EEEE1039
Applied Electrical and Electronic Engineering: Construction Project

Advanced Line Following with Object Recognition and Reaction

Department of Electrical and Electronic Engineering
University of Nottingham Ningbo China

Student's Name: Jingxiong YU
Student's ID: 20123738
Group: 7
Tutors' Names: Dr. Jing WANG, Dr. Chunyang GU,
Dr. Sherif WELSEN, Dr. Fei CHEN
Submission Date: 14 June 2020

Abstract

In this report, the process of the development of a line patrol robot integrated with Raspberry Pi and its algorithm for advanced line following with object recognition and reaction were presented. A brief introduction of the robot and the algorithm and methods for hardware integration and algorithm development for advanced line following using optimised PID control and object recognition and reaction using advanced template matching algorithm were covered in this report. The main algorithms used in this report besides PID and template matching were basically OpenCV library based HSV colour space inRange colour filtering, perspective transformation and other image processing methods. The results and discussions for the methods were also covered in detail. The robot and the algorithm were successfully developed in the end which successfully challenged most of the tasks of this project. Further re-deployment was needed on the algorithm for object recognition and reaction.

Introduction

In the previous lab sessions in the last semester, a line patrol robot was successfully developed integrated with Arduino, SunFounder Line Follower module and PID control algorithm to realize basic line following. However, it was not enough for an automatic driverless robot to follow a line only, it also needed to be able to deal with various situations happened while following the line such as running into a red light or a T intersection. The robot should be able to recognize the situations (i.e. symbols) and react properly for each situation. In this lab session, the SCM (i.e. Single Chip Microcomputer) Arduino was replaced by a single board multicore computer Raspberry Pi which can multitask to realize advanced line following with object recognition and reaction.

In this project, the same line patrol robot as last semester was equipped with Raspberry Pi 3B, Pi Camera Board V2, servo motor, HC-SR04 ultrasound distance sensor and LCD 1602 and removed all other components that were not used in this project. The algorithm for advanced line following with object recognition and reaction integrated with OpenCV library and PID control algorithm was developed.

Relevant Background

As shown in Figure 1, OpenCV (Open Source Computer Vision Library) was an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. (https://opencv.org/about/, no date)

Figure 1. OpenCV

As shown in Figure 2, different from last semester, instead of using a SCM Arduino, the single board multicore computer Raspberry Pi 3 Model B was used in this semester. The Arduino used in last semester was a single board microcontroller capable of running a single task which was used for lightweight repetitive tasks, while the Raspberry Pi ran a full operating system such as Linux or Windows 10 IoT Core (Linux was used in this project) which was used for multiple heavy-weight and differing tasks. The technical specifications of Raspberry Pi 3 Model B were also shown in Figure 2.

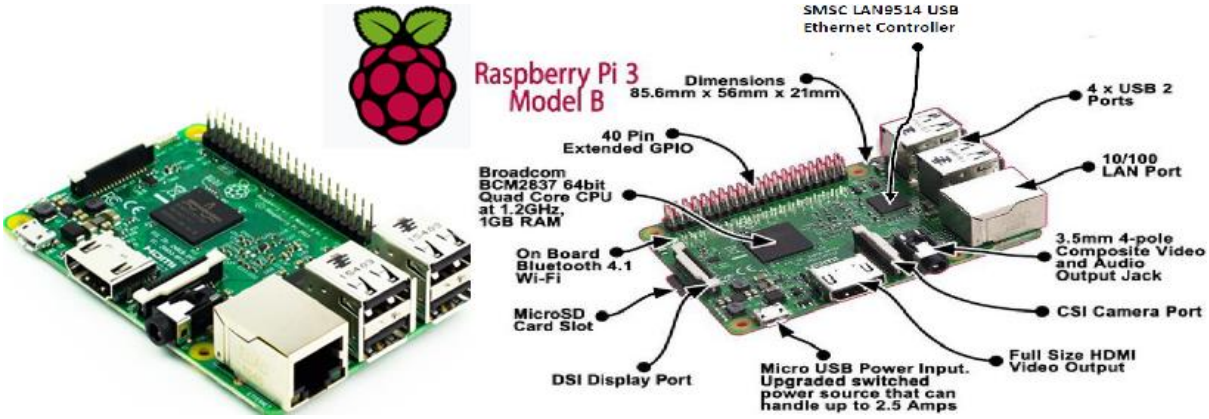

Figure 2. Raspberry Pi 3 Model B and its technical specifications

In the previous lab sessions, the SF-8CHIDTS SunFounder Line Follower module was used to detect the location of the black line. However, the SunFounder module could only detect the reflection rate of the surface which was only used to locate the black line with a low-level accuracy. In this project, the sensor had not only to detect the black line but also to distinguish different coloured short cuts and to recognize different symbols. In that case, the high-performance Raspberry Pi Camera Board V2 as shown in Figure 3 was used in this project to realize object recognition, object tracking and ultimate line following. The image processing and ultimately the resulting decisions that needed to be made to complete the tasks required more processing power and computational resources than the Arduino had, therefore, the Raspberry Pi took its place.

Figure 3. Raspberry Pi Camera Board V2

The Pi camera needed to be moved to certain positions in between the line following task and the symbol recognition task, therefore, the SG90 Micro Servo Motor as shown in Figure 4 was introduced and integrated to motorize the angle control for the Pi camera.

Figure 4. SG90 Micro Servo Motor

Another new hardware component integrated onto the robot was the HC-SR04 Ultrasound Distance Sensor as shown in Figure 5. It was used to measure the 2 cm – 400 cm non-contact distance between the robot and the target with 3 mm accuracy in the distance measurement demo task. The module consisted of ultrasonic transmitters, receiver, and control circuit.

Figure 5. HC-SR04 Ultrasound Distance Sensor

In this project, the robot was designed to be able to distinguish different colours in both line following on coloured short cuts and symbol recognition. Thus, the HSV colour space was introduced which made it easy to isolate colours. In HSV colour space, each pixel of an image was stored as colour data including Hue (i.e. colour tone), Saturation (i.e. intensity of the colour) and Value (i.e. brightness) as shown in Figure 6 below.

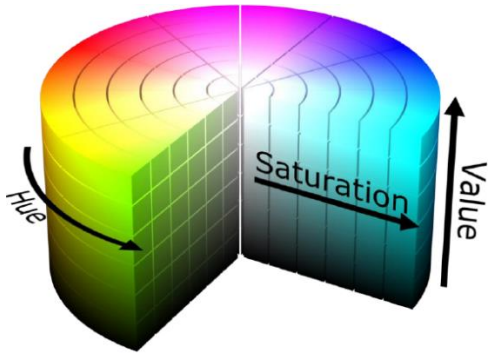

Figure 6. HSV colour space

Methods

Hardware Integration

At the beginning of this project, hardware integration was done. During the hardware integration, all hardware components that were unnecessary for advanced line following with object recognition and reaction in lab session 5 and 6 were removed from the robot, and the Raspberry Pi, Pi camera, servo motor, ultrasound distance sensor and LCD were integrated onto the robot. Pin allocations were as shown in Table 1.

Table 1. Pin allocations

Pin of component	Pin of Pi	BCM	GPIO (wPi)
LCD_RS	15	22	3
LCD_E	11	17	0
LCD_D4	22	25	6
LCD_D5	12	18	1
LCD_D6	18	24	5
LCD_D7	16	23	4
Data pin of servo	13	27	2
HC-SR04_TRIG	36	16	27
HC-SR04_ECHO	37	26	25
Reset pin of robot	7	4	7

As shown in the Table 1 above, the pins listed were necessary data pins to realize communication between Raspberry Pi and other hardware components. Others pins of hardware components were pins to provide power supply which were connected correspondingly to 5 V and GND, therefore, were not listed in the table above.

Level shifters or sets of resistors to realize similar objective were needed, because the signals from the HC-SR04 ultrasound distance sensor and also the baseboard were at 5 V, while signals from Pi GPIOs were at 3.3 V. If a signal which was basically an electric pulse higher than the maximum voltage the Pi was able to tolerate was sent to the receiving port on Pi, the port on Pi was possibly broken down. Therefore, the level shifter or a set of resistors was used to divide the voltage to realize level shifting between 5 V and 3.3 V. A level shifter module was directly used for communication between Pi and the baseboard, but for ultrasound distance sensor, a set of resistors was used. The wire connections of both level shifters were shown in Figure 7 and 8 correspondingly.

