Line Following Robot Using Image Processing

300605198
ENGR101
Dr Howard Lukefahr, Arthur Robers
AVC project
5 June 2022

Abstract—In this report you will find the progress made for the AVC project to achieve the full criteria set. It will also give an in-depth description of the method used to complete this project

Keywords: line following robot, image processing

1. Introduction

1.1 Scope

This report will cover an in-depth analysis of the description and performance of overall program for the line following robot.

1.2 Motivation & aim

My aim for this project is to produce a fully functional line following robot that can be used and improved into real world application. The outcome of this program should theoretically be able to easily adapt to any environment set for it.

1.3 constraints

we have only had 4 2-hour lab times spread across the 4 weeks to finish this project.

2. Background

2.1 Software

The language used in this work is C++ a general-purpose programming language as this is one of the top energy and memory efficient language out there.

Rosetta Code Global Ranking	
Position	Language
1	C
2	Pascal
3	Ada
4	Rust
5	C++, Fortran
6	Chapel
7	OCaml, Go
8	Lisp
9	Haskell, JavaScript
10	Java
11	PHP
12	Lua, Ruby
13	Perl
14	Dart, Racket, Erlang
15	Python

Table 1: programming language efficiency table [1]

2.1.1 C++ Terms

- **Int:** positive or negative integer
- **Float:** decimal number
- **Double:** 8-byte fractional number with greater accuracy than float
- **Unsinged int:** positive number
- **Unsinged char:** single number that represents a character using ASCII with a range
- **Boolean:** a binary variable with one of two values, 0(false) 1(true)
- **Array:** a set of values with a pre-defined size
- **Vector:** a set of values with an adjustable size

2.1.2 Image Processing

Image processing involves processing images or altering an existing image in a desire manner. The first step is to convert a 24-bit color image to PPM (a readable format) it stores each pixel with 3 max values of 255. The routines for reading and writing an image file are as follow,

```
1.  int rows=0, columns=0;
2.  unsigned char *buf1, *buf2;
3.  char *in_name = “input_file.ppm”;
4.  char *out_name = “output_file.ppm”;
5.  buf1 = (unsigned char *) malloc ((3000000)
*size of
(unsigned char));
6.  creat_image_file (in_name, out_name);
7.  get_image_size (in_name, &columns,
&rows);
8.  the_image = allocate_imageArray (rows,
columns);
9.  read_image_array (in_name, imageArray);
10. /*call an image processing routines.*/
11. write_image_array (out_name, imageArray);
12. free_image_array (imageArray, rows);\
```

2.2 PID

PID controllers have the goal of finding the error in our system and setting it to 0. The PID controller is an instrument typically used for industrial control application to regulate temperature, flow, pressure, speed, and other process variables. In this

application we will be using it to control the direction of the robot to keep in the middle of the line. This allows the robot to smoothly go around d curves without going off the set path. The position error calculated by using a scalar product this scans the top row of black pixels in the middle.

2.2.1 Other Terms

- **Closed loop:** A loop that receives inputs that allows it to react in real time to
- **Function:** A chunk of code you can use over and over again

3.Method

3.1 Software

3.1.1 Overview

This application is split into three different algorithms as the robot needs to adapt to the different situations of the course. The code that can reused for each of the three sections are stored in a class which allows for faster development and testing. The overall program in a main closed loop when the code initialize it starts the main algorithm which checks the top pixels for red pixels if present it will execute the function for the next section of the course. The overview of the main closed loop is showing in Figure 1.

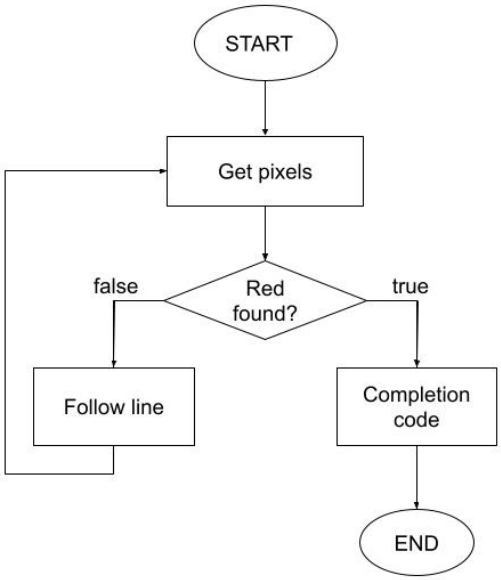

Fig 1: Overview of program

3.1.2 Section One

Section one is just curves this means we just need to use a PID control loop to get the robot to follow the line. Figure 2 demonstrates the PID loop for section one scanning the top row of the image if black pixels is present it is added into a vector else if there are no black pixels present the robot will move backwards until present. The array is then fed into the PID algorithm which returns the required motor speed for left and right motors. Algorithm 1 shows how the adjustments are calculated. This allows the robot to use the middle of the line to move around the curve until a red pixel is detected. We can find out if the pixel is red by seeing if half the redness is greater than green and blue in the pixel.

