FACE-RECOGNITION BASED SECURITY ROBOT INCORPORATING OMNIDIRECTIONAL VISION

MOHAMED TAHIR AHMED SHOANI

UNIVERSITI TEKNOLOGI MALAYSIA

FACE-RECOGNITION BASED SECURITY ROBOT INCORPORATING OMNIDIRECTIONAL VISION

MOHAMED TAHIR AHMED SHOANI

A thesis submitted in fulfilment of the
requirements for the award of the degree of
Master of Engineering (Electrical)

Faculty of Electrical Engineering
Universiti Teknologi Malaysia

MAY 2015

Dedicated to my parents, family & friends

ABSTRACT

Security robots are gathering an increasing interest as a supplement to current security systems due to their advantages of continuous effective surveillance and low cost of operation. Current security robots, however, have their own limitations which is preventing them from being used on a large scale in real world environment. The main objective of this research is to develop a security robot that can efficiently be used in real-world environments by overcoming the limitations of most security robots, to guard properties and humans from intruders that may compromise the safety of the premises or its inhabitants. The robot developed in this research uses motion detection by image subtraction. When motion is perceived a face detection programme using the Viola Jones method coupled with skin-color-content detection is used to find out whether the motion was caused by a person or otherwise. If a face of a person or more is found, then the face recognition stage is utilized to verify the familiarity of the person(s) detected by using three different face recognition algorithms (Fisher faces, Eigen faces and LBPH (Local Binary Pattern Histogram)) to ensure maximum identification under different pose, facial expression and lighting conditions. If the person(s) is determined to be a stranger, the robot would raise an alarm and navigate towards the subject to authenticate him/her through a password entry check.

ABSTRAK

Robot-robot keselamatan menambahkan minat sebagai penambahbaikan ciri keselamatan yang sedia ada yang disebabkan oleh kelebihan mereka mengawal secara efektif dan kos operasi yang rendah. Robot keselamatan semasa, bagaimanapun, mempunyai kekurangan mereka sendiri yang menghalang mereka daripada diguna secara besar-besaran dalam persekitaran dunia sebenar. Objektif utama kajian ini adalah untuk menghasilkan sebuah robot keselamatan yang berkesan yang boleh digunakan dalam persekitaran dunia sebenar dengan mengatasi kekurangan kebanyakan robot keselamatan, untuk menjaga keselamatan harta benda dan manusia daripada penceroboh-penceroboh yang mungkin menjejaskan keselamatan premis atau penghuninya. Robot yang dihasilkan dalam kajian ini menggunakan pengesanan gerakan oleh penolakan imej. Apabila gerakan dikesan program pengesanan muka menggunakan kaedah Viola Jones ditambah pula dengan pengesanan kandungan warna kulit digunakan untuk mengetahui sama ada pergerakan itu disebabkan oleh seseorang atau sebaliknya. Jika wajah seseorang atau lebih ditemui, maka peringkat pengiktirafan wajah digunakan untuk mengesahkan pengetahuan orang yang dikesan dengan menggunakan tiga algoritma pengecaman wajah yang berbeza (Wajah Fisher, Wajah Eigen dan LBPH *(Local Binary Pattern Histogram)*) untuk memastikan identifikasi yang maksimum di bawah cara yang berbeza, ekspresi wajah dan keadaan pencahayaan. Jika orang disahkan untuk menjadi orang yang tidak dikenali, robot itu akan menghidupkan penggera dan menavigasi ke arah subjek untuk mengesahkan dia melalui entri kata laluan.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AAM	-	Active Appearance Model
ASV	-	Autonomous Sea Surface Vehicle
ATTfa3FR	-	Accumulative Training Time for all 3 Face Recognizers
CIE	-	International Commission on Illumination
CIWS	-	Close-In Weapon System
Cnt	-	Continuous
DoF	-	Detection of Face
DoM	-	Detection of Motion
Dst	-	Distance
ECR	-	Eigen Correct Recognition
ECRP	-	Eigen Correct Recognition Percentage
Err	-	Erroneous
ERT	-	Eigen Recognition Time
ERTPI	-	Eigen Recognition Time Per Image
ETT	-	Eigen Training Time
ETTPI	-	Eigen Training Time Per Image
FCR	-	Fisher Correct Recognition
FCRP	-	Fisher Correct Recognition Percentage
FD	-	Face Detection
FOV	-	Field of View
fps	-	Frames Per Second
FRT	-	Fisher Recognition Time
FRTPI	-	Fisher Recognition Time Per Image

FTT	-	Fisher Training Time
FTTPI	-	Fisher Training Time Per Image
HD	-	High Definition
HMM	-	Hidden Markov Model
HSL	-	Hue, Saturation, Lightness
HSV	-	Hue, Saturation, Volume
ICA	-	Independent Component Analysis
LBPH	-	Local Binary Pattern Histogram
LCR	-	LBPH Correct Recognition
LCRP	-	LBPH Correct Recognition Percentage
LDA	-	Linear Discriminant Analysis
LRT	-	LBPH Recognition Time
LRTPI	-	LBPH Recognition Time Per Image
LTT	-	LBPH Training Time
LTTPI	-	LBPH Training Time Per Image
MOG	-	Mixture Of Gaussian
OTP	-	One-Time-Password
PCA	-	Principal Component Analysis
PDA	-	Personal Digital Assistant
PDBNN	-	Probabilistic Decision Based Neural Network
PIN	-	Personal Identification Number
PTZ	-	Pan-Tilt-Zoom
RF	-	Radio Frequency
RFID	-	Radio Frequency based Identification
RGB	-	Red, Green, Blue
SVM	-	Support Vector Machines
Thld	-	Threshold
VGA	-	Video Graphics Array
XYZ	-	Axis of CIE color space
YCrCb	-	Yellow Chroma-Red Chroma-Blue

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Problem Statement (Problem Background)

Security is one of the main objectives of every creature. Everyone wants to feel safe within their home and work environment. This security requirement increases as the importance of the premises increases. For this reason it is usual to see guards in places such as banks and other establishments of high importance. Security is also highly required in areas and buildings which may be vulnerable to attacks, such as border crossings, military departments and many others.

Security was and in many cases still is, carried out by security personnel which carry out patrols and may use surveillance cameras for monitoring the designated area. This approach, however, has the following disadvantages:

- Guard fatigue which may result in inadequate monitoring or even sleeping on the job.
- Boredom which may result in the guard(s) being busy with something else other than the main objective.
- Intrinsic human inadequacy. It's hard for a human to monitor a number of screens showing video streams from surveillance cameras continuously for a long period of time.

- Ineffectiveness, an intruder may sneak behind a guard or even attack the guard and neutralize him/her towards achieving the intruder's objective.
- High cost of employing many guards to secure the area 24/7.

For these reasons, automated security systems have been sought by security managers to limit or inhibit the disadvantages and weaknesses of human based security systems.

Although automated security systems can be in the form of a laser beam which signals an alarm when it's broken or a similar method of intruder detection, a far better approach would be a mobile robot that has the capacity to monitor an area in all directions continuously and intercept strangers to verify their identity, thereby solving many of the problems mentioned earlier and providing a much better solution to securing a facility using a 'machine + human' approach.

1.2 Research Objectives

The general aim of this research is to design a robot capable of performing security related tasks to help organizations as well as individuals with safeguarding their properties or other important assets with low cost and better efficiency.

The specific aim of this research is to give a security robot several extra capabilities that most current security robots lack such as:

- 360 degree vision and surveillance using a different approach than the catadioptric (fisheye) camera method
- Full HD vision to capture a clearer image of an unidentified person's face.
- Subject tracking, approach and following.
- Subject verification through biometric means and authentication through specific knowledge.

1.3 Research Question

There are many queries as well as difficulties surrounding robots in general, and more specifically to those that are intended for security operations. Questions relating to the nature of operation as well as the extent of autonomy and not least, the real-life effectiveness of such robots are all issues that require answers.

In this project the aim is to answer the following questions:

1- Can a security robot use several wide angle cameras instead of a catadioptric camera to conduct 360 degrees intruder detection in an indoor environment?
2- What threshold value should be used to allow a security robot to detect intruder motion in an indoor environment using a digital camera in different lighting conditions?
3- What is the maximum face detection distance capability of a security robot in well-lit indoor environment using a video camera without optical enhancement gear?
4- How efficiently can a security robot detect human faces in an uncontrolled live indoor environment?
5- What is the best face image dimensions to be used in a security robot for the purpose of face recognition in an uncontrolled live indoor environment.

1.4 Research Methodology

The research methodology used in this project is as follows:

- Review recent and past literature to find out what has been achieved so far and to identify possibilities and limitations of security robots under research.
- Design a robot that possesses the strengths and possibly avoids the limitations of past security robot.
- Plan a method of conducting surveillance and security checks using the added capabilities of the designed robot.

- Test the operation and correct mistakes and errors.
- Improve the operation of the robot as much as possible.
- Verify the robot's efficiency using real-life scenarios.

1.5 Expected Findings

This research is expected to find out the following:

- The possibility and if so, the method of using an omnidirectional vision system employing three wide angle cameras rather than one fisheye camera for the purpose of continuous surveillance of an indoor environment as well as face detection of subjects within the environment at practical distances.
- The possibility of using face recognition efficiently to determine strangers from known individuals by an autonomous security robot using only its on-board resources.
- The possibility and method of building an autonomous security robot that is fast enough to efficiently interact with strangers in real-life environments.
- The possibility of applying speed on a tall robot to conduct its operations and to what extent.

1.6 Scope and Assumptions

The scope of this work is as follows:

1- The Robot:

   a. Autonomous.
   b. Mobile.
   c. Built by candidate student.
   d. Uses electrical dc motors.
   e. Height: minimum 1 meter, maximum 2 meters.

2- The robot's environment:
    a. Indoors.
    b. Unstructured.
    c. Even dry floor.
    d. Lighting is sufficient for a human to see around.
3- Motion Detection:
    a. Detect motion of a human subject.
    b. Use one or more digital cameras.
4- Human subjects:
    a. Height is from 50 to 200 cm.
    b. Not putting excessive makeup.
    c. Not wearing a mask or other face occluding / altering objects.
    d. Approach the robot in a normal walking manner.
    e. Subject distance can be 1 – 7 meters from the robot's camera.
    f. Subject color and clothes color is different from background color.
    g. Subjects are capable of hearing and seeing.
    h. Subjects can read and write English.
    i. Subjects have used a computer in the past.
5- Face Detection:
    a. Live motion.
    b. Uncontrolled (the subject can be anywhere within the camera's view).
    c. Subjects' faces may be slightly facing right or left (a few degrees).
    d. Subjects' faces may be slightly tilted clockwise or counterclockwise (a few degrees).
    e. More than one face can be present in the camera's image.
    f. More than one face can be detected in a single video frame image.
6- Face Recognition
    a. Face images should be gray colored.
    b. Face images are cropped from one or two sides to decrease background.
    c. Face images sharpness should be acceptable (moderate to high).
    d. Faces can be tilted and/or rotated by a few degrees.
    e. Use one or more face images.

f. Use three face recognizers.
g. A member should have a minimum of two different face images in the database to start with.

7- Face Tracking
   a. The face is detectable by the robot (see 'Face Detection' above)
   b. Subject face movement is moderate or slow.

8- Navigation
   a. The target (subject) is detectable by the robot's camera's (see 'Face Detection' above)
   b. The target's (subject's) face is traceable by the robot's camera's (see 'Face Tracking' above)
   c. The floor surface is suitable for the robot's movement.
   d. Obstacles are detectable by the ultrasonic sensors.

9- Obstacle Avoidance
   a. Obstacles are usual objects found in an office or lab environment.
   b. Obstacles can be static or dynamic.
   c. Obstacles may not change position or orientation at high speed.

10- Authentication
   a. Members know their passwords.
   b. Members do not share their passwords with strangers.
   c. Members do not reveal methods to overcome the robot's security.
   d. Strangers obey the robot's voice commands.
   e. Administrators can hear (or become notified of) the robot's alarm when it goes off.

1.7 Structure of Thesis (Outline of thesis)

This thesis is divided into five chapters. The first chapter is the introduction which has covered topics related to the research objectives, methodology and expected findings. The second chapter presents a literature review of past and recent research done on security robots and their operation as well as the two employed methods of authentication. The third chapter outlines the methodology used in

implementing the robots operation. Chapter four presents the different experiments that were conducted and discussion of the results obtained. Finally, chapter five presents a conclusion on the work carried out and possible future work.

CHAPTER 2

LITERATURE SURVEY

Development of a security robot is well covered in the literature. Many types of security robot with different capabilities have been built and tested inside and outside university laboratories. In this chapter we will try to give an overview of current and past research that is directly related to this project.

Section 2.1 outlines the types of security robots. Subsection 2.1.1 gives an overview of teleoperated security robots, while subsection 2.1.2 covers autonomous security robots, and finally, subsection 2.1.3 provide an overview of hybrid security robots. These subsections provide a deeper look at the capabilities and limitations of each of the three reviewed types of security robots when equipped with normal and omnidirectional vision capability.

Section 2.2 outlines the approach used in conducting surveillance operations. Subsection 2.2.1 gives a general overview of surveillance conducted by security robots. Subsection 2.2.2 provides an outline of cameras used in surveillance and monitoring operations. Subsection 2.2.3 provides an overview of camera usage in security systems and security robots, and subsection 2.2.4 provides a deeper look into surveillance operations conducted by security robots.

Section 2.3 outlines the approach used in detecting motion. Section 2.4 provides an overview of face detection methods. Subsection 2.3.1 takes a closer look at skin-color based face detection, while subsection 2.3.2 provides an overview of the Viola-Jones method for face detection. Section 2.5 outlines face recognition as well as the methods used in OpenCV to accomplish it.

Section 2.6 covers authentication. Subsection 2.2.1 reviews biometric baased authentication methods. Subsection 2.2.1.1 takes a deeper look at face recognition, which is one of the biometric methods used for authentication. Subsection 2.2.2 reviews knowledge based authentication, while its subsection 2.2.2.1 takes a deeper look at password authentication which is used in this system.

2.1 Security Robots

Many types of security related robots are mentioned in the literature, each with its own strengths and weaknesses. In general, security related robots can be classified into three main categories:

- Teleoperated security robots
- Autonomous security robots.
- Hybrid security robots.

The following subsections offer an overview of the robots and research done in these fields.

2.1.1 Teleoperated Security Robots

Teleoperated security robots are remotely controlled to carry out many different operations such as surveillance, inspection, hostage retrieval and other law-enforcement operations [1].

Human teleoperated robots come in a large variety. They are used for many different purposes such as inspection, bomb disposal, as well as other operations with high risks [1]. The main two reasons for using teleoperated robots rather than autonomous robots are [1][2]:

1. The utilization of artificial intelligence may result in an emergent response when an autonomous robot is programmed with an unconstrained learning algorithm. Consequently the robot may exhibit a non-deterministic response and take a critical decision, such as shooting a human with a weapon.
2. An environment is too unstructured, unpredictable or complex to be modelled efficiently and as such a robot may not be controlled effectively.

Teleoperated robots have some advantages [3] such as:

1. Getting humans out of harm's way, such as in hazardous or polluted areas.
2. To access areas unreachable by humans for any reason.

However, the disadvantage of teleoperated robots is that they do not operate on their own unless equipped with some degree of autonomy, and therefore would be useless in helping a human unless another human controls them to do so. Teleoperated robots may be equipped with a normal or an omnidirectional camera.

2.1.1.1 Teleoperated Security Robots without an Omnidirectional Camera

There are many types of teleoperated robots that do not employ an omnidirectional camera. However they use many different sensors to perform their operation such as color cameras, thermal cameras, ultrasonic sensors and others.

In [4] and [5], models of teleoperated robots which utilize a range of different sensors are offered to carry out diverse missions such as surveillance, explosive ordinance disposal, vehicle inspection, and route clearance. The different models have different capabilities depending on their design and sensors they employ.

In [6] a ball shaped robot is designed and used for surveillance and rescue operations. It has video transmission capability, and due to its unique design, it can be maneuver to narrow places and, hence, can also be used to find survivors in rescue operations. In [7] a robot is used to scan an indoor location such as a house, gather images, and send them to a website (where a user can see and use the images) and can be controlled through a browser window through the internet and in [8] a remote controlled robot is used to scan an area for fire or intrusion. The robot is equipped with camera to send back video and a manipulator for firefighting purposes.

In [9] a Packbot Scout robot's operation and utility is tested by a SWAT unit in their operations. It can be used for scanning certain site as well as conduct negotiations with suspects or people inside, while in [10] the prospects of using natural language to communicate with and supervise a robot is investigated. The robot is proposed to be used by military and security personnel in their operations. In [11], a robot has a different 'unique' design. It can transform from a vehicle to a walking robot. When the terrain is even, the vehicle mode can be used to move fast with less energy, while the walking mode can be invoked to navigate a rough terrain. The robot is teleoperated and is equipped with a camera to enable the user to view the robot's environment.

In [12], a teleoperated, semi-autonomous robot is equipped with a camera and a GPS and can be used to conduct surveillance of an outdoor environment. The robot's six-wheel design enables it to traverse the rough terrain of the outdoor environment. The robot can be controlled wirelessly and instructed to move to any location; it then autonomously navigates the terrain and avoids obstacles to reach the specified location. In [13] another semi-autonomous robot is used to monitor an indoor environment. Again, the operator can set a target location, and the robot will move to that location while avoiding obstacles in its path. The robot is equipped with smoke and fire sensors and can notify the operators of such events during its operation. The operators can also use the onboard camera to monitor the robot's environment.

In [14][15][16] a security robot can be controlled using a CDMA based mobile phone. The system allows the user to view images from the camera onboard

the robot as well as command the robot's movements, while in [17] the robot is remotely controlled over the internet by security personnel. The robot streams the camera view to a server which performs identification, and if an intruder is detected a message is sent to the operator as well as the intruder.

In [18] a robot equipped with a wireless camera is controlled with a joystick to approach desired locations. The robot is equipped with an ultrasonic sensor which stalls the robot for five seconds when an obstacle is encountered allowing the user to respond afterwards by moving the robot away from the obstacle. The camera is used to capture images of any location or event in the environment.

In [19] a quad-copter is equipped with a camera and used to track people by extracting body information. The problem with such robots is the limited payload capability resulting in reducing the usefulness of the acquired information while requiring high operation costs in terms of energy spent on flying and controlling the platform.

In general, teleoperated robots may not employ omnidirectional vision as they may not need it for their mission, however, adding such as sensor may endow the robot and/or user with extra and improved capabilities to carry out the mission at hand.

2.1.1.2 Teleoperated Security Robots with an Omnidirectional Camera

In [20] a robot equipped with an omnidirectional camera is used for fire and intruder detection. When a fire or intruder is detected, the PC alerts the owner by sending an SMS to his/her mobile phone as well as posting the captured images on website. Although the proposed system can detect fire and alert the user, it does not take any other action, which is a very important point in case of a fire. Also, the robot is controlled by a PC which makes it vulnerable in locations where the signal is lost or damaged by noise. Finally intruder detection is done by background extraction. While being an effective method for stationary robots and fixed cameras,

it may be less robust on a mobile robot as the background would change constantly during the robot motion.

2.1.2 Autonomous Security Robots

Autonomous security robots have the advantage of performing their tasks without human intervention, thus alleviating the burden on humans by repetitive chores or lengthy operations hours. An excellent example of such robots are those used in security, as these robots do not require sleep and do not lose focus due to extended working hours. In the following subsections, we will examine robots without an omnidirectional camera as well as those employing an omnidirectional camera in their operation.

2.1.2.1 Autonomous Security Robots Utilizing a non-Omnidirectional Camera

Many security robots perform their operation without utilizing an omnidirectional camera. They depend on other sensors to carry out their tasks such as pan-tilt-zoom (PTZ) cameras, thermal cameras and other sensors.

In [21] and [22], multisensory fusion is used for environment monitoring, motion detection and surveillance using a mobile robot equipped with a number of sensors. In [23] a set of three robots, each equipped with a CCD camera, ultrasonic sensor, fire-detecting module and an RFID based localization unit, are used to detect fire, gas and radiation. When detected, the robot sends a message to a web-control center and another robot comes to confirm the accident.

In [24] a sensor network is used to detect intruders. When an intruder is detected, the robot is instructed to jump towards the sensor location and start taking photos once it gets there. The photos are sent to a gateway which in turn sends them to the user.

In many systems, a simple autonomous robot is used to monitor an environment and report any abnormal events such as a sound, motion, fire or other. Examples of such robots are exhibited in [25][26][27][28][29][30][31][32] .In [25] and [26] a robot uses a camera as well as a set of microphones to detect abnormal activity such as fighting, running or other behaviors that may produce certain sounds inside a home, verify the source of the sound and report an image to the master's mobile phone. In [27] a robot uses a microphone as well as a networked camera to detect intruders and report to a local monitoring station. It can also report to a remote monitoring station using the internet. In [28], [29], [30] and [31] the robot is also used to monitor an indoor location using a camera or other sensor and when an intruder or other event is detected the user is notified through an SMS or email, and in [32] a dog-shaped robot was tested in a supermarket environment to interact and report accidents and events to its supervisor center.

In [33] two robots, one fully autonomous and one partially autonomous were used to patrol an indoor environment. The autonomous robot is equipped with an IR sensor for detecting intruders and RFID scanner to verify them. If the intruder fails to present a correct RFID tag, then the robot will attack the intruder, otherwise it will continue its normal operation.

In [34] a robot operates autonomously to map and then navigate the environment. The robot utilizes three-layer control architecture for primitive behaviors, complex tasks and control algorithms. The robot is programmed to detect missing object and detect intruders, when an intruder is detected, the robot will follow him/her until the intruder stops or disappears from the scene.

In [35] and [36] a robot with a thermal sensor is used to detect the presence of a person, and another camera is used for the face detection and recognition purposes and in [37] a robot navigates a pre-determined path and uses a camera to recognize the face of an intruder and an alarm is raised if the face is not recognized.

In [38] an autonomous system incorporating a camera-on-rail is used to monitor an indoor environment and detect intruders. The merit of such a system is that it does not require any obstacle detection or complex navigation capabilities as

well as being non-obstructive to humans. The disadvantage however, is the inability to track a subject beyond the limits of the rails being used.

In [39] a mobile platform is controlled using a number of stationary cameras. The direction of motion is decided based on the location of the person to be tracked or followed and the presence of an obstacle which is detected by the robot. The problem with this system is that it will be incapable of tracking a subject beyond the scope of the stationary security cameras.

Although it is possible to design and use a robot without employing an omnidirectional camera, it is obvious that such a sensor would increase the robot's awareness of its surrounding and possibly reduce the number of sensors onboard or exterior to the robot, as well as improve the robot's ability to track a subject and apply suitable navigation and obstacle avoidance methods.

2.1.2.2 Autonomous Security Robots Utilizing an Omnidirectional Camera

Robots that utilize an omnidirectional camera are able to detect motion or environment variations in any direction without having to turn or maneuver. In [40] five different types of sensors were used; a video motion detector, a passive infrared array, an acoustic sensor array and an ultrasonic array. Each sensor arrays covered 360 degrees and was independent of other sensor arrays. The information from the sensors was fused to reach a conclusion of an intruder detection, which would result in raising an alarm. Although the authors report a 99% success rate in detecting intruders, the presented design falls short of recognizing the person's identity. An intruder can be one of the security personnel happening to pass by for any reason.

In [41] an autonomous sea surface vehicle (ASV) uses a set of 6 cameras to capture a 360 degree view to monitors an area searching for other vessels and determine whether it's adversarial. Such a sea robot (in the form of an autonomous ship) is highly preferential as it relieves service personnel from enduring different conditions at sea as well as being on guard 24/7. The efficiency of such a platform, however, is highly dependent on the sensors provided and the algorithms used.

In [42] a robot is trained manually to follow a certain route, during which it registers the environment by capturing a panoramic image every few centimeters. Later, the robot follows the same path and compares the environment images captured by the camera with those previously stored, if a difference (anomaly) is detected, then an alarm is raised to alert a security person to intervene. This approach would be robust in situations where the environment does not change such as a night shift at a museum, library or office, but would be unsuitable for situations where the environment is volatile due to the addition or removal of any item within, or even the passing by of a security worker for any reason.

2.1.2.3 Comparison of Autonomous Security Robots

To better highlight possible research gaps, Table 2.1 provides a general comparison between the robot presented in this work (highlighted in blue) and previous robots mentioned in the literature.

Heading Description

Ref. No.	:	The reference number in which this robot was mentioned
Robot Name	:	The name of the robot; if any.
Area of Operation	:	Indoors, outdoors, underwater, factory … etc.
Vision Coverage	:	The area covered by the robot's vision system: e.g.: 90, 180, or 360 degrees.
Size	:	Small (5 - 35), Medium (36 - 75), Large (76 - 175) Huge (176 - 300), Gigantic (over 300) cm
Speed of Operation	:	Fast, slow, or the actual speed if mentioned in the literature.
Sound Det.	:	Sound detection capability.
Motion Det.	:	Motion detection capability.
Face Det.	:	Face detection capability.
Face Rec.	:	Face recognition capability.
Authent.	:	Intruder authentication / verification capability.
Sensors Used	:	The sensors used by the robot
Research Gap	:	What is the research gap if any?

Table 2.1: Comparing the robot in this work with robots mentioned in the literature

Ref. No.	Reference Title	Robot Name	Area of Operation	Visoion Coverage (Deg)	Size	Speed	Sound Det.	Motion Det.	Face Det.	Face Rec.	Authent.	Sesors Used	Problem / Research Gap
-	**Face Detection Based Autonomous Security Robot Incorporating Omnidirectional Vision**	**ASR 1.0**	**Indoors**	**360**	**Large**	**Fast**	**No**	**Yes**	**Yes**	**Yes**	**Yes**	**Camera, Ultrasonic**	
21	Multi-sensor surveillance of indoor environments by an autonomous mobile robot	Robot Platform	Indoors	<= 90 *	Large	Slow *	No	Yes	No	No	No	Monocular camera, Laser scanner, Encoders, and RFID Devices	1- Expensive sensor: Laser 2- Monitor specific locations rather than the whole area
22	The development of a general type of security robot	-	Indoors	<= 90 *	Medium	Slow *	No	Yes	No	No	No	Smoke, Humidity , Flame, CO, Temperature, infrared, Ultrasonic, Dual-Axis accelerometer, Micro-Gyro, Camera, Audio & Encoders	Uses wall following algorithm, hence may not be suitable for non-structured rooms.
23	Design and implementation of sensor fusion based behavior strategies for a surveillance and security robot team	SSR	Indoors	189 **	Medium	Slow *	No	No	No	No	No	RFID Devices, Electronic compass, Ultrasonic, Flame, Temperature, and Gas	1- Wall following model. 2- RFID localization 3- No intruder detection
24	An indoor security system with a jumping robot as the surveillance terminal	Jumping Robot	Indoors	<= 90 *	Small	Slow *	No	Using off-board sensors	No	No	No	Camera, PIR, Angle, Infrared	Jumping may not be practical with the presence of humans or fragile equipment onboard or off-board the robot.
25	Surveillance Robot Utilizing Video and Audio Information	Surveillance Robot	Indoors	<= 90 *	Medium	Slow *	Yes	Yes	No	No	No	Camera, Microphone	No obstacle avoidance has been mentioned in the paper
26	Intelligent household surveillance robot	Surveillance Robot	Indoors	<= 90 *	Medium	Slow *	Yes	Yes	No	No	No	Camera, Microphone	No obstacle avoidance has been mentioned in the paper
27	A design of mobile robot based on Network Camera and sound source localization for intelligent surveillance system	Mobile Robot	Indoors	<= 90 *	Small	Slow *	Yes	Yes	No	No	No	Camera, Microphone, Ultrasonic,	Does not distinguish humans from non-humans
28	A surveillance robot with human recognition based on video and audio	Surveillance Robot	Indoors	<= 90 *	Medium	Slow *	Yes	No	Yes	Yes	No	Camera, Microphone, Infrared, Smoke, and CO	1- Subject detection distance is not mentioned. From images in the paper, it appears to be three meters or less. 2- The calculations are possibly done off-board the robot.
29	Ensuring security in a closed region using robot	Hector	Indoors	<= 90 *	Small	Slow *	Yes	Yes	No	No	No	Camera, Microphone, Ultrasonic,	Does not distinguish humans from non-humans
30	GPRS Based Guard Robot Alarm System Design	Autonomous Robot	Indoors	<= 90 *	Medium	Slow *	No	No	No	No	No	Camera, Encoders	Robot only reports to user; no action is taken.
31	Development of user-friendly intelligent home robot focused on safety and security	Home Robot	Indoors	-	Small	Slow *	No	Yes	No	No	No	PIR, RFID, Infrared, Ultrasonic, Gas and Fire	1- Wall following model. 2- RFID localization
32	Development of a mobile platform for security robot	Mastiff-I	Indoors	<= 90 *	Medium	Slow *	No	Yes	No	No	No	Camera, Microphone, Ultrasonic, Infrared, Compass, Smoke, and Encoders	1- Low resolution camera 352 x 288 2- Does not distinguish humans from non-humans.
33	Student Projects; Security Robot Design	Robot - A	Indoors	-	Small	Slow *	No	Yes	No	No	Yes	Infrared, Sonar (Ultrasonic)	1- Does not distinguish humans from non-humans. 2- What if a member lost or forgot to carry the RFID tag ?
34	An autonomous mobile robotic system for surveillance of indoor environments	Robot Platform	Indoors	<= 90 *	Large	Slow *	No	Yes	No	No	No	Monocular camera, Laser scanner, Encoders, and RFID Devices	1- Expensive sensor: Laser 2- Monitor specific locations rather than the whole area
35	Active people recognition using thermal and grey images on a mobile security robot	Peoplebot	Indoors	<= 90 *	Large	Slow *	No	No	Yes	Yes	No	Normal camera, Thermal camera, Laser range finder,	1- Uses a thermal camera and color camera combination to track people. 2- Low resolution 320x240 3- No navigation is mentioned
36	People tracking by mobile robots using thermal and colour vision	Peoplebot	Indoors	<= 90 *	Large	Slow *	No	No	Yes	Yes	No	Normal camera, Thermal camera, Laser range finder,	The research is mostly aimed at tracking people using a combination of thermal and color camera.
37	Vision Based Mobile Robot for Indoor Environmental Security	WITH	Indoors	<= 90 *	Small	??	No	No	Yes	Yes	No	Camera	The research is focused on face recognition. The robot is given a route to follow. It is anticipated that either encouders or time delays were used, as the method used for controlling the robot's path is not mentioned
38	Flow-based Motion Perception Technique for an Autonomous Robot System	EEyeRobot	Indoors	<= 90 *	Small	Slow *	No	Yes	No	No	No	Camera	It's a camera on a rail with motion detection
39	Automated Surveillance Systems with Multi-Camera and Robotic Platforms	-	Indoors	360 ***	Medium	Medium	No	Yes	No	No	No	Laser scanner, Web camera, Omnidirectional camera, PTZ camera,	1- The robot uses a laser scanner (expensive) for obstacle avoidance. 2- The calculations are possibly done off-board the robot.

Legend

* Not mentioned, therefore anticipated based on the robot design

** Pan & Tilt, not a built in viewing angle

*** Omnidirectional camera

2.1.3 Hybrid Security Robots

In [43] a simple robot is used to monitor an environment. In autonomous mode, the robot wanders about the environment while avoiding obstacles, and in manual mode, a user can move the robot in any desired direction while viewing images from the onboard camera.

In [44] a robot can autonomously navigate the environment and report a fire or intruder event to a mobile phone or client computer. The robot also has two modes of manual control; direct control and behavior control, in which the user can control the robot over the internet. In [45] the same robot is used to control appliances using and RF module, and in [46] another robot is similarly used to detect intruders and capture images of the intruder. The user can also manipulate the robot over the internet through a wireless network and control electrical devices at home using an RF transmitter and receiver.

In [47] and [48], a service-oriented architecture is used to design and implement a system and three robots. The robots can autonomously navigate to map the environment. Once mapping is done, the robot will navigate the environment to detect the presence of new objects or intruders. When a new object is detected the robot sends a warning signal to the control center, and when an intruder is detected, the robot moves towards the intruder and allows him/her to enter the correct passcode, failing which will trigger the alarm.

In [49] and [50] a sensor network incorporating different types of sensors is used to detect fire, intrusion and other events. When an event is detected by the one or more sensors, the robot is moved to the event location using triangulation and dead-reckoning using cricket nodes fixed to the ceiling. After arriving at the event location, the robot starts transmitting images to the server which in turn forwards them to the user. When the user is notified, he/she has the ability to give instructions to the system. The paper, however, does not outline what commands the user can give to the system. In [51] a number of sensor nodes are also used to detect events. When an event is detected, the robot navigates to its location and transmits images to

the user. The paper emphasizes on the robot design rather than the security operation of the robot.

Many surveillance robots have been mentioned in the literature. These robots take on the general action of ‘Monitor & Report’. They work autonomously and when an event occurs, the user can command the robot with varying degrees of control. In [52] a robot is equipped with a robotic arm as well as several sensors to detect fire, smoke or gas leak. The robot monitors the environment and sends a text message to the user when an event occurs. The user can then control the robot using his/her mobile phone. In [53] a robot is designed to detect intruders using a body sensor and in [54] and [55] a home security robot is used to detect events such as a fire, gas leak or intruders. In all of these robots, when an event is detected an alert is sent over the internet and the user can also control the robot over the internet through a user interface showing the camera image and the sensor readings for [53] and [54]. The robot in [55], however, has no camera installed. In [56] , the robot is provided with face detection capability in order to track the face of an intruder and possibly follow him/her. The user would be notified of the intrusion and images would be sent to the user’s mobile phone or PDA.

In [57] a security robot following a predesigned route is assisted by two fixed cameras to monitor a location and notifies the user through MSN or Facebook when an intruder is detected. The robot can also be controlled via a smart phone or website over a Wi-Fi network to move to a desired location to acquire real-time images of a certain view. In [58] a system of surveillance cameras and mobile robots are deployed to protect an oil facility in South Korea. The system consists of stationary robots for surveillance and tracking as well as mobile robots for surveillance over large fields. The mobile robots can patrol a specific area autonomously. When flame or gas leak is detected, the operators are notified; the operators can then control the mobile robots from their consoles.

In [59] and [60] RFID tags are used to localize the robot which follows a pre-determined path and uses two cameras to monitor a home or another location. If an intruder is detected, the robot notifies the user through MSN and SMS. The user can

use his mobile phone to navigate the robot a particular destination to verify the intruder or for any other purpose.

