1. This has the disadvantage of causing the robot to overshoot more often.
Using four or more sensors the robot will seldom overshoot and this would be very good for a fast robot. Coupling this with the pattern shown in figure 1.1 creates a nice setup that is reasonably accurate and also will almost never overshoot (see figure 1.2).

Figure 1.1

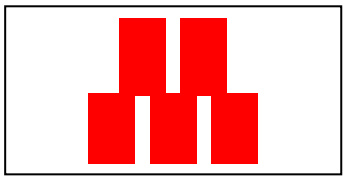
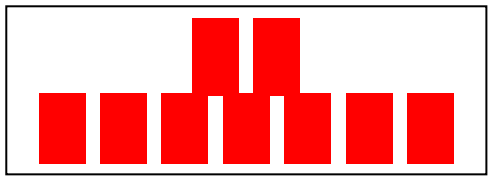


Figure 1.2



In short, keeping the sensors closer together makes it more accurate and reduces the sawtooth pattern and having more sensors or moving them further apart will stop the robot overshooting.

1.3 – Sensor Height

The optimal height (when there is the biggest difference between the values from the grey tiles and the white tape) for the sensor is 3mm, but it will work from 0.5mm to 25mm. When using PID loops (discussed in the next section), the optimal height is about 20mm. This is so that at least one sensor can partly see the line at all times (the sensing angle is about 45°).

Section 2 – Software

2.0 – Software Overview

The mounting and positioning of the sensors is the easy part. The software side, though not hard, can take a lot of time to adjust and fine tune to get it working exactly how you want it.
The hardware mounting method is very dependent on the type of programming you choose.
The software basically consists of a loop to check the sensors, perhaps do some calculations such as used with PID loops and then control the drive motors accordingly. Such a loop is best placed in another task so it can run at the same time as the robot does other tasks such as measuring distances or raising an arm. See listing 2.0 for an example.

Listing 2.0

```
1  task follow_line()
2  {
3  //line following code goes here
4  }
5
6  task main()//Main Programme
7  {
8  StartTask(follow_line);//Start the line following task
9  //The programme will run both tasks at the same time
10
11 wait1Msec(10000);//Wait ten seconds
12
13 StopTask(follow_line);//Stop following the line
14 }
```

The software will most likely also have to do other functions such as finding the line (driving forward or turning and then switching to line follow mode when it finds it), and possibly dealing with + junctions.
When doing any line following, the ambient lighting affects the readings so much that there is not one value that can be used for all fields. One solution is to take a reading from the grey tiles as soon as the game starts, but it poses problems, namely that the reflectance of the tiles is not constant, and the external lighting might change in the middle of the game. Also, the robot might have to start on a coloured tile In which case the readings would be wrong. The best solution is to use PID loops for the main function and grey level sensors for finding the line. These solutions are both discussed below.

2.1 – Grey Level Sensing

As mentioned above, the lack of consistent lighting creates a need for a more flexible sensing system. Grey level sensors are the best way to easily and reliably detect the line. Refer to figures 2.0 and 2.1 for the positioning of the sensors. When spinning (e.g. a 180° turn at the end of a line) it is best to use the sensor on the side that corresponds to the way you are turning as the main sensor and use the one on the opposite side as the grey level sensor (See figure 2.2).
The programming is simple: turn or move forward until one of the main sensors (usually the middle one, unless you are spinning) equals the output of the grey level sensor minus a predetermined value found by experimentation (with sensors mounted 20mm above the surface a value of 150 worked well for me). See listing 2.1. Do not set the value to low or the sensors will mistake the line down the join of the tiles or the small difference in reflectance between different ones for the tape.
If you put the grey sensors too far away from the main sensors, the lighting levels might be different, especially with spot lights, so they will not be so reliable.

