HUMAN FOLLOWING ROBOT USING IMAGE PROCESSING FOR MEDICAL APPLICATION

GOH KUAN CHEIN

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons.) Electrical and Electronic Engineering

Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

January 2016

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : _____

Name : Goh Kuan Chein

ID No. : _____

Date : _____

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"HUMAN FOLLOWING ROBOT USING IMAGE PROCESSING FOR MEDICAL APPLICATION"** was prepared by **GOH KUAN CHEIN** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons.) Electrical and Electronic Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : _____

Supervisor : Mr. Teoh Boon Yew

Date : _____

HUMAN FOLLOWING ROBOT USING IMAGE PROCESSING FOR MEDICAL APPLICATION

ABSTRACT

This paper presents a project of designing and building a human following robot. This project is related to human following robot using image processing for medical application. The aim of this project is to assist and tracking a specific patient in a hospital. This system is mainly run in indoor application. A human tracking algorithm will be designed to run the human following robot on tracking a target person. A camera is used to capture image that will be process by the Raspberry Pi which will control the robot to follow the target person. On the other hand, ultrasonic sensor is used to perform the task of safe distance tracking, obstacles avoidance and precision tracking. Raspberry Pi is used as a platform of processing the data that collected from the sensors then execute the command on controlling the movement of the human following robot. The robot was made sure to have an accuracy on tracking the target person and avoid the obstacles during the following period. The idea of the system was experimentally verified with the ease of the prototype in which all the proposed ideas are implemented and the outcome was successfully obtained.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 General Introduction

As the robotic industry growing rapidly, human following robot is also one of the robotic scope that many people has taken as a research project. Human following robot has wide scope of usage and many type of application can be applied into the robot in daily life. It can be linked in many fields such as medical, logistic, defence, underwater, and even as household robot. There are many researches have propose to use sensors like ultrasonic sensor, Infra-red sensor, PIR sensor, laser rangefinder, stereo camera, Kinect sensor and so on in this human following robot system. The human following robot usually have a few basic application system which are target person tracking system, static and motion obstacle sensing and avoiding system and the robot movement control system.

In this paper, the human following robot is in the field of medical application where the priority of this robot is to improve the service in medical field. The human following robot system here utilize ultrasonic sensors, camera and other sensors. The ultrasonic sensor is use for obstacle tracking and target person tracking while the camera is to ensure the right target is followed. The Bluetooth module is use for indoor positioning system. It is necessary to integrate a few sensors together to get a precise location of the target person and can avoid obstacles during the following period.

1.2 Aims and Objectives

The main objective of this project is as following:

- To design and build a human following robot
- To assist and tracking a specific patient in a safe distance
- To make sure that the human following robot follow the right person

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In this chapter, five techniques of human following robot from the papers were researched and reviewed. Different techniques of human following robot is used in this five paper where each technique has its own pros and cons. Every human following robot that the researchers built have their own limitation, however the limitation can be improved and solved in the future by other designers or researchers where this process is endless until the best solution of human following robot is invented.

A basic human following robot consists of three main parts, which the most important part is the processor of the robot. The remaining two parts is the robot body which consist of the chassis, motor and the wheel, and the detection device which is used to detect the person. Figure 2.1 shows the main parts that build up human following robot.

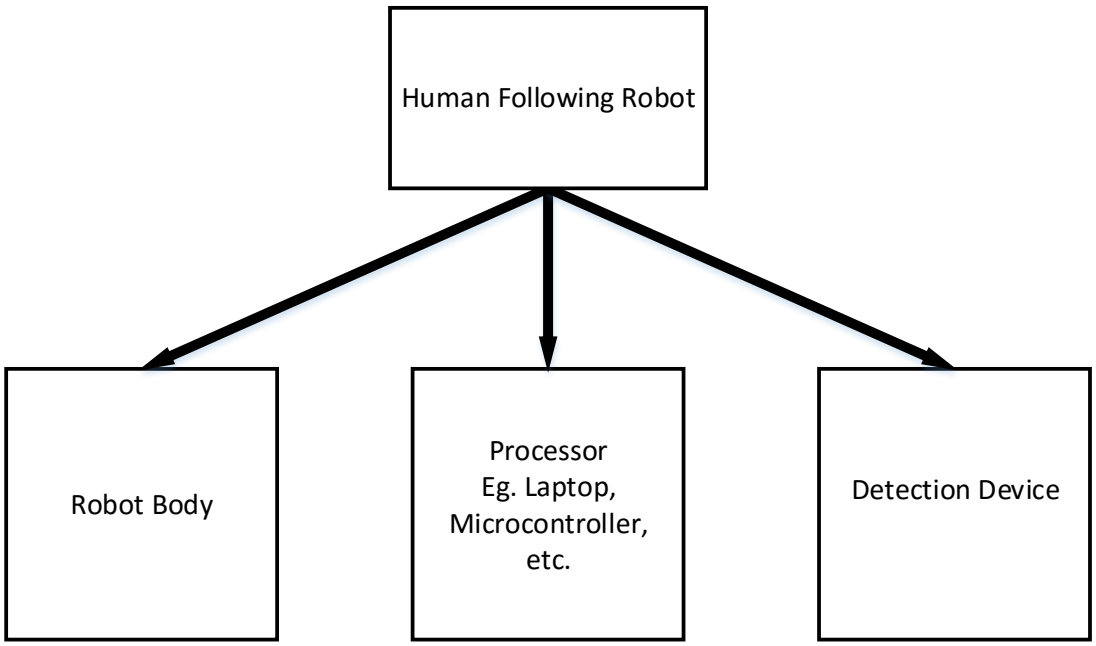

Figure 2.1: Main parts of Human Following Robot

2.2 Challenges on Human Following Robot

When designing a human following robot, there are many challenges that the researchers and designers has to face. The researchers has to consider many kinds of situation that will be faced during the robot following period. The researchers have to think of ways to overcome the challenges.

One of the challenges is in a certain crowded area, the human following robot should be able to function well. When there is a lot of obstacles and movement around, it should not be lost in tracking the person. Besides that, the robot should be able to avoid obstacles either that the obstacles are moving or static. The design of the human following robot should be consider that the robot will operate on the floor that is uneven. The human following robot should be able to follow and distinguish the right person during the tracking. Furthermore, the robot should follow the person in a safe distance and know when to stop to avoid collision with the person.

2.3 Example Techniques on Human Following Robot

The following is some journal researched which contain different technique of human following robot. These human following robot information with different kind of tracking systems can be used as the reference in this human following robot project development.

2.3.1 S. Shaker, 2008

Based on the journal (S. Shaker, 2008), the researchers implement laser range finder into their robot. They detect and follow a person movement by using leg tracking algorithm. The laser range finder provides the data to the algorithm to detect the targeted person's leg. The algorithm can calculate out the velocity of the targeted person's leg movement with the respect to the human following robot. The information that generated by the algorithm is then passed to a fuzzy controller which will control the follow speed of the robot when following the targeted person's leg. Fuzzy interference system is to deal with the problem which is humanistic, complex and situation that the use of mathematical is too precise, but is imprecise with nature. This system is to control and smoothen the human following robot's motion when following the targeted person. This system also ensure that the robot follow the targeted person in a safe distance. Figure 2.2 shows the Human following robot that is developed by S. Shaker

Figure 2.2: Human Following Robot Developed by S. Shaker

2.3.2 Z. Chen, 2007

In the journal (Z. Chen, 2007), the researchers use two stereo camera for image capturing. The two captured images are then being analysed using stereo based Lucas-Kanade approach. Binocular Sparse Feature Segmentation (BSFS) algorithm is use in the human following robot for vision-based capture. This algorithm uses Lucas-Kanade approach to detect and track the feature points in the images and calculate the sparse disparity map and then remove the disparity from the images. Random sample consensus approach is a technique that calculate the motion of the target in a static background using the match point of the two images. The stereo and motion information that obtained from the approaches are fused, then the BSFS algorithm separates the moving target from the static background. This entire system does not use color-based approach, it uses only gray level information and the targeted person does not necessary have to wear different color clothes to differ from the background. Face detection algorithm is also included into the robot to increase the accuracy of tracking, although the target person does not necessary facing the human following robot. The Figure 2.2 shows that the process flow of the system on processing the information and tracking the target person.