In [61] a robot system is proposed in which RFID tags are used to identify authorized personnel which are allowed to control the robot. The robot is equipped with a normal camera for daylight operations and an infrared camera for night time operations. The robot has an autonomous and a manual mode of operation, in which it can be controlled using RF signals.

In [62] a robot is equipped with a video camera, a microphone, a laser scanner, a motion sensor and a bumper. The robot localizes itself using triangulation based on two IR tags mounted on the ceiling and an onboard infrared sensor. The robot can either work autonomously or be controlled by user using a smart phone.

In [63] a spherical robot is equipped with two cameras that provide 360 degree vision. The robot is equipped with an ‘Adjustable Autonomy’ control architecture which allows changing the level of autonomy the robot is equipped with. The robot can be used for security surveillance, human search and rescue as well as disaster area inspection.

Autonomous robots with manual override capability provide a good option for security operators to intervene when a situation requires their attention or a certain decisive action, however the main drawback being the possible hostile takeover by an adversary (hacking) resulting in a neutralization or even using the robot against its own purpose.

2.2 Surveillance

The surveillance operation is the main task of the security robot and it encompasses many operations performed by the robot components.

2.2.1 Overview

Surveillance and/or patrolling operations are the major tasks carried out by many security robots. Depending on the robot model, one or more of these tasks may be carried out continuously until a certain action halts the operation.

Robot surveillance operations in general involve monitoring a certain location without changing a robot's position. Once an event has occurred then a certain preprogrammed action is taken accordingly. Actions may include sounding an alarm or notifying the owner or security personnel [25][26][27][28][29]. In some cases other retaliatory actions may also be considered [33].

In the system of the current project, the security robot performs surveillance operations as a precursor and originator of other action to be taken accordingly. In the following sections, the surveillance methodology of the security robots used in this project will be described and details will be given about its operation.

2.2.2 Cameras

In any surveillance operations, cameras would be used to monitor different locations for possible events such as in [58] and [39]. The number of cameras used can be one or more depending on the system specifications and required area coverage.

Low cost security robots usually utilize webcams for their operation as they cost less than specialized security cameras and are easier to recognize and setup in

the system. The number of cameras used by the robot depends on the task to be accomplished and the approach followed for completing that task.

Surveillance cameras can be categorized into many different categories [64][65], such as:

- Static or dynamic: Static cameras are fixed, while dynamic cameras can usually able to rotate around any or both of two axes giving the ability to pan, tilt and possibly zoom in on the target.
- Closed circuit or Networked: Closed circuit cameras cannot be accessed from any entity outside the surveillance system, while networked cameras can be accessed by anyone having the right (or ability) to do so.
- Analog or digital: Analog cameras are usually of lower video and image quality (DVD quality at best) compared to digital cameras which can be Megapixel cameras (e.g. 5 MP) with full HD video (1920x1080p) capability.

2.2.3 Camera Usage

In standard security systems, cameras are used by security personnel to monitor the area for any events that require their attention and/or intervention. In robotic systems, cameras are mostly used to monitor a particular area, while intervention is left to the security personnel in most cases [58].

Security robots usually use one camera, which in some designs has the ability to rotate around two axes giving it the ability to pan and tilt [13] or can be omnidirectional [42]. Other robot models may employ two cameras to attain stereo vision [32]. However, robots employing more cameras have also been designed using up to six cameras for 360 degrees of vision for continuous surveillance in all directions [41].

In robots, cameras are not only employed for security purposes; rather, their usage can be extended for any of the following tasks:

- Motion detection.
- Face (or object) detection.
- Face (or object) tracking.
- Face (or object) recognition.
- Localization.
- Mapping.
- Other actions related to computer vision.

2.2.4 Surveillance Operations with Robots

Surveillance operations involve the monitoring of an area for any event which may be of interest to the entity performing the surveillance operation. Usually it involves guarding a facility against intruders or unauthorized people who may not be allowed to exist inside.

Surveillance started first with human guards only. Cameras were later introduced to enable monitoring several locations simultaneously thereby reducing the number of guards needed to cover a large area. Finally with the advent of security robots, the interest now is to use these to replace most of the security guards due to the many traits of robotic systems:

- Robots do not become tired or sleepy.
- Robots may have capabilities which humans don't such as seeing in total darkness or seeing using an infrared camera.
- Robots can perform in high risk areas in which humans may prefer to stay away from.
- Robots do not take leave, require salary or pension.

However, autonomous robots used in the security field, have largely been used for surveillance operations as they are not well equipped to handle diverse situations; for example, a robot would have a hard time discriminating an enemy pointing a weapon at it from a little girl pointing an ice cream cone at it [66]. This is

due to the lag of artificial intelligence resulting in the possibility of taking the wrong decision by a robot in a fatal situation. That being said, autonomous military robot do exist such as the US Navy's Phalanx CIWS [2] which are programmed to be autonomous in taking action against the enemy.

2.3 Motion detection

Motion detection in general can be accomplished using many different sensors, such as microphones to detect sound of moving objects [25], radio frequency energy such as in radars [67][68], infrared sensors [69], vibration sensors [70][71], magnetic sensors such as in vehicle detection and classification for traffic measurement [72] and cameras [38][73][74].

For security-robots operating indoors, visual motion detection using a simple camera and some computer vision technique is a viable low cost solution as it can be simply set up and used effectively to detect motion within the robot's environment.

In computer vision, motion detection generally involves sensing a change in the perceived camera image which may be interpreted as an object motion. Needless to say that not all changes in the perceived image are indicative of object motion, hence, other steps must be taken afterwards to verify the source of change.

In this system, the reasons for using a camera do detect motion rather than any of the other sensors are the following:

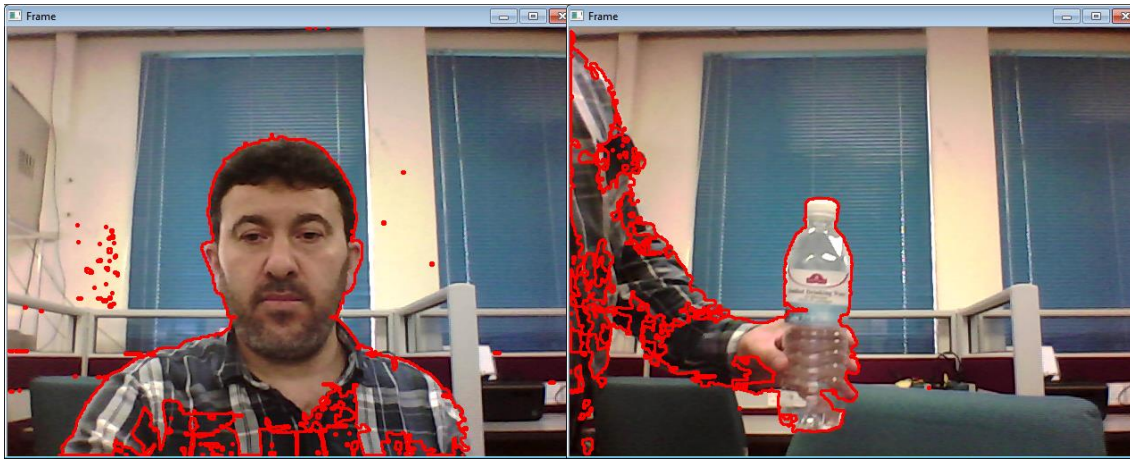
- Need to detect motion indoors rather than outdoors.
- Need to detect motion a far (up to ten meters) and at close range.
- Need to distinguish different object at any visible distance.
- Need to detect object that do not produce sound.
- Need to be stealthy.
- Need to be fast.
- Need to have high resolution motion detection.

- Need to determine precise location.
- Need to be low cost.

Using a small number of cameras can achieve all the above requirements.

2.3.1 Motion Detection in OpenCV

OpenCV is an open source software library that is dedicated towards computer vision functions [75]. In version 2.4.8 of this software library (released in 2014), motion detection is usually done using the 'BackgroundSubtractorMOG' function which is based on [76]. There are many implementations around the web. One of these implementations was tested as shown in Figure 2.1 below:



(a) (b)

Figure 2.1: Motion detection using the 'BackgroundSubtractorMOG' of OpenCV

As can be seen from Figure 2.1, OpenCV's 'BackgroundSubtractorMOG' method has the advantage of pinpointing the exact motion source; a person or otherwise; which is very important in surveillance systems, however, as can be seen in Figure 2.2, it has the following disadvantages:

- Slow: 2.5 frames per second or less per camera. This results in less than 1 frame per second in three cameras.
- Is not suitable for mobile robots as the background has to remain static for a period of time in order to detect motion.
- Any motion during the 'history' buildup of the 'background' image results in less-effective motion detection.
- Any lighting difference during the 'history' buildup of the 'background' image also results in less-effective motion detection.

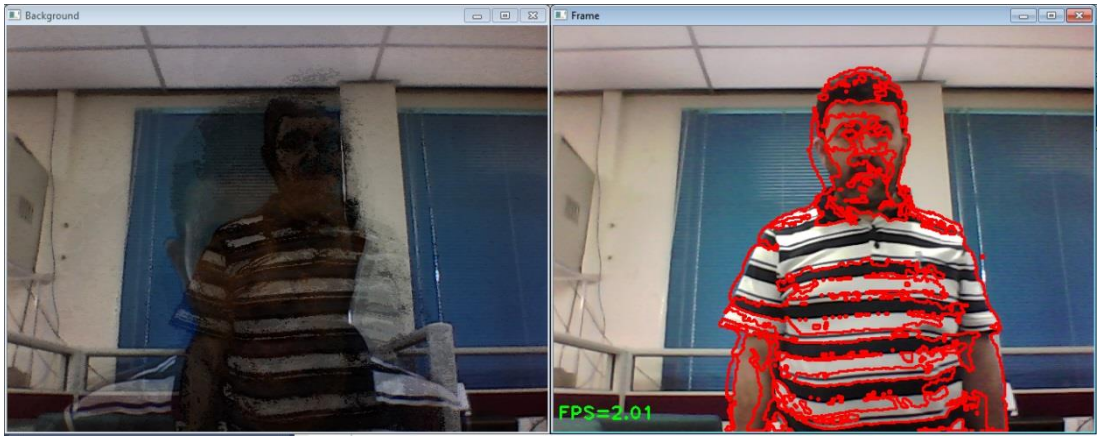


Figure 2.2: Shadows and low FPS when using OpenCV's 'BackgroundSubtractorMOG'

The above disadvantages make OpenCV's 'BackgroundSubtractorMOG' of limited usefulness in autonomous mobile surveillance robots. In such systems, the required solution must satisfy the following criteria:

1- Can be used with low resources.
2- Must be fast.
3- Must be robust against lighting changes.
4- Must be robust against noise.
5- Can quickly and adequately handle relocation of observing platform.

2.4 Face Detection & Methods

Face detection is a challenging problem. Not least due to the complexity of human faces in terms of definition by standard shapes such as circles, ovals, rectangles, triangles …etc. Some of the problems associated with recognizing a human face can be summarized as follows [77]:

- Posture: A person's pose towards the camera can be in any angle.
- Structural Components: A person may have beard, mustache, glasses …etc.
- Facial Expression: A person's facial expression such as smiling, frowning ...etc., affects the appearance (and sometimes the geometry) of the face.
- Occlusion: Partial occlusion by other faces or objects changes the outline of the face.
- Orientation: A face may be turned clockwise or counterclockwise by an angle, although normally it would be less than 90 degrees.
- Other conditions: Lighting, camera characteristics (such as sensitivity to light and colors …etc.) may have a large effect on how a face looks in an image.

For the reason that a face image may be hard to identify, many methods have been proposed to solve this problem [78], some with higher degree of success others with lower computational requirements. Depending on the technique used, face detection methods can be classified according to one or more of the following approaches [77][78][79]:

- Knowledge Based face detection.
- Feature Based face detection.
    - Facial features.
    - Texture.
    - Skin-color.
- Template matching face detection.
    - Predefined templates.
    - Deformable templates.
- Appearance based face detection.

- Eigen-faces.
- Distribution-based methods.
- Neural networks.
- Support Vector Machines.
- Sparse network of Winnows.
- Naïve Bayes classifier.
- Hidden Markov Model.
- Information-Theoretical Approach
- Inductive learning.

The above methods not only differ in their approach, but also in their performance and requirements. While some employ rules for detecting and/or localizing a face in an image, such as the Knowledge based and Feature invariant approaches, others, such as the Appearance based methods require training images which captures the facial appearance variability in order to detect the faces in an image.

While these methods have been used by different systems, our focus will be mainly on two approaches:

- Skin-color based face detection.
- Viola-Jones method for face detection.

The above methods are widely used, the first one for its simplicity and the second one for its robustness to illumination and color variations.

2.4.1 Skin-Color Based Face Detection

In many face detection systems, skin-color is employed to filter out non-skin colored regions of the image, thus obtaining only the face part [79][80], as can be seen in Figure 2.3.

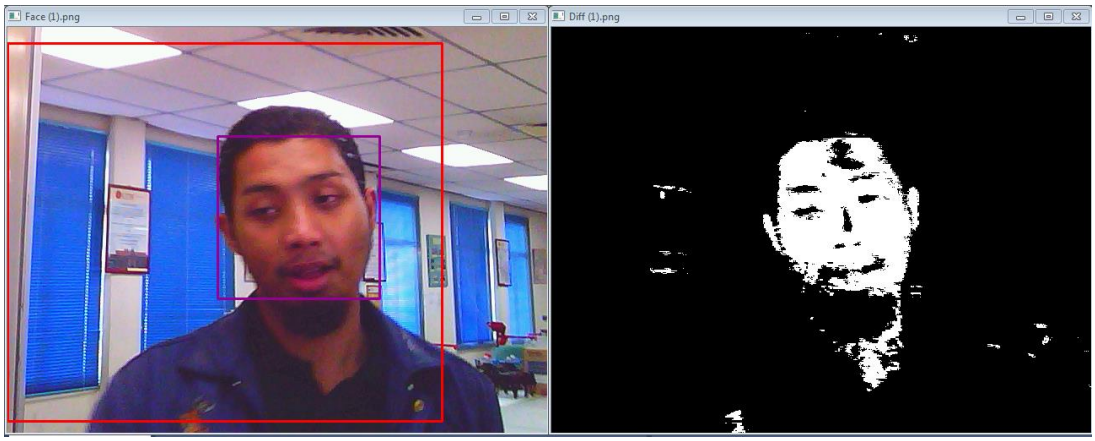


Figure 2.3: Skin-color filter used in face detection

This approach, although useful in some cases, cannot be adequately applied in all situations and circumstances. For example, in situations when an image is not colored as shown in Figure 2.4; no color information is available to be used to detect the face.

(a) Color (b) Grayscale

Figure 2.4: A face image in color and grayscale

This method may also fail if the face illumination conditions change causing the skin color to be beyond the threshold values set for detection, or when another object or the background has a similar color to the skin color, as can be seen in Figure 2.5.

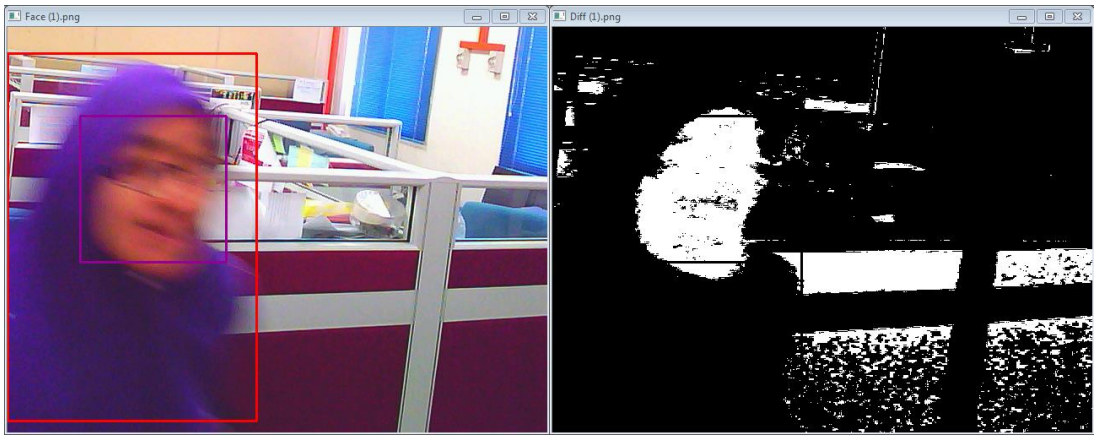


Figure 2.5: Some of the background is within the threshold range for the skin-color detector

Although the HSV color space is widely used in systems that use the skin color to detect a face, other methods that use different color systems also exist. Some systems use the RGB color space, while other systems use the YCrCb color space. Each of these color systems may have its own advantages and disadvantages.

2.4.2 The Viola-Jones Method of Face Detection

The Viola-Jones method [81][82] is one of the more robust approaches to face detection. It has the following advantages:

- Can work with gray as well as color images.
- Is robust to illumination changes.
- Is robust to color changes.
- Can detect partially occluded faces.

The Viola-Jones method employs a cascade of Haar-like classifiers to detect facial features and determine the presence of a face in an image. The Haar-like classifiers; shown in Figure 2.6 are applied across the whole image to scan for faces. The method starts off with small sized classifiers and after doing a complete scan of the whole image, the classifiers are enlarged by a certain percentage and the image is

scanned again. The process continues until either the maximum stated size is reached or the maximum possible size according to the image size is reached.

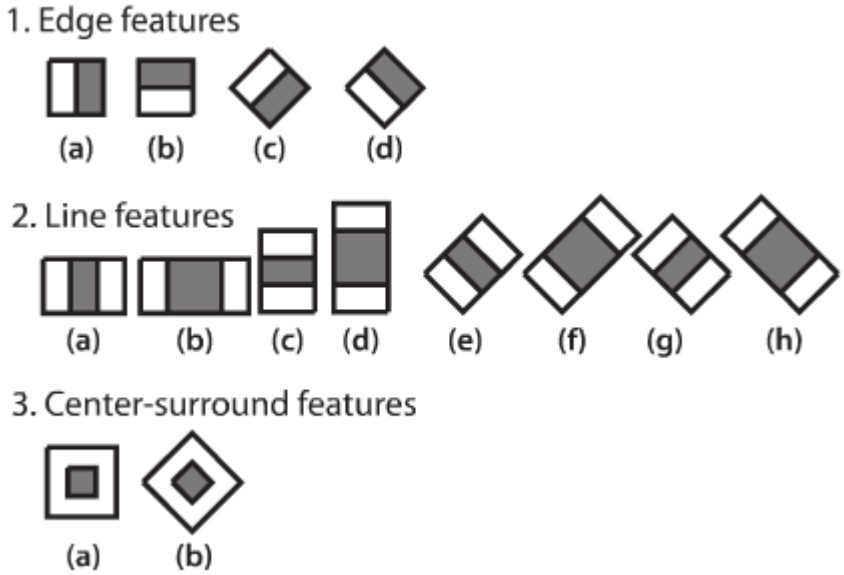


Figure 2.6: Haar-Features used in face detection

With the Viola-Jones method, it is possible to detect the face more than once, as can be seen in Figure 2.7. This can be used to our benefit by setting a threshold value to only accept faces that are detected more than a certain number of times, hence eliminating false positives, as seen in Figure 2.8, in which, the resulting face image is the average area and location of all the detected faces of that particular person within the image.

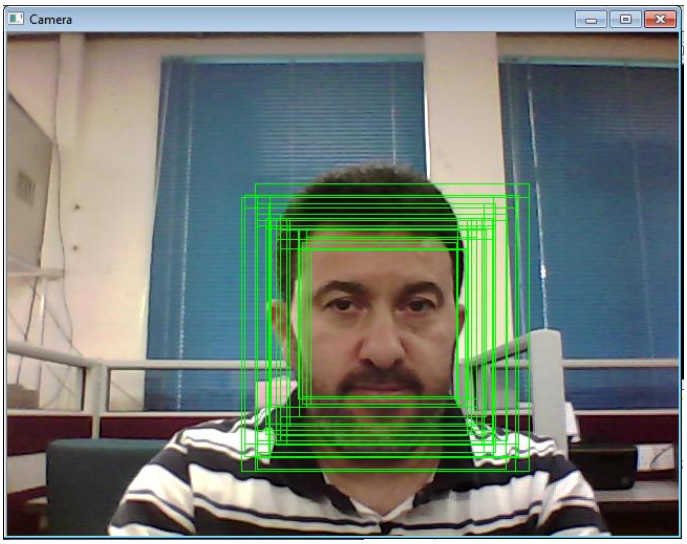
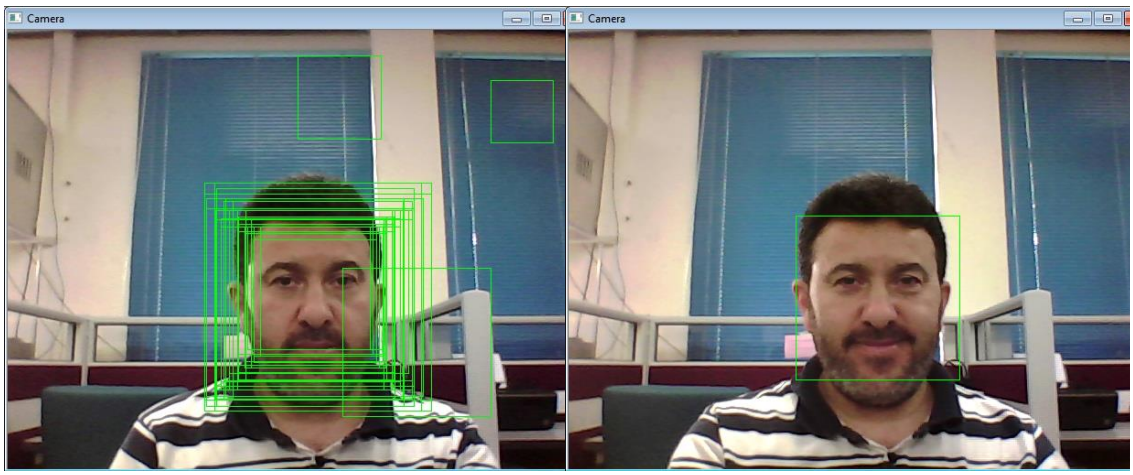


Figure 2.7: A face may be detected many times



(a) (b)

Figure 2.8: Eliminating false positives by requiring the detection of a face several times

Although the Viola-Jones approach is one of the more robust methods around, it has some disadvantages:

- Running it with high accuracy results in slow operation making it not very suitable for real-time face detection.
- Even with high accuracy, false positives can still occur.
- The size and exact location of the detected face area may change from frame to frame without apparent changes in illumination or face location or pose.

2.5 Face Recognition

Facial recognition is used by many security systems and agencies to verify the identity of a person. Many algorithms exist for face recognition, each having its own strengths and weaknesses.

In this system, three different face recognition algorithms are used to identify a detected person. The face recognition approach used will be outlined in the following section.

2.5.1 OpenCV Face Recognizers

OpenCV (version 2.4.8) provides three different face recognizers:

- The Fisher faces face recognizer [83].
- The Eigen faces face recognizer [84].
- The LBPH face recognizer [85].

Each of these face recognition algorithms work in a different way; the Fisher faces face recognizer uses the Linear Discriminant Analysis (LDA) approach to identify a face, while the Eigen faces face recognizer uses the Principal Component Analysis (PCA) approach to identify a face. LBPH uses an approach in which the image is divided into to several sections and a binary histogram of each section is calculated. These histograms are later used to recognize similar faces.

As stated earlier, each of these face recognizers have its own strengths and weaknesses. The Fisher faces face recognizer has high speed learning and recognition, but cannot be updated with new faces in the database during operation. If new face images are to be added to the database, then it has to be retrained.

Similarly, the Eigen Faces face recognizer cannot be updated during operation and has to be retrained when new images are added to the database;

however it has a higher recognition rate in certain situations involving facial expressions and large image dataset.

Although the LPBP face recognizer is similar to the Eigen Faces face recognizer in that it requires a long time to train and has a slower identification time compared to the Fisher faces algorithm, the LBPH face recognizer accepts new additions to the face image database and does not require retraining all over again as well as its robustness to varying lighting conditions.

2.6 Authentication

Authentication is a means of verifying the identity of a person for any reason. It can be done in a number of ways such as checking a person's ID card or other credentials.

Authentication can generally be divided into three types [86][87]:

- Ownership based authentication.
- Knowledge based authentication.
- Biometric based authentication.

Ownership based authentication refers to items that are possessed by a person and can be used to verify the person's identity such as a key, ID card, smart card (RFID) …etc. Knowledge based authentication on the other hand refers to a piece of information that only that person is supposed to know such as a password, an answer to a secret question or a personal identification number (PIN). Biometric authentication on the other hand is based on an attribute of a person such as his face, eye iris, finger print, palm print …etc.

Authenticating a person's identity can be intrusive such as taking a person's fingerprints or non-intrusive such as capturing their images while they walk past a security camera located at a hidden or open location. Authentication can be simple

such as asking a person for his name or elaborate such as requiring their passport as well as other information. Authentication can be manual such as entering the identification data into the system by a person, or automatic such as allowing someone access based on their iris scan while they past a security gate [88][86].

Depending on the purpose of the authentication requirement a system utilizing one of the mentioned approaches would be used to identify subjects and grant or forbid access to certain facilities. However, in all types of authentication systems, the main issue is the precision, hence the correctness of the identification. If the system is not accurate or can be deceived, then it has failed in its task no matter how intricate or expensive it may be.

Each of these methods of authentication has its own advantages and disadvantages, hence there's no best or worst authentication method, but rather, an authentication method is chosen based on its suitability to the security requirements and conditions therewith. Consequently, the mostly used authentication method would be the one that can suitably verify the user for the condition at hand, for example: using a username and password for accessing an e-mail account. In the following sections two of these methods will be looked into in more details regarding their usage, strengths and weaknesses as well as the latest research done on them.

2.6.1 Biometric Based Authentication

One of the widely used methods of authentication is biometric authentication [88][86][89]. Biometrics refers to the unique characteristics that distinguish a person from others, such as the facial structure, fingerprint, eye iris …etc. Figure 2.9 shows some of these characteristics that are used for identity verification.

(a) (b) (c)

Figure 2.9: Some biometrics used for identity verification

a) Face [90], b) Fingerprint [91], c) Iris [92]

Biometric features have many traits and advantages [86][88] that make them an attractive option for identity verification, such as:

- The do not need to be carried by a person, as they are a part of his/her body.
- Presence of the authorized person is required, eliminating the possibility of access by others during the absence of authorized members/users.
- No need to remember passwords or PINs.
- Security tokens used in biometric systems (a person's biometric features) cannot be lost or forged.
- Biometric features are unique to each individual and cannot be shared; this results in creating a personalized access for that user.
- Biometric features cannot be stolen or passed (willingly or unwillingly) to others to be used for illegal access or registration.
- Biometric systems may offer a faster method of access than some other authentication methods such as passwords and PINs.
- Since biometric features cannot be lost, they eliminate the need to produce replacement access tokens such as a new access cards (ID or RFID), hence eliminating unnecessary cost and overhead for system mangers.

These features as well as the non-intrusive nature of some of the techniques have encouraged the use of biometrics to authenticate people's identities allowing only authorized individuals access to locations or resources that are not available for others.

In biometric authentication systems, a person is verified based on an attribute related to his body such as his eye iris, eye retina, face, finger print, finger geometry, palm print, hand geometry, vein pattern, voice, ear shape and others [88].

Although biometric authentication has many advantages, it has several disadvantages [86][88][93], such as:

- In most cased, biometric authentication cannot be 100% accurate (correct/false) as is the case with tokens such as an RFID or password, but rather it gives a confidence level to the examined biometric feature.
- User acceptance can be another challenge, as some users consider holding their biometric information a breach of their personal privacy.
- In some cases, a person's biometric features may change resulting in unrecognition by the system. For example, diabetes may affect the patient's eye, and chemicals may change the affected person's fingerprint or palm-print, and flu may affect a person's voice rendering it inadequate for authentication purposes.
- It is possible that certain biometric features are not available with all people. For example, a person with an amputated arm cannot register or verify himself using a fingerprint or palm-print scanner.
- In some cases, a biometric feature can be similar in two individuals as is the case with twins who may have very similar face images.
- Finally, each particular biometric system has its own limitations, which may be cost, noise, incorrect interaction by the user or other factors.

Although biometric authentication has some limitations, its advantages make it an attractive choice in systems utilizing multi-mode authentication. One of the methods that can provide quick, low cost, unobtrusively authentication is face recognition. In the following section, face recognition authentication will be examined in more detail.

2.6.1.1 Face Recognition Authentication

In some biometric systems face recognition authentication is used to verify a person based on his/her face. This method has many advantages:

- Quick: It takes less than a second to recognize a person from his face.
- Unobtrusive: in contrast to fingerprint, palm-print, retina and other scanners, face recognition can be done as the person walks past a camera which can be hidden.
- Low cost: Although high-end face recognition systems may be costly, face recognition authentication can be achieved with a simple system of a low cost webcam and computer.

But, as with any other biometric system, face recognition has its own limitations, such as [94][95]:

- Lighting and makeup can limit some systems ability to correctly recognize a person.
- Disguise and masks can be used to deceive some face recognition systems.
- People with similar faces such as twins may be hard or even impossible to distinguish by some face recognition systems.

Face recognition techniques can be categorized as Appearance Based, Feature Based, and Hybrid. Many methods have been proposed for each of these techniques [96][97]:

- Appearance Based
    - The Eigen Face method.
    - The Fisher Face method.
    - Support Vector Machines (SVM).
    - Independent Component Analysis (ICA).
    - Probabilistic Decision Based Neural Network (PDBNN).
- Feature Based
    - Face Recognition through geometric features.

  - Hidden Markov Model (HMM).
  - Active Appearance Model (AAM) ((2D Morphable Method))
  - 3D Morphable Model.
- Hybrid

In the appearance based methods, the face image is transformed into face-space, and then a statistical method is applied to it. In the Eigen Faces method [84], Principal Component Analysis (PCA) is used for this goal. The Fisher Faces method [98][83] on the other hand is based on Linear Discriminant Analysis (LDA). The SVM method [99][100] was introduced to develop the classification performance of PCA and LDA. The independent component analysis (ICA) method [101] is a modified version of the PCA method with more representative power. The PDBNN method [102] consists of a face detector, eye localizer and a face recognizer together. The PDBNN method is concerned with the upper part of the face only.

In the Feature based methods, a structural classifier is used based on the face geometry or local features such as the eyes, nose and mouth. In [103], Kanade used the Euclidean distance for correlation between extracted features. The Hidden Markov Model (HMM) approach [104] can be used with images which may vary due to lighting, facial expression and/or orientation since it is based on the arrangement of the face features as discrete parts. In the Active Appearance Model (AAM) [105][106] the appearance of the face is represented as a compact set of model parameters combining shaper variation with appearance variations. In the 3D Morphable Model method [107], a 3D model of the face is constructed using face images of subject. The images are taken in a good lighting conditions and the Morphable model is used to acquire the correspondence information of the facial components and regions.

Hybrid methods use both, the appearance (holistic) and feature based methods to better recognize a face such as the Local Feature Analysis (LFA) method [108], the Shape-normalized Flexible appearance method (or Gabor & Grid) [109], and the Component-based face recognition with 3D Morphable models method [110].

Another method of authentication based on face recognition is the infrared thermal scan of a face image to identify facial characteristics [111][112][94]. The facial thermogram method uses the infrared thermal scan of a face image to identify facial characteristics.

Like all biometric approaches, face recognition cannot give a definite answer regarding the identity or authenticity of a person, but rather a confidence level based on the stored database and acquired data at the time of verification [86][93]. For this reason, a second method which does not depend on biometric authentication is required to be used when the confidence level is below the 'authorization' threshold. For this purpose, knowledge based authentication provides a low-cost and practical method to authenticate a user which the system is unsure of (i.e. when the confidence level lays between the 'authorized' and 'stranger' levels).

2.6.2 Knowledge based Authentication

Knowledge based authentication are based on a piece of information that is specific to a user or a group of users. This piece of information can be a password, a special number (PIN: Personal Identification Number) or an answer to a question. This information may be changed as needed.

This authentication method had some advantages and disadvantage, but it differs from biometric authentication in that it provides solid proof (0% or 100%) of a user's credentials rather than a confidence rating. For this reason, such an authentication approach can be used in situations where biometric authentication fails to provide a confident response regarding the identity or credentials of a certain user.

2.6.2.2 Password Authentication

Password authentication is one of the simplest methods of knowledge based authentication. It involves validating a person using a password. This password can be simple or complex, it can be a one-time password or of a recurring type, it can also be unique to the user or shared by a group of users who are all authorized to use a particular resource.

Although password authentication has some advantages such as being a simple, low cost and easy to use approach, it has disadvantages such as the ability to be lost, forgotten or guessed by unauthorized personnel [87].

The "One-Time-Password" (OTP) technique is a variation of the normal password authentication approach and can be used to limit the vulnerabilities of the ordinary password authentication technique. It involves allowing usage of a password provided by the system only once, after which this password can no longer be used for authentication purposes [113][114]. Some variations of this approach are:

- An OTP list generated by the system of which a password is crossed off by the user and the system once it's used.
- A date-based OTP in which the date forms a part of the password or dictates the formation of the password.
- A function is used to generate a new password based on the old one [115].

Summary

This chapter provided a review of several topics that are related to the security robot presented in this system. Based on the control approach, security robots fall into three categories, teleoperated, autonomous and hybrid security robots. Although omnidirectional vision provides extra capability to a robot, teleoperated and autonomous robots may or may not use omnidirectional vision in their operation depending on their design and application. Surveillance performed by security robots is a primary part of their job which may causes different actions to arise such as alarm activation, subject tracking or other actions specified by the robot builder. Motion detection is a method widely used by many security robots to fulfill their surveillance task enabling them to detect the presence of intruders entering their operation area. Face detection can be done in different ways such as using skin-color detection, cascade classifiers or others, and is used by improved security robots to determine the cause of the motion that was detected and verify whether it was caused by a human or otherwise. Face recognition is an elementary tool for allowing a security robot to identify authorized personnel from strangers. This is done by employing a face recognition method such as Fisher faces, Eigen faces, LBPH or others. Authentication can either be ownership based, knowledge based or biometric based. Authentication is used as decisive step to validate subjects that may not have been acceptably identified as strangers by the face recognition stage.

CHAPTER 3

METHODOLOGY

The development of a security robot involves many disciplines. This is due to the fact that such a robot not only has to sense the environment, but also has to navigate autonomously, avoid obstacles as well as interact with people if and when needed.