Figure 2.0

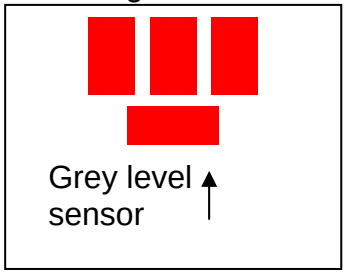


Figure 2.1

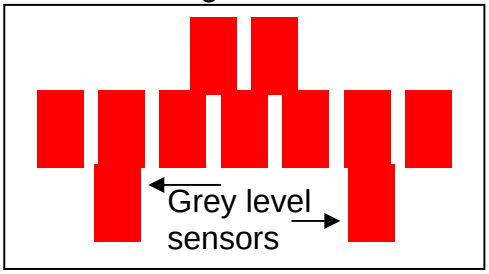
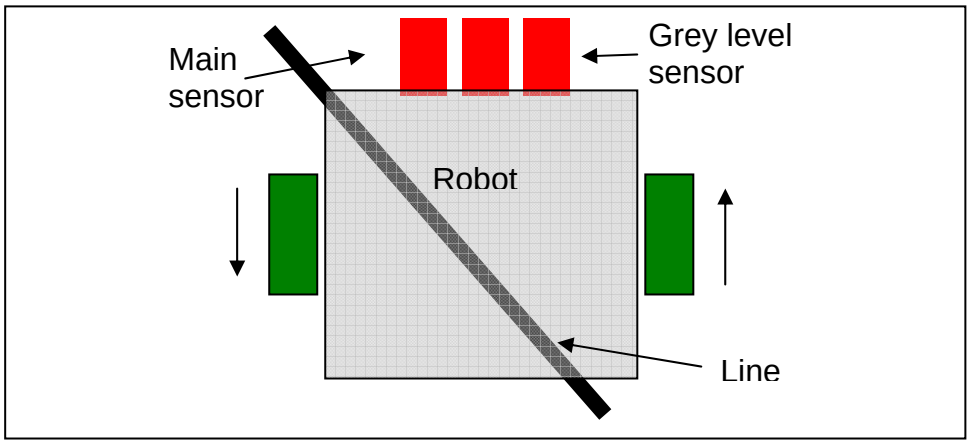


Figure 2.2



Listing 2.1

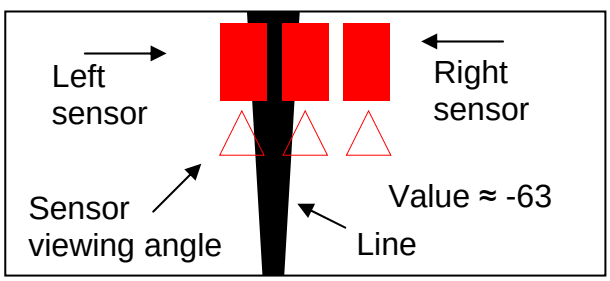
```
#pragma config(Sensor, in1,    line_sensor,         sensorLineFollower)
#pragma config(Sensor, in2,    grey_level_sensor,   sensorLineFollower)
#pragma config(Motor,  port1,           left,                tmotorNormal)
#pragma config(Motor,  port2,           right,               tmotorReversed)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

task main()
{

//Move Forward
motor[port1] = 127;
motor[port2] = 127;

//Wait until the robot finds the line
while(SensorValue[in1] > SensorValue[in2] - 150)
{
}

//Stop
motor[port1] = 0;
motor[port2] = 0;
}
```

2.2 – PID Loops

Most designs suffer from overshooting and inaccuracy, important factors when lining up with goals or other field objects. PID loops are the best solution to this problem, also bettered by the fact that the lighting does not affect them, even when the lighting changes in the middle of the game.

A PID loop takes an analogue value and give an analogue output, meaning that if you calculate a value from a few line sensors that will be -127 when the line is under the left sensor, 0 in the middle, and 127 on the right side (see figure 2.3), you can use a PID loop and the robot will make a small adjustment when the error is small, and a large adjustment when the error is large. On a good programme you will want the following settings:

Figure 2.3



- Maximum Turn Speed
- Minimum Turn Speed
- Spin Speed
- Middle Threshold

The maximum turn speed ensures that the robot does not overshoot the line when correcting itself, while the minimum

turn speed is there to make sure that the
robot does actually turn (e.g. if the robot detected that it needed to turn only slightly left e.g. left motor only 5 units lower than the right motor, the robot probably would not turn).
The spin speed should be set low enough so that if the robot does overshoot, it can spin around the opposite way and not overshoot. It does not matter if it overshoots a little as long as at least one sensor is over the line so it can correct itself.
The middle threshold will set the minimum distance from the centre of the line before the robot corrects itself, a smaller value being more accurate and a larger value theoretically being faster as it can go full speed for longer before it will be made to turn. In practice a smaller value is often faster as the robot does not exhibit so much sawtooth movement.
When the line position is inside the threshold level, both motors will usually be on full speed.
An extra feature is to have another value that calculates the difference between the current value and the previously read value and takes this into consideration. A bigger value will mean that the robot is turning faster the wrong way so you will have to make a sharper turn.
Usual required subs will be for checking whether one or more sensors are on the line (see listing 2.2), a spin sub for if no sensors are on the line (you will need to create a variable so you know which way it was trying to turn when it lost the line)(see listing 2.3), a sub for reading the sensors and calculating the line value (-127 to 127)(see listing 2.4 for an example with three sensors), and a sub for turning (see listing 2.5).
Of course you will need the main programme (see listing 2.6) hopefully in a task, to look at the values and call the appropriate subs.
Note that the sensors should always be about 20mm high for this method.
If your sensor array is small, for example only two or three sensors wide, the robot will overshoot a lot at first making the robot very slow, and take a while to settle down. This effect is reduced if you find the line coming closer parallel to it (e.g. 20° rather than 60°).
PID loops can be complicated, but are very rewarding.

2.3 – Finding The Line

To find the line is a simple procedure, especially with a grey level sensor.
If you are travelling at right angles to the line, use the middle sensor, or better still, the average of the sensors (see listing 2.7) and move forward until the value equals the grey sensor minus your chosen value, or if you don't have a grey level sensor, a predetermined value equal to the value of the sensor when it is over the line plus a small value not so large that it detects false lines, but not to small that it does not detect the line. As the robot cannot start following a line when its relative angle is more then about 60°, this is usually used only to help the robot when using dead reckoning and let the robot know it has hit the line.
If you have a wide sensor array, the robot will turn a lot more smoothly when it hits the line, especially if you turn slightly as you drive towards the line.
When coming at the line from an angle, use the sensor closest to the line as the main sensor, and check its value against your grey sensor, furthest from the line or if you are coming from near to parallel to the line, you can use the main sensor furthest from the line as the grey sensor. The latter method is used when spinning.
When spinning to find the line it will not work if the centre of the turning circle is to far from the line. You can also use a predetermined value as in the example above.

The line finding speed can be reasonably fast, but if your programme is doing a lot of things at once, the robot might miss the line, or even if it does find the line it might overshoot it so that the sensors are completely off the line. This will usually only happen when you travel at right angles to the line. One solution is to find the line travelling at high speed, and then if it overshoots the line back up relatively slowly, to find it again. You can also try setting the priority of the task higher.
When you are intending to travel along the line the best practice is to start towards the line from an angle of around 45°, and then following it, making sure that if you have a overshoot-protect sub as described in section 2.5, that you set the default direction (the direction it turns when it hasn't started following the line and writing it's own last direction) to turn is for example, left if you are coming from the left of the line.

2.4 – Using a Single Sensor

Using a single sensor is the most basic of all line following techniques.
The usual way to do this is to turn left until the sensor detects the tile, turn right until it detects the tile again, then simply repeat the process.
While overshooting is not a problem as far as reliability is concerned as the sensor will simply turn back and find the line again, this method will exhibit a large amount of sawtooth movement, slowing the robot down considerably.
If you place the sensors at the optimum height, the reliability is not to bad, but if you place the sensors higher e.g. 5mm to 15mm, you can use a lower threshold value, which will make it more accurate, but not very reliable when the lighting varies.
It is possible to use PID loops for this method except that it does not compare the value to another sensor, only to a preset value, so that it will not respond to lighting changes so well, although it will probably work better than the traditional way.