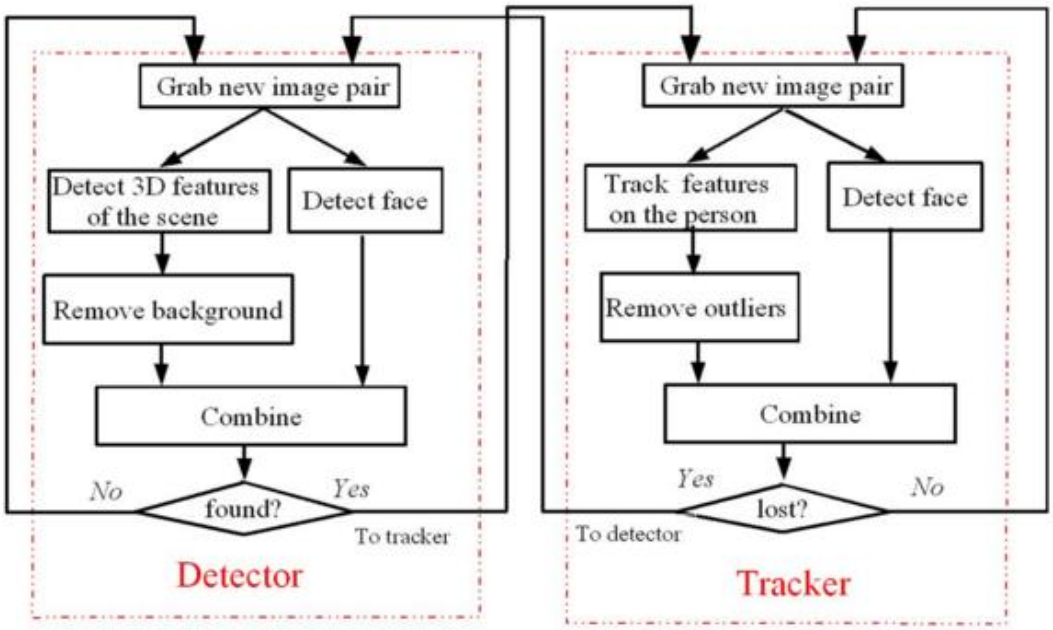

Figure 2.3: The Flow Diagram of The System (Z. Chen, 2007)

2.3.3 J. Satake, 2012

As for the research (J. Satake, 2012), the researchers built a human following robot which uses motion stereo camera that tracking the targeted person using depth templates. The depth template is a template mainly for the human upper body. In this system, the researchers uses three types of template which is in different direction to the body. The depth templates use the depth image which captures by the camera where the targeted person is 2m away. The depth template consist of a binary template that the foreground and background value are adjusted according to the tracking status and the input data. Extended Kalman filter (EKF) tracker is being used for tracking a target person by providing predicted scene positions. This tracker will check continuously whether there is new objects appear in the image. Besides that, support vector machine (SVM) based is being used to remove the false detection from other object with similar characteristic to the target person.

In this research, the laser rangefinder is used for obstacle avoidance. The color detection is included, but when the human following robot is in a low light

surroundings, the robot cannot detect and differentiate between two or more person with similar color clothes. To overcome this problem, scale invariant feature transform (SIFT) is being introduced. SIFT feature is an image feature that is powerful in scaling and rotation in the image plane and also resistant to lighting condition. The error point from the SIFT in the image is being filtered by the Random Sample Consensus. The SIFT feature decreases when the distance of the target person with the camera increases. To solve this problem, the distance of the camera and the target person is being set to limit the decrease of SIFT features. Figure 2.4 shows the structure of the human following robot developed by J. Satake.

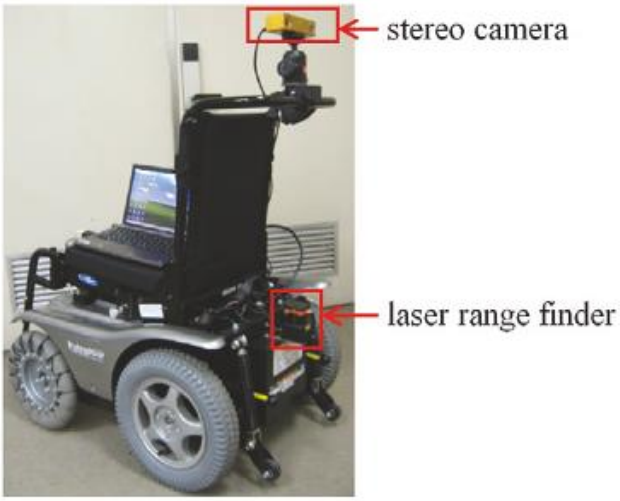

Figure 2.2: Structure of Human Following Robot by J. Satake

2.3.4 B. Ilias, 2014

The research done by (B. Ilias, 2014) uses Kinect high speed sensor to detect and track the movement of the target person. When initialize state, the target person needs to raise his hands in front of the Kinect sensor in order to calculate the human skeleton. Then the information is passed to the Processing.Org software by using laptop. After the human skeleton is being traced out by the software the command is sent to the BASIC stamp 2 Kinect to execute the direction of movement of the robot, then the BASIC stamp 2 ultrasonic will check out for obstacles. If there is no

obstacles in front the robot, the BASIC stamp 2 motor will move the robot to the target person. However there is an issue when using Kinect sensor outdoor where the sensor unable to operate due to the ultraviolet rays. To overcome this issue, they uses four layers of 5% tinted film cover at the Infra-red depth sensor.

2.3.5 K.S. Nair, 2014

In this research (K.S. Nair, 2014), the researchers uses the ultrasonic sensor as the key element in detecting the target. The ultrasonic sensor module will sense the presence of the target human and the robot will move according to the direction of the target person. As for the obstacle detection, the Infrared sensor is used to detect the obstacles and the robot will avoid the obstacle by changing the direction for static obstacle or by stopping to wait for the motion obstacles to move away. The system uses AVR Atmega 32 as the processor on controlling the human following robot.

2.3.6 Comparison and Summary

The human following robot of the research can be different in various aspects such as the sensor it used, the tracking method and the part of the target person being track. Comparison of the techniques of human following robot from all the paper is in Table 2.1.

Table 2.1: Processing Time for Different Production Line

Author	Detection and Tracking Devices	Methods	Which part of the target person being detected
S. Shaker, J.J.Saade, D.Asmar(2008)	Laser range finder	Leg detection algorithm, Fuzzy inference system	Leg
Z. Chen, S.T. Birchfield(2007)	Two stereo cameras	Lucas Kanade approach, Random Sample Consensus, face detection algorithm	Face
J. Satake, M.Chiba, J. Miura(2012)	Laser range finder, and stereo camera	Random Sample Consensus, SIFT features, SVM-based verifier, EKF Tracker	Upper body
B. Ilias, S.A.Abdul Shukor, S.Yaacob, A.H. Adom and M.H.Razali(2014)	Kinect sensor	Human skeleton Method	Human skeleton method
K.S. Nair, A.B. Joseph, J.I. Kuruvilla(2014)	Ultrasonic sensor, IR sensor	Obstacle detection, human target detection	Leg

CHAPTER 3

METHODOLOGY

3.1 Processing Unit

In this project, Raspberry Pi is chosen as the main processing unit that is going to run the whole system of human following robot. The Raspberry Pi will be receiving the input information from the sensors where all aspect of situation will be take in consideration. A human following algorithm will be implement in this system where to ensure the robot follows the target person. The input information from the sensor included the safe distance tracking of the robot, the obstacles avoidance of the robot and the precise tracking system. Different input from the sensors gives out different movement of the robot while tracking the target person. The motor controller control the direction of the motor and the speed of the motor when following the target person. The camera is use to confirm whether the robot follows the right target person. All the system are integrated together to ensure the robot works perfectly. Figure 3.1 shows the overall system of the human following robot.