In this chapter, an overview is given of the different methods and techniques used in the operation of this project's security robot. Section 3.1 describes the robot's design, body and hardware. Subsection 3.1.2 provides an outline of the construction of the robot's body. Subsection 3.1.2 outlines the robot's hardware; sensors, controllers and actuators. Subsection 3.1.3 describes the camera system used and how its connected to the robot's main controller, the laptop.

Section 3.2 describes the surveillance operation conducted by the robot, while section 3.3 outlines the approach used in detecting motion. Subsection 3.3.1 gives a general overview of motion detection. Subsection 3.3.2 provides an outline of the approaches used in detecting motion in video. Subsection 3.3.3 outlines some methods provided by the OpenCV library for motion detection, while subsection 3.3.4 describes the approach used in this system for detecting motion.

Section 3.4 outlines face detection methods and the approach used in this system for detecting faces. It contains three subsections: subsection 3.4.1 gives a

general overview of face detection. Subsection 3.4.2 outlines the use of motion detection in video-based face detection. Subsection 3.4.3 provides a description of skin-color based face detection. Subsection 3.4.4 provided a simple overview of the Viola-Jones method for face detection. Subsection 3.4.5 describes the approach used in this system for the detection of faces in the environment where the robot operates.

Section 3.5 outlines the many different aspects of face recognition as well as the approach used to accomplish it. Section 3.6 outlines the methodology used for tracking the face of a subject. Section 3.7 outlines the many different aspects of obstacle avoidance as well as the methodology used in this system for accomplishing that on the robot. Section 3.8 outlines robot navigation and the methodology used in this system, and section 3.9 outlines the different aspects of identity authentication and the methodology followed in this system to implement it.

3.1 The Robot's Design

In order to well perform the operations of surveillance, face detection, recognition as well as subject tracking and navigation, the robot was designed in a certain way to meet the requirements of the tasks at hand.

The requirements were to construct the robot tall enough to be able to capture subject faces from a front perspective rather than from below as would be the case with a 'short' or a small robot. Also, the robot should have sufficient speed to be able to navigate to a subject in a relatively short time. These two requirements necessarily resulted in a third obligation, which is to make the robot stable so that it doesn't fall when it moves, turns and stops during its navigation operation. The following subsections, present in some detail the robot's body, hardware and camera system's design.

3.1.1 Body

The robot's body was constructed in three steps:

1. Robot's base
2. Robot's upper body
3. Improvements

The robot's base was borrowed from a large radio controlled vehicle sized 1:6 of the original. This approach simplified and speed up the robot's building process due to the fact that the base body, driving motors and wheels all come pre-assembled and tested for correct operation. Figure 3.1 below shows the base of the robot.

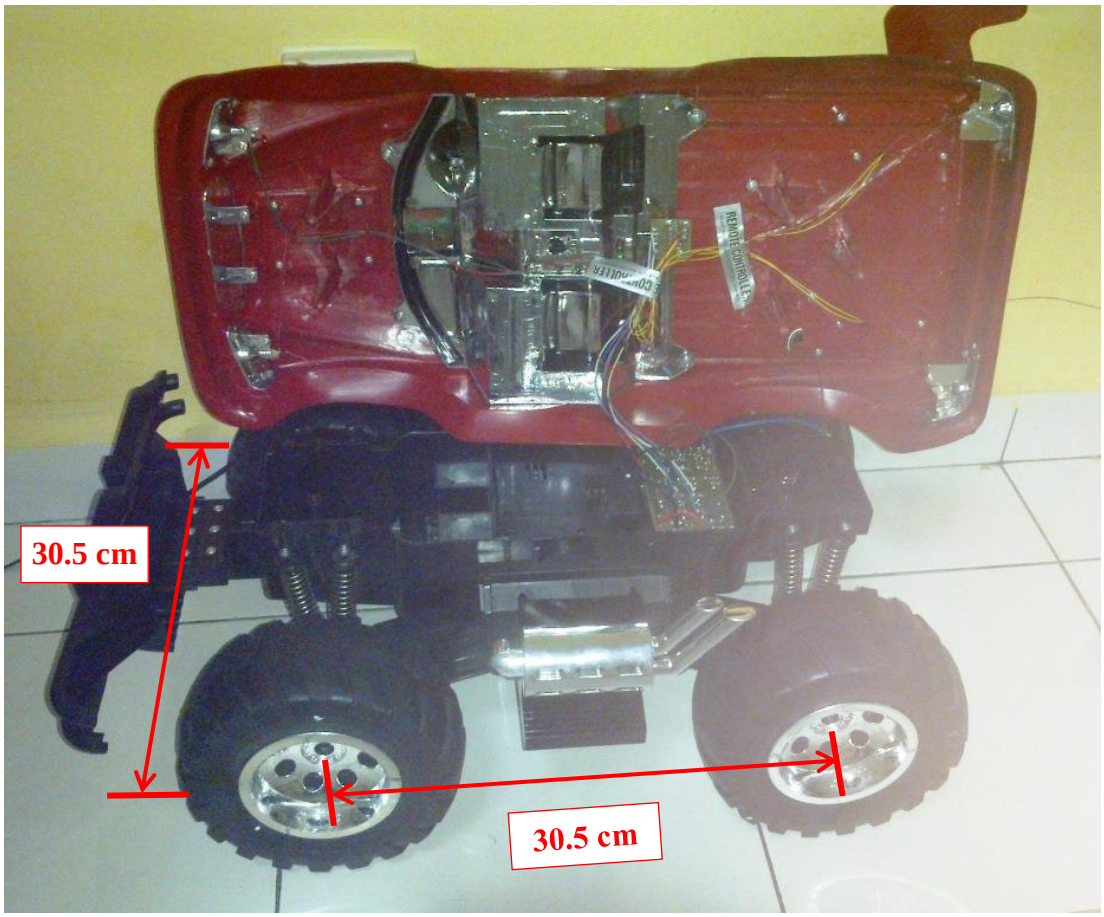


Figure 3.1: The robot's base

The robot's body was built from angle slot aluminum having dimensions of 18x18 mm and a cross section of 0.5mm. This angle slot metal was chosen for its light weight as well as its relative strength which made it suitable for the robot's operation. The metal slot was cut to form the robot's body. Figure 3.2 shows the design and dimensions of the robot's body.

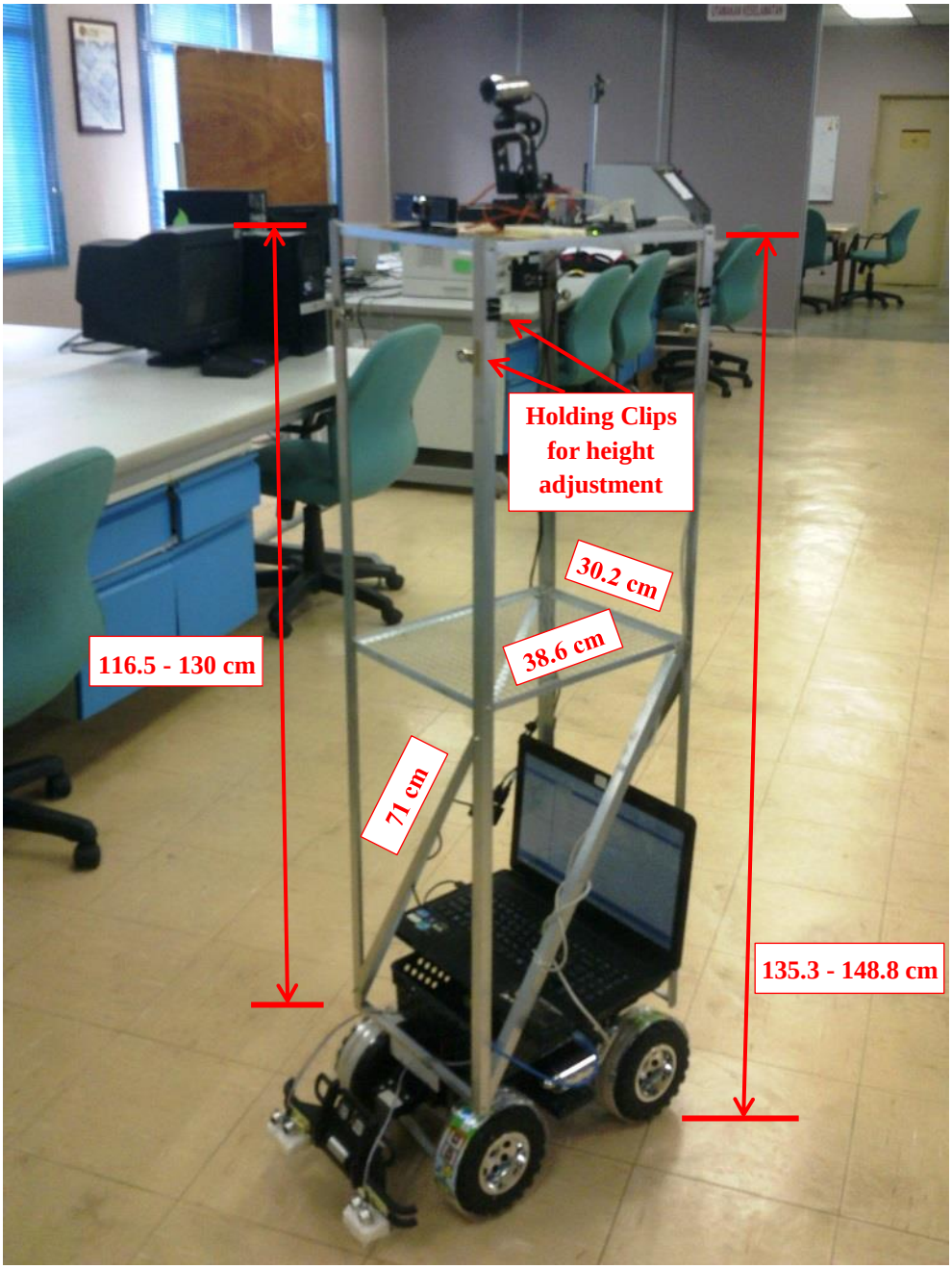
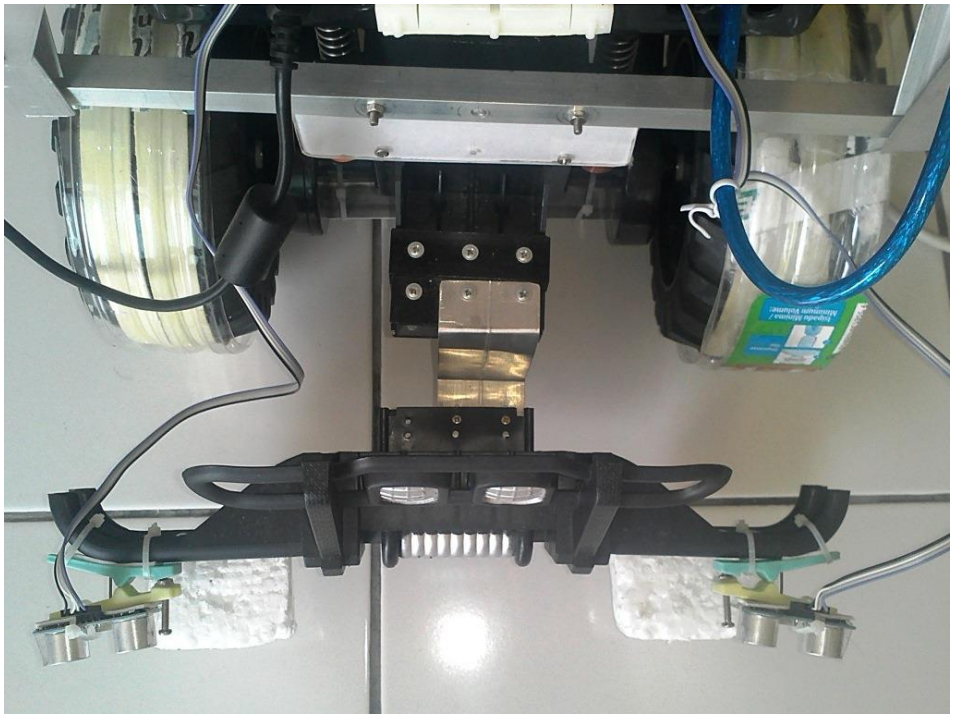


Figure 3.2: The design and dimensions of the robot's body

After building the robot, several improvements were applied to enhance the robot's operation thus giving it variable height as well as better stability during motion and navigation operations.

(a)



(b)

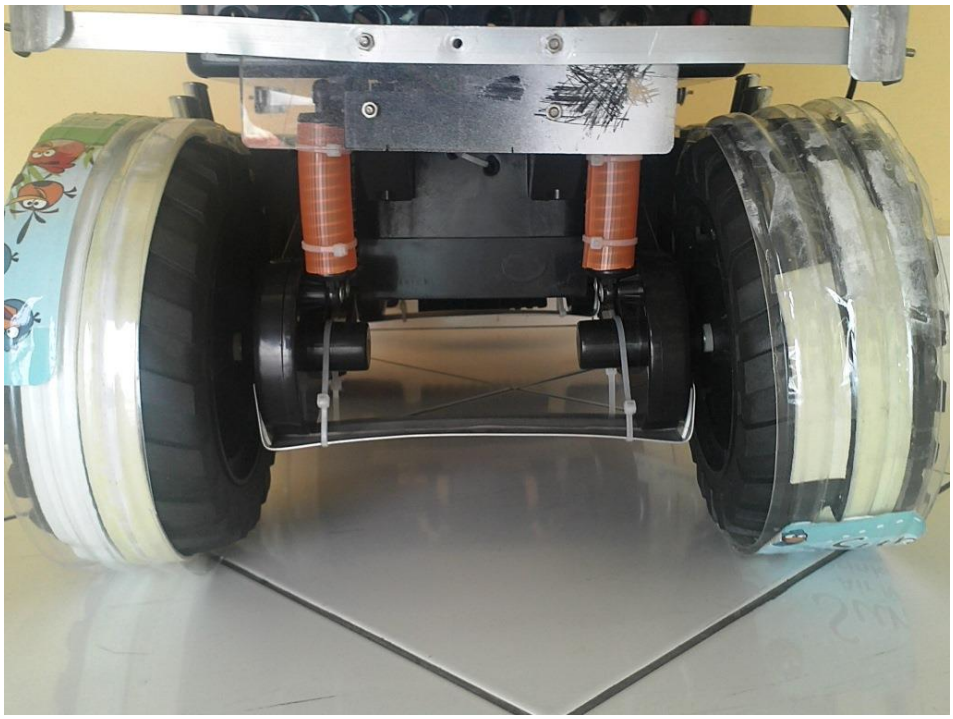


Figure 3.3: The improvements that were applied to the robot's base and body: (a) Improving the front bumper, (b) Improving the suspension and the wheels

As can be seen in Figure 3.2, the robot's height can be manually adjusted, making it possible to extend or contract it to suit the operation environment. Other improvements are shown in Figure 3.3 (a) where the front bumper is being attached to a bent metal strip to improve its crash handling, so it minimizes the crash force without breaking, as well as a couple of Styrofoam blocks to protect the ultrasonic sensors during a crash. Figure 3.3 (b) shows the applied improvements to the robot's suspension and wheels. The suspension was enforced to eliminate tilting during navigation, as well as the addition of a metal strip between the two wheels to limit their separation due to the robot's body and components weight. Figure 3.3 (b) also shows the improvements that were applied to the wheels, where a hard plastic cover was put on the wheels to eliminate vibrations due to the ridges which affected the cameras' picture quality.

3.1.2 Hardware

The robot's hardware consists of sensors, controllers and actuators. It is responsible for maintaining correct operation during the robot's navigation towards a subject allowing it to avoid obstacles and move in the desired path:

- Sensors: Four ultrasonic sensors were used to detect obstacles in the robot's path.
- Controllers: Three Arduino controllers and one DC motor controller were used to receive signals from the sensors and send commands to the actuators.
- Actuators: Two DC motors were used to drive the robot in the desired direction.

The hardware system also included a small breadboard that was used to connect the ultrasonic sensors to Arduino board as well as host the authentication key (DIP switch) which was used in the authentication stage. In that stage, An Arduino controller would read the switch's value and send it to the computer which will in turn compare it with the password entered by the subject during the verification procedure. Figure 3.4 shows the ultrasonic sensors, Figure 3.5 (a) shows the

controllers, while Figure 3.5 (b) shows the robot's operational block diagram. Finally, Figure 3.6 shows the actuators (DC motors) that are used to drive the robot.

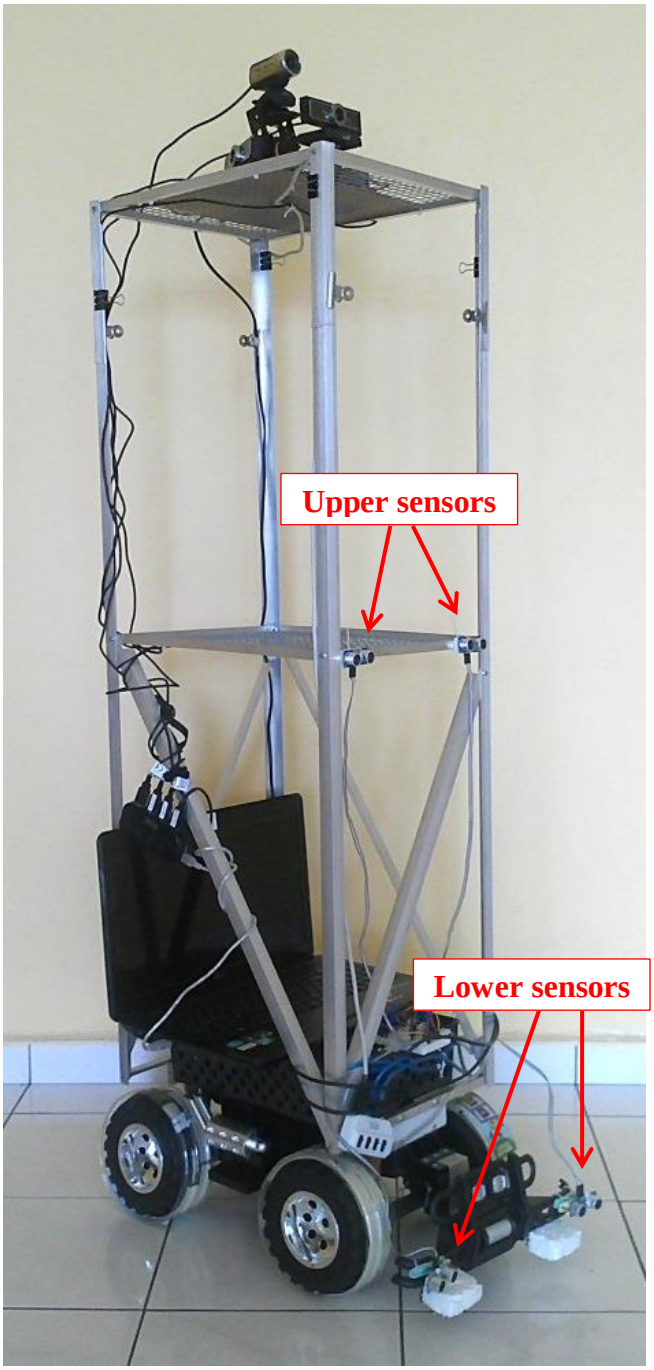
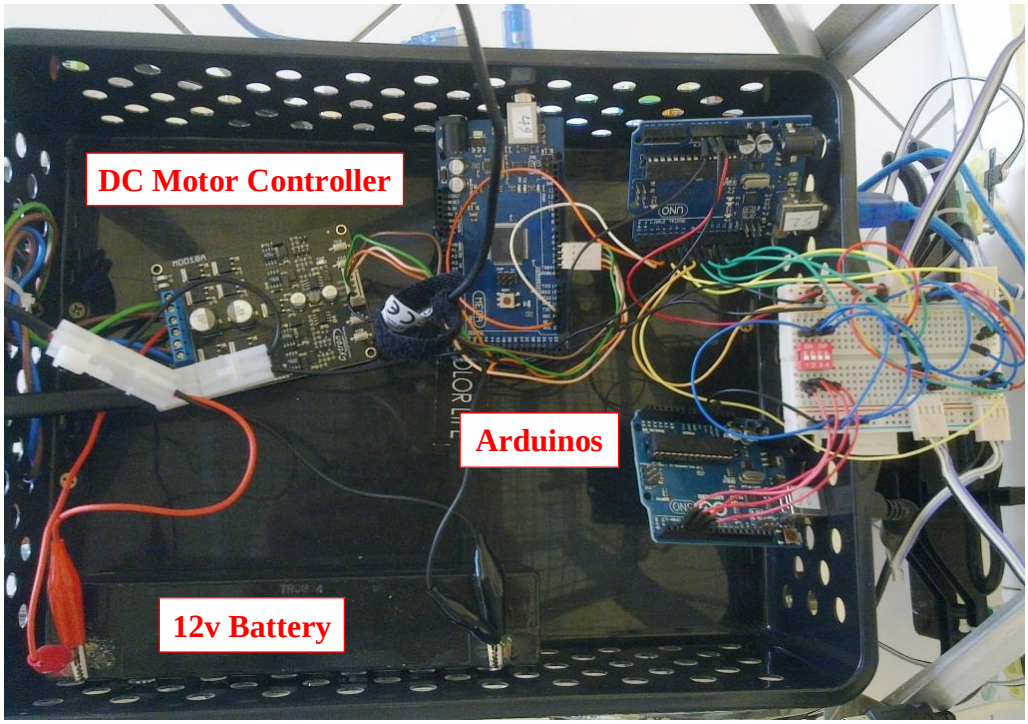


Figure 3.4: The ultrasonic sensors installed on the robot

(a)



(b)

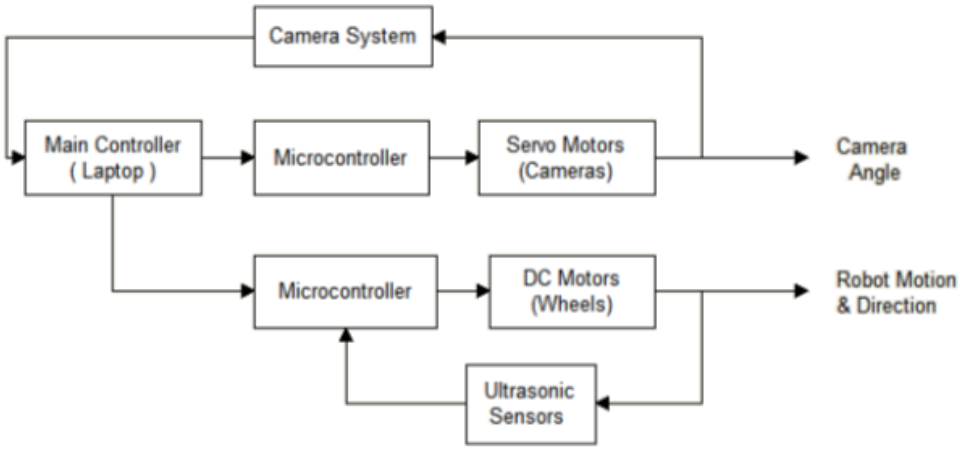


Figure 3.5: (a) The controllers used on the robot, (b) The block diagram

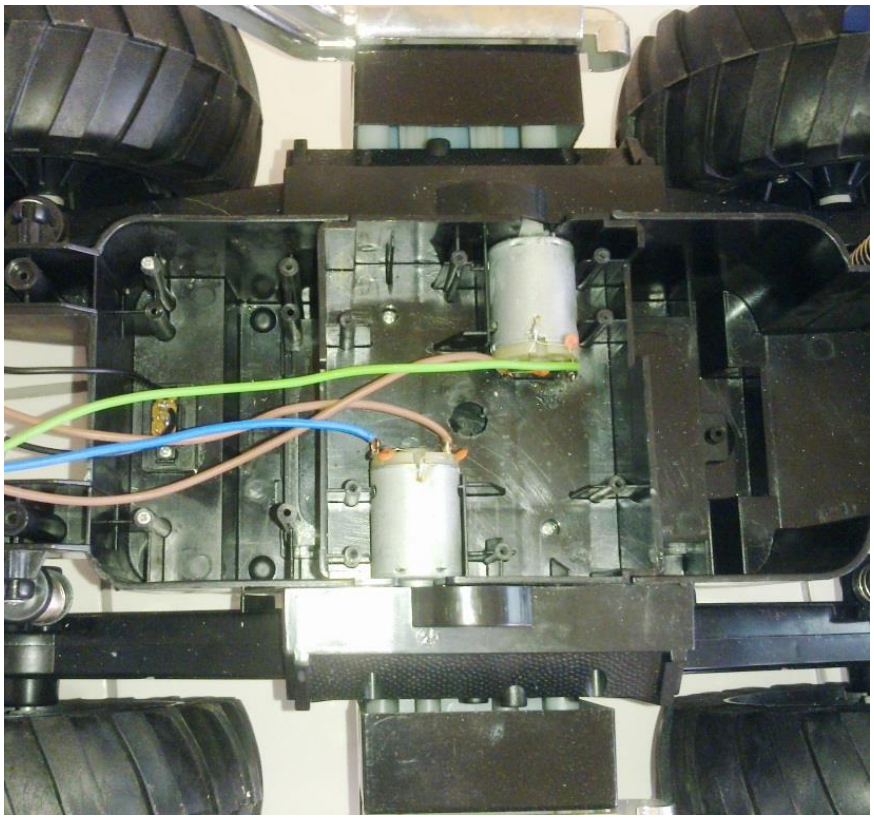


Figure 3.6: The actuators (dc motors) used on the robot

3.1.2.1 Power and Performance

The robot is equipped with a 12 volts battery, as can be seen in Figure 3.5. This battery is the power source for the dc motors that drive the robot as well as the Arduino board that drives the servo motors in the camera system mentioned in section 3.1.3. The rest of the robot hardware (microcontrollers and cameras) are powered by the laptop's USB ports which supplies 5 volts.

The 12 volts battery provided sufficient power to drive the robot at a speed of around 1.9 meter per second. This speed was tested more than once with the robot fully loaded with all its operational equipment.

3.1.3 Software

The program used to control the robot operation was written in C++ using Microsoft Visual Studio 2010 under Microsoft Windows 7 – 64 bit version. The program listing is mentioned in Appendix A.

The written program made use of the open source computer vision software library known as OpenCV [75]. The open source software was used without being modified.

3.1.4 Camera System

The camera system used by the robot was designed to monitor the area surrounding the robot so as to detect any intruder that may enter into the robot's operation area.

The system consists of three Genius F100 wide angle cameras and one full-HD normal angle camera. The three wide angle cameras each has a 120 degrees field of view, thus cooperatively covering 360 degrees, while the full-HD camera is used to capture the detected subject's face image to be later used for face recognition.

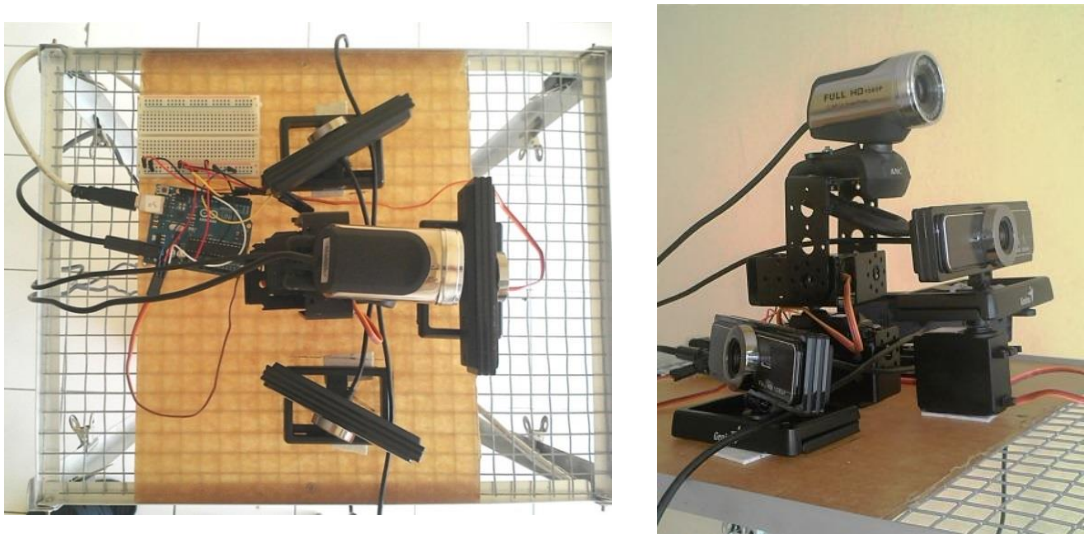
The two rear wide angle cameras are stationary, while the front wide angle camera is stationary during the monitoring operation, but is fixed on top of a servo motor, thus has the ability to rotate right and left during the navigation operation. The full-HD camera is affixed to two servo motors, one rotating horizontally and the other rotating vertically, thus enabling moving the camera left, right, up and down to track the intruding subject. All the cameras used were connected to the main controller (laptop) through USB. A USB hub was utilized to facilitate connecting all the cameras to a single USB port on the laptop. Figure 3.7 shows the camera system used on the robot.

3.2 The Robot's Surveillance Operation

The surveillance operation involves giving the robot a better chance at assessing a situation to enhance its ability in making the right decision at critical moments, such as deciding whether to stop/arrest a subject or allowing him/her access.

To accomplish this, a system must be capable of sensing motion in all direction at once, thus giving the robot the ability to monitor 360 degrees around its location. Furthermore, the robot must be able to identify the presence of subjects (one or more persons) and allow or prevent their passage based on their identity.

To accomplish the above, a system of four cameras has been used on this robot. Three wide angle cameras, each with a 120 degrees viewing angle to cover the 360 degrees of the environment surrounding the robot, and the fourth is full HD camera used to verify the identity of the intruder. Figure 3.7 shows the camera setup.



(a) (b)

Figure 3.7: The setup of the cameras used on the robot

Although the image from a wide angle camera differs from the image of a normal camera; as can be seen in Figure 3.8, the usage of the wide angle cameras instead of normal webcams gave the system a number of advantages:

1- A wider angle of view, both horizontally and vertically. This is important as some subjects are tall, others are short, and at close range either one will be out of the normal camera view.
2- Fast motion detection. Normal cameras with low frame rate may either not be able to detect the quick motion of a running subject, or may detect it incorrectly, while the wide angle camera with a high frame rate can better detect such quick motion.
3- Reduce the system load. Instead of using six or more cameras and having to deal with many environmental images, only three cameras were used to cover the whole 360 degree viewing space.
4- Less power consumption, more efficiency. Three cameras will consume less power and require less power to process their images than six or more cameras.
5- Less calibration, less effort: Three cameras require less calibration and maintenance effort than six or more cameras.

(a) A picture taken with a normal camera (b) A picture taken with a wide angle camera

Figure 3.8: Image comparison of the same location

The system of four cameras can perform the following surveillance operations, each of which can be reported separately if required:

- Detects motion.
- Track the source of motion.
- Detect light.

- Track a light source.
- Detect subjects.
- Track subjects.
- Identify subjects.

Such actions are helpful to security handlers as it gives them the ability to focus on other tasks and get involved only in certain situations when a critical event requires their intervention.

3.3 Motion Detection

This section describes the motion detection system used by the robot. The motion detection feature is used by the robot to discover the presence of an intruder.

3.3.1 Motion Detection in Video

Computer vision based motion detection can be done through the frame subtraction method, background subtraction method, and optical flow method [116].

The frame subtraction method of motion detection in video involves subtracting the current frame from the previous frame, then checking the pixel values of the resulting frame to look for the differences which indicates a change in the perceived image by the camera.

The issue of checking the pixels in the resulting frame involves a large amount of comparison calculations. Video frame dimensions and frame rate per second can result in huge amounts of calculations that need to be performed:

$$P_{Total} = P_{fw} \times P_{fh} \quad \text{--- (3.1)}$$

$$C_{ps} = P_{Total} \times F_{ps} \quad \text{--- (3.2)}$$

Where P_{Total} is the total number of pixels in a frame image, P_{fw} is the frame image width in pixels; P_{fh} is the frame image height in pixels. C_{ps} is the number of calculations per second and F_{ps} is the number of frames per second.

For a webcam providing a resolution of 640x480 at a rate of 15 frames per second, C_{ps} would be:

$P_{Total} = 640 \times 480 = 307{,}200$ pixels

$C_{ps} = 307{,}200 \times 30 = 9{,}216{,}000$ calculations per second

As can be seen above, this huge amount of calculation can be a burden on any system. Moreover, if the three color: Red, Green and Blue are to be checked separately, then the number becomes three folds:

C_{ps} = 3 x 9,216,000 = 27,648,000 calculations per second

For Full HD video at a resolution of 1920x1080, this figure would increase by six folds:

C_{ps} = 6 x 27,648,000 = 165,888,000 calculations per second

3.3.2 The Robot's Motion Detection

One of the methods that can be used for motion detection is the 'Differential Images' or 'Frame subtraction' [117]. This method works by simply finding the difference between two images, if no difference is found then the two images are identical and no motion is present, otherwise a motion has occurred in the region where the two images differ. Although this method is simple, but depending on the implementation, it can have some disadvantages:

- Possible false detection of motion caused by snow, rain or moving trees.
- Changes in illumination can sometimes lead to false detection of motion.
- High camera sensitivity (high ISO) induces noise in the video stream which may result in a false detection of motion.

To overcome these advantages and increase the speed, an approach was used whereby the size of the compared images was reduced to 160x120 pixels, which is $1/16^{th}$ of the original size. This resulted in the following advantages:

- Reduce the load on the system and increase the speed of calculation; a rate of over 34 frames per second (for a single camera) was achieved as seen in Figure 3.9.

- Reduce the sensitivity of the system to noise and miniscule changes (such as flying insects or other miniature objects that may move due to wind blowing from a fan or air conditioner ventilation openings). This reduction in sensitivity results in more robust operation and less false positives.