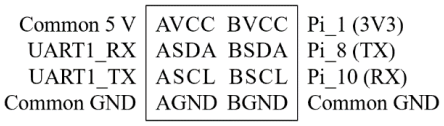

Figure 7. Wire connections of Level Shifter module

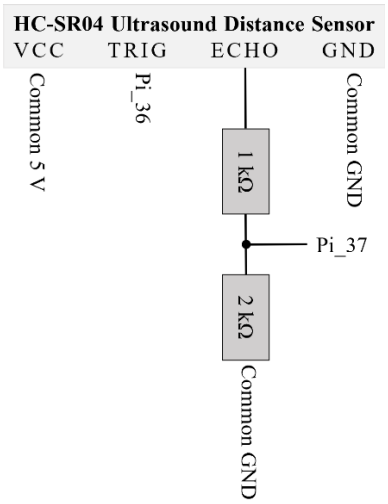

Figure 8. Wire connections of ultrasound distance sensor

Instead of lots of soldering work, in this project, the breadboard was used for wire connections to save time for debugging and software development. However, for USB socket used to power up the Raspberry Pi, soldering work was required. To power up the Raspberry Pi and make it fully functional, a current larger than 2.5 A and a voltage of 5 V were needed. There were two methods for powering up the Pi using the power supply output from the robot. One approach was to use the battery set integrated on the robot to power the Pi directly, another was to use the 5 V output pins from the baseboard. Both two approaches had pros and cons. If the battery set was used directly, the 2.5 A current requirement was satisfied, but the voltage of the battery set was 7.4 V or even higher if the batteries were over charged. Obviously, such a high voltage could not be applied onto the Pi directly, therefore, a DC-to-DC voltage converter was needed to reduce the voltage to 5 V. If the 5 V output pins on the baseboard were used, one set of 5 V and GND (i.e. ground) pins was only able to provide a current of approximately 0.5 A which was not large enough to make the Pi fully functional, therefore, at least 5 sets of 5 V and GND pins were needed to be connected in parallel to increase the output current without increasing the voltage to satisfy the 2.5 A rated current requirement of Pi. In this project, the latter was applied, and the soldering layout and wire connections were as shown in the Figure 9 below.

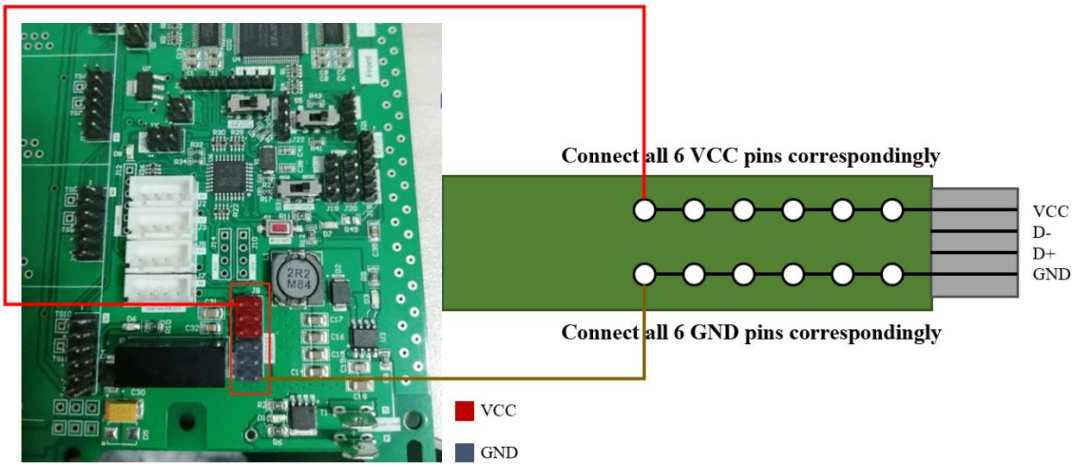

Figure 9. Soldering layout and wire connections of USB socket

Raspberry Pi GPIO

The Raspberry Pi was not only a small computer that was able to run full-featured Linux distributions while driving HDMI displays and processing mouse, keyboard, and camera inputs, but also a hardware prototyping tool. Like Arduino. the Pi had bi-directional I/O pins which were used to drive hardware components such as driving LCD displays, spinning servo motors and driving the line patrol robot. (https://learn.sparkfun.com/tutorials/raspberry-gpio, no date) It was the GPIO interface of Raspberry Pi supporting many of the hardware connections and communications. Unlike Arduino, the Raspberry Pi was able to be programmed in various programming languages such as C, C++, and Python. C and C++ (mainly C++, because C programmes were portable to C++ programmes) were used in this project. Also, 27 GPIO pins' voltage levels were 3.3 V and were not 5 V tolerant, therefore, the voltage levels had to be restricted to avoid breaking the Pi's port down.

There were two different pin-numbering schemes, one was to use the BCM pin-numbering scheme (i.e. the Broadcom GPIO pin numbers) and the another was to use wiringPi's simplified number system, but only one of them was to be used throughout the rest of the programme. To initialize the BCM pin-numbering scheme, the "wiringPiSetupGpio()" was to be called, while the "wiringPiSetup()" being called to initialize the simplified number system. The latter pin-numbering scheme was used in this project. After setting up the pin-numbering scheme, the rest such as pin mode declaration was exactly like programming in Arduino. One piece of essential code for wiringPi setup and pin mode declaration was shown in Figure 10.

```
1   #include<wiringPi.h>
2
3   void setup(void){
4       wiringPiSetup();
5       pinMode(7,OUTPUT);
6       return;
7   }
8
9   int main(){
10      setup();
11      digitalWrite(7,HIGH);
12      return 0;
13  }
```

Figure 10. Example code for wiringPi

The pin allocations for all hardware components' data pins and their corresponding BCM pin numbers and wiringPi simplified pin numbers were listed in the Table 1 above, but only the wiringPi simplified pin numbers were used to realize communication between Pi and other hardware components. To get the status and numbering systems of all GPIO pins, the "gpio readall" command was used in the terminal of Linux distribution, and the results returned were as shown in Figure 11 below.

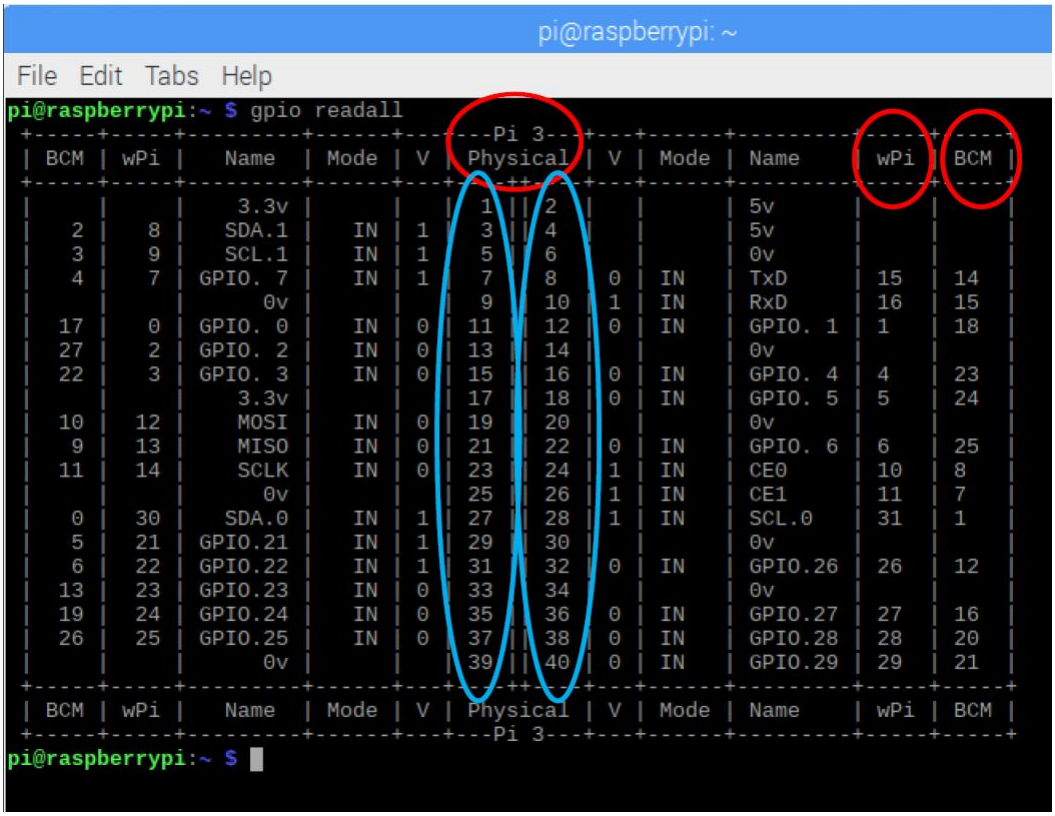

Figure 11. Returned results of "gpio readall" command in terminal

Enhanced PID Control

In previous lab sessions, a basic line following algorithm was developed based on Bang-Bang control and immature PID control algorithms. However, due to the instability of Bang-Bang control and the poor design for PID control, the line following was not smooth and accurate, also the speed had to be restricted under 20% of the maximum. In this project, an advanced line following algorithm based on enhanced PID control was developed and integrated onto the programme to realize automatic movement control for the robot.