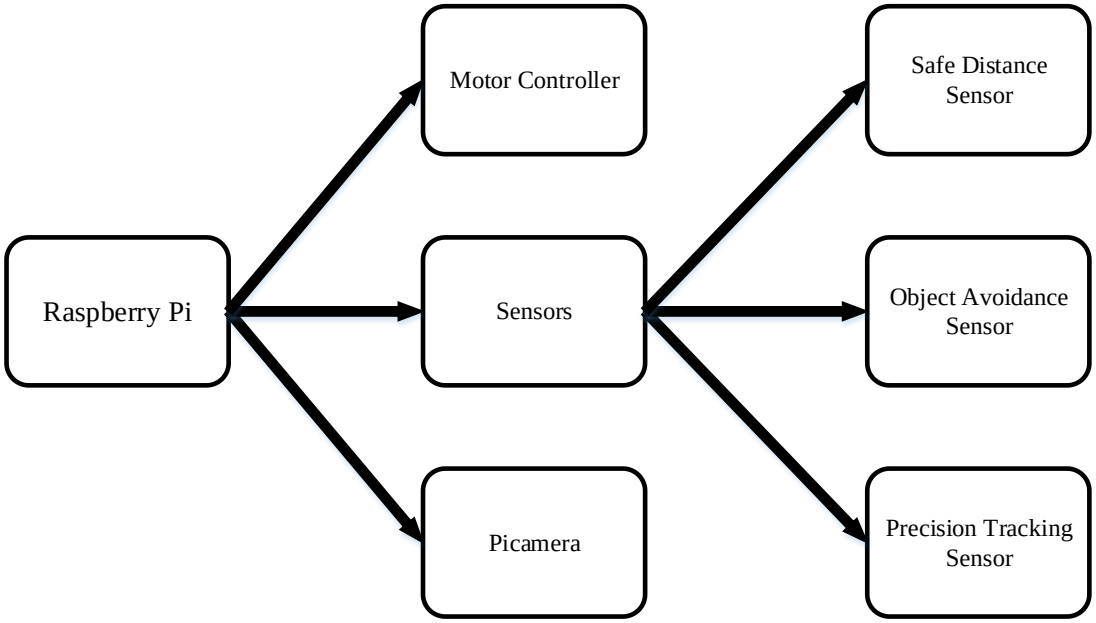

Figure 3.1: Overall system of the human following robot

3.2 Safe Distance Tracking and Obstacle Avoidance

For safe distance tracking and obstacle avoidance, ultrasonic sensor is chosen as the sensor to achieve them. About three ultrasonic sensor will be place in front of the human following robot. One of them is placed in the center of the robot, while the other two is placed at the side of both end of the robot. The center ultrasonic sensor will be a 5m range precision ultrasonic sensor which will act as the safe distance tracking from the leg of the target person. Approximate of 1m of safe distance tracking will be set as the range for the robot to follow the target person. While the both corner front ultrasonic sensor is a 30cm range ultrasonic sensor which is use as obstacle avoidance guide. When the obstacle is sense by one of the corner, the robot will move to the other direction to avoid the obstacles. Figure 3.2 shows the 5m range high precision ultrasonic sensor that use for safe distance tracking. Figure 3.3 shows the 30cm ultrasonic module for the obstacles avoidance.

Figure 3.2: HRLV-MaxSonar EZ Series Ultrasonic Sensor

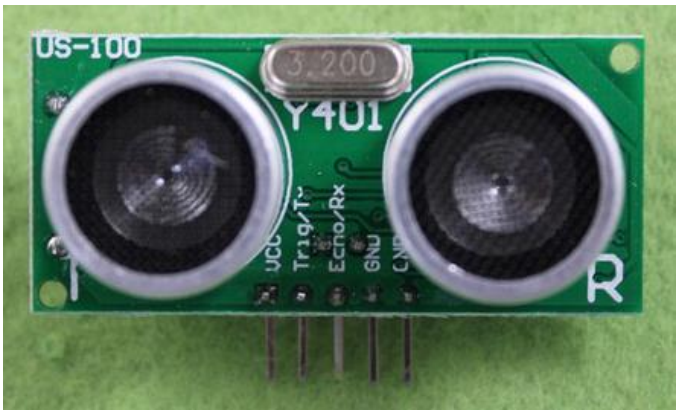

Figure 3.3: 30cm Ultrasonic Module

3.3 Object Tracking

For object tracking, the Raspberry Pi camera is chosen for capturing the image of the logo that indicates the target person. Which then the image of the certain logo will be image processed by using the OpenCV software with the Python interface. The specific logo will be identified whether is the correct target person on following. The logo will be customized to be only recognized the robot as the correct target person. . Figure 3.4 shows the Raspberry Pi with the Pi camera module. Figure 3.5 shows the tracking logo that is used for the human following robot.

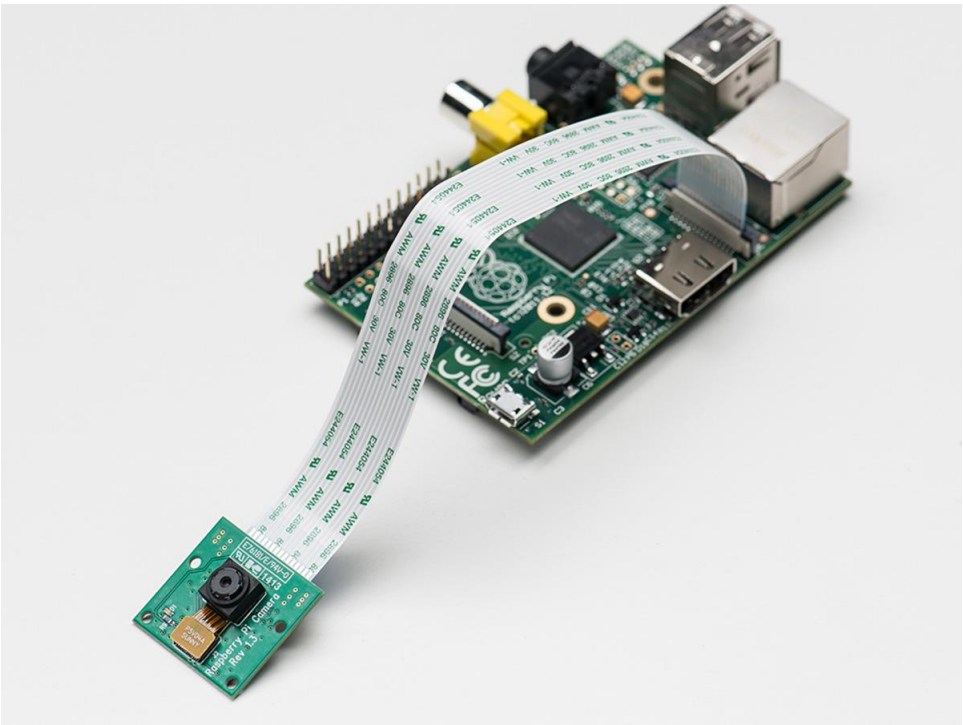

Figure 3.4: Raspberry Pi with Picamera

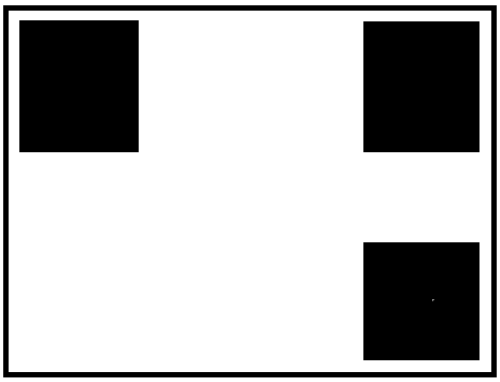

Figure 3.5: The Logo of The Tracking System

3.4 DC Geared Motor

The DC motor that used in this human following robot is SPG30-60 DC geared motor. This motor is a 1.1W DC Brushed motor and its rated voltage is 12V with stall torque of 254.8mNm. It has a speed pf 58rpm at free run, 70mA current at no

load and maximum rating of 410mA of current at loaded. Besides that, the DC motor has a gear ratio of 60:1. Two SPG30 DC geared motor is used in this project. Figure 3.6 shows the 1.1W SPG30-60 DC geared motor that used in the project, while Figure 3.7 shows the Mounting of the SPG30-60 DC geared motor and the free wheel on the base of the robot.

Figure 3.6: DC Geared Motor, SPG30-60

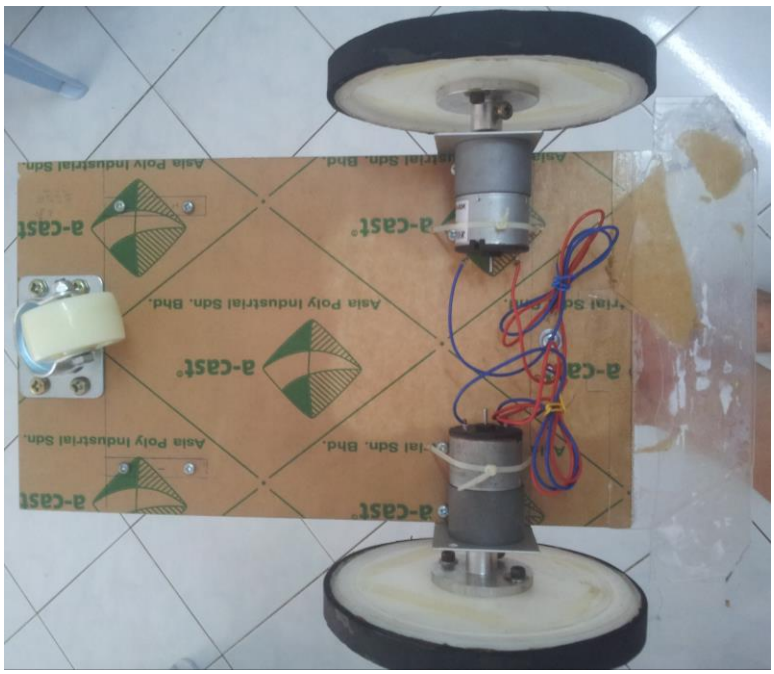

Figure 3.7: The Base of Robot with Mounted DC Geared Motor and Free Wheel

3.5 DC Motor Driver

The motor driver used in the project is MDD10A. This motor driver is a dual channel which designed to drive two DC brushed motor with the current up to 10A continuously and 30A peak current for 10 second for each channel. The speed of the DC motor can be controlled with the motor driver through the PWM signal that provided by the microcontroller. It supports motor voltage from 5V to 25V. This motor driver also supports sign-magnitude PWM signal and locked-antiphase. Besides that, this motor driver uses full solid state components which gives faster response in time. It eliminates the wear and tear of the motor driver mechanical relay. Figure 3.8 shows the MDD10A dual channel DC motor driver.