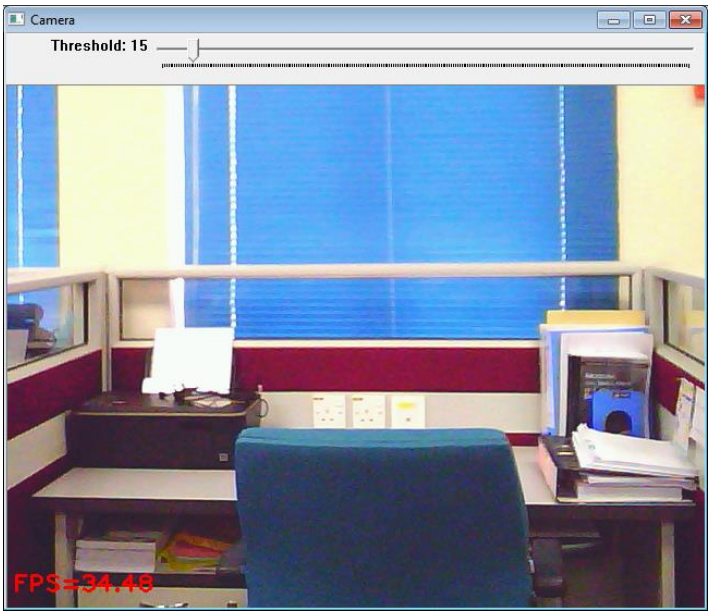


Figure 3.9: A frame rate of over 34 frames per second was achieved

In the implementation of the motion detection in this project, a threshold of '15' was used to detect motion. This number refers to the difference between the luminance of two pixels. A difference value below this would not be considered.

Although the motion detection approach used relies on luminance, the threshold value used; which refers to the sensitivity of the detection, was determined practically by using a slider as seen in Figure 3.9 to test different values. This value resulted in the best sensitivity/performance ratio for the lighting conditions used in the robot environment.

Choosing a value larger than 15 results in a low sensitivity towards change in low light and dark areas, hence a moving object may not be detected correctly or not detected at all. Figure 3.10 and 3.11 shows a person passing in front of the camera in the lab during working hours (daytime), but using a threshold of 25 and 15 respectively, while the overhead lights were switched off.

Figure 3.10: A passing person was not detected in low light with a threshold of 25

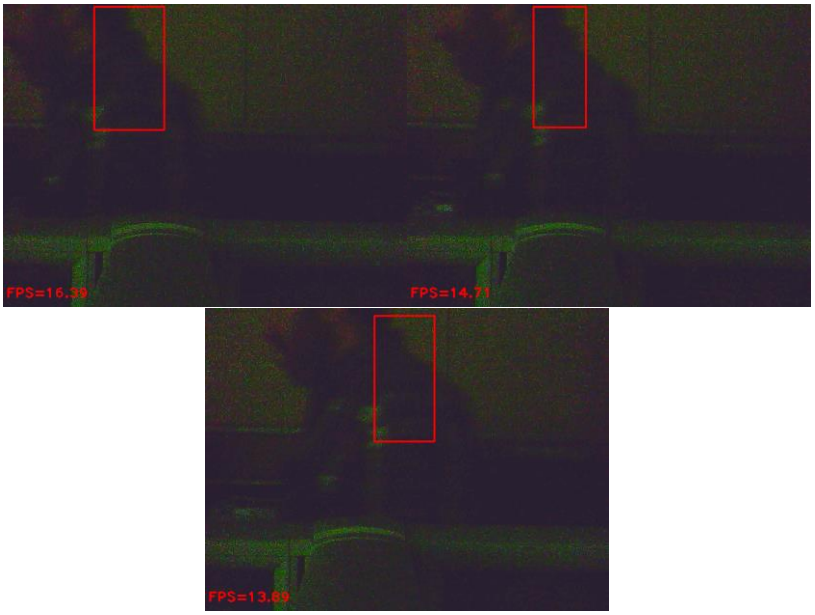


Figure 3.11: A passing person was detected in low light with a threshold of 15

Choosing a value less than 15 increases the sensitivity towards changes in low light, but may lead to false detection of motion in normal or brightly lit areas due to camera sensitivity or any miniscule changes in pixel luminance intensity, as can be seen in Figure 3.12 where no motion is present, yet a motion is being reported by the motion detection function.

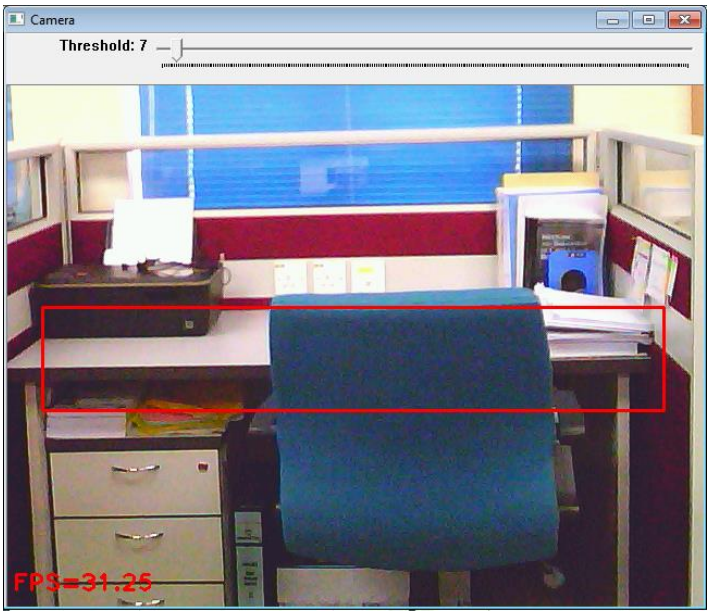


Figure 3.12: False motion detection when a low threshold value is used

When motion is detected, the detection program used in this project returns a rectangular region where motion was detected, as shown by the red rectangle in Figure 3.13.

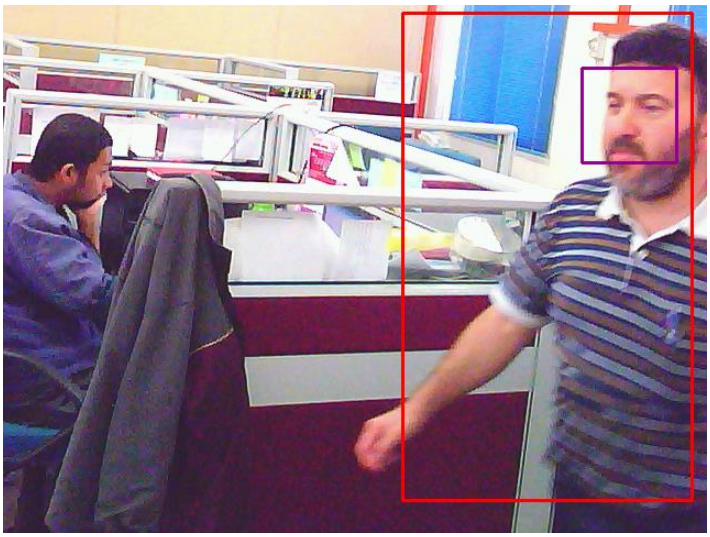


Figure 3.13: The region of the frame image where motion was detected

3.4 Face Detection

3.4.1 Introduction

Face detection is the process of identifying a face in an image. Many techniques exist to try and find the face or faces within an image. Some techniques are faster, others are more accurate.

In this section different methods of face detection will be outlined, a description of skin-color based face detection and finally a practical approach to face detection will be presented.

3.4.2 The robot's Face Detection

As presented earlier, each of the face detection approaches has its own advantages and disadvantages. However, when using face detection on a security robot, several requirements must be met:

- Detection must be fast.
- Detection must be light on system resources.
- Detection must be robust against illumination and color changes
- Detection must be accurate:
    - Very little or no false positives (i.e. non faces recognized as faces).
    - Very little or no false negatives (i.e. faces recognized as non-faces).

In order to achieve all the above requirements, the following approach was followed:

- **Detect motion:** Hence, static objects would be ignored.
- **Apply the Viola-Jones method:** In the region of the motion, look for faces using the Viola-Jones method with parameters that result in faster operation.

- **Check the pixel density:** Check the pixel density (sharpness) of the faces found so far and discard any face which has more than 70% pixel density.
- **Check skin color content:** Check faces found by the Viola-Jones method for skin color content, and discard any face which contains less than 50% of skin color.
- **Check for the presence of eyes:** Finally, if the face image does not contain eyes, then it may not be a face, but something else, such as a part of the human body or a shape that looks like a face and has a similar color.

The above approach resulted in an accuracy rate of around 91%.

In the following sections, the steps mentioned above will be outlined in more details:

3.4.2.1 Motion Detection

Motion detection was used (as outlined in section 4.1) to determine areas where motion has been detected. For face detection, this has the following advantages:

- It avoids static objects which may contain facial features.
- It reduces the area which would be searched for the presence of a face.

The motion detection program used in this project returns a rectangular region where motion has been detected, hence only this area needs to be scanned for faces.

First, the area where motion has been detected is checked, if both: its width and height are more than 35 pixels, then it's scanned for faces otherwise it is ignored.

The reason behind this is that faces with dimensions less than 35 pixels across would be too unclear to identify later with the face recognizer; hence there is no

point in capturing the face image and spending resources recognizing it. To illustrate this point, Figure 3.14 shows the face image of a person which has a dimension of 35x35 pixels. The image is enlarged to facilitate visual comparison by the reader; keeping in mind that the person was static to enable the capture of a good quality face image. Had the person been in motion, the face image would have been even lower in quality.

(a) (b) (c)

Figure 3.14: A face image of less than 35 pixels is hardly useful for face recognition

When the area of detected motion has dimensions larger than 35 pixels across, the second phase of the face detection commences.

3.4.2.2 The Viola-Jones Method

The second phase of the face detection in this system is to apply the Viola-Jones method. As stated earlier, this method is one of the more powerful methods in finding faces in an image; however, it can be slow if the image to be searched has a large size. For example: in an image of 640 x 480 pixels (the normal video frame size), the Viola-Jones method can take as much as 959 milliseconds to complete scanning it for faces sized 25 pixels and above, as can be seen in Figure 3.15.

Figure 3.15: The time taken to scan for a face of size 25x25 pixels

When checking video for faces in real time, much higher speed is required. A good frame rate in a security system should be 15 frames per second or higher in order to capture all moving subjects correctly. Therefore the parameters governing the operation of the Viola-Jones method in the program had to be adapted to reduce the required time while maintaining the effectiveness of the method.

The function implementing the Viola-Jones method in OpenCV is as follows:

detectMultiScale(Image, Array, Scale Factor, Min. No. of Neighbors, flags, Min Size, Max Size)

The parameters' descriptions are as follows:

Image: The image to search in

Array: This will hold the locations of all the faces.

Scale Factor: The increment factor (each time the method re-scans the image it will increase the size of the Haar features by this factor; for example 10%)

Min. No. of Neighbors: The minimum number of times a face is detected (The Viola-Jones method may detect a face more than once; as pointed out in the previous section)

Min Size: The smallest size of a face to search for.

Max Size: The largest size of a face to search for (This parameter is optional. If omitted, the function will continue searching until it reaches the largest possible face size within the image)

Normally, to achieve high accuracy, the following parameters are used:

Scale Factor: 1.1; means a 10% increase on each scan

Min. No. of Neighbors: 4; this will skip any face which has been detected less than 4 times.

Min Size: 20; this will search for faces that are 20 pixels across or larger.

Figure 3.16: Searching for a face of size 20x20 can result in a speed of 0.85 FPS

However, as can be seen from Figure 3.16, using the parameters above will result in a speed of 0.85 frames per second (1177 milliseconds per frame) for a 640x480 camera that normally operates at 30 frames per second; which is a very slow speed. Therefore the following changes were applied to speed up the face detection operation without greatly undermining the accuracy of the method:

Scale Factor: 1.2; means a 20% increase on each scan. This has the effect of doubling the speed.

Min. No. of Neighbors: 2; this will skip any face which has been detected less than 2 times. This is important in order to capture any face. The old value of 4

may not capture a moving person since his face image may be distorted due to motion.

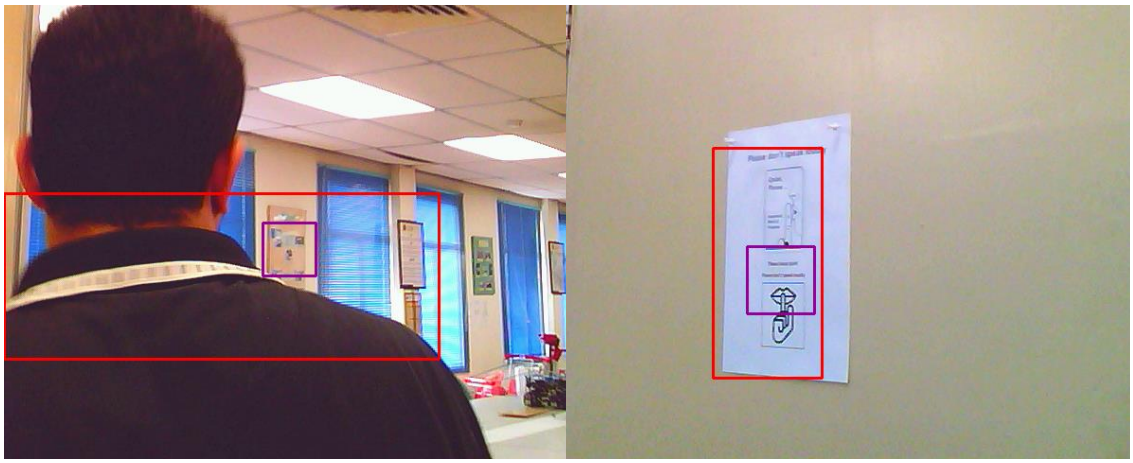
Min Size: 35; this will search for faces that are 35 pixels across or larger. This results in another three folds increase in the speed of detection.

As can be seen from Figure 3.17, using the above parameters as well as scanning a sub-area of the whole frame (the area where motion was detected) resulted in an adequate speed of 2.2 frames per second (450 - 455 milliseconds per frame) for scanning the same area of 640x480 pixels. This makes the system more capable of detecting subject quickly and efficiently.

Figure 3.17: Less time is required

3.4.2.3 Skin Color Content

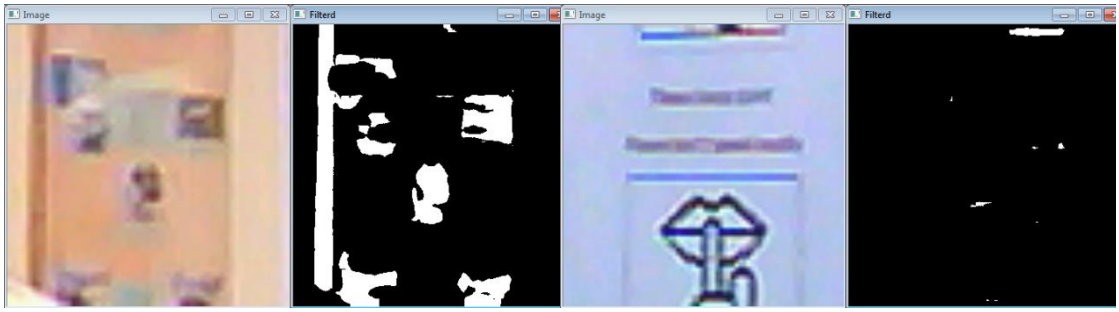
Although the Viola-Jones method is one of the more robust approaches to face detection, still, it may return falsely detected faces, as can be seen in Figure 3.18. The red rectangle is the detected motion area, and the purple rectangle is the detected face within.



(a) (b)

Figure 3.18: False face detection by the Viola-Jones method.

To counteract this problem, the third phase in the face detection process is to check the detected face for skin color content. This process filters out many falsely detected images as can be seen from Figure 3.19.



(a) (b)

Figure 3.19: Skin-color based filtering can exclude many false positives

a) Skin Content = 15.19 %, b) Skin Content = 0.58 %

As pointed out earlier, many color spaces exist such as RGB, HSV, HSL, YCrCb, and XYZ. When tested, it was found that the YCrCb color space gave very good results compared to HSV or HSL as it was robust to illumination changes. A program was written to test the best parameters to use with this color space. Figure 3.20 shows the results of choosing these parameters:

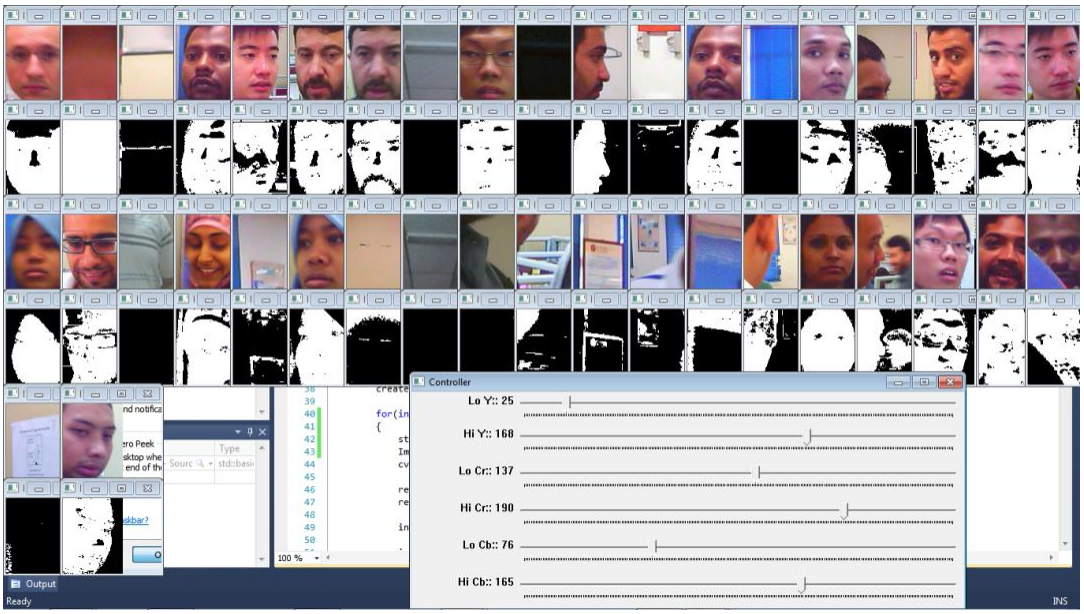


Figure 3.20: Face images and the thresholds used to filter non-face images

The method used to detect the presence and percentage of skin color in a face image is as follows:

1. The face image is converted to the YCrCb color space.
2. OpenCV's function 'InRange' is used to black-out all the pixels that do not fall within the predefined range (the skin color range; which was determined through experimentation as seen in Figure 3.20)
3. Non-black pixels are counted.
4. The percentage of non-black pixels to the total number of face-image pixels is found.
5. This percentage is the percentage of Skin-Color-Content.

3.4.2.4 Pixel Density

Another step in the process of filtering false-positive face detection is to create a pixel density or sharpness filter that would filter out images which contain too many details. During experimentation it was found that face images usually have a pixel density of less than 70%, therefore detected face image which have a pixel

density higher than 70% are ignored. This filter proved useful in filtering out certain images which bypassed the previous phases of detection and filtering. Figure 3.21 shows a few examples of blocked false face detections:

(a) (b)

Figure 3.21: Falsely detected face images with high sharpness

a) Sharpness = 91.67, b) Sharpness = 85.75

The method used to find the sharpness (Pixel-Density) of a face image is as follows:

1. The image is converted to gray color.
2. OpenCV's function 'adaptiveThreshold' is used to white-out all pixels in the image except the pixels representing edges.
3. Black pixels are counted.
4. The percentage of black pixels to the total number of face-image pixels is found.
5. This percentage is the image sharpness (Image-Pixel-Density).

3.4.2.5 The Presence of Eyes

It's interesting to know that after all the filtering process done so far, some images of false face detection can still pass. Therefore, the final step was to use a filter that would exclude images which do not contain eyes. For this purpose, OpenCV's eye-detection capability was used as a tool (filter) to achieve the desired

outcome. This tool (filter) proved useful in eliminating certain images which bypassed the previous phases of detection and filtering. Figure 3.22 shows a few examples of blocked false face detections:

(a) (b) (c)

Figure 3.22: Falsely detected face images which do not contain eyes

The method used to find the presence of eyes in a face image utilized OpenCV's 'detectMultiscale' function in conjunction with three eye cascade classifiers:

- SmallEyeCascade.
- BigEyeCascade.
- OtherEyeCascade.

The following is a line of code representing one of the operations:

```
SmallEyeCascade.detectMultiScale(FaceImage,DetectedEyesIndicator,1.1,1,0|CV_HAAR_SCALE_IMAGE,Size(FaceImageHeight/10,FaceImageWidth/10))
```

The function would return the number of eye-pairs detected (DetectedEyesIndicator). If the number of detected eye-pairs is zero, then that means there are no eyes present in the face image, and that it may probably be a non-face image.

3.5 The Robot's Face Recognition

In this system, face recognition is used to verify the person's identity. Practical face recognition using real time video from surveillance cameras has to achieve certain qualities to be adequate:

- It must be fast: slow face recognition will slow the whole system down. A face should be recognized in a few milliseconds.
- It must be accurate: Accuracy is important. Failing to recognize a person correctly in a security robot could mean disaster in certain situations where a stranger is allowed passage to the facility.
- It must be flexible: The ability to accommodate changes during operation is an important feature. Lacking this feature may require a robot operator to restart the face recognizer training every time a number of new images are to be added the database; an event which may take a long time to complete.
- Can cope with sharp and blurry images: This is hard to achieve as blurry images lack the facial details that sharp images possess.
- Can handle misaligned images.

In the current system, an effort to meet the first three requirements was made, while the last two requirements were skipped as they required extra computations which may slow the system performance without providing high value. For example, since the operation is done in real-time, blurry images may not need to be sharpened as the subject's face image may be clear in the next video frame and recognition operation can be done without losing resources on sharpening a blurry facial image. The same issue goes to misaligned images. Normally a subject is walking or standing upright, therefore misalignment may be rare, hence system resources are not spent on re-aligning face images in this real-time setting.

The face recognition approach used in this system works as follows:

- Using Multiple Face Recognizers to increase the accuracy.
- Refreshing the Face Recognizer database to improve recognition rate during operation.
- Face image cropping to reduce disadvantageous parts of the face image.

Following is a description of the above in more detail.

3.5.1 Using Multiple Face Recognizers

In order to maximize the accuracy of the face recognition operation, all the face recognizers provided by OpenCV (three) were used to correctly identify the face:

- The Fisher-faces face recognizer.
- The Eigen-faces face recognizer.
- The Local Binary Pattern Histogram face recognizer.

When the three face recognizers return the same identity for a given face image, then the decision is considered conclusive. However, it is possible that the three different face recognizers would disagree about the identity of a face image. In such a situation, the decision taken would be in favor of the most votes. In the unlikely event of having three different identifications by the three different recognizers, then the highest confidence is taken to determine the identity of the person. This approach is outlined in Figure 3.23:

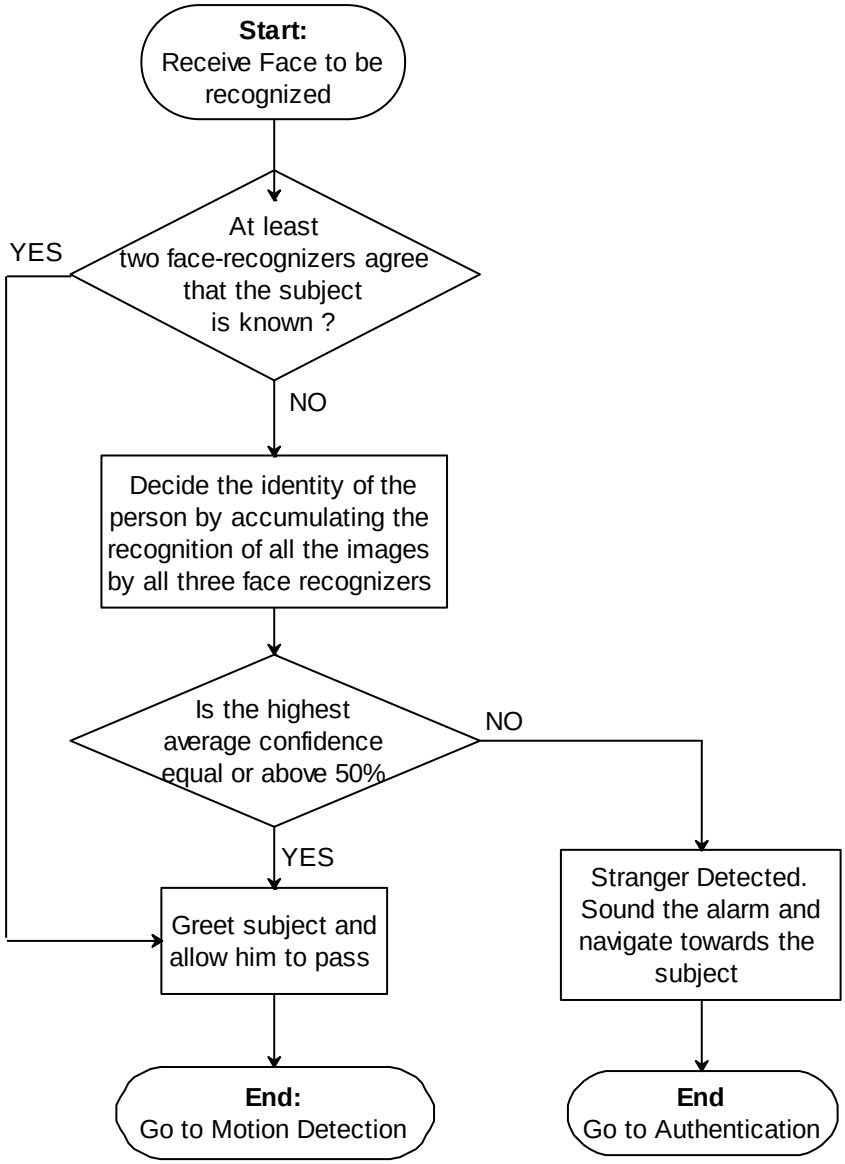


Figure 3.23: Deciding the correct identity of a subject

Although using this approach may consume more time than using a single face recognizer, but the benefits outweigh the consumed resources, especially that the total recognition time for all three recognition operations is a few milliseconds.

3.5.2 Refreshing the Face Recognizer

One of the important aspects in face recognition is the ability to update the face recognizer with more images regularly in order to increase its effectiveness.

The Fisher-faces and Eigen-faces face recognizers cannot be updated during the robot operation, while the program is running. In order to add new images to the training set used by these two algorithms, retraining (from the start) is necessary. Since retraining cannot be conducted during the robot's surveillance operation, this is done at the end of the robot's working shift.

During operation, the robot gathers new images of subjects which it identifies as 'unknown' or 'unfamiliar'. These images are stored, and at the end of the day (working shift), the robot operator scans the new images and relocates each one to the correct folder of the relative person.

The LBPH face recognizer differs from the previous two face recognizers in its ability to be updated during operation. This allows newly acquired images to be added to the face recognizer database and be used to increase the effectiveness of the LBPH face recognizer in identifying subjects.

When a new image is acquired, it is checked by the three face recognizers, if all three face recognizers agree on the identity of the person, the LBPH face recognizer checks to see if its confidence level is low for this image, if so, it will add it to the database to increase the confidence the next time it comes across a similar image. If the confidence is already high, then this means that the image has high similarity to one or more of the images in the current database, hence there's no need to add it. Figure 3.24 shows an outline of this operation:

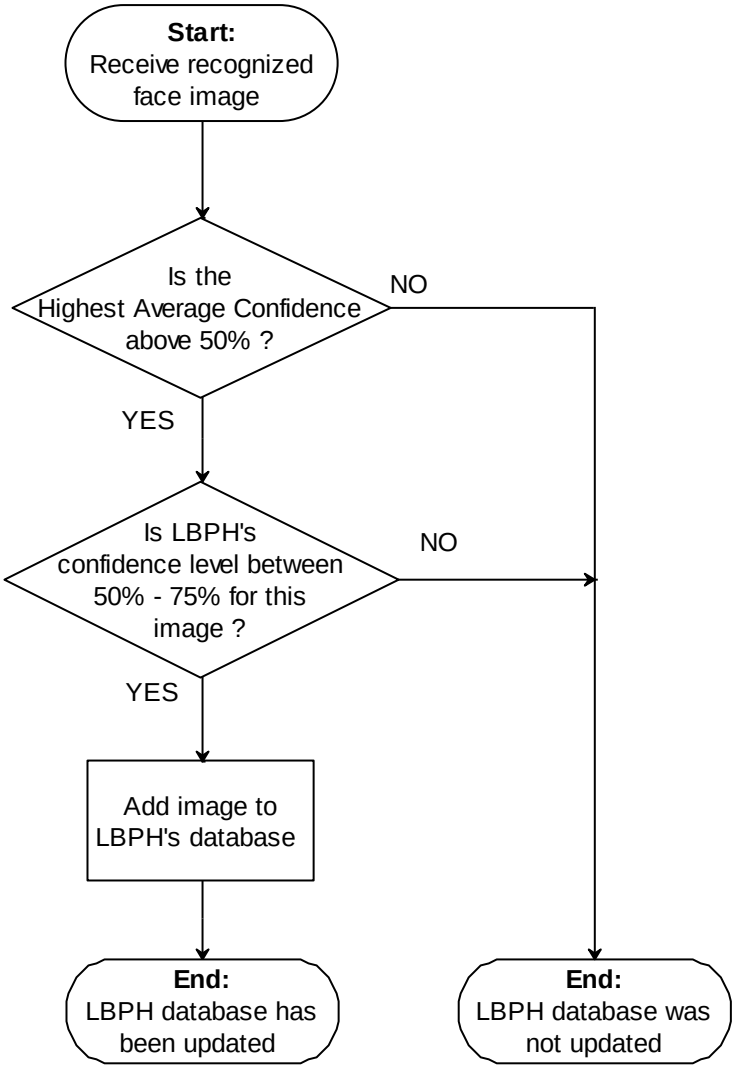


Figure 3.24: The LBPH face recognizer can add an image to its dataset

3.5.3 Face Image Cropping

The face image returned by the face detector usually contains undesired extra parts or sections as can be seen in Figure 3.25.

(a) (b) (c)

Figure 3.25: Detected face images normally contain undesired extra sections a) Extra area on both sides of the image, b) Extra area on the left side of the image, c) Extra area on the right side of the image

These extra parts or sections of the image negatively affect the operation of face recognition as they contribute image information belonging to the background rather than the face itself. For this reason, it is necessary to apply a cropping operation in order to eliminate as much as possible of the undesired parts or portion of the image while retaining as much as possible of the subject's face.

Since faces may have different postures and have different sizes, cropping a face image correctly is a challenging operation that has to be done in a certain way to ensure acquiring the face image with as little as possible of the background regardless of the face's posture, size or location within the image.

For this purpose, the skin-color detector was employed to scan the sides of an image and incrementally crop the image from the side that contains less skin color until a certain predefined width has been attained. Figure 3.26 shows a face image before and after cropping.

(a) (b) (c)

Figure 3.26: Cropping the undesired parts of a face image

a) Side cropped, b) Left cropped, c) Right cropped

As can be seen from Figure 3.26, the size that yielded the most satisfying results was found to be 100 pixels for an image of 120 pixels. If the image was cropped to become less than 100 pixels wide (e.g. 90), as required by Figure 3.26 (b), then we would lose a part of the face from both images (a) and (c).

3.6 Face Tracking

3.6.1 Introduction

In systems where the vision sensor is non-stationary, the process of visually tracking an object involves rotating or moving the vision sensor in such a way as to keep that particular object within the vision sensor's field of view. Face tracking is a particular case of object tracking in which the object of interest is the face of a person.

2.6.2 Challenges in Face Tracking

There are many challenges involved with face tracking, some of which are:

- The need to determine the existence of a face in the camera's view.
- If there's more than one, then it is required to determine which face, out of the group, should be tracked.
- Keeping up (mechanically and/or electronically) with the quick motion of a subject may be hard, especially if the person running (indoors), is too close to the camera or is riding a vehicle (outdoors).
- Other mechanical limitations such as the maximum angle a camera can rotate horizontally and vertically.

Other challenges related to the face pose change, occlusion and lighting changes all fall into the primary point in which a face needs to be determined in the first place in order for tracking to occur.

2.6.3 Practical Face Tracking

Practical face tracking involves solving all or at least most of the challenges mentioned above. For this purpose, face tracking operations implemented in this project use two servo motors and a webcam to track a subject's face.

The two servo motors are mounted in such a way to allow for the first motor to rotate in the horizontal (around the 'y') axis, while the other motor rotates in the vertical (around the 'x') axis, while the camera is attached to the second motor. Figure 3.27 shows a picture of the setup.

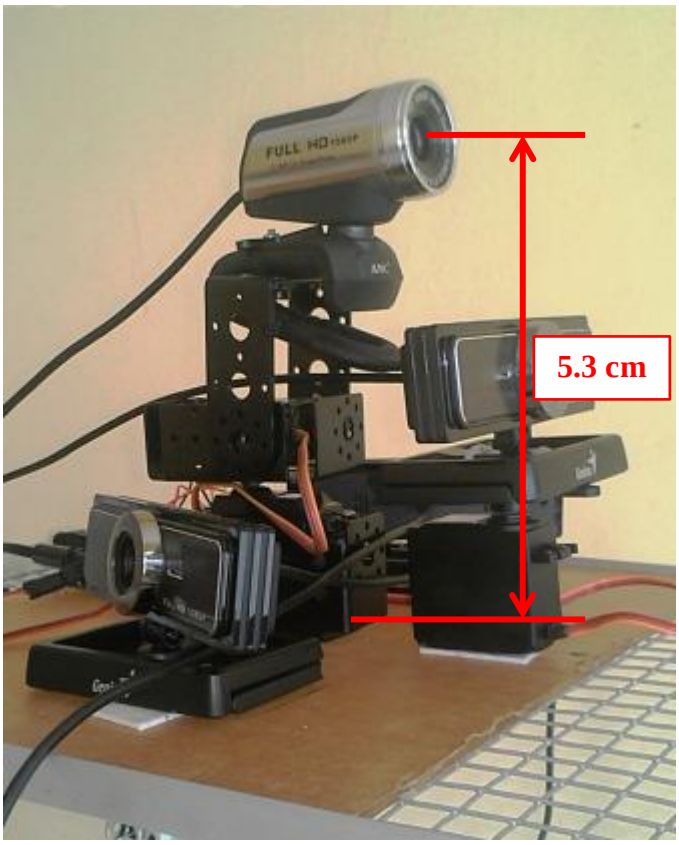


Figure 3.27: The face tracking rig.

When a human face has been detected by the system and is required to be tracked, a signal is sent to the Arduino board controlling the two servo motors to align the first (horizontal) servo motor such as the face is in the vertical middle of the screen, and align the second (vertical) servo motor such as the face is in the horizontal middle of the screen.

In this way, the system will try to keep the face in the middle of the screen at all times. This approach help compensate for any sudden or quick motion that may otherwise result in the face being off limits to the camera's field of view.