2.5 – Using Two Sensors

Using a pair of sensors is reasonably accurate, since you can follow the edge of the line, instead of the middle.
This method is slightly less prone to overshooting than that of the previous section, but when the robot does overshoot you need to turn back the opposite way from that which it was turning when it overshot the line. This is best calculated by storing in a variable the last turn direction and then turning the opposite way from the value stored. PID can be used with this method, or you can detect the white line by using the sensor not on the line as a grey level sensor.
You can detect when it has overshot the line by comparing the two values. If they are very similar you have probably overshot the line.
The programming for two sensors simply turns left when the left sensor detects the line, and turns right when the right sensor detects the line. Make sure that the sensors are not to close together.
An alternative method is to put the sensors as close together as you can get them, and turn left when the right sensor detects the grey, then turn right when the left sensor detects the grey. This is different from the single sensor method in that it usually goes straight, and only turns when one of the above mentioned sensors detect the edge of the line.

2.6 – Using Three Sensors

Have a look at most line following robots and they will probably have three sensors. Three sensors are easily implemented, do not overshoot to often (or at all, providing the speed is low enough), and have a reasonable level of sawtooth. These can be used with the middle sensor in front of the back two (refer back to figure 1.2.0), to create a much more accurate line follower but has the disadvantage of requiring and overshoot-protect system.
This is made much like the above mentioned system except that you have to compare the middle sensor with the two outside sensors. If they are all similar then you can safely assume that you are not on the line. A better way is to calculate the average of the three sensors, and compare the value of each sensor to this (see listing 2.).

2.7 – Using 4+ Sensors

With four or more sensors, you can create a sensor pattern that will never overshoot, be very accurate, or both.
To use four sensors the programming is almost the same as the two-sensor design, but you can make the robot turn faster when an outside sensor is on the line, and slower when an inside sensor is on it. This will increase the accuracy, and also make it less likely to overshoot than the two-sensor design.
Using six or eight sensors etc, you can completely eliminate the overshoot-protect system, and have a lot of different speeds for when the line is under different sensors.
When using 5, 7 or 9 sensors etc, you can use similar programming to the three sensor design. Refer back to figure 1.1 for one type of pattern. The pattern shown will increase accuracy (not needed when using a PID loop).
If you are not using a PID loop, you can use a similar pattern to the previously mentioned one, but with more sensors on the sides, shown in figure 1.2. This will greatly reduce (or eliminate) overshooting. When using the traditional method you can have a lot of different turning speeds based on which sensor the line is under as described above.

2.8 – ╬ Junctions

In most games there is at least one ╬ Junction.
This can create problems for some types of line following, causing it to think that it has overshot the line (if it compares the values from the sensors).
If your robot travels fast enough, even if it starts to spin to find the line again, it will already be past the junction and will resume normal operation. If you are using a predetermined threshold for the line following, you can tell the robot to keep moving straight when all the sensors detect the tape.
One solution is to position two grey level sensors one on each side of the line, and then compare the average of the two outside main sensors to the average of the two grey level sensors, you can tell when there is a junction, and make the robot keep travelling straight until it does not detect the junction anymore.
If you need to go until you hit the junction, then turn 90°, it is better to stop before you get to the junction, and turn 45°, initiate the find line function, and then start following the other line. This will save time rather then going until the robot detects the junction, spinning, turning 45°, and then following the line. If you do choose the latter method, remember that this will not work if the turning circle is to close to the line sensors.

You could also drive until the robot detects the junction, drive forward a little more so that the centre of the turning circle is over the line, turn 45° and then use your “find line while turning” sub. Another way (though not so reliable) is to follow the line a certain distance so you stop before the junction, drive forward manually until you are past the junction and then resume your line following.

Conclusion

Hopefully these pages have given you an insight of some of the many designs and concepts used in line following. Using line following can take your autonomous to the next level.

Written By:

Michael David Lawton – Programmer
Free Range Robotics (Homeschoolers)
www.robotics.org.nz

2009 New Zealand Champions
2009 New Zealand Programming Skills Champions
2009 World Semifinalists
2009 World Robot Skills 3rd Place
2009 World Programming Skills Champions

Note: Code is untested and provided as is. Free Range Robotics will not take responsibility for any damages incurred to robots or accidents resulting from bugs or problems in the code.