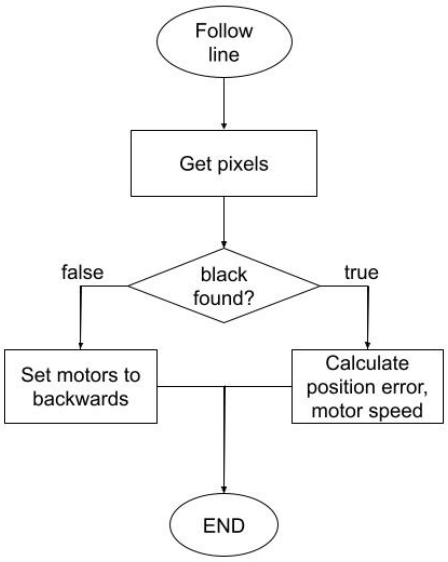

Fig 2: Follow line function

Algorithm 1: Follow line (calculate position error)

```
1: for pixel in pixel_in_row do
2:         if pixel is black then
3:                 black_pixels← true
4:         else
6:                 black_pixels← false
7:         end if
8: end for
9: middle ← middle(pixel_in_row)
10: p ← 0 – middle
11: for pixel in black_pixels do
12:         if true then
13:                 position error←p
14:         else
15:                 position error←0
16:         end if
17:         add one to p
18: end for
19: adjust ← kp * position error
20: move motors(adjust)
```

Array of indexes - middle:

-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6

Array of white pixels:

0	0	0	0	0	1	1	0	0	0	0	0	0

Table 2: Table for the inner_product [2]

3.1.3 Section Two

Section two is a bit different in terms of the course as we now must deal with intersection, by using maze theory and keeping to the left, if possible,

then right if possible. Figure 3 demonstrates this in practice the left, right and top rows are all scanned for black pixels it will then output the first true priority intersection to turn into. If the robot ever ends up at a dead end it will set the motors for a U-turn. We can find out if the pixel is blue by seeing if half the blue is greater than green and red in the pixel.

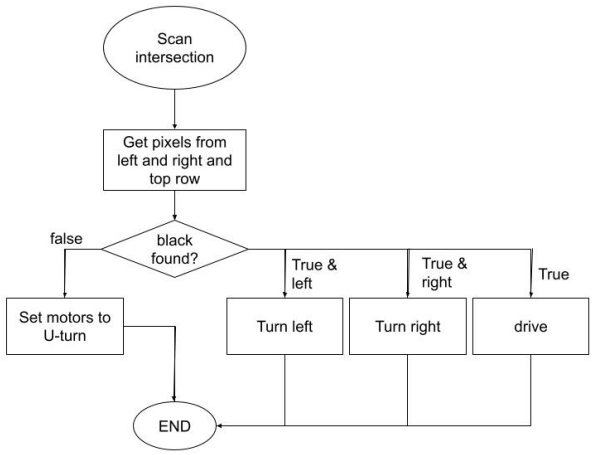

Fig 3: Scan intersection function

Algorithm 2: Scan intersection

```
1: for pixel in pixel_in_row do
2:         if pixel is black at top then
3:                 top_black_pixels← true
4:         else
5:                 top_black_pixels← false
6:         end if
7: for pixel in pixel_in_col do
8:         if pixel is black at left then
9:                 left_black_pixels← true
10:        else
11:                left_black_pixels← false
12:        end if
13:        if pixel is black at right then
14:                right_black_pixels← true
15:        else
16:                right_black_pixels← false
17:        end if
18: return left, right, top
```

3.1.4 Section three

Section three is very similar to Section two but requires a little bit of knowledge of the course layout. Figure 4 shows us the turns the robot would need to take to reach its goal using this we can hard code an algorithm to work for the section. We know that the robot needs to priorities the right intersection and turn left if there is no right present. We can find out if the pixel is green by seeing if half the blue is greater than blue and red in the pixel.