In the previous version of PID control algorithm, there were lots of "if-else" conditional loops being applied to divide the line following into different situations and different PID control units with different parameters were then applied onto each situation to realize an immature PID control. The newly designed PID control algorithm enabled the robot to automatically modify the speed of its wheels based on the position differences it obtained and calculated to realize accurate and smooth line following without human interference. The essential pieces of code were as shown in the Figure 12 below.

As shown in the Figure 12 below, there were mainly 2 functions in the PID control algorithm including a function to calculate the PID output (i.e. the speed difference calculated based on the position difference input) and another function to apply the calculated speed difference onto the motors to realize automatic PID movement control for the line patrol robot.

```
void lineFollow_pid(double output){//basic line following using pid ctrl
    leftMotorSpeed=leftMotorBaseSpeed+output;//calc the modified motor speed
    rightMotorSpeed=rightMotorBaseSpeed-output;
    //apply new speed and direction to each motor
    if(leftMotorSpeed>=0&&rightMotorSpeed>=0){
        leftMotorSpeed=constrain(leftMotorSpeed,0,max_speed);rightMotorSpeed=constrain(rightMotorSpeed,0,max_speed);
        serialPrintf(robot,"#Bafffr0%d,0%d,0%d,0%d",(int)leftMotorSpeed,(int)leftMotorSpeed,(int)rightMotorSpeed,(int)rightMotorSpeed);delay(50);}
    else if(leftMotorSpeed<0&&rightMotorSpeed>=0){
        leftMotorSpeed=constrain(leftMotorSpeed,min_speed,0);rightMotorSpeed=constrain(rightMotorSpeed,0,max_speed);
        serialPrintf(robot,"#Barrff0%d,0%d,0%d,0%d",-(int)leftMotorSpeed,-(int)leftMotorSpeed,(int)rightMotorSpeed,(int)rightMotorSpeed);delay(50);}
    else if(leftMotorSpeed>=0&&rightMotorSpeed<0){
        leftMotorSpeed=constrain(leftMotorSpeed,0,max_speed);rightMotorSpeed=constrain(rightMotorSpeed,min_speed,0);
        serialPrintf(robot,"#Baffrr0%d,0%d,0%d,0%d",(int)leftMotorSpeed,(int)leftMotorSpeed,-(int)rightMotorSpeed,-(int)rightMotorSpeed);delay(50);}
    else if(leftMotorSpeed<0&&rightMotorSpeed<0){
        leftMotorSpeed=constrain(leftMotorSpeed,min_speed,0);rightMotorSpeed=constrain(rightMotorSpeed,min_speed,0);
        serialPrintf(robot,"#Barrrr0%d,0%d,0%d,0%d",-(int)leftMotorSpeed,-(int)leftMotorSpeed,-(int)rightMotorSpeed,-(int)rightMotorSpeed);delay(50);}
    return;
}
double pid(int pos){//calc the pid output
    //pid loop
    errorOld=pid_error;//save the old error for differential component
    pid_error=pos;//the error in pos
    errorSum+=pid_error;
    //calc the components of pid
    double proportional=pid_error*kp;
    double integral=errorSum*ki;
    double differential=(pid_error-errorOld)*kd;
    double output=proportional+integral+differential;//calc the result
    return output;
}
double constrain(double a,double b,double c){//constrain var a between val b & c
    if(a<b)return b;if(a>c)return c;return a;
}
```

Figure 12. Essential code for PID control

To call the PID functions, the position difference between the centre of the robot and the black line was calculated from the captured video frame ahead and input into the PID functions as a parameter to calculate the speed difference. The methods for calculating the position difference were discussed later along with the HSV colour space image filtering.

There was an additional enhancement in the newly designed PID control algorithm. Though the advanced PID control algorithm was able to conquer most of the corners with large degrees, when the robot overcame a right-angle corner, it was easily running off track. Hence, an enhancement which allowed the robot to run backward was developed. When the camera on the robot no longer detecting the black line, which meant the robot had already been off track, in that case, the robot was programmed to run backward to deal with this situation. Instead of simply programming the robot to go straight backward, the robot was designed to run backward with a direction. This was achieved by recoding the last position (i.e. left or right side) where the robot saw the black line and when the function was triggered, the robot was to run in the opposite direction. A flag was used to record the position (i.e. left or right side) where the robot saw a large angle corner when the robot was line following and the flag was reset and updated if the robot was detecting the black line. The essential pieces of the code for this enhancement were as shown in the Figure 13 below.

```
//backward movement control code in main funciton
if(flag_run&&is_inline)lineFollow_pid(output);//basic line following using pid output
else if(flag_run&&!is_inline){//run backward if off track
    if(flag_direction==1){serialPrintf(robot,"#Baffrr030,030,060,060");delay(50);}//left backward
    else if(flag_direction==2){serialPrintf(robot,"#Barrff060,060,030,030");delay(50);}//right backward
    else {serialPrintf(robot,"#Barrrr020,020,020,020");delay(50);}//straight backward
}
//reset and update the flag recording the black line's last location in the subfuction
if(count>0){is_inline=1;flag_direction=0;}else is_inline=0;//counter for black line pixels
int x_avg=(int)sum_x/count;//calculate black line pixels' average position
if(count>160){//if large degree corner detected
    if(x_avg>=(int)(img.cols/2))flag_direction=1;
    else flag_direction=2;
}
```

Figure 13. Essential code for backward movement control

HSV Colour Space In Range Colour Filtering

In this project, one of the most important methods was HSV colour space inRange colour filtering, because it was applied in several situations including distinguishing the black line from the environment, distinguishing different coloured short cuts, isolating the symbols from the captured video frames and so on. After the filtering, only the target colour was visible in the thresholded frame. The HSV inRange thresholds for isolating different colours were listed in the Table 2 below. And the essential codes for HSV inRange colour converting were as shown in the Figure 14 below.

Table 2. HSV inRange thresholds

Colour	Situation	H		S		V	
		Low	High	Low	High	Low	High
Black	Basic Line Following	0	179	0	255	0	66
Red	Short Cut	169	179	61	255	53	255
Yellow	Short Cut	17	28	35	255	35	255
Green	Short Cut	58	92	25	255	35	255
Blue	Short Cut	109	166	49	255	68	255
Pink	Block & Symbol	161	166	104	255	0	255

```
1  Mat frameHSV;//convert the frame to HSV and apply the limits
2  cvtColor(frame,frameHSV,COLOR_BGR2HSV);
3  int lowH=0,highH=179,lowS=0,highS=255,lowV=0,highV=66;
4  inRange(frameHSV,Scalar(lowH,lowS,lowV),Scalar(highH,highS,highV),frameHSV);
```

Figure 14. Essential code for HSV inRange colour converting

To obtain the HSV inRange thresholds for isolating different colours as shown in the Table 2 above, a GUI with 6 trackbars was used to modify limits of Hue, Saturation and Value directly via trackbars, and the converted frame was shown below the trackbars to show the immediate change so that it was convenient to find the sets of HSV limits within one try.

After filtering the original frame captured from the camera, the black line was distinguished from the environment, therefore, the position difference between the centre of the robot and the black line was then calculated. Three rows of pixels at the middle, one third and two thirds of the thresholded frame were used to determine the position of the black line. The black line was converted to white while the background being converted to black, thus, the white pixels' columns were added and then the sum of the white pixels' columns were divided by the total number of white pixels on the three rows to get the average white pixel column representing the black line's centre. And the black line's centre calculated was then compared to the centre of the frame to obtain the position difference between the black line and the centre of the robot. Essential codes were as shown in the Figure 15 below.

```
int getLoc(Mat& img){//get loc (error)
    //check for row pixels at middle, 2/3, 1/3
    int i=(int)img.rows/2,i1=(int)img.rows*2/3,i2=(int)img.rows*1/3;
    int count=0,sum_x;
    for(int x=0;x<img.cols;x++){
        uchar p=img.at<uchar>(i,x),p1=img.at<uchar>(i1,x),p2=img.at<uchar>(i2,x);
        int pixel=p,pixel1=p1,pixel2=p2;
        if(pixel>200){count++;sum_x+=x;}
        if(pixel1>200){count++;sum_x+=x;}
        if(pixel2>200){count++;sum_x+=x;}
    }
    if(count>0){is_inline=1;flag_direction=0;}else is_inline=0;
    if(!count)return 0;
    int x_avg=(int)sum_x/count;
    if(count>160){
        if(x_avg>=(int)(img.cols/2))flag_direction=1;
        else flag_direction=2;
    }
    return x_avg-(int)(img.cols/2);
}
```

Figure 15. Essential code for calculating error (i.e. position difference)

Advanced Template Matching Algorithm

The template matching algorithm was applied throughout this project which was used to realize object recognition. While the robot running on the track, the camera was not only detecting the black line but also the environment, therefore, if a pink block near the line indicating a task to be done was detected, the robot was to stop running and the camera on it was to be lifted up to identify the specific task symbols via template matching algorithm.