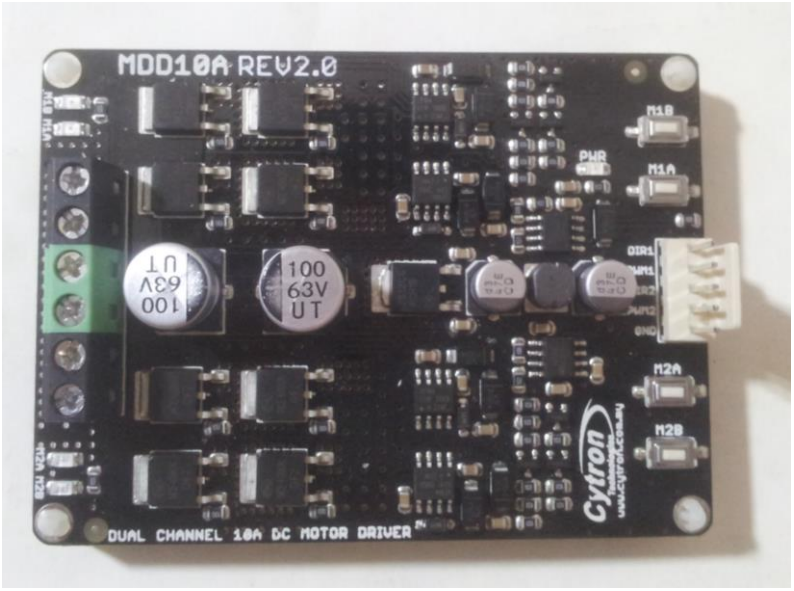

Figure 3.8: Dual Channel DC Motor Driver, MDD10A

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

In this chapter, the results obtained and the problem encountered throughout the project development process are discussed. The functionality of the human following robot hardware is tested in an indoor environment to ensure the robot is fully operating indoor. The test results are then be evaluated and has become the benchmarks for the robot operation. This evaluation is important where it ensures the objectives is being achieved.

4.2 Results

In this part, the project involves in both mechanical part and programming part. The mechanical part is consist of the robot base and the speed control of the motor. As fpr the programming part, the image processing is processed by the Raspberry Pi by using the software Python with OpenCV and the motor controlling part.

4.2.1 Mechanical Development

Firstly, the material that mostly used in this project to build the robot chassis is acrylics. Acrylic is a type of plastic that has good durability and support for the robot

body. The two motors is placed in the front of the robot and a free wheel is placed at the back. The reason that the motors are placed in front of the robot is that the drag force of two motor in front is better than putting two free wheel and two motors pushing at the back. Less friction and weight for the free wheel to contact the ground by comparing placing two free wheel in front of the robot. The reason which there are only two motor is used in this project is that the two motor has the enough torque to support the robot total weight and saves battery power by comparing on using four motor to run the robot. Figure 4.1 shows the human following robot chassis with mounted DC motor and the free wheel.

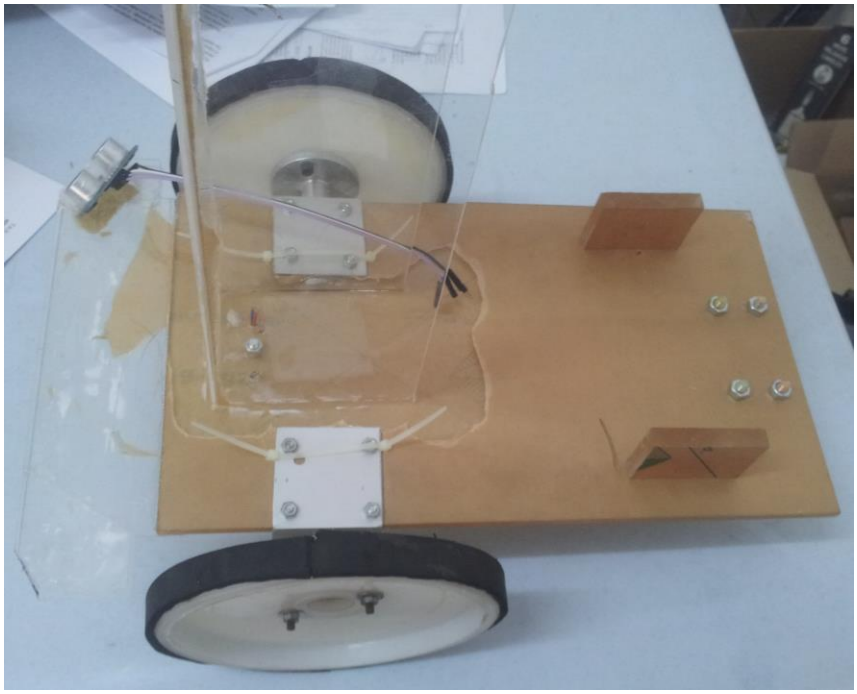

Figure 4.1: The Chassis of The Human Following Robot

Secondly, the Raspberry Pi camera is installed at the highest spot of the robot chassis. This is to ensure that the Pi camera get a better view in tracking the logo without any obstructions from the surroundings and the hardware. Besides that, a 5m high range precision ultrasonic sensor is placed just below the Pi camera which allows the ultrasonic sensor to keep track the distance of the targeted person and the robot which fixed at least 1m in distance. Then two 30cm distance ultrasonic sensor

is placed in front of the robot which sense the front corner of the robot whether there are any obstacles in front and avoid them.

Thirdly, 12V lead acid battery is placed on the robot body to power up the motor driver and runs the motor. Asides that, a spare 12V battery is used to power a up a high power LED light below the high range ultrasonic sensor which act as the lighting for the camera to run at a low lighting environment. A mobile power bank is used as the supply for powering up the Raspberry Pi where the power bank supplies 5V at 2A. Power bank is used to power up Raspberry Pi because the processor of the Raspberry pi needs more power to process data that collected from the image processing. The ideal case for the power supply for powering up Raspberry Pi is 5V, 2.5A which the Raspberry Pi draws many current when there is a high usage of the processing speed in the processing unit. Figure 4.2 shows the overall setup of the human following robot.

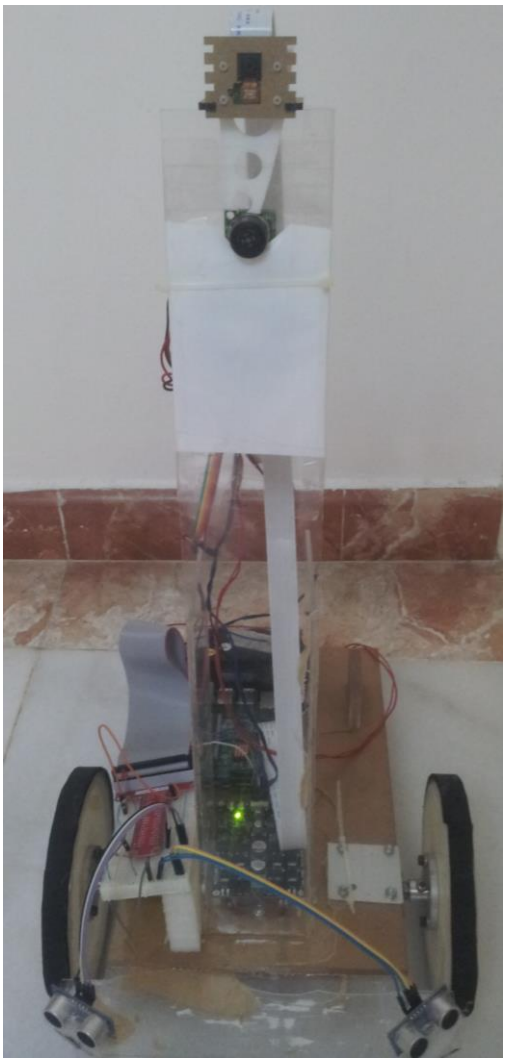

Figure 4.2: The Overall Setup of The Robot

4.2.2 Programming Development

As for the programming development, it consists of two part which are the object tracking and the robot movement controlling part.