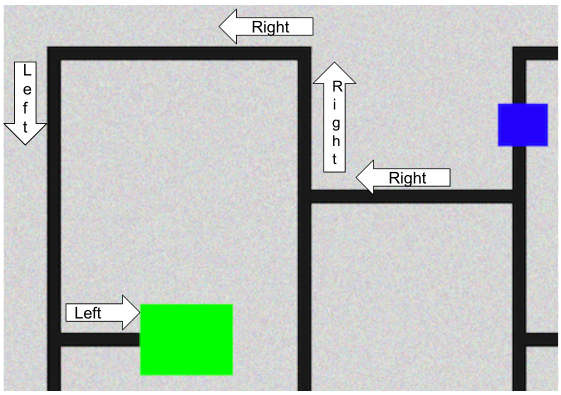

Fig 4: Section three layout

3.2 Testing

- **Section One:** when testing the robot for section one we need to test the value of kp to ensure our robot can turn at every type of sharpness curve and sticking to the middle of the line.
- **Section Two:** when testing the robot for section two we need to test the different types of Intersections and turning.
- **Section Three:** as Section three is very similar to section two the testing will be the same mostly just trialing our algorithm to see if it gets to the green box.

4. Results

4.1 Software

4.1.1 Section One

Section one had the calibration of the kp values for the robot to be able to drive around the curves smoothly without it driving off the path. The way we did this was through trial and error.

Test num	Kp value	Results
1	0.02	Undereats at corners
2	0.04	Undereats at corners
3	0.06	Undereats at most corners
4	0.08	able to do corners not smooth
5	0.1	able to do corners not smooth
6	0.12	able to do corners correctly
7	**0.14**	**able to do corners correctly**
8	0.16	able to do corners not smooth
9	0.18	overeats at some corners
10	0.2	overeats at some corners

Table 3: Kp calibration testing table

4.1.2 Section Two

Section two had an error we ran into when turning a intersection as the camera is in-front of the robot quite a few pixels this means the turn function would start before the middle of robots (the point it turns) would be a the intersection this means we had to calculate the distance in pixels from the

intersection the robot would have to move forward to get there.

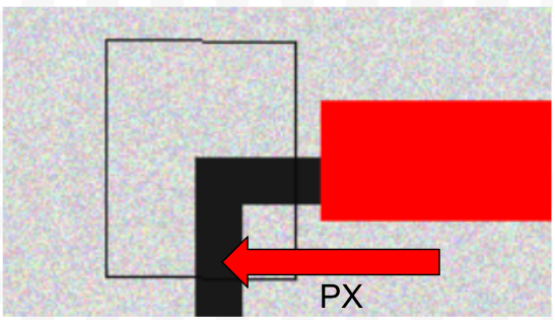

Fig 5: Camera + Intersection error

4.1.3 Section Three

Section three did as expected and since we fixed the turning error in section two there wasn't anything we need to bug fix. The robot is able to move through the whole maze without and errors every time.

5. Discussion

5.1 Software

The outcome of the software was successful for the project but for the implantation of the final product it is not fit for the real-world implantation as we hoped. This is due to the line following robot not being fully operational with some human input a real-world example for this application is heavy material transport, the problem with this software is it doesn't have a lot of secure programming implemented as it wasn't needed for the brief. The software is not suitable for the workplace environment as there are a lot of unknowns that aren't accounted for, if the black line was ever tampered with for any reason the software would not be able to overcome the problem. The kp calibration was also tested in a closed environment and would not adapt to the friction or slippiness of the factory.

6. Conclusion

In conclusion the line following robot that our team developed was able to meet all the criteria that was declared in the purpose. This was mainly achieved my splitting the code into multiple parts as it made each method much more efficient and accurate throughout development.

7. Future work

While this project achieved all three goals set it also shows there is a lot of improvement that could be implemented. One of the main improvements for section three is to have the robot remember where it has been, so it won't get stuck in a loop instead of needing to help it by hard coding an algorithm this is important in the real world uses as talked about in the discussion.

8. References

[1]Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. and Saraiva, J., 2022. *Ranking programming languages by energy efficiency*.

[2]Ecs.wgtn.ac.nz. 2022. *AVC Project video processing and movement control*. [online] Available at: <https://ecs.wgtn.ac.nz/foswiki/pub/Courses/ENGR101_2022T1/LectureSchedule/ENGR101_Lecture15.pdf> [Accessed 7 May 2022].

[3]Ieeexplore.ieee.org. 2022. *Line Follower Robot: Design and Hardware Application*. [online] Available at: <https://ieeexplore.ieee.org/document/9197968> [Accessed 1 June 2022].

Pawar, S., 2022. *Implementation of PPM Image Processing and Median Filtering*. [online] researchgate. Available at: <https://www.researchgate.net/publication/49607469_Implementation_of_PPM_Image_Processing_and_Median_Filtering> [Accessed 5 June 2022].