There were different OpenCV embedded functions to be called to realize template matching. As shown in the Figure 16 below, due to time limitation, the template was directly matched in the thresholded frames using “matchTemplate()” function to find the most similar area and “minMaxLoc()” function to get a similarity to be used to set the thresholds between different symbols. Also, both the template and the frames were filtered in HSV colour space.

```
cvtColor(imgRaw,imgRaw,COLOR_BGR2HSV);
inRange(imgRaw,Scalar(146,25,70),Scalar(167,255,132),imgRaw);
Mat temp1=imread("temp1.png");
cvtColor(temp1,temp1,COLOR_BGR2HSV);
inRange(temp1,Scalar(145,37,0),Scalar(179,255,115),temp1);
Mat similarity;
matchTemplate(imgRaw,temp1,similarity,CV_TM_CCORR_NORMED);
double maxVal;
minMaxLoc(similarity,0,&maxVal,0,0);
```

Figure 16. Essential code for template matching algorithm

However, the template matching algorithm introduced above was low efficient and of low accuracy, and the reason why was discussed in “Results and Discussions” part. An advanced template matching approach was to use “findContours()” function to find the corners of the symbol’s frame, and then use “transformPerspective()” function to correct the perspective of the image, and lastly use “compareImages()” function to compare the processed image with each of the symbol files to get a similarity which was then used to set the thresholds. The flow chart for this advanced template matching algorithm was shown in the Figure 17 below.

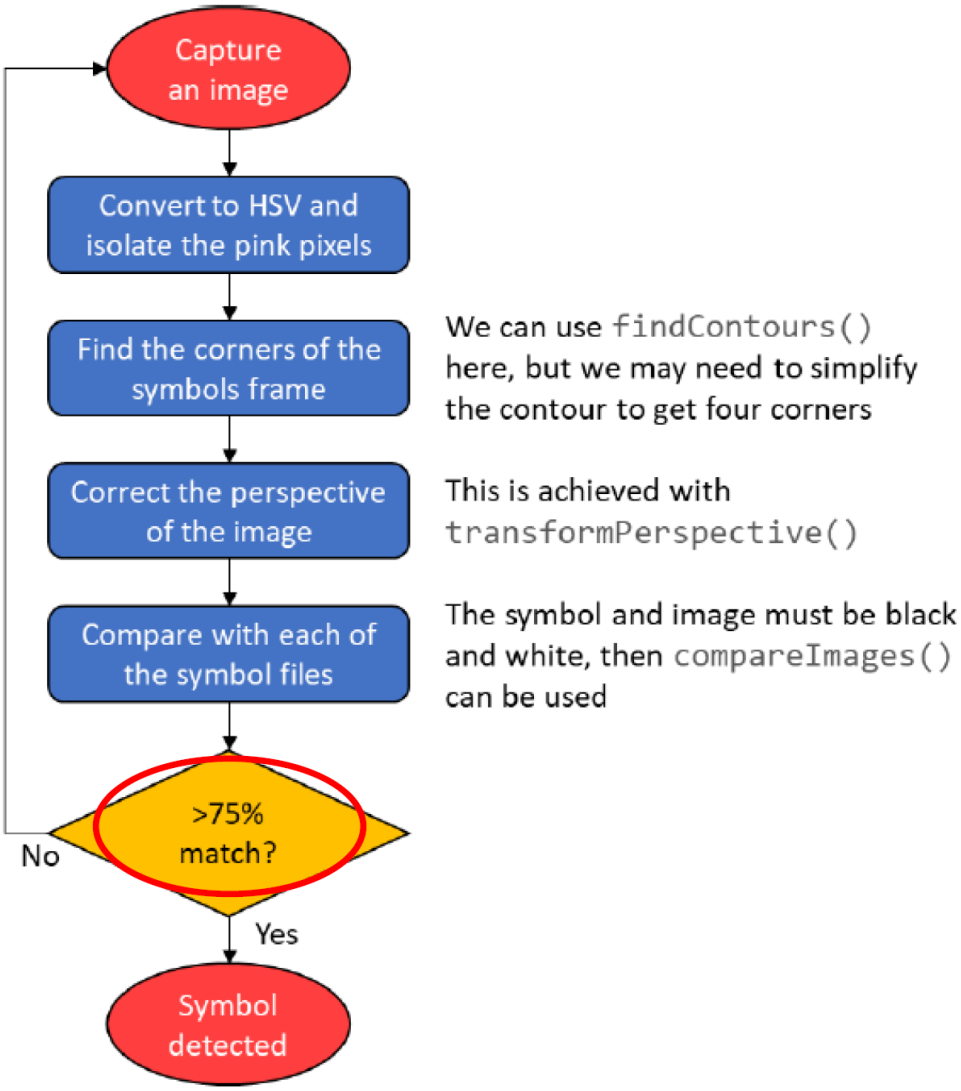

Figure 17. Flow chart for advanced template matching algorithm

Algorithms for Driving Hardware Components

In this project, several new hardware components were integrated onto the robot, also the operating system was changed to Linux, therefore, different algorithms were required for driving different hardware components. The algorithms for driving the robot (i.e. UART communication between the Pi and the baseboard), driving the LCD display, driving the servo motor, and driving the ultrasound distance sensor were covered in this part.

Since the line patrol robot was to recognize objects and reaction while line following, the communication between Pi and the baseboard was the most important part of this project which was studied, tested, and integrated onto the programme first. For serial communication, the UART serial communication method was applied and the UART1 interface on baseboard was used for wire connections and serial communication. As shown in the Figure 18 below, the ttyAMA0 was serving as GPIO serial port instead of the original Bluetooth port, because ttyAMA0 had better performance comparing with ttyS0.

```
1  //configure the serial port for the robot
2  int robot=serialOpen("/dev/ttyAMA0",57600);
3  serialPrintf(robot,"#Bafffff0%d,0%d,0%d,0%d",(int)speed,(int)speed,(int)speed,(int)speed);
4  delay(50);//a short delay allowing the robot to process cmd
5  serialPrintf(robot,"#Ha");delay(50);
6  serialClose(robot);//called to disable the serial port for robot
```

Figure 18. Essential code for UART communication between Pi and baseboard

Besides the basic line following, all other demo tasks required an LCD display to indicate the task that was undergoing and also display certain parameters such as the distance measured in the distance measurement task. Thus, the algorithms for driving the LCD display was developed secondly. The “lcdPuts()” was only able to put a string to the LCD, therefore, the “sprintf()” function under <cstring> header file was called to write integers to a string which was then be put to the LCD. Essential codes for setting up the LCD and displaying messages on LCD were shown in the Figure 19 below.

```
1  //initialise lcd
2  int lcd=lcdInit(2,16,4,lcd_rs,lcd_e,lcd_d4,lcd_d5,lcd_d6,lcd_d7,0,0,0,0);
3  lcdClear(lcd);//clear the display
4  char msg[16];
5  sprintf(msg,"Dist: %.2f cm",dist);
6  lcdPosition(lcd,0,1);//cursor: second line, first column
7  lcdPuts(lcd,msg);//print txt msg at current cursor pos
```

Figure 19. Essential code for LCD display

The servo motor was as important as the LCD display for demo tasks, because it was used to motorize the camera’s position which enabled the camera to look at different directions including looking down for basic line following and looking forward for symbol recognition. There were basically two methods for driving the servo motor including software PWM control methods and simply using “delay()” function to realize servo control. The latter was used in this project because the accuracy of the motorization was not required in this project. Essential codes for driving servo motor were as shown in the Figure 20 below.

Also as shown in the Figure 20, the “delayMicroseconds()” function was called instead of the “delay()”, because the integer was the only data type that was able to be input to the “delay()” function, which made 1.5 the same as 2 in which case the domain of the angles was narrowed, while the “delayMicroseconds()” operating float numbers avoiding the problem.

```
void setServoLoc(int mode){//set camera loc via servo
    switch(mode){
        case(0):{//default for line following
            int angle=80,i=0,k=180;float x=0;
            while(k--){x=11.11*i;
                digitalWrite(servo,HIGH);delayMicroseconds(500+x);
                digitalWrite(servo,LOW);delayMicroseconds(19500-x);
                if(i==angle)break;i++;
            }break;
        }
        case(1):{//only when pink blocks were detected
            //same as above except the angle was 20
        }
        default:break;
    }
    return;
}
```

Figure 20. Essential code for servo motor

In this distance measurement task, the distance between the ultrasound distance sensor and the symbol was measured and displayed on the LCD. Essential codes for setting up the ultrasound distance sensor and using it for distance measurement were shown in the Figure 21 below. The “timeval” structure and the “gettimeofday()” function under Linux embedded system header file <sys/time.h> were used for getting the system time at the start and the end to calculate the time interval.

```
#include<sys/time.h>
void getDist(void){//distance measurement using ultrasound sensor
    struct timeval start_time,stop_time;
    digitalWrite(TRIG,LOW);delay(10);
    digitalWrite(TRIG,HIGH);delayMicroseconds(10);
    digitalWrite(TRIG,LOW);
    while(!(digitalRead(ECHO)==1))gettimeofday(&start_time,NULL);
    while(!(digitalRead(ECHO)==0))gettimeofday(&stop_time,NULL);
    double start,stop;
    start=start_time.tv_sec*1000000+start_time.tv_usec;
    stop=stop_time.tv_sec*1000000+stop_time.tv_usec;
    double dist=(stop-start)/1000000*34000/2;
    //display distance on LCD
    return;
}
```

Figure 21. Essential code for ultrasound distance sensor

Object Oriented Programming

In this project, the robot was no longer programmed to do a single task repeatedly but to react correspondingly to the different tasks it detected and recognized. Thus, the OOP (i.e. Object-Oriented Programming) was applied. The flow chart for this OOP designed programme was shown in the Figure 22 below including algorithms that were not successfully developed.