4.2.2.1 Object Tracking

For the object tracking, OpenCV with Python coding is used. This OpenCV is used to use the pi camera to capture frames and process the frames to trace the tracking

logo out of the frames and keep track of the logo. The tracking position of the logo is sent to the movement controller coding of the robot. The tracking position is basically separated into three position which is center , left and right. The OpenCV coding tracks down the tracking logo and then identify the position of the logo. The resolution of the camera is lowered down to 320x240 of width and height. Figure 4.3 shows the coding of live video stream while the logo is being tracked. Figure 4.4 shows the coding of the tracing logo from the frames capture. Figure 4.5 shows the logo being track which the position is to the right. Figure 4.6 shows the logo is at the left position. Figure 4.6 shows that the logo at the center of the footage.

```
# Create a Video Stream Tread
class PiVideoStream:
    def __init__(self, resolution=(CAMERA_WIDTH, CAMERA_HEIGHT), framerate=CAMERA_FRAMERATE, rotation=0, hflip=False, vflip=False):
        # initialize the camera and stream
        self.camera = PiCamera()
        self.camera.resolution = resolution
        self.camera.rotation = rotation
        self.camera.framerate = framerate
        self.camera.hflip = hflip
        self.camera.vflip = vflip
        self.rawCapture = PiRGBArray(self.camera, size=resolution)
        self.stream = self.camera.capture_continuous(self.rawCapture,
            format="bgr", use_video_port=True)

        # initialize the frame and the variable used to indicate
        # if the thread should be stopped
        self.frame = None
        self.stopped = False

    def start(self):
        # start the thread to read frames from the video stream
        t = Thread(target=self.update, args=())
        t.daemon = True
        t.start()
        return self

    def update(self):
        # keep looping infinitely until the thread is stopped
        for f in self.stream:
            # grab the frame from the stream and clear the stream in
            # preparation for the next frame
            self.frame = f.array
            self.rawCapture.truncate(0)

            # if the thread indicator variable is set, stop the thread
            # and resource camera resources
            if self.stopped:
                self.stream.close()
                self.rawCapture.close()
                self.camera.close()
                return

    def read(self):
        # return the frame most recently read
        return self.frame

    def stop(self):
        # indicate that the thread should be stopped
        self.stopped = True
```

Figure 4.3: The Coding of The Video Stream

```
# Grab a Video Steam image and initialize search rectangle
cam_cx1 = sw_cx
cam_cy1 = sw_cy
cam_cur_cx = cam_cx1
cam_cur_cy = cam_cy1
image1 = vs.read()   # initialize first image
img = cv2.imread("cam.jpg") #image that has chosen to be the search triangle
search_rect = img[sw_y:sw_y+sw_h, sw_x:sw_x+sw_w]  # Initialize centre search rectangle
cam_track_cx = 0   # initialize cam horizontal cam movement tracker
cam_track_cy = 0   # initialize cam vertical cam movement tracker
while True:
    image1 = vs.read()  # capture a image from video stream thread
    # Look for search_rect in this image and return result
    result = cv2.matchTemplate( image1, search_rect, cv2.TM_CCORR_NORMED)
    # Process result to return probabilities and Location of best and worst image match
    minVal, maxVal, minLoc, maxLoc = cv2.minMaxLoc(result)  # find search rect match in new image
    # Get the center of the best matching result of search
    cam_cx2, cam_cy2 = get_center( maxLoc[0], maxLoc[1], sw_w, sw_h )
    # Update cumulative camera tracking data
    cam_track_cx = cam_track_cx + cam_cur_cx - cam_cx2
    cam_track_cy = cam_track_cy + cam_cur_cy - cam_cy2
    # Check if search rect is near edges and meets search accuracy threshold
    if not ( maxLoc[0] > sw_buf_x and maxLoc[0] + sw_x + sw_buf_x < CAMERA_WIDTH and
             maxLoc[1] > sw_buf_y and maxLoc[1] + sw_y + sw_buf_y < CAMERA_HEIGHT
             and maxVal > sw_maxVal):
        # check value of lowest matching result and reset search rectangle
        if minVal < sw_minVal:
            if debug:
                print("    Reset search_rect cur_cx=%i cam_track_cx=%i cur_cy=%i cam_track_cy=%i"
                % (cam_cur_cx, cam_track_cx, cam_cur_cy, cam_track_cy))
            search_rect = img[sw_y:sw_y+sw_h, sw_x:sw_x+sw_w]

            cam_cx2, cam_cy2 = get_center( maxLoc[0], maxLoc[1], sw_w, sw_h )
            cam_track_cx = cam_track_cx + cam_cur_cx - cam_cx2
            cam_track_cy = cam_track_cy + cam_cur_cy - cam_cy2
            cam_cx1 = sw_cx
            cam_cy1 = sw_cy
            cam_cx2 = cam_cx1
            cam_cy2 = cam_cy2
    cam_cur_cx = cam_cx2
    cam_cur_cy = cam_cy2
```

Figure 4.4: Coding of The Tracking System

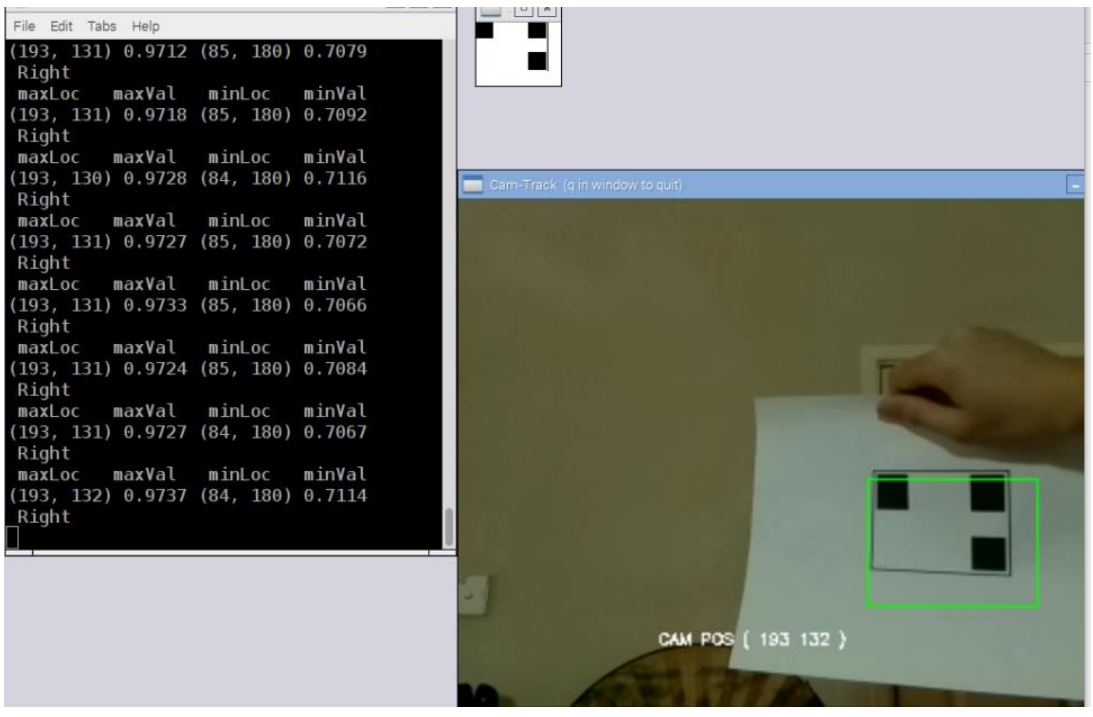

Figure 4.5: The Tracking Logo which Position to the Right.