Tracking continues until either the face is no longer detected, or is no longer required to be tracked.

3.7 Obstacle Avoidance

3.7.1 Introduction

In general, the act of obstacle avoidance in robotics involves steering a robot in such a way so as to prevent a collision between the robot and the obstacle. Obstacle avoidance is considered a cornerstone of autonomous mobile robotics; in which a method for avoiding a collision between the robot and an obstacle is in place.

Challenges to autonomous mobile robots in obstacle avoidance involve first the detection and second the avoidance of the obstacle. The sensor(s) used tackle the first part; the detection, while the robot's drive mechanism tackles the second part, the avoidance.

Since the robot used in this project is an autonomous mobile robot, an obstacle avoidance method has been used to prevent collision with objects that could be in the robot's path.

3.7.2 Types of Obstacles

In the real world, there are many types of obstacles that a robot may come across. From stationary object such as walls and table legs to dynamic objects such as a person's foot or leg.

That's not all, there are also objects that are hollow such as a metal or plastic mesh, or even a pipe, there are objects that are transparent such as glass or some types of plastic, object that change shape such as cloth and finally objects that deflect (scatter) or absorb a sensors signal rendering it invisible to the robot. Finally it is not impossible to come across an object which combines two or more of these properties; for example being deflective to the sensor's signal as well as being dynamic and deformative; an example of such an object is may be some types of cats, dogs and/or birds.

3.7.3 Types of Sensors

Many types of sensors are used by the different robots to detect the multitude of obstacles that may lie ahead. Logically, a robot designer would equip the robot with sensors capable of detecting the obstacles most likely to appear in the robot's environment. A robot intended for work in an office would encounter different obstacles from a robot intended to work outdoors, and a robot that flies may very well encounter different obstacles to those encountered by robots moving on the ground.

The sensors used to detect obstacles in autonomous mobile robots generally fall into two categories: active and passive. Contact or mechanical sensors that trip when pushed by an obstacle fall into the passive type of sensors.

Active sensors emit a signal and detect an obstacle based on the reflected signal's properties such as the time of flight (time between sending the signal and receiving a reflection). Active sensors include the following:

- Infrared sensors.
- Laser scanners.
- Ultrasonic sensors.

Passive sensors on the other hand make use the signals naturally available in the environment (such as light) for the purpose of obstacle detection. Passive Sensors include the following:

- Infrared sensors.
- Photo diodes
- Cameras
- Gas sensors
- Contact (mechanical) sensors.

Depending on the type and operation, an infrared sensor can either be an active or passive sensor. Active infrared sensors use an infrared transmitter to emit an infrared signal and then measure the reflection. Passive infrared sensors, only measure the level of infrared signal received without emitting any.

3.7.4 The Robot's Obstacle Avoidance

As mentioned in the previous section, there are numerous challenges associated with the detection of obstacles, resulting in the failure of detection and ultimately resulting in the robot crashing into the obstacle.

For the purpose of this project, an approach has been devised to detect obstacles naturally present within the robot's environment such as:

- Walls: Flat
- Boxes: Flat or cornered
- Chairs and Tables: The legs can be of any shape, cylindrical or otherwise.
- Other miscellaneous objects that may be transparent, hollow or have irregular shapes.

The sensors used in the robot in this project were ultrasonic sensors. These sensors have many limitations, the most relevant to the current system are:

- Depending on the approach angle, the sensor's signal can deflect (not reflect back to the receiving sensor) of the obstacle making it undetectable.
- The sensors may reflect of other surfaces resulting in false detection of obstacles or false reporting of obstacle distance from the robot.

To solve for these limitations and use the sensors to successfully detect obstacles in the robot's path, the sensors were tilted inwards at an angle. This allowed for detecting objects that were previously undetectable by employing such

sensors in an outward inclination as usually implemented on mobile robots. Figure 3.28 shows the sensors and their inward inclination.

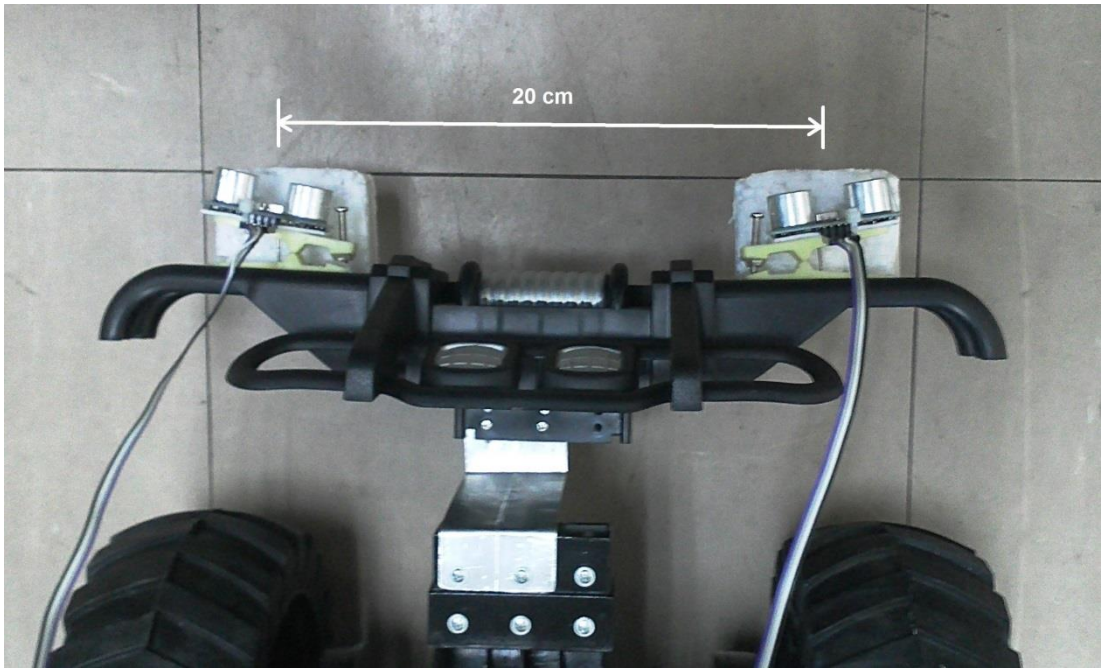


Figure 3.28: The ultrasonic sensors used with the robot

The inclination angle was calculated based on the desired minimum detection distance of 0.8 meters as follows: R1 = 3 m, R2 >= 0.8 m, d = 0.1 m (Figure 3.29)

$$\tan \emptyset = \frac{R_2}{d} \qquad \text{--- (3.3)}$$

By substituting the values of R_2 and d, we find that:

$$\tan \emptyset = 8$$

Hence,

$$\tan^{-1} 8 = 82.875\ degrees$$

From this we conclude that θ (the inward inclination angle) is 7.125 degrees, as can be seen in Figure 3.29.

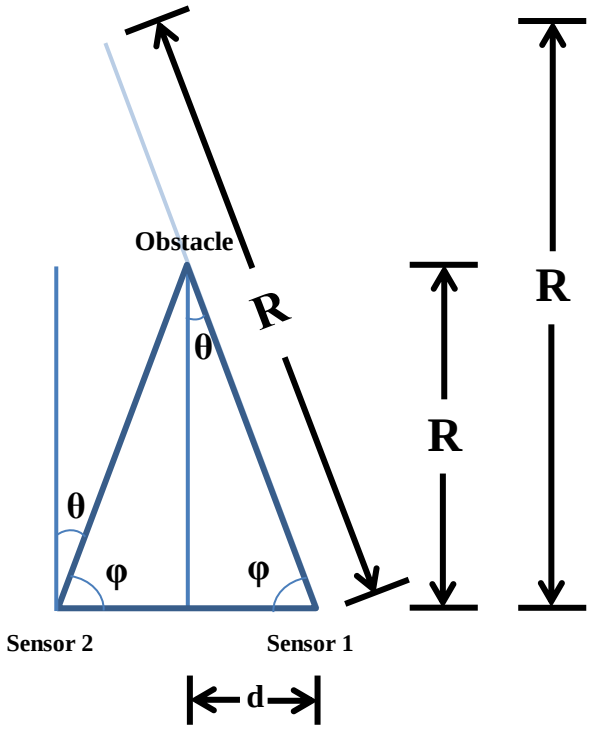


Figure 3.29: Calculating the sensor angle

Since the sensor's range (R_1) is 3 meters, using this angle, the sensor would also be capable of detecting objects that are 2.98 meters away.

As can be seen from Figure 3.29, two sensors were used to detect the obstacles, so that if one sensor fails to detect the obstacle due to the angle of approach or the obstacles' shape, the other sensor would apply the signal from a different angle thus allowing the robot to discover the obstacle ahead.

3.8 Navigation

3.8.1 Introduction

Navigation in robotics generally involves moving the robot from one location to another. To accomplish this task the robot has to be equipped with some means to move it in a controlled manner.

The main challenge to robot navigation is the ability to traverse the environment efficiently. However, the real world is a dynamic place; hence the environment and the surface the robot is treading can change resulting in slippage or unintended change in direction.

As the robot used in this project will be used indoors, it is equipped with four wheels, the motion of which control the direction of the robot.

3.8.2 Types of Robot Drives

Depending on the environment in which a robot will operate, it should be equipped with a suitable locomotion. Choosing a particular type of robot drive affects the speed, maneuverability, and stability of the robot. Surface robots can be equipped with any of the following types of drive systems [118][119][120]:

- Wheels: wheels are very common with many robots; they allow for good stability, steering and speed, however, they may exhibit slippage and jamming in certain situations. Wheeled robots can have any number of wheels depending on the task and purpose of the robot.
- Continuous Track Locomotion (Tank Tread, Caterpillar Tracks): this locomotion belongs to the wheel category, however they differ in having a sort of a belt surrounding the wheels which results in a better ability to traverse more complex and uneven terrain, but require more energy to do so. In some cases steering can cause problems due to the large area that the

caterpillar belt has to sweep across the ground to accomplish the change in direction.

- Pedals: pedals are used with more sophisticated robots. Their main advantage is the ability to traverse rough and irregular terrain such as outdoors as well as the ability to climb up and down stairways. The two main disadvantages are:
    - They comprise of a larger number of motors compared to wheeled robots, hence they are more complex and require a greater level of control than wheel drive systems.
    - They are generally slower than wheel drive systems.
- Crawl: Certain robots are not equipped with wheels or pedals, instead they use their body motion to creep or crawl on the surface. This kind of locomotion has the advantage of being able to travel across any surface, but is limited in speed and consumes more energy.
- Hybrid: Some robots employ a hybrid locomotion system in which a combination of drives may be used to avail the advantages while avoiding the disadvantages of using a single type of locomotion.

As can be concluded from the above, the more complex the terrain that the robot has to traverse, the more complex the drive system, the slower the speed, and the more energy the robot has to devote to achieve locomotion. Therefore it is necessary for a robot designer to choose wisely a drive system that allows the robot to accomplish its tasks while consuming the least amount of energy.

3.8.3 Path Planning

When a robot needs to move to a certain place or location, it has to conduct what is known as path planning. Path planning is deciding the best route to pursue in order to get to the desired location.

Depending on the environment and the way that the task is to be accomplished, path planning can be a sophisticated problem as it may involve taking a longer path instead of a shorter one in order to evade a certain obstacle or danger.

Figure 3.30 shows an example of such a situation in which a robot has to follow a longer route to arrive at the desired destination.

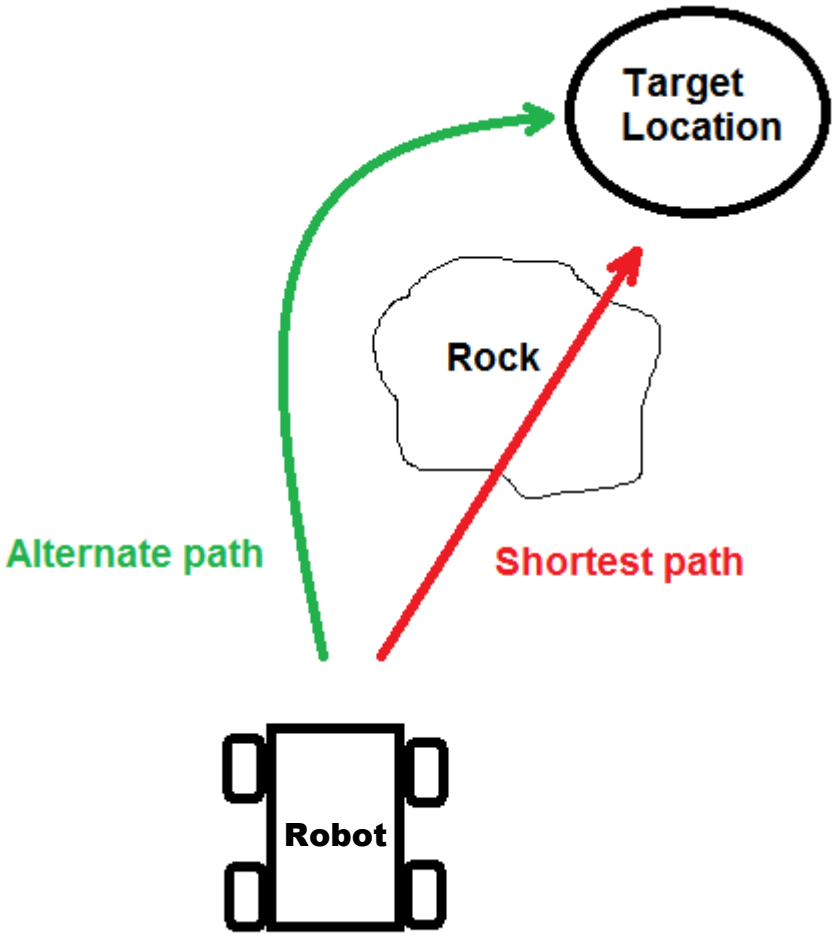


Figure 3.30: Choosing a different path

Path planning may be local; in which a robot has to decide how to avoid an obstacle, or global; in which a robot optimizes the overall travelled distance to the target [121].

3.8.4 The Employed Navigation Approach

Practical navigation relies in part on the type of locomotion the robot is equipped with. This is important as the robot's type of locomotion may allow it to

skip or drive over an obstacle rather than go around it thus gaining shortening the distance to the target and possibly saving time and energy in the process.

In the following sections, the locomotion used in the robot of this project is outlined as well as the approach used in navigating the robot to its target.

3.8.4.1 The Robot's Locomotion

For the robot used in this project a drive system comprising of four wheels was chosen to navigate the robot across its indoor environment. The four wheels have a diameter of 16 cm which makes them large enough to overcome any changes in the surface of an indoor environment such as an uneven floor. The size of the wheels is also adequate for overcoming small or thin obstacles that it may encounter in an indoor environment such as a pen, some paper, a wire …etc. Figure 3.31 shows the robot wheels besides some objects to give a sense of its size.

In this drive system, steering is accomplished by operating two wheels on one side while stopping or reversing the wheels on the other side. This combination of wheels and steering system has the following traits:

- The ability to steer left or right with a lower energy requirement compared to track locomotion.
- The ability to rotate in place without requiring to drive forward or backwards to accomplish that.
- Better traction compared to single and two wheeled robots (with one or two castor wheels).
- Better stability compared to single and two wheeled robots as the robot remains on four contact points all the time.
- Simpler control in comparison to castor wheels as only the side that needs to advance forward is operated.

Figure 3.31: The robot's wheels

Practical robot navigation not only involves the robot's drive system, but also the planning of its path to arrive at the desired destination in the least amount of time while spending the least amount of energy.

3.8.4.2 The Robot's Navigation

In this system an approach is used which uses local navigation with dynamic rerouting. This is done to maximize the robot's response to real world, real-time changes in target location or other factors that may affect its navigation path or target.

Local robot navigation involves obstacle avoidance as these block the robots path and consequently the robot would have to navigate around or away from them. The challenge in this system is to navigate the robot to accomplish the following tasks in the shortest possible time with the least amount of energy:

- **Approach subject:** The difficulty is that the target subject may not be static; hence the robot may have to constantly change its path (route).
- **Avoid obstacles:** Avoid different types of static and dynamic obstacles while approaching the main target.

The subject is first tracked with a camera, and when the conditions requires the robot to approach the subject; the camera angle is transmitted to the Arduino board which controls the motion of the robot wheels. The direction of motion is decided based on the presence of obstacles in front of any or both of the ultrasonic sensors. Figure 3.32 is a flowchart illustrating the approach.

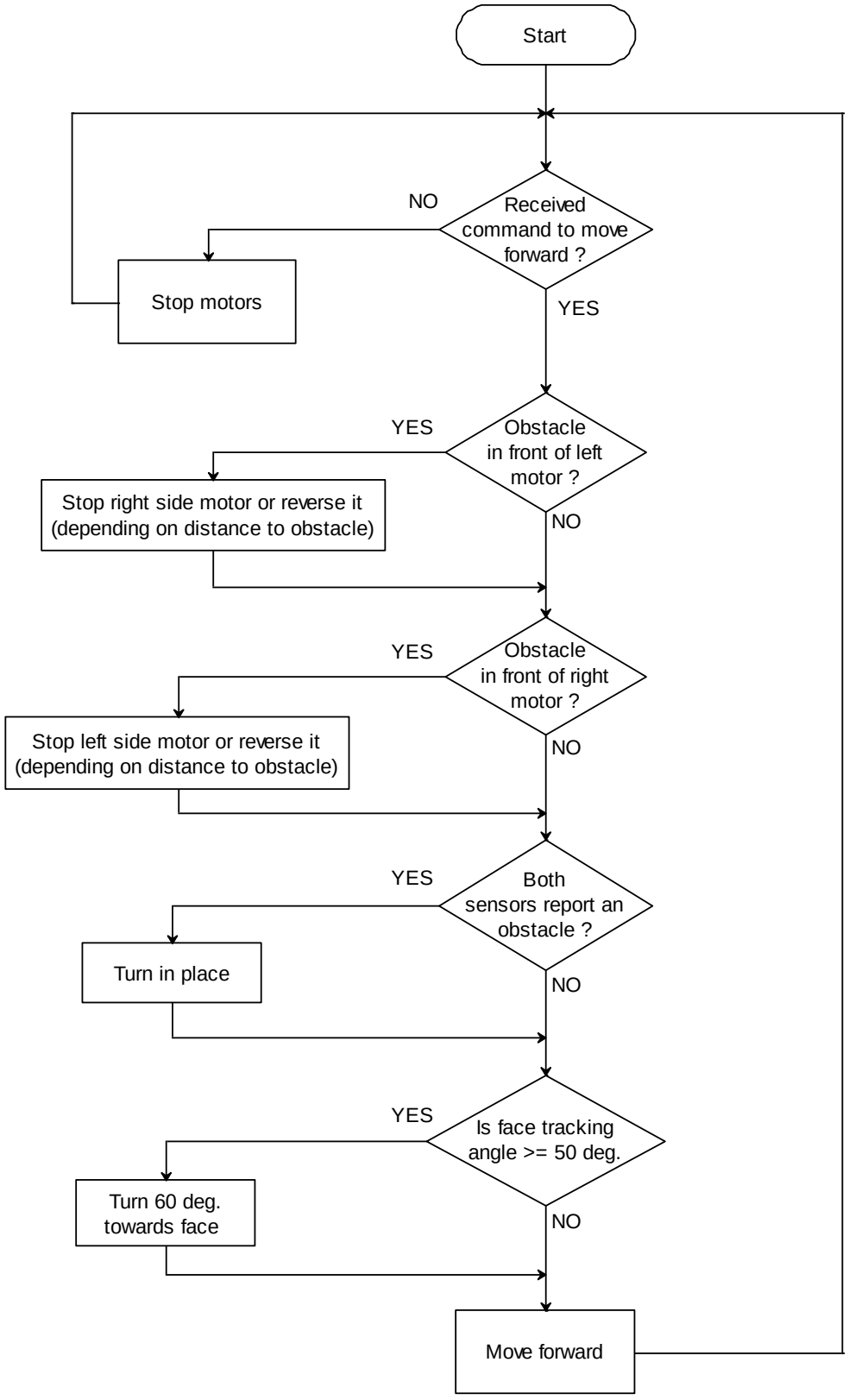


Figure 3.32: The approach used to arrive at the desired subject

3.9 Authentication

3.9.1 Introduction

Authentication generally involves verifying the identity of a person. This action may be required to allow the person access to certain facilities or resources. Authentication normally includes comparing of information supplied by the person in question against information stored in the system for the purpose of confirming the permissions available for that particular person [122].

In this system, authentication is used to verify the identity of subjects entering or using the environment in which the robot is located. This is based on information previously stored inside the robot for the purpose of confirming the authorized persons from those who are not allowed access to the robot's environment.

3.9.2 The Employed Authentication Approach

For our practical purposes, an authentication operation has to be automatic, quick and non-intrusive in order to enable subject identification without impeding their normal operation and to accomplish a swift and accurate verification of their identity.

Practical authentication also involves using an approach to strengthen its effectiveness and accuracy in determining the identity of authorized individuals from those who should not be allowed access.

In the system used in this project, a two-tier authentication scheme has been utilized to ensure better verification of a subject's identity. The first level of authentication is facial identification of the subject; when the confidence level of facial identification is low, a second authentication action is activated requiring the subject to enter a password to prove his/her affiliation with the facility and/or its resources.

The facial identity of all authorized personnel has been previously stored in the system, and when a human subject is encountered, his/her face is checked against the images in the database to verify its identity.

Each member of staff has his/her own password. The password is used in case facial identification of an authorized staff fails, in which case entrance of the password will grant him/her access, while extra facial images are captured for later addition to the database. The user can change his/her password to prevent possible compromise as shown in Figure 3.33 where a user is entering his/her password on the robot.

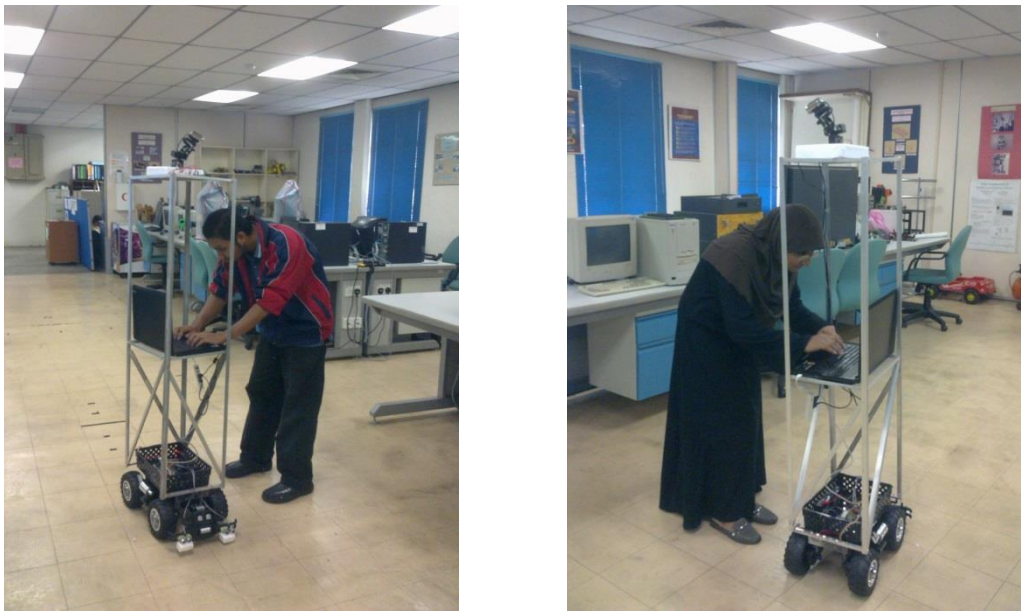


Figure 3.33: A user enters his/her password on the robot

Summary

This chapter discussed the design and function of the security robot conducting its operations in an indoor location. The methodology can be divided into three main operations: Motion detection to detect the presence of a moving object, which may indicate the presence of a person. Face detection in the area where the motion was detected. And finally face recognition is conducted on the detected face. Each of these three operations contains sub-operations to maximize its effectiveness and yield the best possible results in terms of speed and accuracy. The OpenCV library of functions was used extensively in conducting these three operations.

CHAPTER 4

EXPERIMENTS

In this chapter, the different experiments that were conducted and their settings will be outlined. Several experiments were performed to test the different aspects of the system and reach the best setting for a practical system

The experiments can be broadly categorized into software and hardware experiments. Software experiments are the ones related to motion detection, face detection and face recognition, while hardware experiments are related to face (subject) tracking, navigation and obstacle avoidance.

4.1 Motion Detection Related Experiments

4.1.1 General Objective

The objective of these experiments is to examine the aspects related to the implementation of motion detection in the current system. Two experiments were carried out to find out the best devices and software setting for an efficient motion detection operation.

4.1.2 Camera's Field of View Experiment

4.1.2.1 Objective

There are two objectives behind this experiment:

1. To find out the field of view of different cameras in order to use the camera with the widest field of view in this project.
2. To compare the wide angle image of the different cameras in terms of angle and view.

4.1.2.2 Description

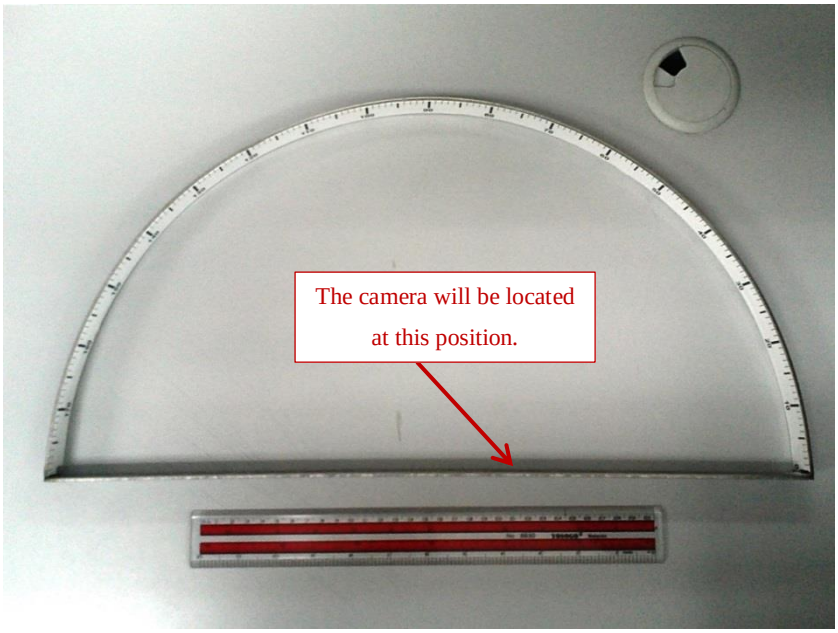
In order to find the field of view of each camera, a simple apparatus was designed to assist in determining any camera's field of view. This apparatus was used for measuring the camera's field of view as follows:

1. A camera is positioned at the center of the circle as shown in Figure 4.1-a
2. A picture is captured by the camera.
3. The picture shows the markings on the apparatus, which is used to calculate the camera's field of view by subtracting the higher degree value on the left of the image from the lower degree value on the right. Figure 4.1-b shows the marking on the apparatus.

For each camera two images are captured. The first one is a normal image, while the second one is an image taken with a fisheye lens placed on the camera's sensor. The experiment begins with placing a camera at the middle point, in front of the view angle measuring apparatus; then an image of the FoV measuring apparatus (the scale) is captured using that camera at that particular location. The image is saved and is later viewed to determine the maximum angle of view.

(a)

The Apparatus



(b)

Finding the field of view from the markings

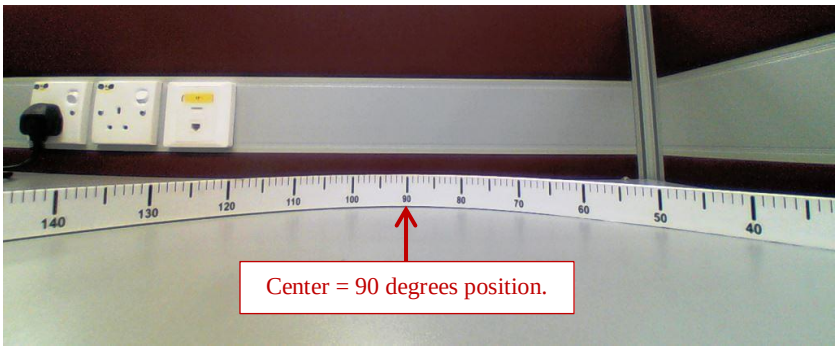
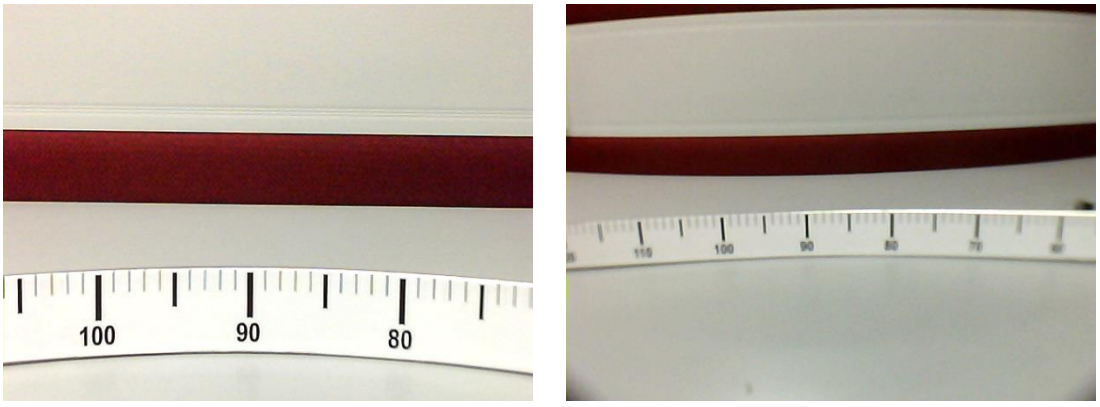


Figure 4.1: Determining the camera's field of view

A second image is then captured with a fisheye lens fixed on the same camera's sensor. The image is also saved and then the two images are compared to find the total increment in the camera's field of view using the fisheye lens. Figure 4.2 shows the cameras used in the experiment, while Figure 4.3 shows an example of the two images captured by one of the cameras during the experiment.

Figure 4.2: The camera's used in the experiment, from left to right: The slim webcam, the A1-Pro webcam, the fisheye lens, the E72, the Genius F100 and the Full-HD webcam



(a) (b)

Figure 4.3: The images captured by the A1-Pro webcam

a) Normal Image, b) Fisheye Image

4.1.2.3 Setting

For conducting the experiment, an apparatus was used to measure the camera's field of view. The apparatus, shown in Figure 4.1 is made up of a half-circle, the diameter of which is 50 cm. The length of the half circle circumference is 78.54 cm and contains marking for the 180 degrees. The captured images from all the cameras have a resolution of 640x480, except for the Genius F100, with which three images were captured, one at 640x480, one at 1280x720 and one at 1920x1080. Figure 4.4 shows the setting of the experiment while testing the Genius F100 webcam.

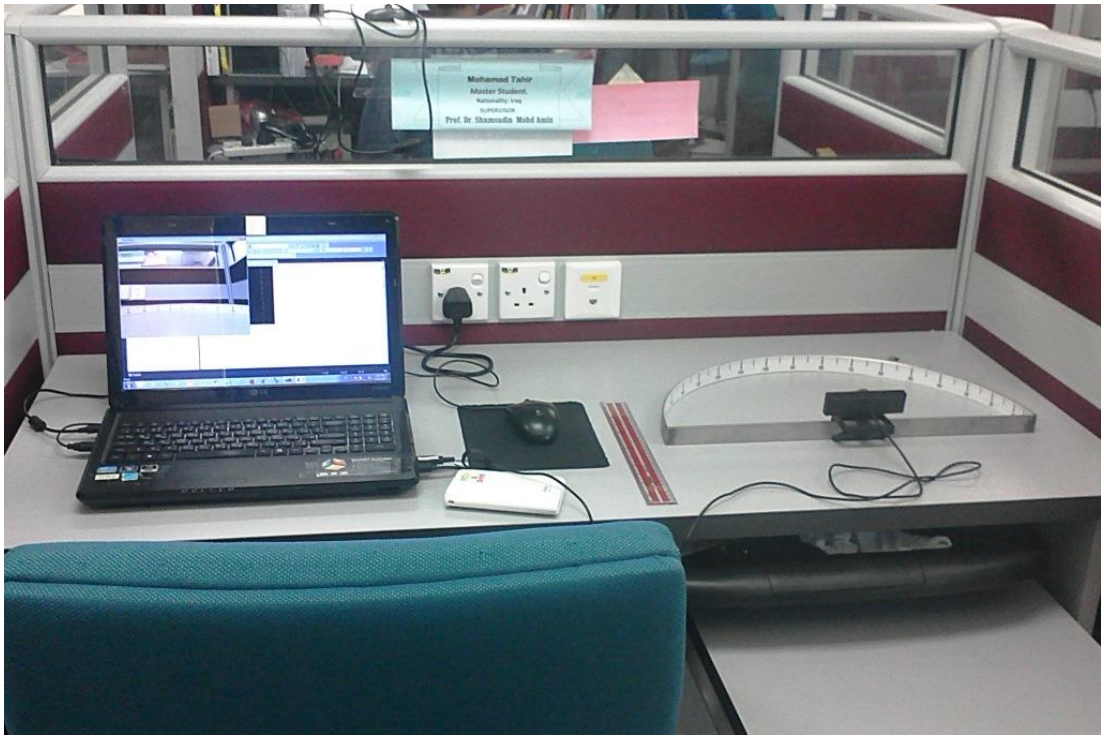


Figure 4.4: The camera field of view experiment setting

4.1.2.4 Results

The measured field of view angles from the captured images for all the tested cameras were tabulated as shown in Table 4.1:

Table 4.1: The results of measuring the field of view for different cameras

Camera	Normal FOV	Wide FOV	Using
A1-Pro Webcam	34	65	Fisheye Lens
Slim Webcam	39	75	Fisheye Lens
Full-HD Webcam - VGA mode	46	93	Fisheye Lens
Full-HD Webcam - HD mode	60	98	Fisheye Lens
Genius F100 Webcam	88	112	HD & Full-HD Mode
E-72 Camera	51	100	Fisheye Lens
LG P920 Camera	51	100	Fisheye Lens

Legend: FOV: Field of View (in degrees).

4.1.2.5 Discussion and Analysis

From the results presented in Table 4.1, we find that the Genius F100 camera has the largest field of view in comparison to all the cameras, both in normal and wide FOV modes. This makes this camera the best choice to be used in surveillance operations as it can cover a wide area. Using three camera of this mode in the wide-angle FOV mode can efficiently cover the whole 360 degrees which would normally require a minimum of eight cameras of the A1-Pro or Slim-Webcam type to do the same job.