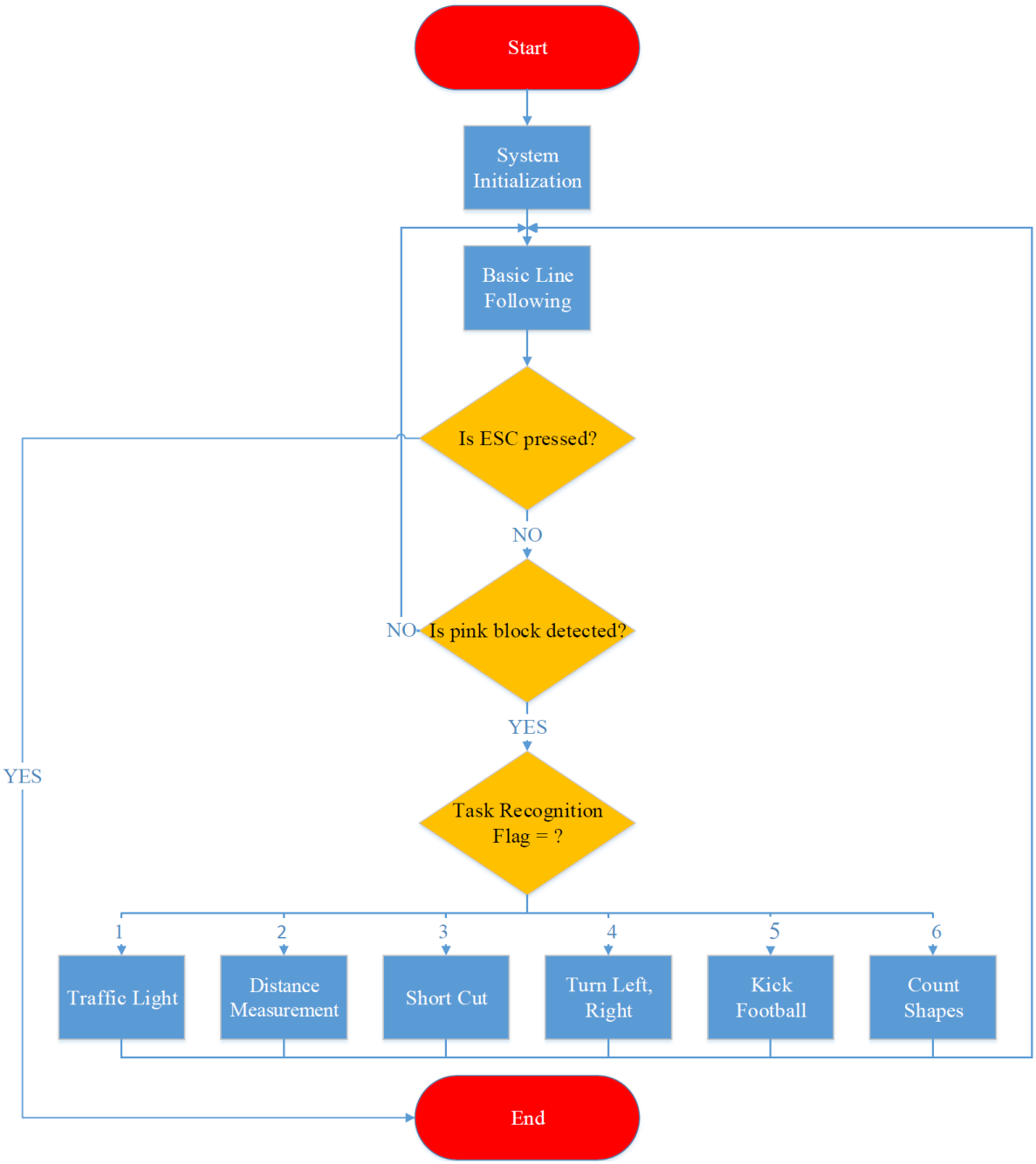

Figure 22. Flow chart for OOP designed programme

Demo Tasks

A brief introduction to demo tasks was covered in this part. In this project, there were basically 7 demo tasks to be challenged and another task which was to achieve multitask. The seven individual demo tasks were divided into two types, one type was line following related and another was template matching related. There were two demo tasks of the former type including basic line following on black line and line following on coloured short cuts which were studied, developed, and challenged first. The other 5 demo tasks including distance measurement, shape counting, traffic light, turning left or right, and football kicking were of the latter type which were studied, developed, challenged later.

The robot was programmed to realize basic line following using PID control first, then the robot was enhanced to be able to detect and distinguish different coloured short cuts and react differently correspondingly to different colours including normal speed on yellow short cut, higher speed on green short cut, lower speed on red short cut, and running between split blue short cuts. For other demo tasks, symbol recognition based on template matching algorithm was developed beforehand to enable the robot to be able to recognize the task symbols as shown in the Figure 23 below.

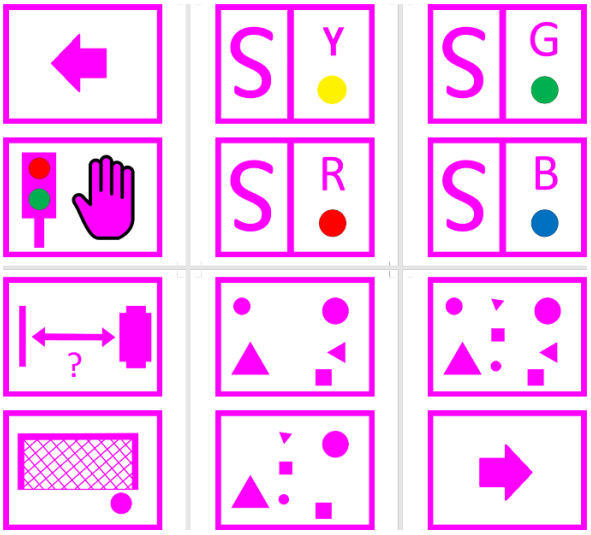

Figure 23. Task symbols

In the distance measurement task, the distance between the symbol and the robot was measured using ultrasound distance sensor and then displayed on LCD display. In the shape counting task, the numbers of each shape were obtained and then displayed on the LCD display. In the traffic light task, the robot was stopped when the red light on and continued line following when the green light on. In the left/right turning task, the robot was to react correspondingly to the symbol (i.e. turn to the direction indicated by the symbol) at the T intersection. In the football kicking task, after the task symbol being recognized, the robot was to rotate at the spot until it detected a green ball and then speed up to kick the ball into the goal, and the robot was able to run back onto track to continue line following. Each task was challenged individually first, but the multitask was not challenged due to time limitation.

Results and Discussions

Raspberry Pi Setup

The Raspberry Pi was set up first. The first lesson learnt from experiment in this lab session was that the micro SD card was so vulnerable that it could suddenly break down out of no reason. Actually, there was one SD card broke down during the mirror burning, and another broke down during regular operating. Therefore, all codes developed or modified on Pi were backed up to mobile hard drive in case that the SD card broke down and erased all data on it, and also, there was always another burned SD card for back up.

Also, at the very beginning of this lab session, the HDMI-HDMI wire was used for Pi display. Although it had quick response and was easily set up, it was noticed that because of the tenacity of the HDMI-HDMI wire, it was hard to debug or operate the robot while it was connecting to the display, and also, the robot was unable to be controlled remotely during the demo. Therefore, the VNC remote display was necessary to be set up despite the time wasted for establishing the communication between the VNC server on Pi and the VNC viewer on laptop after the Pi connected to the network automatically. With the VNC remote display, the robot and the Pi were able to be controlled anywhere by the laptop as long as the Pi and the laptop were connected to the Internet (better if connected under same local network) which gave the robot portability.

Hardware Integration and Debugging

Hardware components used to be soldered onto stripboard to make the entire system tight and neat, however, in this project, the breadboard was used instead for several connections such as LCD display. Because the soldering work always took a lot time while the software development was the core of this project in which case time was saved for software.