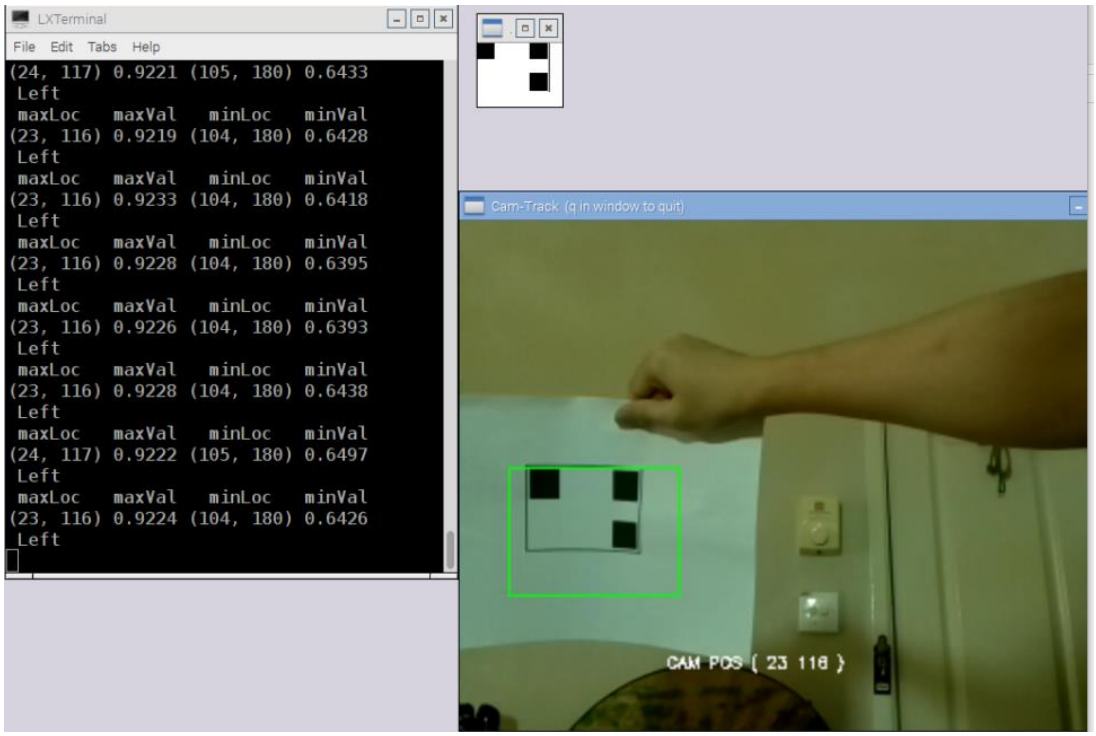

Figure 4.6: The Tracking Logo which Position to the Left.

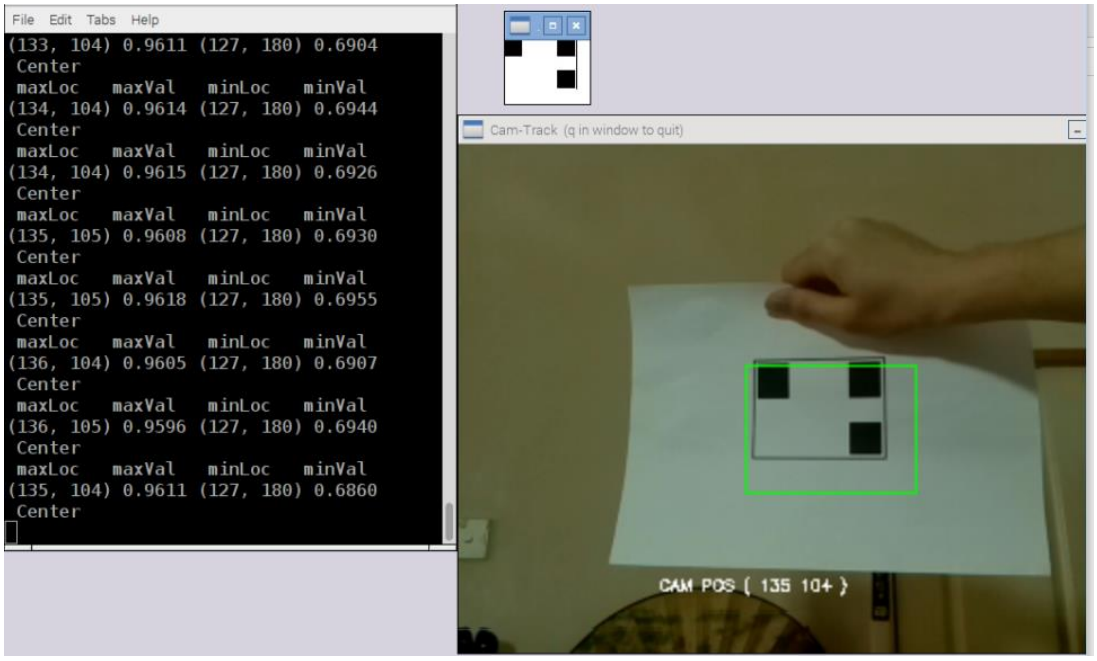

Figure 4.7: The Tracking Logo which Position in the Center.

4.2.2.2 Robot Movement

For the robot movement, it involves the Python coding that controls the direction of the DC geared motor through the motor driver. Besides that, the tracking position that obtained from the image processing tracking also gives the direction of the robot movement the follow the targeted person. Figure 4.8 shows the coding of the movement of the human following robot. Figure 4.9 shows the coding of checking the surroundings obstacles and the tracking distance of the robot and the targeted person.

```
def forward(tf):
    gpio.output(13, True)
    gpio.output(15, False)
    gpio.output(16, True)
    gpio.output(18, False)
    time.sleep(tf)
    gpio.cleanup()

def reverse(tf):
    gpio.output(13, False)
    gpio.output(15, True)
    gpio.output(16, False)
    gpio.output(18, True)
    time.sleep(tf)
    gpio.cleanup()
```

(a)

```
def turn_left(tf):
    gpio.output(13, False)
    gpio.output(15, False)
    gpio.output(16, True)
    gpio.output(18, False)
    time.sleep(tf)
    gpio.cleanup()

def turn_right(tf):
    gpio.output(13, True)
    gpio.output(15, False)
    gpio.output(16, False)
    gpio.output(18, False)
    time.sleep(tf)
    gpio.cleanup()
```

(b)

```
def pivot_left(tf):
    gpio.output(13, False)
    gpio.output(15, True)
    gpio.output(16, True)
    gpio.output(18, False)
    time.sleep(tf)
    gpio.cleanup()

def pivot_right(tf):
    gpio.output(13, True)
    gpio.output(15, False)
    gpio.output(16, False)
    gpio.output(18, True)
    time.sleep(tf)
    gpio.cleanup()
```

(c)

Figure 4.8**: Picture (a) shows the coding of forward movement and the reverse movement. Picture (b) shows the coding of turn left and turn right. Picture (c) shows the coding of pivot left and pivot right.**

```
def check_left():
    init()
    dist = distance1()

    if dist < 15:
        print('too close,',dist)
        init()
        turn_left(3)
        dist = distance1()
        if dist < 15:
            print('too close,',dist)
            init()
            pivot_left(3)
            dist = distance1()
            if dist < 15:
                print('too close,',dist)
                sys.exit()
```

(a)

```
def check_right():
    init()
    dist = distance2()

    if dist < 15:
        print('too close,',dist)
        init()
        turn_right(3)
        dist = distance1()
        if dist < 15:
            print('too close,',dist)
            init()
            pivot_right(3)
            if dist < 15:
                print('too close,',dist)
                sys.exit()
```

(b)

```
def check_front():
    init()
    dist = distance3()

    if dist < 100:
        print('too close,',dist)
        init()
        reverse(1)
        dist = distance1()
        if dist < 100:
            print('too close,',dist)
            init()
            reverse(2)
            dist = distance1()
            if dist < 100:
                print('too close,',dist)
                sys.exit()
```

(c)

Figure 4.9: Picture (a) and (b) shows the coding of sensing the obstacles in front of the robot. Picture (c) shows the coding of the tracking distance of the robot and the target person.

4.3 Discussion

Based on the results obtained, there are many problems that faced for the tracking system. The first problem is the programming language of the software. The software that are used in the project is Python. The problem encountered when sorting out the Open CV coding where many aspect are needed to consider. One of the aspects is the coordinates of the tracking frame.

Besides that, the other challenge is to trace out the logo where in some point the logo count not be track out properly which many condition that affects the efficiency on tracking the logo. The quality of the frame captured by the Pi camera is not good enough where the frame captured has a colour defect which the overall colour of the picture is yellowish. The one of factors that affects the quality of the frame and the tracking logo efficiency is the lighting of the surroundings. A high power LED is added to the robot to improve the lighting effect when the lighting surroundings are not so good.