Also, by inspecting the results carefully, we find that normal webcams have a narrower field of view than the mobile phone camera which in turn has a narrower field of view than the Genius F100 camera which is originally manufactured for a wide angle perspective. This is expected as normal webcams need only show the person directly in front of the camera and in some cases it may be preferred to hide as much of the surrounding as possible for privacy reasons. Mobile phone webcams are wider than webcams as they are intended for general purpose image and video capturing.

Finally, although the fisheye lens is advertised to enable 180 degree field of view, practically, it only provides an increase in the camera's viewing angle to almost double.

4.1.3 Threshold and Lighting Effect Experiment

4.1.3.1 Objective

There are two objectives for this experiment:

1- To determine a suitable threshold to be used for motion detection of humans or intruders in low and high lighting conditions.
2- To determine the minimum motion by a human subject that would trigger detection with the chosen threshold.

The first objective is important to enable the robot to detect motion in spaces with low lighting conditions as well as spaces that are well lit. The second objective is important as it measures the robot's ability to detect motions that are as equal to or larger than as a person's body part such as a hand, head, arm or leg movement, while ignoring motion equal to or smaller than those caused by flying insects or other miniature objects.

4.1.3.2 Description

To conduct the experiment, a webcam operated by a motion detection program was used in the research room of the lab. The program was started during daytime and in normal lighting to verify the operation of the setup, then the experiment commenced.

The experiment started by placing the motion detection camera at one end of the lab and switching off all the ceiling lights as well as closing all the curtains, so only a small amount of light was coming from the sides of the curtains. Motion detection was then measured by trying to capture the walking motion towards the camera from the other end of the lab at varying speeds.

The same actions were repeated without ceiling lights, with one row of ceiling lights, two rows and the full three rows switched on. This was done to verify the effect of lighting conditions on the operation of the motion detector program. Also, the motion detector program was used to check to find out whether false detection of motion can be triggered by camera noise in bright and low lighting conditions.

4.1.3.3 Setting

The experiment was carried out in lab P08L02 of the faculty of electrical engineering at Universiti Technologi Malaysis. The lab contains three rows of ceiling lights as well as windows with blue blinds. Distance signs were placed at suitable intervals to show the distance of the subject while in motion, and the Genius F100 camera was used to conduct the experiment. Figure 4.5 and Figure 4.6 show the setting of the experiment.

Figure 4.5: The lab in zero and full lighting conditions

Figure 4.6: The experiment setting

During the experiment a subject would walk from the far end of the lab towards the camera at varying walking speeds. The experiment is repeated three times, first slow, then normal, then at a fast walking pace, and the average value of the distance at which a motion is detected is calculated. Other settings are as follows:

- The Genius F100 camera was capturing video at HD quality (1280x720) with wide angle view.
- The video speed was at around 10 frames per second.
- Distance markings were placed at the 1 – 8 meters from the camera.
- The experiment was carried out during daytime with the windows blinds down. This allowed minimal light to pass through allowing humans to see while being dark for a normal camera as can be seen from Figure 4.5.

4.1.3.4 Results

The experiment was repeated three times to consolidate the results and the average values for all iterations was calculated and recorded. Table 4.2 shows the gathered results, and Figure 4.7 shows the results in graphical format.

Table 4.2: The average values of the three repetitions of the experiment

Lights	Thld	Dst for 1st DoM	Dst for 1st DoF	Dst for Cnt FD	Err MD
0	10	5.67	3.83	0.00	Yes
0	15	5.17	0.50	0.00	No
0	20	2.75	0.00	0.00	No
0	25	1.75	0.00	0.00	No
1	10	12.67	11.67	9.67	Yes
1	15	13.00	13.00	7.08	No
1	20	13.00	10.17	6.08	No
1	25	11.83	7.00	5.42	No
2	10	13.00	12.17	6.92	Yes
2	15	13.00	12.50	7.50	No
2	20	13.00	6.42	4.08	No
2	25	12.33	4.25	2.50	No
3	10	13.00	13.00	10.50	Yes
3	15	13.00	9.50	7.92	No
3	20	12.83	7.67	4.92	No
3	25	12.50	6.17	4.08	No

In Table 4.2, the headings have the following meanings:

Lights : The number of ceiling lighting used

Thld : The threshold used

Dst for 1st DoM : The distance (in meters) for the first detection of motion. This is the furthest distance at which motion was detected.

Dst for 1st DoF : The distance (in meters) for the first detection of a face based on the detected motion area. This is the furthest distance at which a face was detected based on the detected motion area.

Dst for Cnt FD : The distance (in meters) for continuous face detection based on motion detection. This is the distance at which face detection starts and continues in the frames that follow.

Err MD : Erroneous detection of motion

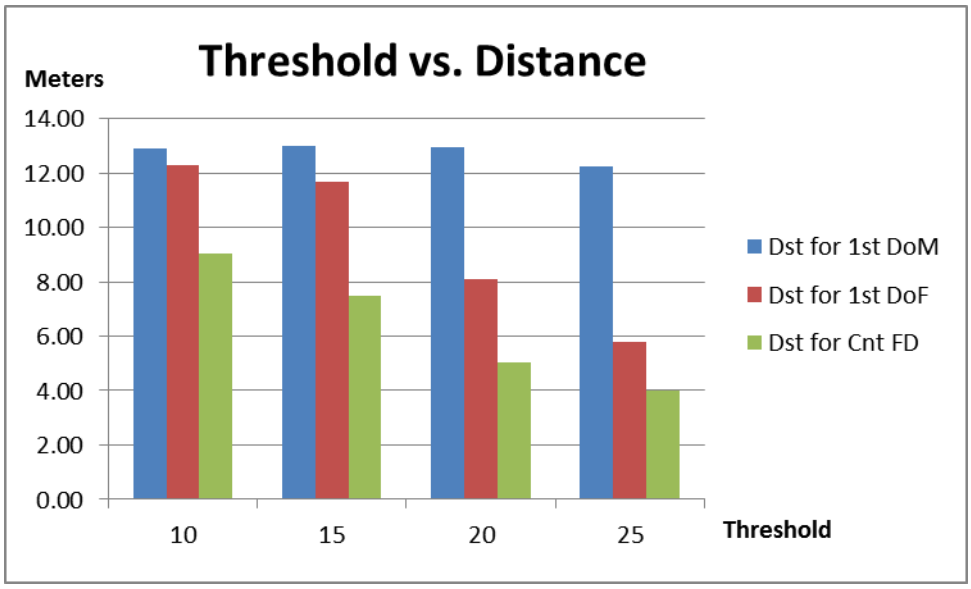


Figure 4.7: A summary of the gathered results from the experiment

4.1.3.5 Discussion and Analysis

By examining Table 4.2, we can observe the following:

1. For the condition in which all the lights were off, the detected motion area does not include the face of the subject in a continuous manner; sometimes the face is within the detected motion area and sometimes not.
2. When all the lights are off, a face may be detected at close range using a low threshold, however, using a threshold higher than 10 severely degrades inclusion of face area within the detected motion area rendering the operation of looking for a face in the detection motion area fruitless.
3. In all lighting conditions, a threshold of 10 (and presumably less) will cause false detection of motion. Therefore a higher threshold value should be used.
4. In all lighting conditions, as the threshold increases, the sensitivity and hence the detection distance increases. For this reason, the lowest possible threshold should be used.
5. By combining the observations of 3 & 4, a threshold of 15 seems to be a practical solution.
6. Increasing the lighting from 1 to 2 or 3 does not much affect the distance at which motion or face detection can occur. Therefore, it is possible to operate

the robot efficiently with only a single strip of ceiling lights being on (the middle one).

7. Motion detection can occur as far as 13 meters (the end of the lab), while continuous face detection occurs at around 7 meters. This suggests that the robot would only be able to recognize subjects that are 7 meters away. This may not be a limitation, as the size of the face may too small for recognition beyond that distance. The reader is advised to refer to section 4.2.2 for more details regarding face size relation to distance from the camera.

By examining Figure 4.7, we observe that:

1. There seems to be no difference in the distance of first motion detection between threshold 10 and 15. Therefore, choosing threshold 15 instead of 10 as pointed out in point 5 above, may have no effect on the motion detection distance.
2. The difference in the furthest distance at which a face can be continuously detected does not degrade substantially by choosing threshold 15 over threshold 10 and is practically acceptable.
3. Choosing a higher threshold value, such as 25, may not significantly affect the furthest distance at which motion is first detected, but it greatly affects the distance at which a face is included in the detected motion area.
4. Effectiveness of the motion detection operation as well as the inclusion of the face area within the detected motion is inversely proportional to the threshold value applied.

Due to the above, it is recommended to use the lowest threshold value which does not cause false detection of motion. In this case, a value of 15 seems to be a suitable choice.

4.2 Face Detection Related Experiments

4.2.1 General Objective

The objective of the face detection experiments is to test the influence of various factors on the robot (computer) ability to detect a human face. Different experiments were carried out to find out the best possible setting for a reliable face detection outcome.

4.2.2 Face-Size vs Distance Experiment

4.2.2.1 Objective

This experiment has three objectives:

- The first objective of this experiment is to determine the smallest detectable face using a camera that may be used during the robot's operation.
- The second objective is to try and find a possible relation between subject distance and face size.
- The third objective is to find the approximate distance at which there's enough detail to distinguish a face of one subject from another, in order to use this information in face recognition.

4.2.2.2 Description

This experiment was carried out in the lab. During the experiment several stationary cameras (of different types) were simultaneously used to capture video footage of the environment, and a face detection program was later used to detect the face of the subject appearing in front of each of the cameras.

In order for the experiment to be comprehensive, eight different subjects from four different races and different genders took part in the experiment. In each iteration, one of the subjects would stand in front of the cameras for a few seconds at each floor marking to allow the cameras (and later, the face detection software) to capture the environment image and find the face within.

The cameras were placed at one end of the markings to capture the video footage that will be used to detect and calculate the face size. In order to relate the face size to the distance at which the subject is standing, the subjects were requested to hold a cue card showing the distance at which they are standing from the cameras. Figure 4.8 shows a subject standing at a floor marking holding a cue card.

Figure 4.8: A subject holding a cue card and standing 4.0 meters away from the cameras

The face image, and face size were all determined and saved. Later they were used to deduce a relation associating the subject's face size to the subject's distance from the camera. Several face images were captured for each subject in order to verify the face size across multiple frames.

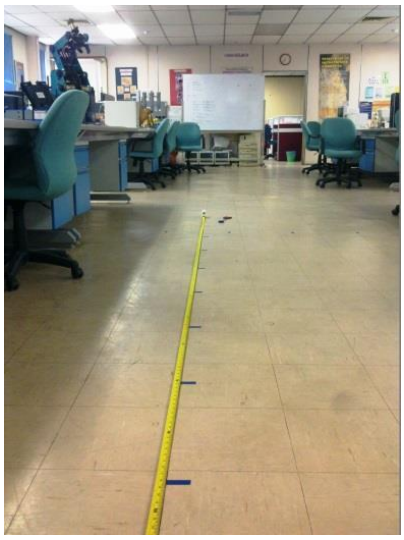
4.2.2.3 Setting

The experiment settings, shown in Figure 4.9, were as follows:

- Floor markings were places 0.5 meters apart for a distance of 6.0 meters as shown in Figure 4.9 (b).
- A built-in laptop's camera was used to capture the images. The camera has a resolution of 640x480.
- In this experiment several cameras were used: the laptop's camera (640x480), the Genius F100 (wide angle), the E-72 (320x240) and a Full-HD camera.
- The laptop's camera was placed at a height of 150 cm to be able to capture the face image of all the subjects, tall and short.
- The normal overhead lab lighting was used. No extra assistive lighting was used.
- No extra lenses or other optical gear were used.

During the experiment subjects kept a standard facial expression of a normal face. Only at very close distance of 0.5 meters that some subjects smiled or laughed. The change in the facial expression at such a short distance had no effect on the detection effectiveness as their faces were detected without issues.

(a)



(b)

Figure 4.9: The setting used for conducting the experiment
a) The laptop and its built-in camera, b) The floor markings

Table 4.3: The face size in pixels and subject distances gathered from the experiment

	Asian	Malaysia Male		Malaysia Female		Chinese	Middle Eastern		Av
Distance	Habib	Azrai	Luqman	Amirah	Naqiah	Goh	Alaa	MTA	F
0.5	233.0	253.0	237.0	197.0	196.0	251.0	215.0	254.0	
	231.0	251.0	241.0	189.0	201.0	253.0	220.0	267.0	
	248.0	253.0	240.0	190.0	205.0	251.0	219.0	265.0	
	232.0	262.0	240.0	188.0	205.0	273.0	228.0	264.0	
	236.0	254.0	255.0	189.0	208.0	259.0	224.0	278.0	
	231.0	260.0	238.0	191.0	208.0	256.0	228.0	266.0	
	236.0	249.0	238.0	186.0	211.0	252.0	225.0	270.0	
	231.0	259.0	232.0	183.0	208.0	254.0	221.0	266.0	
	234.8	255.1	240.1	189.1	205.3	256.1	222.5	266.3	
1.0	123.0	123.0	120.0	116.0	107.0	125.0	129.0	127.0	
	127.0	120.0	117.0	119.0	104.0	126.0	126.0	118.0	
	126.0	122.0	123.0	118.0	103.0	131.0	127.0	129.0	
	123.0	127.0	123.0	119.0	107.0	128.0	123.0	124.0	
	127.0		121.0	117.0	107.0	128.0	121.0	123.0	
	127.0		120.0	120.0	107.0	129.0	122.0	129.0	
	126.0		120.0	117.0	105.0	134.0	122.0		
	128.0		121.0	115.0	106.0	126.0	123.0		
	125.9	123.0	120.6	117.6	105.8	128.4	124.1	125.0	
1.5	80.0	74.0	76.0	74.0	74.0	89.0	86.0	83.0	
	78.0	79.0	79.0	77.0	71.0	88.0	87.0	83.0	
	78.0	78.0	76.0	78.0	71.0	90.0	86.0	82.0	
	78.0	78.0	76.0	76.0	69.0	83.0	85.0	88.0	
	80.0		77.0	80.0	73.0	85.0	90.0	85.0	
	81.0		78.0	78.0	82.0	85.0	86.0	87.0	
	81.0		76.0	79.0	72.0	82.0	88.0	86.0	
	80.0		81.0	80.0	68.0	86.0	86.0		
	79.5	77.3	77.4	77.8	72.5	86.0	86.8	84.9	
2.0	56.0	58.0	56.0	55.0	54.0	60.0	61.0	62.0	
	60.0	59.0	58.0	60.0	53.0	62.0	62.0	63.0	
	60.0	59.0	58.0	58.0	54.0	62.0	64.0	63.0	
	61.0	60.0	55.0	58.0	54.0	66.0	66.0	63.0	
	61.0	58.0	59.0	59.0	55.0	63.0	64.0	64.0	
	58.0		59.0	58.0	54.0	63.0	64.0		
	59.0		57.0	59.0	52.0	64.0	67.0		
	61.0		57.0	59.0	53.0	64.0	63.0		
	59.5	58.8	57.4	56.0	53.6	63.0	63.9	63.0	
2.5	48.0	49.0	48.0	48.0	46.0	49.0	52.0	50.0	
	48.0	50.0	47.0	46.0	45.0	51.0	51.0	50.0	
	48.0	49.0	47.0	47.0	46.0	51.0	51.0	50.0	
	47.0	50.0	47.0	46.0	44.0	49.0	52.0	51.0	
	49.0	48.0	47.0	45.0	46.0	52.0	51.0	51.0	
	48.0		47.0	46.0	46.0	51.0	51.0		
	50.0		48.0	47.0	46.0	51.0	53.0		
	47.0		48.0	48.0	47.0	52.0	52.0		
	48.1	49.2	47.4	46.6	45.8	50.8	51.6	50.4	
3.0	43.0	43.0	41.0	42.0	40.0	43.0	47.0	45.0	
	44.0	43.0	40.0	40.0	40.0	44.0	46.0	44.0	
	43.0	43.0	42.0	39.0	40.0	47.0	43.0	44.0	
	43.0	42.0	41.0	39.0	39.0	44.0	44.0	44.0	
	44.0	43.0	42.0	41.0	40.0	44.0	46.0	45.0	
	44.0		42.0		40.0	43.0	43.0	44.0	
	43.0		42.0		40.0	44.0	43.0	46.0	
	43.0		42.0		41.0	46.0	44.0		
	43.4	42.8	41.5	40.2	40.0	44.4	44.5	44.6	

	Asian	Malaysia Male		Malaysia Female		Chinese	Middle Eastern		Av
Distance	Habib	Azrai	Luqman	Amirah	Naqiah	Goh	Alaa	MTA	F
3.5	36.0	38.0	35.0	35.0	33.0	40.0	39.0	40.0	
	38.0	38.0	39.0	30.0	34.0	39.0	38.0	39.0	
	39.0	38.0	35.0	34.0	35.0	38.0	40.0	39.0	
	38.0		35.0	36.0		37.0	40.0	38.0	
	38.0		35.0	34.0		39.0	41.0	40.0	
	34.0		35.0			38.0	40.0	39.0	
	36.0		35.0			39.0	43.0		
			34.0			39.0	41.0		
	37.0	38.0	35.4	33.8	34.0	38.6	40.3	39.2	
4.0	29.0	32.0	30.0	26.0	28.0	34.0	34.0	32.0	
	34.0	30.0	30.0			37.0	36.0	33.0	
			31.0			37.0	34.0	29.0	
			28.0			34.0	36.0	32.0	
						36.0	34.0	32.0	
						35.0	34.0	31.0	
						34.0	36.0	31.0	
						35.0	36.0		
	31.5	31.0	29.8	26.0	28.0	35.3	35.0	31.4	
4.5	26.0	28.0	26.0	24.0	24.0	30.0	29.0	28.0	
	26.0	28.0	26.0	25.0	24.0	28.0	29.0	28.0	
	28.0	27.0	25.0		25.0	30.0	29.0	28.0	
	26.0		28.0		25.0	32.0	28.0	27.0	
	26.0		26.0			30.0	30.0	28.0	
						31.0	29.0	28.0	
						28.0	29.0		
						29.0	28.0		
	26.4	27.7	26.2	24.5	24.5	29.8	28.9	27.8	
5.0	25.0	26.0	23.0	22.0	22.0	24.0	25.0	26.0	
	25.0	24.0	23.0		23.0	24.0	25.0	26.0	
	27.0		23.0		23.0	25.0	26.0	26.0	
	24.0		22.0		22.0	26.0	26.0	26.0	
	25.0		22.0			25.0	25.0	26.0	
	24.0		24.0			26.0	26.0	25.0	
	26.0		22.0			25.0	24.0		
	25.0		22.0			27.0	26.0		
	25.1	25.0	22.6	22.0	22.5	25.3	25.4	25.8	
5.5	22.0	23.0				23.0	23.0	23.0	
	24.0	21.0				24.0	24.0	23.0	
	22.0	21.0				23.0	24.0	22.0	
						24.0	23.0	23.0	
						26.0	24.0	24.0	
						27.0	26.0	23.0	
						23.0	23.0		
						24.0	22.0		
	22.7	21.7				24.3	23.6	23.0	
6.0	21.0					21.0	23.0	22.0	
	23.0					24.0	23.0	23.0	
						22.0	22.0	21.0	
						22.0	22.0		
						21.0	23.0		
						21.0	24.0		
						21.0	23.0		
						21.0	23.0		
	22.0					21.6	22.9	22.0	

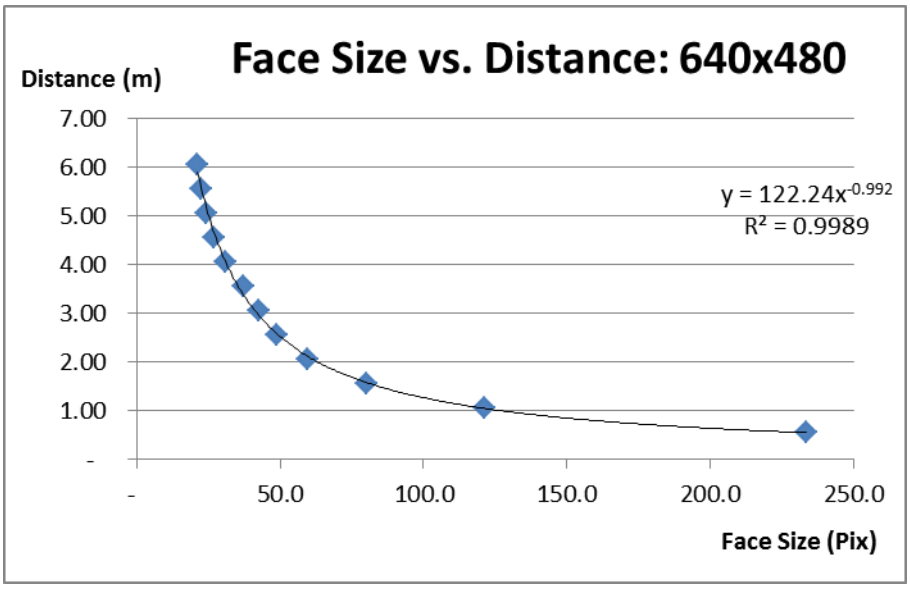


Figure 4.10: A graph showing the relation between distance and average face size

Habib Abdul-Rahman Mohammad Alaa

Figure 4.11: Face images captured at a distance of 6 meters with a 640x480 pixel webcam

4.2.2.4 Discussion and Analysis

By examining the results in Table 4.3, we find the following:

- Not all faces can be detected at far distances: some faces were detected all the way up to 6 meters, while others were not. Although enough time was given to the software to detect the faces, it was unable to find the face in the frame image. This seems to be due to two main factors: Skin color and sharpness of facial features such as the eyes, nose and mouth. From this we can also deduce that low lighting will negatively affect face detection.

- Faces have different sizes: at the same standing distance, some subjects show a larger face size (in pixels) than others.
- Males have larger faces than females: overall, all female faces seem to be smaller in size (in pixels) than their male counterparts.

From the results found, a relation connecting the average face size to the subject distance was deduced as presented in Figure 4.10:

$$y = 122.24\, x^{-0.992}$$

Finally, by inspecting the images in Figure 4.11, we see that facial features are minimal at a distance of 6 meters when face sizes are 20 – 22 pixels. Therefore, it is advised that facial recognition should occur when the face size is larger than 22 pixels. More details regarding the relation of face size to recognition effectiveness is outlined in section 4.3; Face Recognition.

4.2.3 Face-Detection Effectiveness Experiment

4.2.3.1 Objective

The objective of this experiment is to measure the effectiveness of the face detection approach used in this project. The effectiveness in terms of detecting faces in a dynamic, uncontrolled environment, while rejecting false-positives as well as faces of unusable quality.

4.2.3.2 Description

The experiment was conducted in the lab of P08-L01 of the Electrical Engineering Faculty of UTM. The experiment was conducted over a period of two weeks; one week in each of the two locations in the lab. The locations where chosen to capture most of the passing humans during working hours. Figure 4.12 shows the

two locations chosen for the experiment. The arrow locations are the locations the cameras, and the arrow directions are the directions of the cameras' field of view.

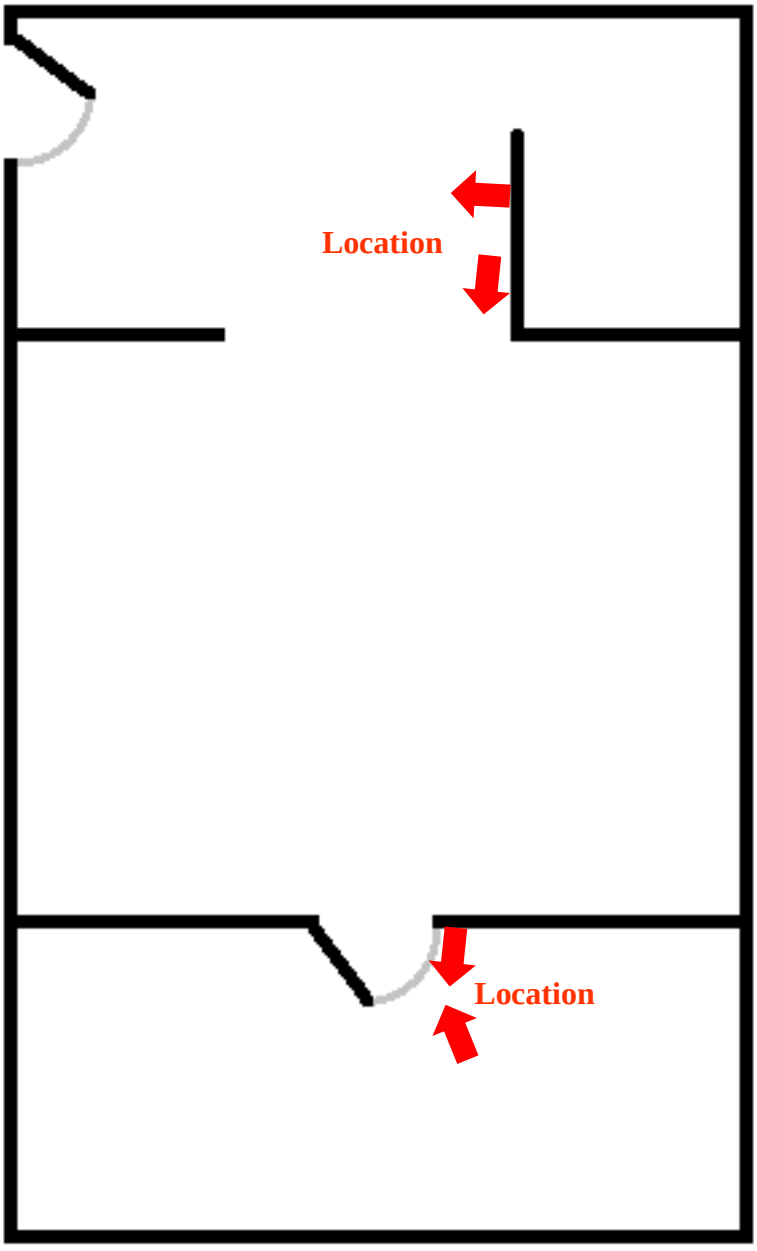


Figure 4.12: Locations where cameras where placed to capture faces.

When the program senses a motion in the camera view, it will check for faces in the motion area, and if a face is found, then it is checked against the following factors:

- **Sharpness:** Faces with a sharpness of 70% or more will be discarded. It was found during testing that in most cases, face images have a sharpness index of less than 60% while images having a sharpness index of over 70% were found to be mostly of furniture or other items. It was found that only in rare cases that a face sharpness index would be between 60% and 70%, such as, for example, when a subject was STANDING STILL, facing the camera at a close distance, well-lit and wearing thick glasses.
- **Skin color content:** Faces that contain less than 50% of skin color are considered false positives. The skin color threshold was chosen experimentally with face images of different subject of different skin color as well as false-positive facial images. The best threshold values were then recorded and used to filter out images with low skin-color content.
- **Presence of Eyes:** The last check is to try and detect whether a pair of eyes do exist in the face image or not. This last filter ensures that face images which passed the previous two filters for any reason are checked one last time before being discarded or allowed to pass to the face recognition phase. Face images that may pass the previous two filters may be those of unusable quality which are faces that are not good enough to be used for recognition purposes.

4.2.3.3 Setting

For this experiment, an A1-Pro webcam was used. This webcam has a resolution of 640x480 and a frame rate of 30 fps. The cameras were placed to face the walking direction of subjects.

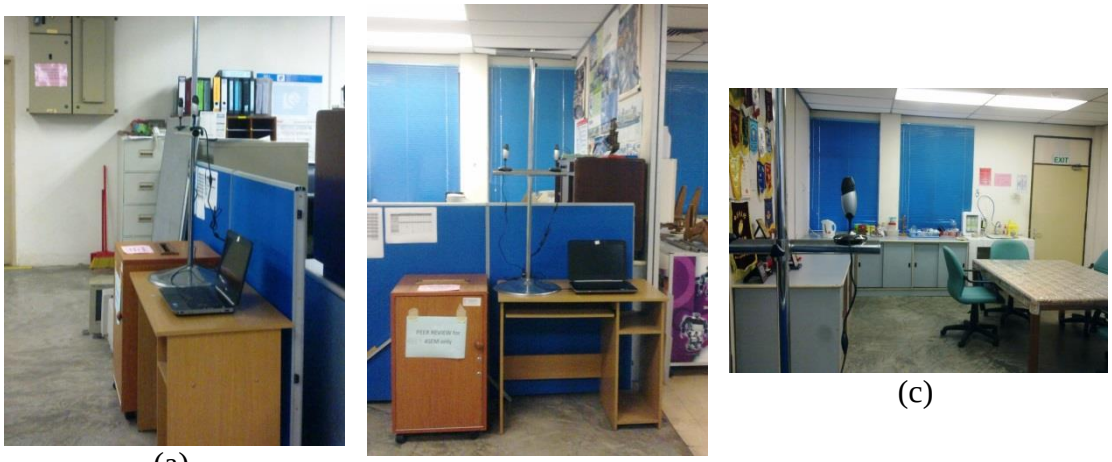
In the first location, the first camera was placed with its back towards the door, so that the face of any subject leaving the room would be captured as he/she approached from distances of 0.5 to 6 meters. The camera would also capture any subject in the room which is in its field of view. The second camera was placed

facing the door, so that when a subject enters, the camera captures an image of his/her face at a distance of one meter or less. In the second location, the cameras were located to capture subjects walking towards camera 1 or past camera 2. Figures 4.13 and 4.14 show the camera setting for both locations and the camera's view in these two locations.

(a) (b) (c)

Figure 4.13: The experiment setup at location 1

a) Setting, b) View of camera 'A', c) View of camera 'B'



(a) (b) (c)

Figure 4.14: The experiment setup at location 2

a) The setting from the right, b) The setting from the front, c) The view of one of the cameras

Both locations were well-lit with ceiling florescent lights. However this also resulted in the faces being better lit in the spot directly under the florescent light, while less so in other areas.

4.2.3.4 Results

Once the experiment was concluded, the results were gathered and tabulated as shown below in Table 4.4:

Table 4.4: The results from the face detection experiment

	Location 1	**Location 2**	**Totals**	**%**
Skin Pass	1641	840	2481	56%
Eye Pass	184	315	499	11%
Sharpness Reject	39	3	42	1%
Blocked	204	1210	1414	32%
Total	2068	2368	4436	100%

Once the results were gathered, the number of correctly and incorrectly blocked face images was calculated. The results from this calculation are shown in Table 4.5 below:

Table 4.5: The results from calculating the correct and incorrect detections

		Location1	**Location 2**	**Totals**	**%**
Correct Skin Pass		1632	797	2429	98%
Incorrect Skin Pass		9	43	52	2%
Correct Eye Pass		184	315	499	100%
Incorrect Eye Pass		0	0	0	0%
Correct Sharpness Reject		39	2	41	98%
Incorrect Sharpness Reject		0	1	1	2%
Blocked Images	**Correctly Blocked**	119	897	1016	72%
	Incorrectly Blocked	85	313	398	28%
Total Correct Detected		1974	2011	3985	**90%**
Total Incorrectly Detected		94	357	451	**10%**

4.2.3.5 Discussion and Analysis

By examining the results in Table 4.4, we see that the total number of blocked images is 32%; almost one third. This either means that the system is strict, allowing only images that are surely face images, or that the initial phase is capturing non-phase images and consequently, the filtration process is blocking most if not all of them.

However, by examining Table 4.5, it becomes evident that over 1000 of those images (roughly 25%) are in fact non-faces, which leads us to believe that the method used for the initial detection of faces is not highly accurate, or at least, is not well-suited to face detection of an uncontrolled environment video stream.

Also, by examining table 4.5, we observe that although using three different filters to block false face images, around 10% (451 images out 4436) managed to go through. This indicates that the conditions surrounding the capture of these faces is rough; i.e. contains a lot of "noise" that affects the operation of the filters and results in allowing non-faces to pass.

However, when examining the operation of each filter individually, we find that the filters work to a very good standard. The skin-color detector filter as well as the sharpness filter both have an efficiency of 98%, while the eye-detection filter has a remarkable efficiency of 100%. However, this is the passing efficiency rather than the blocking efficiency. This means that 98% or 100% of what passes through them is correct.

To conclude, an efficiency of 91% for a video stream of an uncontrolled environment with VGA video quality where indoor distances are anywhere from one to six meters seems as a good achievement when compared with what other research and commercial systems can achieve considering that this system is operating on a live camera stream in an uncontrolled environment. Table 4.6 presents a concise comparison with some of the previous research:

Table 4.6: Face detection efficiency comparison

Title	Source	Image Size	Environment	Method Used	Detection Rate
A multimodal face detection system for elderly companion robot*	Images from live camera[1]	640x480**	Uncontrolled	Integral Image + Adaboost + Cascade Classifiers	84.30% [A]
A Robust Face Detection for Human Interactive Mobile Robot	Still images	200x150	Controlled	Skin-Color-Content + Correlated based matching	89.15% [B]
A robust skin color based face detection algorithm	Still images[2]	640x480**	Controlled	Combinational Skin-Color-Content	95.18%
Face Detection in Low-resolution Color Images	Still images[3]	6x6 and up to 24x24	Controlled	12- bit Modified Census Transform	89.75% [C]
Current Method	Live video feed from camera	1920x1080	Uncontrolled	Cascade Classifiers + Filters (Skin Color, Sharpness, Eye Presence)	91%

The table headings refer to the following:

Source : Live Camera, Pre-recorded Video, Still Images or other
Environment : Controlled / Uncontrolled
Method Used : AdaBoost, Skin-Color or others.
Detection Rate : Average (mean) rate of face detection achieved.

*: Focuses on face tracking

**: Based on specifications obtained from the internet

[1] 10Moons USB Camera is used. Specifications can be found at: http://www.dx.com/p/10moons-d804rc-2-0mp-usb-2-0-web-camera-webcam-w-built-in-microphone-black-blue-145cm-cable-173402#.VNrqtnvjWK8

[2] The face images in the used IIKT database are 140x100 pixels. More details can be found at: https://books.google.com.my/books?isbn=3642133649

[3] Georgia Tech color frontal face database

A: This is the percentage obtained without using environmental perception (microphone, ultrasonic and infrared sensors)
B: Slow operation; face detection requires one second for an image size of 200x150; therefore it's impractical for real-time operation.
C: This is the average of the (highest detection rate + lowest detection rate) / 2.