The greatest challenge came across in the first week's lab sessions was debugging for the integrated robot started from Tuesday afternoon and ended on Thursday afternoon which wasted a lot time. After the components were integrated onto the robot on Tuesday morning, the communication between the Raspberry Pi and the baseboard were tested, however, the robot was not responding to the commands sent from the Pi. Then the debugging for the integrated system started. When testing and debugging for the robot, the debugging procedure was divided into several small sections and then each of the sections was debugged to make sure each one of them was fully operational before continued debugging for the next section. In that case, the range where there might be problems was narrowed. During the debugging process, it was learnt that the robot was not supposed to be integrated and fixed before everything were debugged and tested fully operational, otherwise, it wasted a lot time taking everything that already integrated apart and reassembling them back together. The problem was eventually found that the commands gave to the baseboard via Pi inside the code were outdated, however, the original problem was possibly not only caused by this mistake, because other mistakes might have been fixed during the debugging process.

UART Serial Communication between Pi and Baseboard

As discussed above, the outdated commands were replaced with the latest in which case the robot was successfully driven. However, when several commands were sent continuously, the robot was not responding correctly as it was asked to. In further discovery, it was found that if the commands were sent all together, the port had not got enough time to process the commands it received before the buff was input a new command, therefore, the robot was not responding correctly to the commands. This was solved by adding a short delay for 50 microseconds after each command sent to the serial port to allow the port to process the command and the robot to react correctly to the command.

In the PID control algorithm, the speeds of the motors were not manually set but automatically calculated and set based on the position difference between the black line and the centre of the robot obtained and the PID algorithm, therefore, integers representing speeds needed to be input to the commands which were then sent to the baseboard. Initially, the command "serialPrintf(robot,"#Baffff%d%d%d%d",20,20,20,20);"" was used, however, the wheels were spinning at an unusual fast speed which was impossible to be 20% of the maximum speed set. Also, the robot was unable to be stopped. In further discovery, it was found that the commands received and processed by the port on the baseboard were basically strings, therefore, the 20% of maximum speed was "020" instead of the "20" sent to the baseboard initially. The "0" before the speed percentage was necessary because the string sent to the baseboard was stored in the buff, and if the uncomplete command was not identified by the serial port and remained in the buff, the newly input command was added to the previous command which made the port process the incorrect commands resulted in the misbehaviour of the robot. And also, the commas between speed percentages were missing which also resulted in confusing the serial port. Sample correct codes allowing the speed integers to be sent as commands to baseboard UART port were shown in Figure 24 below.

```
1   serialPrintf(robot,"#Bafffff0%d,0%d,0%d,0%d",(int)speed,(int)speed,(int)speed,(int)speed);
2   delay(50);
```

Figure 24. Sample codes for sending speed integers to baseboard

Basic Line Following

In this project, the line following not only based on the PID control but also the image processing methods because of the advanced sensor Pi camera introduced and integrated. The frames captured from the video captured by the Pi camera were processed first to generate position differences between black line and the centre of the robot which were then applied in the PID control to realize line following. There were different methods for image processing to threshold the captured frame to distinguish the black line from the environment. For example, using the binarization processing or HSV colour space inRange colour filtering to threshold the original frame to largely increase the contrast between the black line and its environment. The HSV space inRange image processing method was tested to be the most reliable and stable and the least influenced by the environmental lightening change.

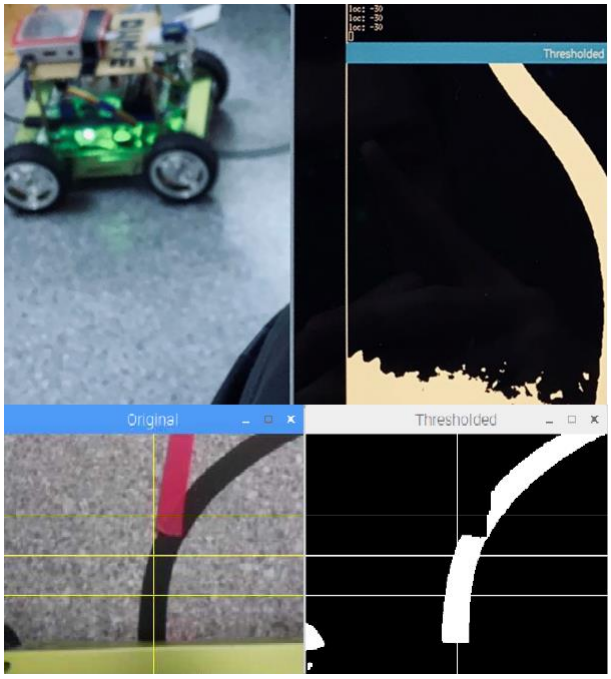

Figure 25. HSV inRange thresholded results for basic line following

As shown in the Figure 25 above, after the colour filtering, the black line was converted to white pixels while the rest converted to black pixels. Except for a few noise occurred around the edge of the robot caused by the baseboard's lightning, the thresholded frames satisfied the needs for separating the black line from the environment in the basic line following.

There were three different algorithms applied to get position differences between black line and the centre of the robot including counting and comparing total white pixels of left side of the frame and right side of the frame, finding contour of the black line and calculating its centre, and calculating the average column of white pixels on the row in the middle of the frame. The last method was applied for it requiring least computational resource while maintaining the accuracy at a relative high level. Initially, only one row of pixels was used, however, light spots on the ground due to environmental lightening made the black line more reflective which was then converted to black pixels during filtering which made the algorithm not fully functional. To solve this problem, other two rows of pixels at one third and two thirds of the frame were used to increase accuracy and made the algorithm more robust.

With a fully designed PID control algorithm integrated, the robot was successfully enhanced with smooth and accurate line following. Although the PID control algorithm was well designed enabling the robot to modify the speed of four wheels based on the position difference returned to stick to the black line without any human interference, also this algorithm was strong enough to conquer most of the corners with large degrees, it was not able to get the robot on the line coming across a right-angle corner. Because when coming across a right-angle corner, even though the corner was detected, the robot was unable to react in time before the camera was no longer above the black line and nothing was detected. So, an algorithm was developed to get the robot back to the black line by running backward every time the camera was no longer detecting the black line (i.e. the robot was off track),

and instead of simply running straight backward, the robot was programmed to go backward with a direction by remembering the side (left or right) it last saw a large degree corner was at and going backward in the opposite direction. By integrating this enhancement onto the PID control, not only did the robot be enabled to turn at right-angle corners but also if the robot were off track by mistake, it could always find its way back onto the track.

As for setting the parameters for PID control, Kp and Kd were used only. Kp was used for differentiating the speeds of the motors on two sides, and Kd was used to stable the robot to stick precisely to the track after the robot came across a corner and the Kp made the robot conquer the corner and also made it shaking. It was calculated that the base speed was 20 and the maximum speed limit was 50 in which case there was a maximum 30 speed difference between the speed at equilibrium position and the speed at maximum amplitude of the position difference. When the robot was at the maximum amplitude, the position difference was around 100, therefore, the Kp was set to 0.3 to make the range of the speed difference to fit the range of the position difference correspondingly. And after several tests, it was found that Kd was to be slightly larger than the Kp to function properly. Thus, Kd was set to 0.6.

When the basic line following demo task was challenged, the robot was able to run precisely following the black line, however, there was a corner where the black line was quite near the edge of the table and the robot was easily running off table while turning on that corner. Then it was noticed that while turning, the speed of the wheels inside the track was not reduced to zero therefore the robot had a net speed forward which made the amplitude of turning too large to avoid falling off the table. This indicated that the Kp was still not big enough, thus the Kp was set to 0.4 and Kd was set slightly larger to 0.8 correspondingly. Then when the robot coming across that corner, the inner side wheels were not spinning in which case the robot was rotating at a point and was not falling off table. Also, by modifying the parameters of PID control, the performance at corners was improved and was more stable and smoother.

Short Cut

There were two approaches for challenging the line following on the coloured short cuts. One was to recognize the symbols of short cut task after a pink block was detected and react correspondingly to the symbol. And another was to set a higher priority for coloured short cuts in which case if the coloured short cut was detected near the black line, the line following on coloured lines was done rather than on black line. The latter method was applied to challenge for the short cut task, because by the time the task was challenged, the object recognition algorithm had not been developed yet while the latter method enabled the robot to realize line following on coloured short cuts without lifting the camera via servo motor and recognizing symbols. As shown in the Figure 26 below, it was similar for the robot to realize line following on the coloured short cuts by applying different HSV colour space inRange filters. All 5 filters for 4 coloured short cuts along with the black line were applied onto each raw frame captured from the camera correspondingly, and the image processing for all thresholded frames was done correspondingly each time. Whenever a coloured short cut was detected, the robot was to run on the coloured line detected. The algorithm for four short cuts was developed and integrated which was successfully tested on the self-made track.