Next problem encountered is the combination of the coding of the object tracking and the robot movement takes times and there is still some error when both of the coding is run together. A few improvement is needed to add into the coding to improve the overall system.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

Human following robot can give a lot of benefits to the society. In the future development, the usage of different types of robot in helping human's task in any aspects will be massive. Human following robot will be one of the robot that will take over simple task, such as being a following assistant, maid, goods carrier or guider. Some application that can be useful to the hospital where the human following robot can be attached to the nurse cart and to the airport where it can become a luggage cart which brings the passenger luggage and follows the passenger. It is necessary for human to explore the knowledge of the robotics as it knows as no boundaries.

5.2 Recommendations

For future development, there are a few improvement that needed to make to enhance the robot's tracking and following system. Firstly, a better tracking camera is needed to improve the image quality of the frames capture, where the current camera that uses in this project is not in a good condition. The image that captured from this camera shows a bit yellowish background. Secondly, a better processor is needed to run the image processing system. The Raspberry Pi is a good small size processor which can runs the system on doing image processing, but there is a

limitation on the Raspberry Pi where the image process cannot be a high resolution image. When the Raspberry Pi runs image processing in high resolution the processor will be occupied by the processing and it slows down the processing speed of the other features. So a better processor is needed to run high revolution image processing.

Besides that, other tracking system can be implement into the human following robot such as Indoor Bluetooth Positioning System. This system covers the weakness of the Global Positioning System (GPS) that GPS is not available indoor where the satellites cannot track or trace the device indoor where the building blocks the signal waves. The Indoor Bluetooth Positioning System can track and trace out the robot position indoor and increases the efficient of the human following robot on tracking the targeted person.

REFERENCES

Service Robot Statistics. (n.d.). Retrieved April 12, 2016, from http://www.ifr.org/service-robots/statistics/

S. Shaker, J.J. Saade, D. Asmar. Fuzzy Inference-Based Person-Following Robot. *In Proceedings of International Journal of Systems Applications, Engineering & Development*, Issue 1, Volume 2, 2008, pp. 29-34.

Z. Chen, S.T Birchfield, *Person Following with a Mobile Robot Using Binocular Feature-Based Tracking*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) San Diego, California, October 2007

J. Satake, M. Chiba, J. Miura. *A SIFT-Based Person Identification using a Distance-Dependent Appearance Model for a Person Following Robot*. Proceedings of the 2012 IEEE International Conference on Robotics and Biomimetics, December 11-14, 2012, Guangzhou, China, pp. 962-967.

B. Ilias, S.A. Abdul Shukor, S. Yaacob, A.H. Adom and M.H. Mohd Razali. *A Nurse Following Robot with High Speed Kinect Sensor*. ARPN Journal of Engineering and Applied Sciences, vol. 9. No 12, December 2014, pp. 2454-2459.

K. S. Nair, A. B. Joseph, & J. I. Kuruvilla (2014). Design of a low cost human following porter robot at airports. *International Journal on Advanced Computer Theory and Engineering (IJACTE), 3*(2), 34-37.