Other papers have been reviewed, but they contain insufficient information to be included in the comparison table:

- A Multiple Face Detection and Tracking System Based on TLD.
- Face Detection and Tracking for Human Robot Interaction through Service Robot.
- Face Detection and Tracking using OpenCV
- Face detection in color images.

4.3 Face Recognition Related Experiment

4.3.1 General Objectives

The general objectives of the face recognition experiments is to examine the effects of different parameters on the performance of the face recognizers hence finding the best possible combination of values to use for face recognition.

4.3.2 Image Size vs Performance Experiment

4.3.2.1 Objective

This experiment has three objectives:

1. To compare the training speed of the different face recognizers when using different sizes of training images.
2. To compare the recognition speed of the different face recognizers when using different sizes of training images.
3. To compare the recognition effectiveness of the different face recognizers when using different sizes of training images.

The results from this experiment will be a used as a guideline to determine the image size that guarantees the best performance of this system during operation.

4.3.2.2 Description

Six sets of varying sizes of the same training images were prepared. The experiment used a computer program to read the training images from the hard disk and store them in a stack, then a timer (within the program) is started and a face recognizer is allowed to use the images in the stack for its training. When the training operation has completed, the timer is stopped and the elapsed time is measured and recorded.

Then for the recognition phase, a timer is started prior to the recognition event and stopped immediately after it. The elapsed time was calculated and stored in a variable. The variable used accumulated all the recognition times for the all the test images. When the recognition phase is over, the accumulative recognition time for all the test images was saved.

The same operation was repeated for all the three face recognizers. The experiment was then repeated for the same images, but of different dimensions. The experiment started with a training image set of size 50x60 pixels, and progressively used image sets of a larger size until it reached 300x360 pixels.

The same was done for the test images. Six sets were used starting with a size of 50x60 pixels and gradually increasing up to 300x360. The number of test and training images used was the same in all iterations.

4.3.2.3 Setting

The experiment was carried out on a laptop with the following specifications:

- Processor: i5 second generation, running at 2.3 GHz.
- Memory: 4 GB of RAM
- OS: Windows 7, 64 bit

The number of training images used was 219 facial images of 18 different subjects with different posture, skin colors and facial expressions. The number of test images used was 35, some relating to the people in the training set and others who are considered as strangers.

Since the aim is not to measure the recognition capability of the recognizers themselves, but rather to compare their relative capability and to give a sense of practicality to the experiment, the number of training images was not uniform for all the subjects. This is done to relate to practical situations in which a database may contain a large number of training images for some subjects, but less training images

for other subjects for any reason. Table 4.7 shows the number of training images for every subject:

Table 4.7: The number of images per subject

No.	Name	No. of Images
1	Luqman	3
2	Amirah	4
3	Marwan	5
4	Amri	8
6	Azrai	10
8	Vivi	10
10	Saifuddin	11
11	Ibraheem	13
12	Abdul-Rahman	15
13	Mahmood	15
14	Muhaimin	19
15	Go	21
16	Alaa	22
17	Habib	27
18	Mohammad	38

4.3.2.4 Results

During the experiment, the results were saved in a text file. The results were later tabulated and Table 4.8 below shows the results from the experiment.

Table 4.8: The results from the experiment

Image Size	FTT	ETT	LTT	FRT	ERT	LRT	FCR	ECR	LCR
50x60	4.798	4.229	3.451	0.023	0.168	1.751	19	13	15
100x120	11.828	15.365	14.919	0.078	1.198	4.178	22	15	19
150x180	23.207	31.088	33.010	0.132	2.670	7.425	22	13	17
200x240	37.699	54.025	59.537	0.216	4.989	11.663	21	14	14
250x300	57.335	83.760	93.398	0.286	7.528	17.060	21	15	12
300x360	78.193	115.522	131.602	0.387	10.777	23.252	20	15	11

Following is the legend for the abbreviations mentioned in the table above:

FTT : Fisher Training Time (milliseconds)
ETT : Eigen Training Time (milliseconds)
LTT : LBPH Training Time (milliseconds)
FRT : Fisher Recognition Time (milliseconds)
ERT : Eigen Recognition Time (milliseconds)
LRT : LBPH Recognition Time (milliseconds)
FCR : Fisher Correct Recognition (no. of images)
ECR : Eigen Correct Recognition (no. of images)
LCR : LBPH Correct Recognition (no. of images)

Table 4.9: The results per image, and recognition percentage

Image Size	FTTPI	ETTPI	LTTPI	FRTPI	ERTPI	LRTPI	ATTfa3FR	FCRP	ECRP	LCRP
50x60	21.909	19.312	15.760	0.105	0.766	7.995	8.866	54%	37%	43%
100x120	54.009	70.161	68.123	0.355	5.469	19.076	24.900	63%	43%	54%
150x180	105.967	141.953	150.729	0.604	12.192	33.903	46.699	63%	37%	49%
200x240	172.143	246.688	271.858	0.986	22.781	53.254	77.021	60%	40%	40%
250x300	261.805	382.466	426.476	1.306	34.373	77.900	113.578	60%	43%	34%
300x360	357.044	527.496	600.921	1.767	49.209	106.175	157.151	57%	43%	31%

Following is the legend for the abbreviations mentioned in Table 4.9:

FTTPI : Fisher Training Time Per Image (milliseconds)
ETTPI : Eigen Training Time Per Image (milliseconds)
LTTPI : LBPH Training Time Per Image (milliseconds)
FRTPI : Fisher Recognition Time Per Image (milliseconds)
ERTPI : Eigen Recognition Time Per Image (milliseconds)
LRTPI : LBPH Recognition Time Per Image (milliseconds)
ATTfa3FR : Accumulative Training Time for all 3 Face Recognizers (milliseconds)
FCRP : Fisher Correct Recognition Percentage
ECRP : Eigen Correct Recognition Percentage
LCRP : LBPH Correct Recognition Percentage

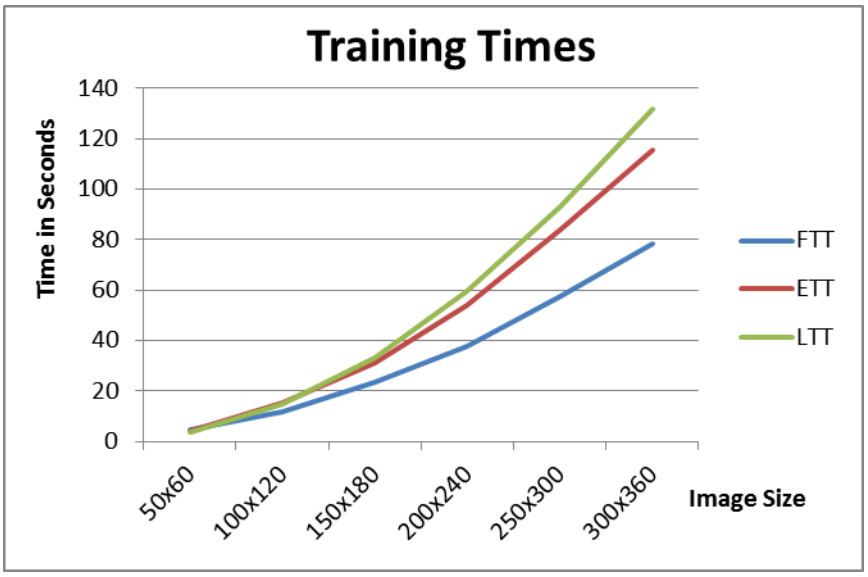


Figure 4.15: Individual training times for the three face recognizers

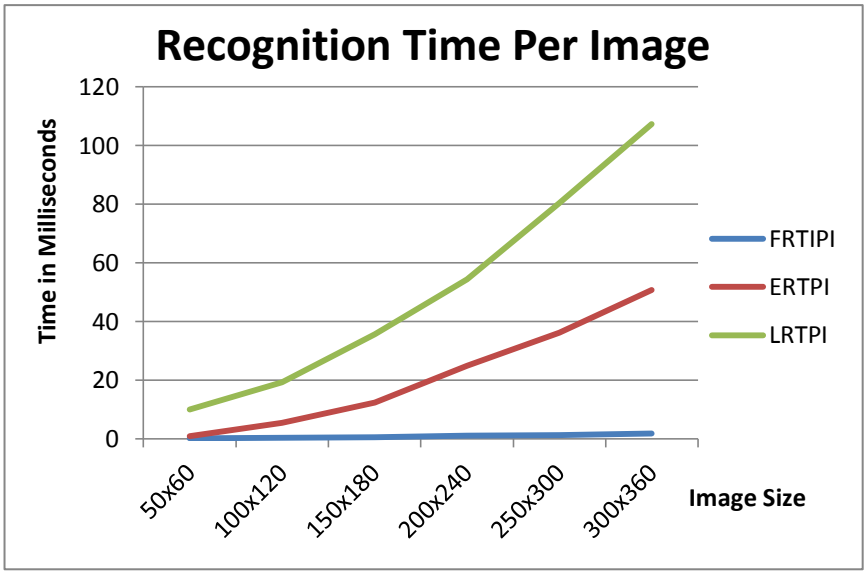


Figure 4.16: Individual recognition times per image for the three face recognizers

4.3.2.5 Discussion and Analysis

By examining the results in Table 4.8, we observe the following:

1. The training times are directly related to the size of the test images.
2. The recognition times are also directly correlated to the size of the test images.
3. The image size of 100x120 seems to be the one that results in the highest recognition rates across all the tested face recognizers.
4. Increasing the face image size does not ensure better recognition rate as can be seen from the results. Using a moderately sized image size may yield better results.

Examining Table 4.9 shows that the Fisher Faces face recognizer achieves the highest recognition rate in all iterations of the experiment. This, coupled with the low recognition time per image, makes the Fisher Faces face recognizer the best out of the three. The Eigen Faces shows an almost steady recognition rate across all face sizes, with a slight decrease at image sizes of 150x180 and a slight increase in smaller and larger images. Also, as observed in Table 4.8, the image size of 100x120 achieves the best recognition results across the board, while requiring the minimum (excluding the 50x60 size) time for training and recognition.

By examining Figure 4.15, we observe that the training time for all the face recognizers is quite low at the beginning for small sized images such as 50x60 pixels, however, as the image size increases; the recognition times increases more rapidly for the Eigen Faces, and even more so for the LBPH face recognizer. This means that the rate-of-increment for the required training time vs. the image size is the lowest for small images and highest for the largest image set to be used. Also the rate-of-increment for the required training time vs. the image size is the lowest for Fisher Face recognizer and the highest for the LBPH face recognizer.

By examining Figure 4.16, we observe the following:

1. The recognition time for all the Fisher and Eigen Faces face recognizers is almost the same at the beginning for images sized 50x60 pixels, however, as the image size increases; the recognition times increases rapidly for the Eigen Faces as well as for the LBPH face recognizers. Since all three face recognizers will be used in this project, Figure 4.16 suggests that using a small image size such as 50x60 or at most 100x120 will ensure a low accumulative recognition time, while using large images will increase the accumulative recognition time to over 150 milliseconds rendering the system less capable in responding to a volatile environment.
2. The recognition time for the Fisher face recognizer increases very little with the increment in image size, while the recognition time for the LBPH face recognizer increases by around ten folds and is almost 90x the recognition time of the Fisher Faces face recognizer. This strongly rules out the use of LBPH in systems where a large image size is to be used in face recognition operations, and advices against its use in dynamic face recognition systems unless necessary.

From the above figures and tables we conclude:

1. Using an image size of 100x120 yields the best results in training time, recognition time, and recognition rate.
2. If the system seems to be slow in responding to a more dynamic environment, then it may be possible to drop the Eigen Faces as well as the LBPH face recognizers and only keep the Fisher Faces for face recognition operations.

3. In another scenario, it could be possible to use the Fisher face recognizer for initial recognition, and only use the Eigen Faces or the LBPH face recognizers, if the confidence returned by the Fisher Faces face recognizer is low.

4.4 Obstacle Avoidance Related Experiments

4.4.1 General Objectives

The objective of this experiment was to test the effectiveness of the obstacle avoidance capability feature of the robot and to adjust the sensor angles if necessary. Two experiments were carried out and the specific objective of each experiment is mentioned thereof.

4.4.2 Minimum and Maximum Detection Distance Experiment

4.4.2.1 Objective

The objective of this experiment is to test whether the sensors can detect different objects or not, and if so at what distance. This is very important with ultrasonic sensors as they have a limited range and their signal can deflect in different directions depending on the geometry of the sensed object. This is the reason why ultrasonic sensors fail to detect some objects in certain cases.

4.4.2.2 Description

This experiment was done in the lab while the sensors was stationary and the different objects were advanced and/or retreated to measure the reading of the sensors in different distances. Figure 4.17 shows the laptop and some of the objects used in the experiment.

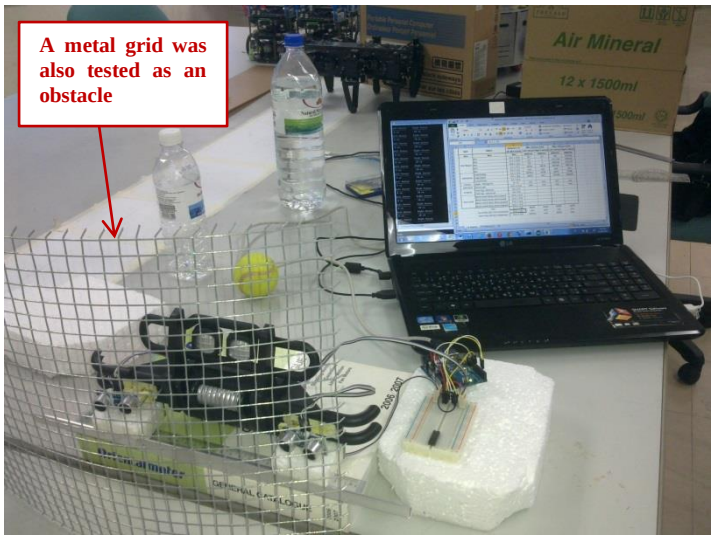


Figure 4.17: The laptop, circuit and some of the objects used in the experiment

In sequence, the different objects were placed at varying distances and the sensor readings were recorded for the minimum and maximum detection distances.

4.4.2.3 Setting

The experiment was done in the lab on a table. This facilitated conducting the experiment. Other settings are as follows:

- Sensor range is reported by the supplier (Cytron) to be 2 – 400 cm.
  Other specifications can be obtained from Cytron's website [123]
- Wall facing the sensor was 356 cm away (reported by the sensor as 3099 (that means infinity according to the used sensor's capability of 3 meters))
- Sensor angle = 7.8 degrees (This is due to the difficulty in obtaining an angle of 7.125 degrees which was found by mathematical calculation in section 3.6.4)
- Sensor height from the ground (table top) = 8.8 cm. This was similar to the actual height of the sensor when it is on the robot on the floor.
- Distance between the two sensors = 20 cm. This is corresponding to the value in section 3.6.4 and matches the sensor placement on the robot.
- Sensors were horizontally placed.

Figure 4.18 shows different images taken before and during the experiment that show the experiment setting.

(a) (b) (c)

Figure 4.18: The experiment setup

4.4.2.4 Results

The results obtained from the experiment show the readings from the left and right sensors for each of the objects. Table 4.10 lists the values for the minimum and maximum detection distance for the objects used in the experiment.

The measured distance columns contain two numbers each: the detected distance (by the left of right sensor) and the actual measured distance using a measuring tape. Although the sensor's vendor specifies 2 cm as the minimum detection distance, the sensor was practically capable of reporting distances as short as 1 cm during detecting the distance to a carton box, as mentioned in Table 4.10.

Table 4.10: Results from the experiment of the ultrasonic sensors

Type	Object	Diameter x H OR W x H (cm)	Min. Distance (cm)		Max. Distance (cm)	
			Left / Actual	Right / Actual	Left / Actual	Right / Actual
Non	**Non**	**Non**	**3099/356**	**3096/356**	**3099/356**	**3096/356**
Flat Objects	Box	13.5 x 31.5	9/7	9/7	263/263	134/132
	Box	27.7 x 37.0	1/1	1/1	153 / 154	220/218
	Box	39.0 x 31.5	1/1	1/1	220/218	232/234
	Box	1.5 x 12.7	72/70	24/19	102/101	120/118
	Box	12.7 x 1.5	none	none	none	none
Cylindrical	Small Bottle	6.6 x 23.0	24/19.5	32/28	38/35	71/68
	Large Bottle	8.8 x 30.6	20/15	13/7	101/99	104/102
Corners	Angle = 90 Deg.	H = 37	8/0	6/0	47/20	22/17
Spherical	Tennis ball	6.25 x 6.25	none	none	none	none
Irregular	Chair leg - wheel in	4.0 x 9.5	29/4	33/27	97/90.2	128/97.3
	Chair leg - wheel out	4.0 x 9.5	45/0	23/0	90/70	43/38
Non-Solid	Mesh metal sheet, bent inward	37.0 x 31.0	8/9	10/9	88/81	119/119
	Mesh metal sheet, bent outward	37.0 x 31.0	7/3.3	6/3.3	69/64	70/64
	Pole	25.5 x 132	40/36	23/18	68/65	53/66
	Two Poles (36.5 cm in between)	25.5 x 132	non	non	non	non
	Two Poles (24 cm in between)	25.5 x 132	22/19.5	non	45/42	non

4.4.2.5 Discussion and Analysis

The objects used in the experiment are examples of objects that the robot may come across laying on the floor; flat objects resembling boxes, doors or walls, cylindrical objects resembling plastic water bottles or soda cans as well as many other types of objects of different shapes and sizes.

By examining the obtained results we find the following:

- Flat Objects: The sensors have no problem detecting a flat surface of any size provided it's on the same or higher than level than the sensors, which are at 8.8 cm above ground level. The sensors did not detect the small box which is

only 1.5 cm high, but then again, this kind of obstacle may not obstruct the operation of the robot.

- Cylindrical: The sensors successfully detected the small and large water bottles as well as the soda can at a reasonable distance.
- Spherical: The tennis ball was not detected, but this is not a problem because objects such as a tennis ball, ping pong, or even a large spherical object such as a football can safely collide with the robot and roll on.
- Irregular: Protruding chair legs were successfully detected by the sensors in several orientations.
- Non-Solid: in this category, two distinct items were tested, one being the metal mesh and the other being two steel poles. The sensors successfully detected the metal mesh in both settings; inward bend and outward bent. The poles were used to determine whether the robot would correctly sense the distance in between and pass through or whether it will retract. The results show that when the distance between the poles is large enough for the robot to pass through (36.5 cm, while the robot is only 31 cm wide), then they are not detected by the sensors, but when they are close to each other that they block the passage of the robot (24 cm), then they are detected by the sensors.

An important factor is the maximum detection distance. This is because if the robot can detect the obstacle at a far enough distance, then it can maneuver smoothly without problems. If the distance was too close, then this could lead to a possible collision if the robot failed to stop or change its course of motion.

4.5 Navigation Experiments

4.5.1 Practical Navigation Experiment

During the robot's operation, it may detect subjects that are unrecognized and therefore need to be authenticated to confirm whether they are permitted to be present in the robot's environment or not. For this reason this experiment was conducted.

This experiment is important to test the potential of the system in approaching and/or chasing subjects that are not authorized within the robot's environment.

4.5.1.1 Objective

The objective of this experiment is to test the operation and accuracy of the tracking and navigation system of the robot in different situations. This is important as it will highlight potential issues that may need attention, especially in dynamic and volatile environments.

4.5.1.2 Description

The experiment starts off by allowing a subject to appear within the robot's environment. The robot is programmed to navigate to this person as would normally happen when an unrecognized and unverified subject is encountered by the robot.

First, the robot's motion and face detection cameras would locate the subject's initial location relative to the robot. The location information is passed onto the tracking camera which tracks the motion and change in subject location. Next the most recent location is used to determine the best approach path. Finally the robot starts to approach the subject based on the latest location information.

If an obstacle is encountered the robot will avoid it by adjusting its path. The tracking camera adjusts its direction to keep the subject within sight, once the obstacle is avoided; the robot adjusts its path to continue to approach the subject.

The subject is allowed to move about whether to approach the robot or to try to keep away from it, or even escape through the lab's door. This is done to emulate real-life situations in which a subject may respond by cooperating or holding back from authenticating him/herself. Figure 4.19 shows the robot while navigating a narrow corridor to approach a subject during the experiment.

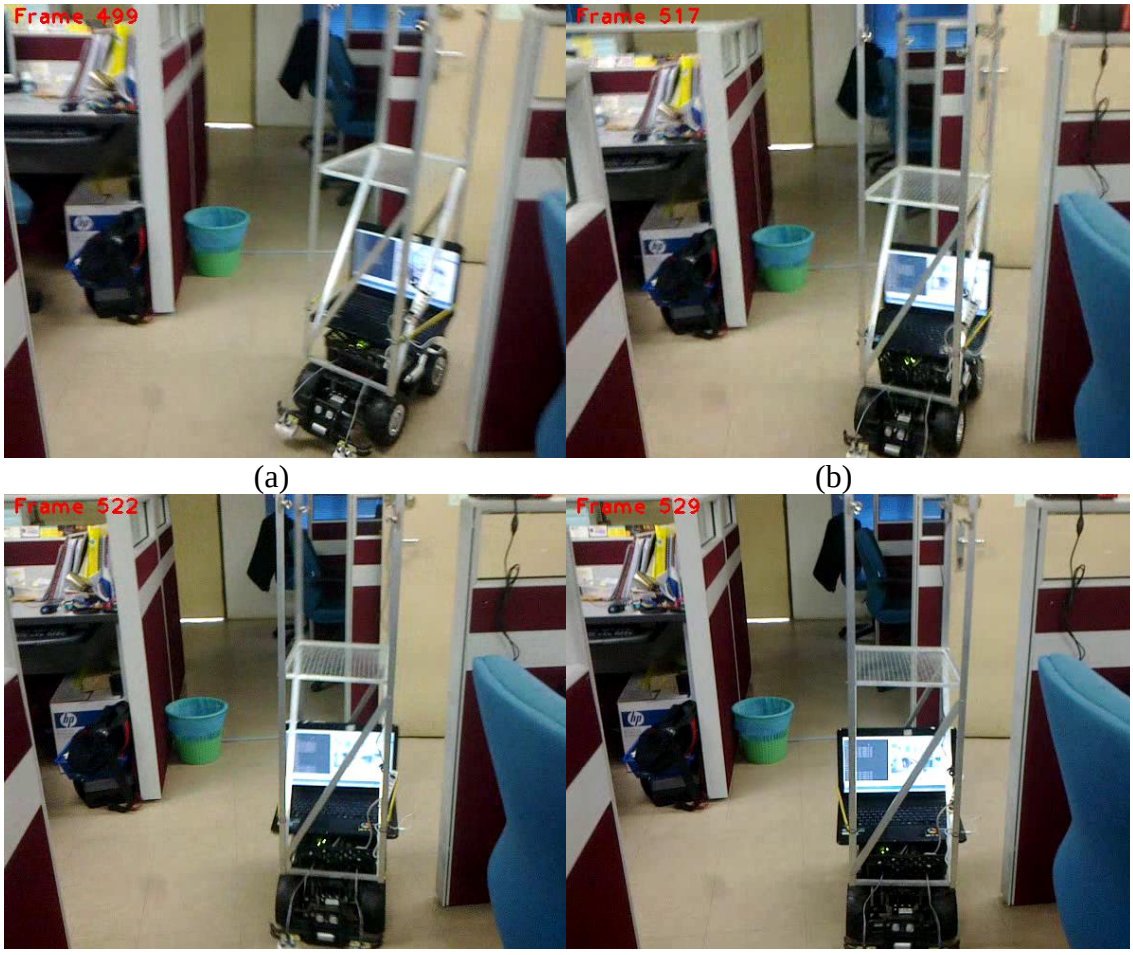


Figure 4.19: The robot navigating and approaching a person during the experiment

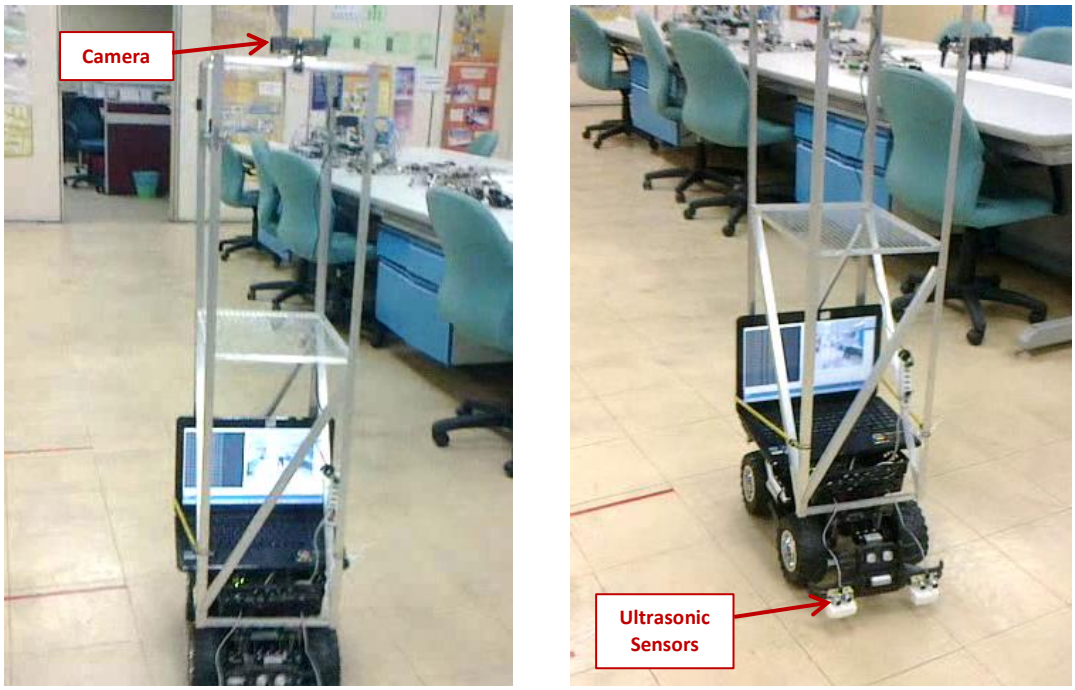
4.5.1.3 Setting

The experiment was conducted in the P08 L02 research lab of the Faculty of Electrical Engineering in UTM. The lab contains tables, chairs, as well as many other different objects. Figure 4.20 shows the experiment environment setting.

Figure 4.20: The lab environment in which the experiment was conducted

A wide angle camera of 120 degrees and a 640x480 resolution was used to locate and track the subject. The camera is positioned at a height of approximately 1.5 meters. The camera height can be adjusted to accommodate different requirements and/or work environments. Positioning the camera at this height allows the detection of short as well as tall humans. Figure 4.21 shows the robot used in the experiment.

The robot was equipped with two ultrasonic sensors tilted inwards at approximately 7 degrees to facilitate detection of different types of obstacles. Kindly refer to the obstacle avoidance section for details regarding the ultrasonic sensors used.



(a) The camera at the top (b) The equipment at the base

Figure 4.21: The robot used in the experiment

During the experiment, the subjects were allowed to move about (be dynamic) to keep the robot from reaching them. The subjects were also allowed to move the obstacles to test the robots maneuverability and change of path capability.

4.5.1.4 Results

The experiment was conducted in lab P08-02 in building P08 of the electrical engineering department of UTM. Several experiments were conducted to test the robot's ability to detect subjects and approach them while avoiding obstacles. Table 4.11 shows the results that were gathered:

Table 4.11: The results from the navigation experiment

Action	Completed Successfully	Notes
Navigating towards a static subject	Yes	The robot turns around to face the subject and then continues to move towards him/her.
Navigating towards a dynamic subject	Yes	The robot detects the subject initial position, and turns around towards him/her. The robot then continues to move towards the subject. When the subject changes his/her location during the robot's approach, the robot changes its course accordingly.
Navigating towards a static subject with an obstacle on the way	Yes	The robot starts to move towards the subject, when the robot encounters the obstacle; it avoids it by changing its direction. When the robot has cleared from the obstacle, it turns around towards the subject to continue its approach.
Navigating towards a dynamic subject with an obstacle on the way	Yes	The robot starts by detecting the subject's initial position and starts moving towards him/her. When the robot encounters the obstacle on the way, it avoids it by changing its movement direction. Since the robot is equipped with a wide angle camera, it can locate the position of the subject after clearing the obstacle and turns around towards the subject to continue its approach.

4.5.1.5 Discussion and Analysis

The results gathered from the experiment give several indications:

1. The robot is capable of detecting a subject from other object in the background.
2. The robot is able to locate the subject with reference to the robot correctly.

3. The robot is able to navigate around obstacles while remaining on the same task of approaching the subject.
4. Dynamic subject relocation does not affect the robot's capability of tracking and continuous approach of the subject.

During the experiment, the following issues were also observed:

- Vibrations resulting from the robot's motion limits or prevents the robots from detect faces in the video frame. Operation effectiveness is inversely proportional to robot speed.
- The robot had to move somewhat slowly. This was due to increased instability as the speed increased. Robot stability is inversely proportional to robot speed.
- The faster the robot moves, the more complicated it becomes to coordinate the programs in the laptop and Arduino (microcontroller). Component operation coordination is inversely proportional to robot speed.
- The robot can steer smoothly towards the subject according to his/her position with reference to the robot's pose.

Summary

Many experiments were carried out to evaluate the different aspects relating to the security robot. Each of the robot's abilities were tested to find out the effects of different parameters on their operation as well as their effectiveness in different situations.

The camera's field of view experiment helped to discover the most practical camera to be used in the project which is capable of covering all 360 degrees with the fewest number of cameras. The motion detection experiments helped in finding the most suitable threshold to use to achieve the best balance between effectiveness and stability in different lighting conditions.

The face size experiment helped in finding an equation to determine the approximate distance of the subject from his/her face size as well as determining the furthest distance for obtaining a practically usable face image using a Full-HD camera rather in comparison to a camera with lower resolution such as normal HD or VGA. The face detection experiment helped in finding the parameters that achieved maximum operation speed while allowing a high accuracy of correct face detection.

The navigation experiment helped in coordinating the robot's direction of motion with the angle of the camera that is tracking the subject's location. This was done by applying proportionate voltage levels to the dc motors driving the wheel relative to the location of the subject and the presence of obstacles.

The obstacle distance detection experiment helped in determining the effectiveness of the sensors as well the used tilt angle, while the navigation experiment helped shed light on the challenges associated with detecting a human face on a moving robot while avoiding obstacles and calculating the required motion to approach that person.

The face recognition experiment helped in finding the operational differences between the different face recognizers as well as determine the best face image size to be used to achieve best performance in terms of speed and accuracy.

The mentioned experiment were not the only conducted experiments. many other minor experiment were also conducted during the project to test the operation of the equipment, synchronizing them, coordinating their operation and determine the best setting.

CHAPTER 5

CONCLUSION

5.1 Research Summary

In this research an attempt was made to design and build an autonomous security robot that conducts its operation based on face recognition and password authentication. The main robot operation involves many sub-tasks:

- Motion detection.
- Face detection.
- Face recognition.
- Navigation.
- Obstacle avoidance.
- Authentication.

Each of these processes is based on previously researched work conducted by fellow researchers around the world. Here these processes were tested and implemented based on a practical evaluation of the parameters involved. This practical evaluation was done in order to fine tune the applied parameters and refine and module's operation and improve the outcome of the methods. In some cases new approaches or novel methods were used to optimize the operation of these modules to increase the efficiency of the robot operation.

The research objectives mentioned in section 1.2 have been met as follows:

- The robot has been equipped with 360 degree vision using three wide angle cameras.
- The robot has been equipped with a fourth camera featuring Full-HD video recording and image capturing capability. This camera was used to capture images of detected subject.
- The robot successfully tracked and approached subjects as mentioned in section 4.5.1.4 and shown in videos provided in the accompanying disk.
- Biometric subject verification was successfully performed using face-recognition as mentioned in section 4.3.2.5
- Subject authentication was also successfully carried out as shown in Figure 3.33.

5.2 Contributions

The contributions of this work are divided into the following areas:

- Using three wide angle cameras to simultaneously conduct motion detection of the area surrounding the robot resulting in the following benefits eliminating the need for catadioptric (fisheye) cameras.

- The use of a Full-HD wide angle camera for face detection using the Viola Jones method with skin-content verification. Face detection systems normally use normal cameras in their operation; however, since the robot used in this project conducts surveillance operations, the same wide-angle cameras used for surveillance have been employed for face detection purposes.

- Simultaneous localization-of and navigation-towards a dynamic target based on face or body location while avoiding static and dynamic obstacles in a non-structured environment by an autonomous mobile robot.

- The use of multiple and diverse face recognition algorithms for recognizing a person's face in live video on a mobile robot. The Fisher faces method, Eigen faces method and Local Binary Pattern Histogram method were used together to recognizer a person's face and determine his/her familiarity.

- Robot design: A different design of a security robot in which two dc motors are used to drive four big wheels in a differential manner to increase stability while maintaining high speed capability. Also, the robot has variable height (150 – 160 cm) with the observation and monitoring apparatus located at the top of the robot while the main robot weight is located at the base to improve stability during operation.

5.2.1 Cost Comparison

One of the objectives of this work was to build a cost effective security robot. To better highlight the contribution of this work, a comparison has been made between the security robot presented in this work as well as commercially available security robots in cost as well as specification.

Cost Comparison: (The prices are in US$)

Cost	ASR 1.0	Knightscope's K5	Gamma2Robotics' Vigilant
Initial	1,000	-	45,000
3-Years Contract	-	164250*	66,000

***K5's cost breakdown:**

1 hour = US$6.25; 24 hours = US$150.0

1 Year = 54,750.0; 3 Year = 164,250.0

5.2.2 Efficiency Comparison

The robot presented in this work boasts better efficiency than previous robots research robots. Table 2.1 presents a general comparison of this robot in with previous autonomous security robots used in research.

As can be seen in the table, the robot presented in this research is the only one that performs all of the following inclusively:

- Better vision: 360 degrees vs. 90 degrees for most other robots.
- Face detection: Many of the other robots do not perform face detection.
- Face recognition: Many of the other robots do not perform face recognition.
- Authentication: Most of the other robots do not perform password authentication.

All the above in addition to the speed of the robot (1.9 meters/second) deduces a more efficient security operation when compared to previous autonomous security robots.