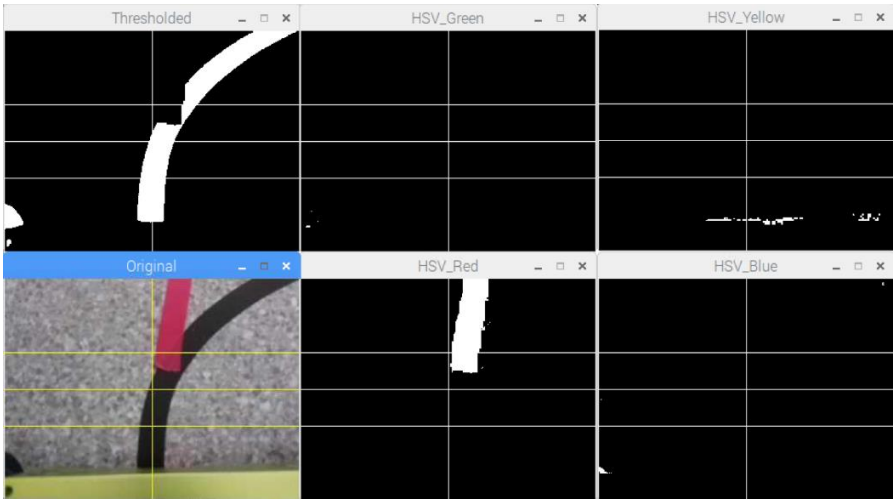

Figure 26. HSV inRange thresholded results for short cuts (e.g. red short cut)

On the self-made track, the robot successfully realized line following on all coloured short cuts, however, when doing demo, the robot was instead of following the coloured line when the coloured short cut was detected but kept line following on the black line. In further discovery, it was noticed that the angle between the coloured line and the black line was lager on the demo track than on the self-made track in which case even though the coloured short cut was detected, the camera was detecting no coloured short cut but the black line thus the robot was to follow the black line. This problem was solved by temporarily disable the black line detection and line following on black line, and by doing so, when the camera was detecting the black line instead of the coloured line after the coloured line was detected, the robot was to react the same as when the robot was off track. Hence, the robot was to run backward in the opposite direction to the side which the last time the line was at.

The temporarily disabling the line following on black line used to be realized by calling “delay()” functions, however, in this highly advanced PID control algorithm, the speed was updated for each frame captured and the process was unable to be paused by the “delay()” function, therefore, a counter was used instead to realize the disabling. When the coloured short cut was detected, for the next several frames, the black line was not being detected. Also, the pink block detection was integrated as a double insurance to improve accuracy. In addition, during the demo, initially the robot was not programmed to stop at the pink block but it was noticed that by doing so, the speed of the robot coming across the short cut was largely reduced and the stability of turning at short cut was increased. Also, the parameters of the number of pixels detected indicating the large angle for both coloured lines and the black line were modified to make sure that the robot was not only able to get itself onto the coloured short cut but also able to get itself back to the black line at the end of the short cut.

Algorithms for all four short cuts were programmed and integrated to the project, however, due to different situations four short cuts were in, it was almost impossible to test and modify parameters for all four short cuts at the same time. Each line needed to be tested ahead and parameters were to be set correspondingly before codes for four individual short cuts being integrated into a single programme. Thus, each short cut demo task was successfully challenged separately.

Object Recognition and Reaction

All other demo tasks required object recognition and reaction, therefore, the algorithm for symbol recognition based on template matching algorithm was developed then.

Due to time limitation, the basic template matching algorithm with "matchTemplate()" function called was developed for symbol recognition. As shown in the Figure 27 below, instead of the detected symbol, the raw frame containing the symbol was applied for template matching. However, even if the frame was perfectly thresholded as shown in the Figure 28 below, the symbol was not directly facing the camera and was relatively small. Also, the size of the symbol in the frame was changing for different position the symbol was placed. Therefore, all the factors discussed resulted in the extreme low accuracy applying the matchTemplate function. The robot was unable to distinguish different symbols with the similarities returned, because the similarities returned were so alike that sometimes the target got lower similarity than other symbols. Hence, to improve the accuracy of symbol recognition, an advanced template matching algorithm was developed and discussed below. However, it was unable to be integrated onto the robot in time by the end of the lab sessions.

Figure 27. Raw frame containing symbol

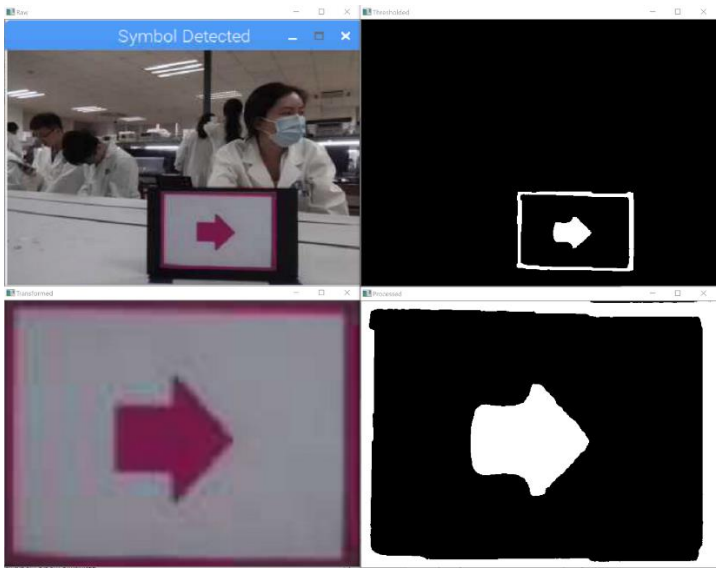

Figure 28. Perspective transformation for advanced template matching

As shown in the Figure 28 above, it was not the raw frame directly applied in the advanced template matching algorithm but the symbol within the capture highlighted by zooming in and modifying the perspective of the largest contour (i.e. the outline of the symbol) using essential codes as shown in the Figure 29 below. By doing so, the similarity between the detected symbol and the template symbol was largely increased therefore increasing the accuracy of the template matching.

```
cvtColor(frame,symbol_HSV,COLOR_BGR2HSV);
inRange(symbol_HSV,Scalar(135,86,45),Scalar(170,255,255),symbol_Bina);//apply pink filter
Mat spotFilter=getStructuringElement(MORPH_ELLIPSE,Size(5,5));
erode(symbol_Bina,symbol_Bina,spotFilter);
Mat maskMorph=getStructuringElement(MORPH_ELLIPSE,Size(4,4));
dilate(symbol_Bina,symbol_Bina,maskMorph);
findContours(symbol_Bina,contours,hierarchy,RETR_EXTERNAL,CHAIN_APPROX_SIMPLE,Point());
approxPolyDP(contours[maxArea],polyContours[maxArea],12,true);
convexHull(polyContours[maxArea],hull,false);
Mat image_transformed=getPerspectiveTransform(symbol_conners,screen_conners);
warpPerspective(frame,symbol_transformed,image_transformed,frame.size());
```

Figure 29. Essential code for perspective transformation and image processing

After the symbol was highlighted by applying several methods as discussed above, it was compared with the template symbol using the algorithm as shown in the Figure 30 below. The matchTemplate algorithm was no longer used because it was used to locate the area where the area had maximum similarity with the template by going through all pixels which costed a lot computational resources. The similarity was able to be obtained using the simple codes as shown in Figure 30. As a result, the similarities returned applying this advanced template matching algorithm were of high accuracy. The matching similarity was around 85 percent or higher while the similarity indicating unmatched symbol was around 50 or less.

```
float compareImages(Mat cameraImage,Mat librarySymbol)
{
    //an incorrect pixel has double weighting
    float matchPercent=100-(100/((float)librarySymbol.cols*(float)librarySymbol.rows)*(2*(float)countNonZero(librarySymbol^cameraImage)));
    return matchPercent;
}
```

Figure 30. Essential code for similarity

The major difference between the basic template matching which was applied for demo and the advance template matching methods discussed above was the image processing part for symbols before the template matching algorithm was applied. The similarities returned from the advanced method were polarized in which case the threshold value was easily being set to distinguish different symbols for accurate object recognition and reaction.

Distance Measurement and Shape Counting

Since the template matching algorithm applied during demo was immature, for the individual distance measurement task, basically the robot was measuring distances all along.