APPENDICES

APPENDIX A: Coding

```
import io
import time
import cv2

from picamera.array import PiRGBArray
from picamera import PiCamera
from threading import Thread
from operator import itemgetter
import numpy as np

#---------------------------------------------------------------------------------------------
# Global Variable Settings
debug = True          # Set to False for no data display
window_on = True      # Set to True displays opencv windows (GUI desktop reqd)
fps_on = False        # Display fps (not implemented)

# OpenCV Settings
WINDOW_BIGGER = 2.0   # increase the display window size
MAX_SEARCH_THRESHOLD = .97  # default=.97 Accuracy for best search result
of search_rect in stream images
MIN_SEARCH_THRESHOLD = .45  # default=.45 Accuracy for worst search result
of search rect in stream images
LINE_THICKNESS = 1    # thickness of bounding line in pixels
```

```
CV_FONT_SIZE = .25    # size of font on opencv window default .5
# SHOW_CIRCLE = True  # show a circle otherwise show bounding rectancle on window
# CIRCLE_SIZE = 8     # diameter of circle to show motion location in window

# Camera Settings
CAMERA_WIDTH = 320
CAMERA_HEIGHT = 240
CAMERA_HFLIP = True
CAMERA_VFLIP = True
CAMERA_ROTATION=0
CAMERA_FRAMERATE = 35  # framerate of video stream.  Can be 100+ with new R2 RPI camera module
FRAME_COUNTER = 1000   # used by fps

#---------------------------------------------------------------------------------------------
# Create a Video Stream Tread
class PiVideoStream:
   def __init__(self, resolution=(CAMERA_WIDTH, CAMERA_HEIGHT), framerate=CAMERA_FRAMERATE, rotation=0, hflip=False, vflip=False):
      # initialize the camera and stream
      self.camera = PiCamera()
      self.camera.resolution = resolution
      self.camera.rotation = rotation
      self.camera.framerate = framerate
      self.camera.hflip = hflip
      self.camera.vflip = vflip
      self.rawCapture = PiRGBArray(self.camera, size=resolution)
      self.stream = self.camera.capture_continuous(self.rawCapture,
         format="bgr", use_video_port=True)

      # initialize the frame and the variable used to indicate
      # if the thread should be stopped
      self.frame = None
```

```
        self.stopped = False

    def start(self):
        # start the thread to read frames from the video stream
        t = Thread(target=self.update, args=())
        t.daemon = True
        t.start()
        return self

    def update(self):
        # keep looping infinitely until the thread is stopped
        for f in self.stream:
            # grab the frame from the stream and clear the stream in
            # preparation for the next frame
            self.frame = f.array
            self.rawCapture.truncate(0)

            # if the thread indicator variable is set, stop the thread
            # and resource camera resources
            if self.stopped:
                self.stream.close()
                self.rawCapture.close()
                self.camera.close()
                return

    def read(self):
        # return the frame most recently read
        return self.frame

    def stop(self):
        # indicate that the thread should be stopped
        self.stopped = True

#---------------------------------------------------------------------------------------------
```

```
# Currently not used but included in case you want to check speed
def show_FPS(start_time,frame_count):
    if debug:
        if frame_count >= FRAME_COUNTER:
            duration = float(time.time() - start_time)
            FPS = float(frame_count / duration)
            print("Processing at %.2f fps last %i frames" %( FPS, frame_count))
            frame_count = 0
            start_time = time.time()
        else:
            frame_count += 1
    return start_time, frame_count

#---------------------------------------------------------------------------------------------
def get_center(x,y,w,h):
    return int(x+w/2), int(y+h/2)

#---------------------------------------------------------------------------------------------
def cam_shift():
    # Process steam images to find camera movement
    # using an extracted search rectangle in the middle of one frame
    # and find location in subsequent images.  Grap a new search rect
    # as needed based on nearness to edge of image or percent probability
    # of image search result Etc.

    # Setup Video Stream Thread
    vs = PiVideoStream().start()
    vs.camera.rotation = CAMERA_ROTATION
    vs.camera.hflip = CAMERA_HFLIP
    vs.camera.vflip = CAMERA_VFLIP
    time.sleep(2.0)

    # initialize the search window (rect) variables
    if WINDOW_BIGGER > 1:  # Note setting a bigger window will slow the FPS
```

```
        big_w = int(CAMERA_WIDTH * WINDOW_BIGGER)
        big_h = int(CAMERA_HEIGHT * WINDOW_BIGGER)
    sw_w = int(CAMERA_WIDTH/4)    # search window width
    sw_h = int(CAMERA_HEIGHT/4)   # search window height
    sw_buf_x = int(sw_w/4)        # buffer to left/right of image
    sw_buf_y = int(sw_h/4)        # buffer to top/bot of image
    sw_cx = int(CAMERA_WIDTH/2)   # x center of image
    sw_cy = int(CAMERA_HEIGHT/2)  # y center of image
    sw_x = (sw_cx - sw_w/2)       # top x corner of search rect
    sw_y = (sw_cy - sw_h/2)       # top y corner of search rect
    sw_maxVal = MAX_SEARCH_THRESHOLD  # Threshold Accuracy of search in image
    sw_minVal = MIN_SEARCH_THRESHOLD  # Threshold of worst search result in image

    # Grab a Video Steam image and initialize search rectangle
    cam_cx1 = sw_cx
    cam_cy1 = sw_cy
    cam_cur_cx = cam_cx1
    cam_cur_cy = cam_cy1
    image1 = vs.read()   # initialize first image
    img = cv2.imread("cam.jpg")
    search_rect = img[sw_y:sw_y+sw_h, sw_x:sw_x+sw_w]  # Initialize centre search rectangle
    cam_track_cx = 0   # initialize cam horizontal cam movement tracker
    cam_track_cy = 0   # initialize cam vertical cam movement tracker
    while True:
        image1 = vs.read()  # capture a image from video stream thread
        # Look for search_rect in this image and return result
        result = cv2.matchTemplate( image1, search_rect, cv2.TM_CCORR_NORMED)
        # Process result to return probabilities and Location of best and worst image match
        minVal, maxVal, minLoc, maxLoc = cv2.minMaxLoc(result)  # find search rect match in new image
```

```
        # Get the center of the best matching result of search
        cam_cx2, cam_cy2 = get_center( maxLoc[0], maxLoc[1], sw_w, sw_h )
        # Update cumulative camera tracking data
        cam_track_cx = cam_track_cx + cam_cur_cx - cam_cx2
        cam_track_cy = cam_track_cy + cam_cur_cy - cam_cy2
        # Check if search rect is near edges and meets search accuracy threshold
        if not ( maxLoc[0] > sw_buf_x and maxLoc[0] + sw_x + sw_buf_x < CAMERA_WIDTH and
                 maxLoc[1] > sw_buf_y and maxLoc[1] + sw_y + sw_buf_y < CAMERA_HEIGHT
                 and maxVal > sw_maxVal):
            # check value of lowest matching result and reset search rectangle
            if minVal < sw_minVal:
                if debug:
                    print("      Reset search_rect cur_cx=%i cam_track_cx=%i cur_cy=%i cam_track_cy=%i"
                    % (cam_cur_cx, cam_track_cx, cam_cur_cy, cam_track_cy))
                search_rect = img[sw_y:sw_y+sw_h, sw_x:sw_x+sw_w]

                cam_cx2, cam_cy2 = get_center( maxLoc[0], maxLoc[1], sw_w, sw_h )
                cam_track_cx = cam_track_cx + cam_cur_cx - cam_cx2
                cam_track_cy = cam_track_cy + cam_cur_cy - cam_cy2
                cam_cx1 = sw_cx
                cam_cy1 = sw_cy
                cam_cx2 = cam_cx1
                cam_cy2 = cam_cy2
        cam_cur_cx = cam_cx2
        cam_cur_cy = cam_cy2

        if debug:
            #print(" Cam at (%i,%i) cam_track_cx, cam_track_cy" % ( cam_track_cx, cam_track_cy, ))
            print(" maxLoc   maxVal   minLoc   minVal")
```

```
        print maxLoc, "{0:0.4f}".format(maxVal) , minLoc ,"{0:0.4f}".format(minVal)
        if maxLoc[0] > 140:
            print(" Right")
        if maxLoc[0] < 100:
            print(" Left")
        if maxLoc[0] > 100 & maxLoc[0] < 140:
            print(" Center")
    image2 = image1

    if window_on:
        # Display openCV window information on RPI desktop if required
        cv2.imshow( 'search rectangle', search_rect )
        #   cv2.circle(image2,(sw_x+sw_w/2,sw_y+sw_h/2),CIRCLE_SIZE,(0,0,255), 2)
        #      cv2.rectangle(image2,(sw_x,sw_y),(sw_x+sw_w,sw_y+sw_h),(0,0,255), LINE_THICKNESS)
        cv2.rectangle(image2,( maxLoc[0], maxLoc[1] ),( maxLoc[0] + sw_w, maxLoc[1] + sw_h ),(0,255,0), LINE_THICKNESS)     # show search rect
        #      cv2.rectangle(image2,( cam_cx1 - sw_w/2, cam_cy2 - sw_h/2  ),(cam_cx1+sw_w/2,  cam_cy2+sw_h/2),(0,0,255),  LINE_THICKNESS) # show current rect
        m_text = ("CAM POS ( %i %i )   " % (maxLoc))
        cv2.putText(image2,  m_text,  (int(CAMERA_WIDTH/2) - len(m_text) * 3, CAMERA_HEIGHT - 30 ), cv2.FONT_HERSHEY_SIMPLEX, CV_FONT_SIZE, (255,255,255), 1)

        if WINDOW_BIGGER > 1:
            image2 = cv2.resize( image2,( big_w, big_h ))



        cv2.imshow('Cam-Track  (q in window to quit)',image2)
```

```
            if cv2.waitKey(1) & 0xFF == ord('q'):
                cv2.destroyAllWindows()
                print("End Cam Tracking")
                break

#-----------------------------------------------------------------------------------------------
if __name__ == '__main__':
    try:
        cam_shift()
    finally:
        print("")
        print("++++++++++++++++++++++++++++++++++++")
        print("%s %s - Exiting" % (progname, ver))
        print("++++++++++++++++++++++++++++++++++++")
        print("")



import RPi.GPIO as gpio
import time
import sys
import Tkinter as tk
from sensor1 import distance1
from sensor2 import distance2
from sensor3 import distance3
import random

def init():
    gpio.setmode(gpio.BOARD)
    gpio.setup(13, gpio.OUT)
    gpio.setup(15, gpio.OUT)
    gpio.setup(16, gpio.OUT)
    gpio.setup(18, gpio.OUT)

def forward(tf):
```

```
    gpio.output(13, True)
    gpio.output(15, False)
    gpio.output(16, True)
    gpio.output(18, False)
    time.sleep(tf)
    gpio.cleanup()

def reverse(tf):
    gpio.output(13, False)
    gpio.output(15, True)
    gpio.output(16, False)
    gpio.output(18, True)
    time.sleep(tf)
    gpio.cleanup()

def turn_left(tf):
    gpio.output(13, False)
    gpio.output(15, False)
    gpio.output(16, True)
    gpio.output(18, False)
    time.sleep(tf)
    gpio.cleanup()

def turn_right(tf):
    gpio.output(13, True)
    gpio.output(15, False)
    gpio.output(16, False)
    gpio.output(18, False)
    time.sleep(tf)
    gpio.cleanup()

def pivot_left(tf):
    gpio.output(13, False)
    gpio.output(15, True)
```

```
    gpio.output(16, True)
    gpio.output(18, False)
    time.sleep(tf)
    gpio.cleanup()

def pivot_right(tf):
    gpio.output(13, True)
    gpio.output(15, False)
    gpio.output(16, False)
    gpio.output(18, True)
    time.sleep(tf)
    gpio.cleanup()

def check_left():
    init()
    dist = distance1()

    if dist < 15:
        print('too close,',dist)
        init()
        turn_left(3)
        dist = distance1()
        if dist < 15:
            print('too close,',dist)
            init()
            pivot_left(3)
            dist = distance1()
            if dist < 15:
                print('too close,',dist)
                sys.exit()

def check_right():
    init()
    dist = distance2()
```

```
    if dist < 15:
        print('too close,',dist)
        init()
        turn_right(3)
        dist = distance1()
        if dist < 15:
            print('too close,',dist)
            init()
            pivot_right(3)
            if dist < 15:
                print('too close,',dist)
                sys.exit()

def check_front():
    init()
    dist = distance3()

    if dist < 100:
        print('too close,',dist)
        init()
        reverse(1)
        dist = distance1()
        if dist < 100:
            print('too close,',dist)
            init()
            reverse(2)
            dist = distance1()
            if dist < 100:
                print('too close,',dist)
                sys.exit()

def autonomy():
    tf = 0.030
```

```
    x = random.randrange(0,4)
    gpio.cleanup()

    if x == 0:
        for y in range(30):
            check_left()
            check_right()
            check_front()
            init()
            forward(tf)
    elif x == 1:
        for y in range(30):
            check_left()
            check_right()
            check_front()
            init()
            pivot_left(tf)
    elif x == 2:
        for y in range(30):
            check_left()
            check_right()
            check_front()
            init()
            pivot_right(tf)
    elif x == 3:
        for y in range(30):
            check_left()
            check_right()
            check_front()
            init()
            pivot_left(tf)

for z in range(10):
    autonomy()
```

```
import RPi.GPIO as gpio
import time

def distance(measure = 'cm'):
    try:
        gpio.setmode(gpio.BOARD)
        gpio.setup(11, gpio.IN)
        gpio.setup(12, gpio.OUT)

        gpio.output(12, False)
        while gpio.input(11) == 0:
            nosig = time.time()

        while gpio.input(11) == 1:
            sig = time.time()

        t1 = sig - nosig

        if measure == 'cm':
            distance = t1 / 0.000058
        elif measure == 'in':
            distance = t1 / 0.000148
        else:
            print('error')
            distance = None

        gpio.cleanup()
        return distance
    except:
        distance = 100
        gpio.cleanup()
        return distance
```

```
print(distance('cm'))
```