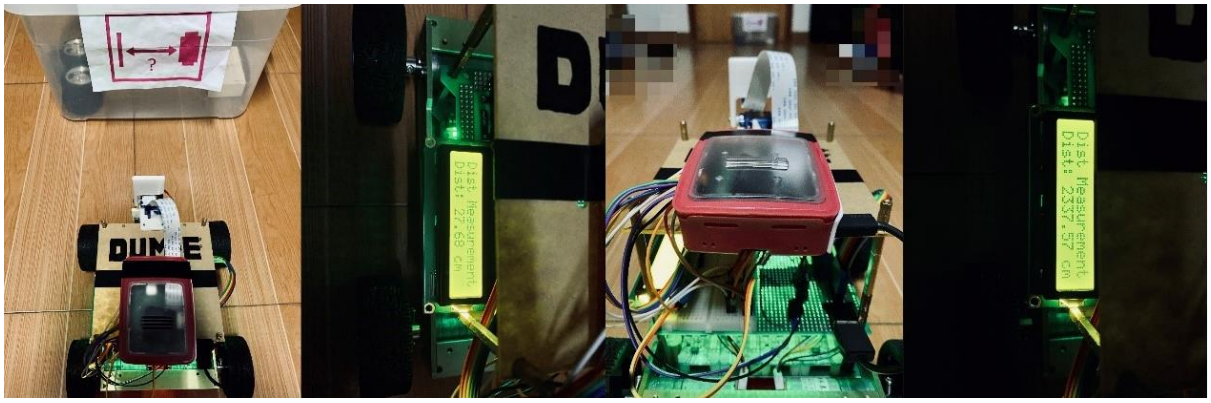

Figure 31. Results of distance measurement

As shown in the Figure 31 above, when testing the robot for distance measurement, it was noticed that when the ultrasound distance sensor was placed near the symbol, the distance measured was accurate but when the symbol was placed further away, the distance returned was too high to be correct. In further discovery, it was found that the module had an operating range between 2 cm and 400 cm for it was relying on ultrasound waves.

For the shape counting demo task, the template matching algorithm was used for distinguishing different shape counting symbols and the pre-set results were displayed on LCD. A more general approach was to find contours and count corners for each contour, and then realizing increment shape count based on number of corners.

Integrated System and Brief Summary

Other tasks were not successfully challenged due to time limitation and the poorly designed template matching algorithm, because the movement and reaction of the robot was determined by the results of the symbol recognition returned for those tasks. Further re-deployment was needed for the robot to achieve fully functional advanced line following with object recognition and reaction. The integrated system “DUM-E” was as shown in Figure 32.

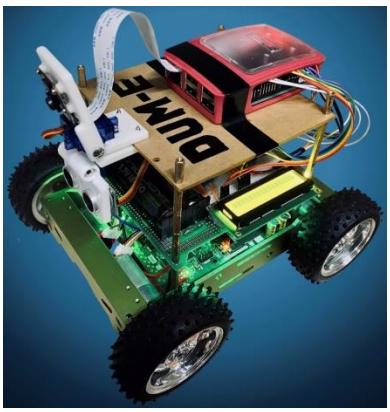

Figure 32. DUM-E

Conclusion

In this project, a robot integrated with Raspberry Pi and the algorithm for advanced line following with object recognition and reaction were developed. The PID control for basic line following was optimised to realize fully automatic speed control and smooth and accurate line following. The robot was able to recognize coloured short cuts of higher priority than the black line and realize line following on the coloured short cuts. The advanced template matching algorithm was not developed in time by the end of the lab sessions, but it was successfully developed after. The distance measurement and the shape counting demo tasks were successfully challenged but the algorithms needed improvement. Several other tasks including football kicking, traffic light, turning at T intersection and also multitask were not successfully challenged in time due to the poorly designed basic template matching algorithm applied during demo. In summary, the line patrol robot with task recognition and reaction was successfully developed in this project. Still, further re-deployment was needed.

An advanced line following algorithm and an advanced template matching algorithm were developed in this project which enabled the robot to realize advanced line following with object recognition and reaction. For PID control, the parameters for P, I, and D were not randomly set or set after multiple experiments simply but set accordingly to the calculation results where the range of the motor speed fitted the range of the error (i.e. position difference between black line and the centre of the robot) and also the performance of the robot corresponding to each set of parameters. As for template matching algorithm, applying matchTemplate function was tested to be a very basic and fundamental template matching algorithm with low recognition accuracy and many application restrictions, therefore, an advanced template matching algorithm was developed using perspective transformation ahead to highlight and isolate the detected symbol to be compared with the template symbol to improve accuracy and make the threshold easier to be set. The HSV colour space inRange colour filtering was applied throughout this project for it was a highly advanced image processing methods allowing frames to be thresholded to a single target colour to distinguish specific targets from the captured frames. These were the major findings in this project.

References

[1] https://opencv.org/about/ (no date) (Accessed: 01 June 2020)

[2] *H61AEE: Introduction to Project V & VI.* Department of Electrical and Electronic Engineering, University of Nottingham Ningbo China, 2020 [Online].

Available:

https://moodle.nottingham.ac.uk/pluginfile.php/6305413/mod_resource/content/1/Introduction%20Lecture.pdf

(Accessed: 03 June 2020)

[3] https://learn.sparkfun.com/tutorials/raspberry-gpio (no date) (Accessed: 03 June 2020)

Appendices

Code for Advanced Template Matching Algorithm

```
//****************************************
//Advanced Template Matching Algorithm.     *
//CopyRight @ JasonYU. All Rights Reserved. *
//****************************************

#include<cstdio>
#include<iostream>
#include"opencv2/opencv.hpp"

using namespace std;
using namespace cv;

Mat processing(Mat);

int main(){
    Mat cap=imread("cap.png");
    Mat result=processing(cap);
    waitKey(0);
    return 0;
}

Mat processing(Mat frame){
    imshow("Raw",frame);

    Mat symbol_HSV;
    Mat symbol_Bina;
    Mat symbol_contours;
    Mat image_transformed;
    Mat symbol_transformed;
    Mat symbol_final;
    bool hull_sort;
    int maxArea;
    vector<vector<Point>>contours;
    vector<Vec4i>hierarchy;
    vector<int>hull;
    Point2f symbol_conners[4],screen_conners[4];
    screen_conners[0]=Point2f(0,0);
    screen_conners[1]=Point2f(640,0);
    screen_conners[2]=Point2f(640,480);
    screen_conners[3]=Point2f(0,480);

    cvtColor(frame,symbol_HSV,COLOR_BGR2HSV);
    inRange(symbol_HSV,Scalar(135,86,45),Scalar(170,255,255),symbol_Bina);//apply pink filter
    Mat spotFilter=getStructuringElement(MORPH_ELLIPSE,Size(5,5));
    erode(symbol_Bina,symbol_Bina,spotFilter);
    Mat maskMorph=getStructuringElement(MORPH_ELLIPSE,Size(4,4));
    dilate(symbol_Bina,symbol_Bina,maskMorph);
    imshow("Thresholded",symbol_Bina);
    findContours(symbol_Bina,contours,hierarchy,RETR_EXTERNAL,CHAIN_APPROX_SIMPLE,Point());
```

Code for Advanced Template Matching Algorithm (cont'd)

```
    symbol_contours=Mat::zeros(frame.size(),CV_8UC3);
    vector<vector<Point>>polyContours(contours.size());
    maxArea=0;
    for(size_t index=0;index<contours.size();index++){
        if(contourArea(contours[index])>contourArea(contours[maxArea])){maxArea=index;}
        approxPolyDP(contours[maxArea],polyContours[maxArea],12,true);
        //drawContours(symbol_contours,polyContours,-1,Scalar(0,0,255),2);
        convexHull(polyContours[maxArea],hull,false);
        //imshow("Contour",symbol_contours);
        if (hull.size()==4){
            hull_sort=false;
            int n=4;
            while(!hull_sort){
                for(int i=1;i<n;i++){
                    hull_sort=true;
                    if(polyContours[maxArea][i-1].x>polyContours[maxArea][i].x){
                        swap(polyContours[maxArea][i-1],polyContours[maxArea][i]);
                        hull_sort=false;
                    }
                }
                n--;
            }
            if(polyContours[maxArea][0].y<polyContours[maxArea][1].y){
                symbol_conners[0]=polyContours[maxArea][0];
                symbol_conners[3]=polyContours[maxArea][1];
            }
            else{
                symbol_conners[0]=polyContours[maxArea][1];
                symbol_conners[3]=polyContours[maxArea][0];
            }
            if(polyContours[maxArea][2].y<polyContours[maxArea][3].y){
                symbol_conners[1]=polyContours[maxArea][2];
                symbol_conners[2]=polyContours[maxArea][3];
            }
            else{
                symbol_conners[1]=polyContours[maxArea][3];
                symbol_conners[2]=polyContours[maxArea][2];
            }
        }
        image_transformed=getPerspectiveTransform(symbol_conners,screen_conners);
        warpPerspective(frame,symbol_transformed,image_transformed,frame.size());
        imshow("Transformed",symbol_transformed);
        cvtColor(symbol_transformed,symbol_transformed,COLOR_BGR2HSV);
        inRange(symbol_transformed,Scalar(135,86,45),Scalar(170,255,255),symbol_final);
        imshow("Processed",symbol_final);
    return symbol_final;
    }
}
```