5.3 Limitations and Future Work

Although the current work has been successful in conducting security operations in the indoor environment of the robot, several possibilities for future work exist:

- Using a PTZ camera to capture the eye image of a subject and use it to extract the image of eye iris. This iris image can be used to further determine the identity of the person. This would result in a more efficient security operation and may limit or eliminate having to request a password entry by unrecognized subjects. This would greatly automate the robot's operation and limit the number of times the robot has to approach a subject to request him/her to verify his/her identity with a password entry or otherwise.

- Adding a capturing capability to enable the robot to capture intruders who fail to verify their identity and refuse to leave the site. This could be a mechanical structure attached to the robot body and activated in predetermined scenarios and situations.

REFERENCES

[1] T. Theodoridis and H. Hu, "Toward Intelligent Security Robots: A Survey," *IEEE Trans. Syst. Man, Cybern. Part C (Applications Rev.*, vol. 42, no. 6, pp. 1219–1230, Nov. 2012.

[2] P. Lina, G. Bekeyb, and K. Abneyc, "Robots in War : Issues of Risk and Ethics," in *Ethics and Robotics*, 2009, pp. 49–67.

[3] D. Gonzales, D. Criswell, and E. Heer, "Automation and robotics for the Space Exploration Initiative: results from Project Outreach," 1991.

[4] "iRobot Robots for Defense & Security," 2014. [Online]. Available: http://www.irobot.com/us/learn/defense.aspx.

[5] A. Birk and H. Kenn, "Roboguard, a teleoperated mobile security robot," *Control Eng. Pract.*, vol. 10, no. 11, pp. 1259–1264, 2002.

[6] M. Rasheed and I. Hussain, "Cost Effective Spy Ball Robot for Surveillance, Rescue and Exploration," *Trans. Electron. Commun.*, vol. 57, no. 1, pp. 3–8, 2012.

[7] S. Ushio, K. Okada, Y. Kido, T. Kitahara, H. Tsuji, S. Moriguchi, M. Narita, and Y. Kato, "A Home Security Service Robot System Using the Network Service Platform and Its Implementation," *2011 IEEE/IPSJ Int. Symp. Appl. Internet*, pp. 402–407, Jul. 2011.

[8] Y. Takahashi and I. Masuda, "A visual interface for security robots," in *Robot and Human Communication, 1992. Proceedings., IEEE International Workshop on*, 1992, pp. 123–128.

[9] C. Lundberg and H. I. Christensen, "Assessment of man-portable robots for law enforcement agencies," *Proc. 2007 Work. Perform. Metrics Intell. Syst. - Permis '07*, pp. 76–83, 2007.

[10] H. R. Everett, E. B. Pacis, G. Kogut, N. M. Farrington, and S. Khurana, "Towards a Warfighter's Associate: Eliminating the Operator Control Unit H.R.," pp. 267–279, Dec. 2004.

[11] C. h. Kuo, C. c. Chen, W. c. Wang, Y. c. Hung, E. c. Lin, K. m. Lee, and Y. m. Lin, "Remote Control Based Hybrid-Structure Robot Design for Home Security Applications," *2006 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, pp. 4484–4489, Oct. 2006.

[12] J. Lee, H. S. S. Oh, J. Hong, J. Kyounghwan, H. Kwon, and J. Kim, "Operating a six-legged outdoor patrol robot," *2007 Int. Conf. Control. Autom. Syst.*, pp. 1034–1039, 2007.

[13] L.-Y. Chung, “Remote Teleoperated and Autonomous Mobile Security Robot Development in Ship Environment,” *Math. Probl. Eng.*, vol. 2013, pp. 1–14, 2013.

[14] J. Ryu, B. Yoo, and T. Nishimura, “Service Robot Operated by CDMA Networks for Security Guard at Home,” in *Service Robot Applications*, Yoshihiko Takahashi, Ed. INTECH, 2008.

[15] J.-G. Ryu, S.-K. Kil, H.-M. Shim, S.-M. Lee, E.-H. Lee, and S.-H. Hong, “SG-Robot: CDMA network-operated mobile robot for security guard at home,” in *IEEE International Conference on Intelligence and Security Informatics*, 2006, pp. 633–638.

[16] Je-Goon Ryu, H.-M. Shim, S.-K. Kil, E.-H. Lee, H.-H. Choi, and S.-H. Hong, “Design and implementation of real-time security guard robot using CDMA networking,” in *ICACT*, 2006, pp. 1901–1906.

[17] J. Liu, M. Wang, and B. Feng, “iBotGuard: an Internet-based intelligent robot security system using invariant face recognition against intruder,” *Syst. Man, Cybern. Part C Appl. Rev. IEEE Trans.*, vol. 35, no. 1, pp. 97–105, 2005.

[18] A. Eydgahi, T. Olowoporoku, and P. Matin, “Design and Implementation of a Wireless Security Robot,” in *laccei.org*, 2011, pp. 1–10.

[19] B. L. Sefidgari, “Human Body Detection and Safety Care System for a Flying Robot,” in *ICAITA*, 2013, pp. 317–325.

[20] Y. Do, G. Kim, and J. Kim, “Omnidirectional vision system developed for a home service robot,” *2007 14th Int. Conf. Mechatronics Mach. Vis. Pract.*, pp. 217–222, Dec. 2007.

[21] D. Di Paola, D. Naso, A. Milella, and G. Cicirelli, “Multi-sensor surveillance of indoor environments by an autonomous mobile robot,” *Int. J. Intell. Syst. Technol. Appl.*, vol. 8, no. 1, pp. 18–35, 2010.

[22] R. Li, L. Zhao, L. Ge, L. Sun, and T. Gao, “The development of a general type of security robot,” in *Robotics and Biomimetics, 2007. ROBIO 2007. IEEE International Conference on*, 2007, pp. 47–52.

[23] T.-H. S. Li, C.-Y. Chen, H.-K. Huang, and Y.-C. Yeh, “Design and implementation of sensor fusion based behavior strategies for a surveillance and security robot team,” *2008 SICE Annu. Conf.*, vol. 2, no. d, pp. 2968–2972, Aug. 2008.

[24] J. Zhang, G. Song, G. Qiao, T. Meng, and H. Sun, “An indoor security system with a jumping robot as the surveillance terminal,” *IEEE Trans. Consum. Electron.*, vol. 57, no. 4, pp. 1774–1781, Nov. 2011.

[25] X. Wu, H. Gong, P. Chen, Z. Zhong, and Y. Xu, “Surveillance Robot Utilizing Video and Audio Information,” *J. Intell. Robot. Syst.*, vol. 55, no. 4–5, pp. 403–421, Jan. 2009.

[26] X. Wu, H. Gong, P. Chen, Z. Zhi, and Y. Xu, “Intelligent household surveillance robot,” *2008 IEEE Int. Conf. Robot. Biomimetics*, pp. 1734–1739, Feb. 2009.

[27] J. Park and K. B. Sim, “A design of mobile robot based on Network Camera and sound source localization for intelligent surveillance system,” *2008 Int. Conf. Control. Autom. Syst.*, pp. 674–678, Oct. 2008.

[28] Z. Cheng, X. Zhang, S. Yu, Y. Ou, X. Wu, and Y. Xu, “A surveillance robot with human recognition based on video and audio,” *2010 IEEE Int. Conf. Robot. Biomimetics*, pp. 1256–1261, Dec. 2010.

[29] P. Bedi, R. Singh, and T. Matharu, “Ensuring security in a closed region using robot,” *Comput. Intell. …*, 2010.

[30] M. Li, L. Sun, and Q. Huang, “GPRS Based Guard Robot Alarm System Design,” in *Internet Computing for Science and Engineering (ICICSE), 2009 Fourth International Conference on*, 2009, pp. 211–216.

[31] K. Lee and C. Seo, “Development of user-friendly intelligent home robot focused on safety and security,” in *International Conference on Control, Automation and Systems*, 2010, pp. 389–392.

[32] Z. DehuaI, X. Gang, Z. Jinming, and L. Li, “Development of a mobile platform for security robot,” in *International Conference on Automation and Logistics*, 2007, no. 200642, pp. 1262–1267.

[33] D. Vu, K. Hoganson, and D. Ph, “Student Projects : Security Robot Design,” in *The 49th Annual Southeast Regional Conference*, 2011, pp. 287–289.

[34] D. Di Paola and A. Milella, “An autonomous mobile robotic system for surveillance of indoor environments,” *Int. J. Adv. Robot. Syst.*, vol. 7, no. 1, pp. 19–26, 2010.

[35] A. Treptow, G. Cielniak, and T. Duckett, “Active people recognition using thermal and grey images on a mobile security robot,” *2005 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, pp. 2103–2108, 2005.

[36] G. Cielniak, “People tracking by mobile robots using thermal and colour vision,” Orebro University, 2007.

[37] S. W. Gordon, S. Pang, R. Nishioka, and N. Kasabov, “Vision Based Mobile Robot for Indoor Environmental Security,” in *ICONIP 2008*, 2009, pp. 962–969.

[38] A. M. Pinto, a. P. Moreira, M. V. Correia, and P. G. Costa, “A Flow-based Motion Perception Technique for an Autonomous Robot System,” *J. Intell. Robot. Syst.*, vol. 75, no. 3–4, pp. 475–492, 2014.

[39] C. H. Chen, “Automated Surveillance Systems with Multi-Camera and Robotic Platforms,” University of Tennessee - Knoxville, 2009.

[40] R. P. Smurlo, “Intelligent Security Assessment for a Mobile Robot,” San Diego, CA, USA, 1993.

[41] M. T. Wolf, C. Assad, Y. Kuwata, A. Howard, H. Aghazarian, D. Zhu, T. Lu, A. Trebi-Ollennu, and T. Huntsberger, “360-Degree Visual Detection and Target Tracking on an Autonomous Surface Vehicle,” *J. F. Robot.*, vol. 27, no. 6, pp. 819–833, Nov. 2010.

[42] P. Chakravarty and A. Zhang, “Anomaly detection and tracking for a patrolling robot,” in *Australasian Conference on Robotics and Automation*, 2007.

[43] M. tiga Zacharie, “Intelligent OkiKoSenPBX1 security patrol robot via network and map-based route planning,” *J. Comput. Sci.*, vol. 5, no. 1, pp. 79–85, 2009.

[44] R. Luo, T. Hsu, and K. Su, “The development of a multisensor based intelligent security robot: Chung Cheng# 1,” *Mechatronics, 2005. ICM’05. IEEE ...*, pp. 970–975, 2005.

[45] R. Luo, T. Hsu, T. Lin, and K. Su, “The development of intelligent home security robot,” in *International Conference on Mcchatronics*, 2005, pp. 422–427.

[46] M. Chiu, T.-S. Lan, and S. Hwang, “Development of a Real-Time Task Editable Service-Oriented Security Robot,” *J. Interdiscip. Math.*, vol. 15, no. 4–5, pp. 239–259, Aug. 2012.

[47] Y. Chen, S. Abhyankar, L. Xu, W. T. Tsai, and M. Garcia-Acosta, “Developing a Security Robot in Service-Oriented Architecture,” in *12th IEEE International Workshop on Future Trends of Distributed Computing Systems*, 2008, pp. 106–111.

[48] Y. Chen and W. Tsai, “Development of a Security Robot in Service-Oriented Architecture,” 2008.

[49] Y. Kim, H. Kim, S. Lee, and K. Lee, “Ubiquitous Home Security Robot Based on Sensor Network,” *2006 IEEE/WIC/ACM Int. Conf. Intell. Agent Technol.*, pp. 700–704, 2006.

[50] Y. Kim, H. Kim, S. Yoon, S. Lee, and K. Lee, “Home Security Robot based on Sensor Network,” in *SICE-ICASE International Joint Conference*, 2006, pp. 5977–5982.

[51] G. Song, K. Yin, Y. Zhou, and X. Cheng, "A surveillance robot with hopping capabilities for home security," *IEEE Trans. Consum. Electron.*, vol. 55, no. 4, pp. 2034–2039, Nov. 2009.

[52] G. K. Dey, R. Hossen, M. S. Noor, and K. T. Ahmmed, "Distance controlled rescue and security mobile robot," *2013 Int. Conf. Informatics, Electron. Vis.*, pp. 1–6, May 2013.

[53] R. Luo and P. Wang, "Navigation and mobile security system of intelligent security robot," in *International Conference on Industrial Technology*, 2005, pp. 260–265.

[54] C. Chang, K. Chen, H. Lin, C. Wang, and J. Jean, "Development of a patrol robot for home security with network assisted interactions," *SICE Annu. Conf. 2007*, pp. 924–928, Sep. 2007.

[55] T. L. Chien, K. L. Su, and J. H. Guo, "The Multiple Interface Security Robot – WFSR-II," in *International Workshop on Safety, Security and Rescue Robotics*, 2005, no. June, pp. 69–74.

[56] R. Luo, Y. Chou, and C. Liao, "NCCU security warrior: An intelligent security robot system," in *The 33rd Annual Conference of the IEEE Industrial Electronics Society*, 2007, vol. Nov., pp. 2960–2965.

[57] H. Lee, W. Lin, and F. Lian, "Hybrid Wireless Indoor Surveillance Robotic System," *Inf. Technol. J.*, vol. 13, no. 13, pp. 2187–2195, 2014.

[58] K. Kim, S. Bae, and K. Huh, "Intelligent surveillance and security robot systems," in *IEEE Workshop on Advanced Robotics and its Social Impacts*, 2010, pp. 70–73.

[59] H.-T. Lee, W.-C. Lin, and C.-H. Huang, "Indoor Surveillance Security Robot with a Self-Propelled Patrolling Vehicle," *J. Robot.*, vol. 2011, pp. 1–9, 2011.

[60] H. Lee, W. Lin, C. Huang, and Y. Huang, "Wireless Indoor Surveillance Robot," 2011, pp. 2164–2169.

[61] P. Prashanth, "Wi-Bot In Defence Using Ad-Hoc Communication Networks," *Int. J. Innov. Res. Dev.*, vol. 2, no. 8, pp. 361–367, 2013.

[62] W. Yu, J. Lee, H. Chae, K. Han, Y. Lee, and M. Jang, "Robot task control utilizing human-in-the-loop perception," *RO-MAN 2008 - 17th IEEE Int. Symp. Robot Hum. Interact. Commun.*, pp. 395–400, Aug. 2008.

[63] M. Seeman, M. Broxvall, and A. Saffiotti, "Virtual 360° Panorama for Remote Inspection," in *International Workshop on Safety, Security and Rescue Robotics*, 2007, no. September.

[64] A. C. Caputo, *Digital Video Surveillance and Security*. Butterworth-Heinemann, 2014.

[65] H. Kruegle, *CCTV Surveillance: Video Practices and Technology*. Butterworth-Heinemann, 2011.

[66] B. P. Lin, "Drone-Ethics Briefing : What a Leading Robot Expert Told the CIA," *The Atlantic*, no. 15, pp. 1–12, 2011.

[67] A. R. Hunt, "Use of a Frequency-Hopping Radar for Imaging and Motion Detection Through Walls," *IEEE Trans. Geosci. Remote Sens.*, vol. 47, no. 5, pp. 1402–1408, 2009.

[68] R. Rinehart and E. Garvey, "Three-dimensional storm motion detection by conventional weather radar," *Nature*, pp. 287 – 289, May-1978.

[69] R. Suzuki and S. Otake, "Monitoring daily living activities of elderly people in a nursing home using an infrared motion-detection system," *Telemed. J. e-Health*, vol. 12, no. 2, pp. 146–155, 2006.

[70] H. Everett, "Robotic security systems," *Instrumentation & Measurement Magazine, IEEE*, no. 4, pp. 30 – 34, 2003.

[71] R. Barnard, *Intrusion detection systems*. Springer Science & Business Media, 2008.

[72] S. Cheung, S. Coleri, and B. Dundar, "Traffic measurement and vehicle classification with single magnetic sensor," *Transp. Res. Rec. J. Transp. Res. Board*, vol. 1917, pp. 173–181, 2006.

[73] T. Gandhi and M. M. Trivedi, "Motion analysis for event detection and tracking with a mobile omnidirectional camera," *Multimed. Syst.*, vol. 10, no. 2, pp. 131–143, Aug. 2004.

[74] R. Stolkin, D. Rees, M. Talha, and I. Florescu, "Bayesian fusion of thermal and visible spectra camera data for region based tracking with rapid background adaptation," *2012 IEEE Int. Conf. Multisens. Fusion Integr. Intell. Syst.*, pp. 192–199, Sep. 2012.

[75] G. Bradski, "{The OpenCV Library}," *Dr. Dobb's J. Softw. Tools*, 2000.

[76] P. Kaewtrakulpong and R. Bowden, "An Improved Adaptive Background Mixture Model for Real- time Tracking with Shadow Detection 2 Background Modelling," pp. 1–5, 2001.

[77] M. Yang, D. J. Kriegman, S. Member, and N. Ahuja, "Detecting Faces in Images : A Survey," vol. 24, no. 1, pp. 34–58, 2002.

[78] C. Patil and G. Dhoot, "Face Detection Techniques-A Review," *Int. J. Curr. Eng. Technol.*, vol. 3, no. 5, pp. 1809–1813, 2013.

[79] R. Hsu, "Face detection in color images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 5, pp. 1–23, 2002.

[80] S. K. Singh, D. S. Chauhan, M. Vatsa, and R. Singh, “A Robust Skin Color Based Face Detection Algorithm,” vol. 6, no. 4, pp. 227–234, 2003.

[81] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001, vol. 1.

[82] P. Viola and M. Jones, “Robust real-time face detection,” *Int. J. Comput. Vis.*, vol. 57, no. 2, pp. 137–154, 2004.

[83] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, “Eigenfaces vs. fisherfaces: Recognition using class specific linear projection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 7, pp. 711–720, 1997.

[84] M. a. Turk and a. P. Pentland, “Face recognition using eigenfaces,” *Proceedings. 1991 IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 586–591, 1991.

[85] T. Ahonen, A. Hadid, and M. Pietikäinen, “Face recognition with local binary patterns,” in *Proc. of the European Conference on Computer Vision (ECCV)*, 2004, pp. 469–481.

[86] Q. Xiao, “Trusted User Authentication Using Biometrics,” Ottawa, Canada, 2002.

[87] A. Jesudoss and N. P. Subramaniam, “A Survey on Authentication Attacks and Countermeasures,” *Indian J. Comput. Sci. Eng.*, vol. 5, no. 2, pp. 71–77, 2014.

[88] R. Awasthi and R. Ingolikar, “A Study of Biometrics Security System,” *Int. J. Innov. Res. Dev.*, vol. 2, no. 4, pp. 737–760, 2013.

[89] B. R. Fussell, “Authentication : The Development of Biometric Access Control,” *ISSA J.*, no. July, 2005.

[90] “Face Authentication,” 2014. .

[91] “Fingerprint scanner,” 2014. .

[92] “Iris Authentication,” 2014. .

[93] K. Delac and M. Grgic, “A survey of biometric recognition methods,” in *46th International Symposium Electronics in Marine*, 2004, no. June, pp. 16–18.

[94] R. Shoja Ghiass, O. Arandjelović, A. Bendada, and X. Maldague, “Infrared face recognition: A comprehensive review of methodologies and databases,” *Pattern Recognit.*, vol. 47, no. 9, pp. 2807–2824, Sep. 2014.

[95] S. G. Kong, J. Heo, B. R. Abidi, J. Paik, and M. a. Abidi, “Recent advances in visual and infrared face recognition—a review,” *Comput. Vis. Image Underst.*, vol. 97, no. 1, pp. 103–135, Jan. 2005.

[96] G. Ramkumar and M. Manikandan, “Face Recognition - Survey,” *Int. J. Adv. Sci. Technol.*, vol. 1, no. 1, pp. 260–268, 2013.

[97] P. W. Han, C. Ji, S. M. Wang, D. X. Zhang, and D. Yang, “Design of an Authentication System Based on Face Recognition and the Second Generation ID Detection,” *Adv. Mater. Res.*, vol. 542–543, pp. 968–971, Jun. 2012.

[98] G. H. Fisher and R. L. Cox, “Recognizing human faces,” *Appl. Ergon.*, vol. 6, no. 2, pp. 104–109, 1975.

[99] G. Guo, S. Li, and K. Chan, “Face recognition by support vector machines,” in *Fourth IEEE International Conference on Automatic Face and Gesture Recognition*, 2000.

[100] B. Heisele, P. Ho, and T. Poggio, “Face recognition with support vector machines: Global versus component-based approach,” in *Eighth IEEE International Conference on Computer Vision*, 2001, no. Vol 2., pp. 688 – 694.

[101] M. Bartlett and T. t Sejnowski, “Independent componentsof face images: A representationfor face recognition,” 1997.

[102] S. H. Lin, S. Y. Kung, and L. J. Lin, “Face recognition/detection by probabilistic decision-based neural network.,” *IEEE Trans. Neural Netw.*, vol. 8, no. 1, pp. 114–32, Jan. 1997.

[103] T. Kanade, *Computer recognition of human faces*. Birkhauser Verlag Basel, 1977.

[104] F. Samaria and S. Young, “HMM-based architecture for face identification,” *Image Vis. Comput.*, vol. 12, no. 8, pp. 537–543, Oct. 1994.

[105] T. F. Cootes, G. J. Edwards, and C. J. Taylor, “Active appearance models ’98,” *Comput. Vision—ECCV’98*, pp. 484–498, Jun. 1998.

[106] T. Cootes, G. Edwards, and C. Taylor, “Active appearance models 2001,” *IEEE Trans. pattern …*, vol. 23, no. 6, pp. 681–685, 2001.

[107] V. Blanz and T. Vetter, “A morphable model for the synthesis of 3D faces,” *Proc. 26th Annu. Conf. Comput. Graph. Interact. Tech. - SIGGRAPH ’99*, pp. 187–194, 1999.

[108] and J. J. A. Penev, Penio S., “Local Feature Analysis: A General Statistical Theory for Object Representation,” *Netw. Comput. neural Syst.*, vol. 7, no. 3, pp. 477–500, 1996.

[109] a Lanitis, C. Taylor, and T. Cootes, “Automatic face identification system using flexible appearance models,” *Image Vis. Comput.*, vol. 13, no. 5, pp. 393–401, Jun. 1995.

[110] J. Huang, B. Heisele, and V. Blanz, “Component-based face recognition with 3D morphable models,” in *4th International Conference, AVBPA*, 2003, pp. 27–34.

[111] S. Angle, R. Bhagtani, and H. Chheda, “Biometrics: A further echelon of security,” in *The First UAE International Conference on Biological and Medical Physics*, 2005.

[112] D. Bryliuk and V. Starovoitov, “Access control by face recognition using neural networks,” *УДК*, vol. 681, no. 01, pp. 681–327, 2002.

[113] C. Adams, “One-Time Password,” in *Encyclopedia of Cryptography and Security*, Henk C. A. van Tilborg, Ed. Springer US, 2005, pp. 443–452.

[114] M. O. Rayes, “One-Time Password,” *Encycl. Cryptogr. Secur.*, pp. 885–887, 2011.

[115] L. Lamport, “Password authentication with insecure communication,” *Commun. ACM*, vol. 24, no. 11, 1981.

[116] L. Zhang and Y. Liang, “Motion Human Detection Based on Background Subtraction,” *2010 Second Int. Work. Educ. Technol. Comput. Sci.*, pp. 284–287, 2010.

[117] Z. Minghan, L. Dayong, and C. Qianxia, “Moving objects detection algorithm based on two consecutive frames subtraction and background subtraction,” *Comput. Autom. Meas. Control*, vol. 3, 2005.

[118] G. McComb, *Robot builder’s bonanza*. McGraw-Hill, Inc., 2003.

[119] L. Bruzzone and G. Quaglia, “Review article: locomotion systems for ground mobile robots in unstructured environments,” *Mech. Sci.*, vol. 3, no. 2, pp. 49–62, Jul. 2012.

[120] O. Jahanian and G. Karimi, “Locomotion Systems in Robotic Application,” *2006 IEEE Int. Conf. Robot. Biomimetics*, pp. 689–696, 2006.

[121] T. W. Manikas, K. Ashenayi, and R. L. Wainwright, “Genetic Algorithms for Autonomous Robot Navigation,” *IEEE Instrumentation & Measurement Magazine*, no. 6, pp. 26–31, Dec-2007.

[122] Y. J. Shen and B. C. Zhong, “Intelligent Access Control and Monitoring System,” *Appl. Mech. Mater.*, vol. 437, pp. 659–662, Oct. 2013.

[123] Cytron, “Cytron’s Website,” 2014. .

APPENDIX A

Below is the software program listing that was used on the laptop driving the robot. The program is written in C++ and uses the libraries of OpenCV 2.4.8.

```
#include "stdafx.h"         // required by Visual Studio

#include <iostream>         // for screen input & output
#include <fstream>          // for file input & output
#include <string>           // for handling strings and wide-strings ((names
and the like))
#include <Windows.h> // for organizing the different windows on the screen
#include <direct.h>         // for 'mkdir' ((Creating folders))
#include <ctime>            // for calculating time intervals
#include <sapi.h>           // for speech

#include "contrib\contrib.hpp"            // for <FaceRecgnizer> and
createFisherFaceRecognizer>
#include "highgui\highgui.hpp"            // for VideoCapture & waitKey
#include "objdetect\objdetect.hpp" // for cascade classifiers
#include "imgproc\imgproc.hpp"            // for cvtColor, resizes &
equalizeHist


#define Ffactor 46000              // A division factor which results in the
Fisher confidence being between 0 & 100 % (sometimes more than 100)
#define Efactor 240000             // A division factor which results in the
Eigen confidence being between 0 & 100 % (sometimes more than 100)
#define Lfactor 3250        // A division factor which results in the LBPH
confidence being between 0 & 100 % (sometimes more than 100)

#define RightCam 1
#define FrontCam 0
#define LeftCam 2
#define FHDcam 4

using namespace std;
using namespace cv;


/*********************************************************************************
////////////////////////  START of EXTERNAL FUNCTIONS  ////////////////////////*/

//========== INITIALIZATION ==========
// A welcome and introduction message
HWND IntializeWelcome(void);

// Reads the images and labels for face recognition for face recognition
purposes
int ReadTrainingImages(vector<Mat>&,vector<int>&);     // vector=the images
and their labes


//========== MAIN OPERATIONS ==========
```

```
// This function finds whether motion has occured, and if so, where in the
frame image
// Rect              : Area of detected motion.
// Image1     : The first image to be compared for motion detection
// Image2     : The second image to be compared for motion detection
// Threshold: The threshold to be used (15 is best)
// Extra      : Expand the are motion of motion by 'Extra' pixels in each
direction.
int LMX3(Rect Motion_Matrix[], Mat Image_1,Mat Image_2,int Threshold, int
Xtra,int Separation_Distance);


// This function aims the Full-HD camera towards the area where motion has
been detected
void AimFHDcam5_2(Mat Frame_Image, Rect Face_Region, int FoV, int FHDAngle[6],
float Increase_in_angle_Percentage);
void CenterFHDcam(void);


// This function aimes the Non-Stationary wide angle camera while approaching
the subject
int AimWAcam3(Mat Frame_Image, Rect Face_Region, int FoV, int Previous_Angle);
int CenterWAcam(void);

// This function finds the largest face inside the 'Image' using the haar
cascades (Viola Jones) method
Rect FindLrgstFace3(CascadeClassifier TheDetector, Mat Image_To_Be_Searched,
int FaceSize_To_Search_For, float Skin_Thrshld, float Sharpness_Thrshld,bool
Save_Image);

// This function finds the largest body inside the 'Image'
Rect FindLrgstBody(Mat Image, CascadeClassifier
FirstBodyCascade,CascadeClassifier SecondBodyCascade,CascadeClassifier
ThirdBodyCascade, int Sz);

// This function makes the robot approach the subject using left and right
motor power level control
void Approach7wi(int Tracking_Camera_Angle,int Power_Level_Percentage, Mat
Frame_Image,int Frame_Img_Counter);

void RobotMotors(int LeftMotor,int RightMotor);

// This function checks the frame image for corruption
bool CheckFrame(VideoCapture Camera, Mat CameraFrame, int Iterations);

// This function shows text on a graphic window
void Print(Mat The_Image_Name,int The_Line_Number,String The_Text);

// This function shows an image with two rectangles in it and waits for a
certain period. It may also resize the image before displaying it
void Show2(Mat Image2BDisplayed,string Window_Title, Rect First_Rect_Red, Rect
Second_Rect_Purple, int Wait_Period, bool Destroy);
void Show2(Mat Image2BDisplayed,string Window_Title, int x_Coord, int y_Coord,
Rect First_Rect_Red, Rect Second_Rect_Purple,
       int Wait_Period, bool Destroy);

void Show2(Mat Image2BDisplayed,string Window_Title, int x_Coord, int y_Coord,
Rect First_Rect_Red, Rect Second_Rect_Purple,
       int Wait_Period, bool Destroy, int Resiz_To_Width, int
Resiz_To_Height);

// Show (graphic window) and wait (waitKey) for some time
```

```
int SnW(int Wait_Tim_Millisec);

// This function outputs the message as speech as well as display it on the
console window
bool Say(string saying);

// This function copies a part of an array to another array (both intger)
void CopyArray(int Source_Array[], int Starting_Cell, int Last_Cell, int
Destination_Array[]);

// This function crops the face image from one or two sides
Mat FixedFaceCrop(Mat Face_Image,int Desired_Face_Width_To_Crop_To);


// This function reads the DIP switch
int ReadDIP(void);

//////////////////////////  END of EXTERNAL FUNCTIONS  ///////////////////////
/*****************************************************************************/


// General Declerations
HWND CMDhandle;                          // CmndWnd is the handle number of
the command window
Mat Status;                              // The image that is dedicated to
showing text information that's related to the program operation
string Names[100];                       // The array that will be used to
find the name of a subject from his label
string PixFolder="D:\\Faces\\100x120\\"; // The folder containing the images
and the text file of the names



//===========================================================================
//-------------------------------START OF MAIN ---------------------------

int main()
{
//############################### DECLARATIONS ################################
// Video & Camera Capture Related
       int CrCm=0;                 // The current camera which is being viewed
       int PrvCm;                  // The previous camera that was being viewed
       string Ans;
       VideoCapture Cam[5];


// Motion Detection Related
       int ImgCounter=0;           // This counter is used to give the saved
images a number so as not to overwrite one another
       int NMR=0;                  // Number of Motion Regions
       int LMR;                    // Largest Motion Region
       int LMW;                    // Largest Motion Width
       Mat WAFrame;                // The Current Frame image to detect motion
       Mat PrvFrame[3];            // The previous frame image that will be
compared with (continue reading in the next line)
       Mat SrchRgnImg;             // The image in which a face will be
searched for
       Mat DisplayedFrame;         // The frame to be displayed
       Rect Motion[50];            // The rectangle containing the motion in
the frame
       Rect LFRg(0,0,0,0);         // Largest Detected Face Region during
motion detection
```

```
	Rect LBRg(0,0,0,0);		// Largest Detected Body Region during
motion detection
	Rect LSRg(0,0,0,0);		// Largest Detected Subject Region. This
will either become LFRg or LBRg during operation depending on calculations
	String DtctdRgn;


//// Face Detection Related
// General
	int fd;				// The face detector loop counter variable
	CascadeClassifier FaceDetector[5];
	string FaceDetectorName[5];
	FaceDetectorName[0]=
"C:\\OpenCV2.4.8\\sources\\data\\haarcascades\\haarcascade_frontalface_alt.xml
";
	FaceDetectorName[1] =
"C:\\OpenCV2.4.8\\sources\\data\\haarcascades\\haarcascade_frontalface_alt2.xm
l";
	FaceDetectorName[2] =
"C:\\OpenCV2.4.8\\sources\\data\\haarcascades\\haarcascade_frontalface_alt_tre
e.xml";
	FaceDetectorName[3] =
"C:\\OpenCV2.4.8\\sources\\data\\haarcascades\\haarcascade_frontalface_default
.xml";
	FaceDetectorName[4] =
"C:\\OpenCV2.4.8\\sources\\data\\haarcascades\\haarcascade_profileface.xml";
	vector <Rect> DetectedFaces;		// A sort of an array holding
all the Detected Faces
// Wide Angle Cam Specific
	int FaceWin;		// The face-search window size
	Rect WALFRg;		// Largest face region in the Full HD camera.

// Body detection related
	Rect FoundBody;
	CascadeClassifier Bdy[3];
	string BodyCascadeName[3];
	BodyCascadeName[0] =
"C:\\OpenCV2.4.8\\sources\\data\\haarcascades\\haarcascade_upperbody.xml";
	BodyCascadeName[1] =
"C:\\OpenCV2.4.8\\sources\\data\\haarcascades\\haarcascade_mcs_upperbody.xml";
	BodyCascadeName[2] =
"C:\\OpenCV2.4.8\\sources\\data\\haarcascades\\haarcascade_fullbody.xml";
	vector <Rect> DetectedBodies;

// Eye detection related
	CascadeClassifier SmallEyeCascade;
	CascadeClassifier BigEyeCascade;
	CascadeClassifier OtherEyeCascade;
	string SmallEyeCascadeName =
"C:\\OpenCV2.4.8\\sources\\data\\haarcascades\\haarcascade_mcs_eyepair_small.x
ml";
	string BigEyeCascadeName =
"C:\\OpenCV2.4.8\\sources\\data\\haarcascades\\haarcascade_mcs_eyepair_big.xml
";
	string OtherEyeCascadeName =
"C:\\OpenCV2.4.8\\sources\\data\\haarcascades\\haarcascade_eye_tree_eyeglasses
.xml";
	vector <Rect> DetectedEyes;


//// Face Tracking Related
	bool AngleError=false;
```

```
	int FaceNotFound;		// 0.2 This variable counts how many times a
face was not found when looking for it.
	int FHDFcCntr;			// The counter which will count the
number of captured faces during tracking
	int FHDAngle[6]={90,90,0,0,0,0};// The horizontal and vertical angles
of the full HD camera to start with.
	int PrvFHDAng[2];		// The previous horizontal and vertical
angles of the full HD camera to start with.
	Mat FHDframe;
	Mat FHDFSimg;			// The face tracking/search area image that
is extracted from the FHD frame.
	Mat FHDFace[15];		// Faces that will be used for recognition.
Up to 5 faces can be captured during tracking
//	Rect FHDFSRgn(400,200,1120,680);  // The face search/tracking area
inside the FHD frame
//	Rect FHDFSRgn(350,150,1220,780);  // The face search/tracking area
inside the FHD frame
	Rect FHDFSRgn(200,100,1520,880);  // The face search/tracking area
inside the FHD frame
	Rect FHDLFRg;			// Largest face region in the Full HD
camera.


//// Face Recongnition Related
	double Fcnf,Ecnf,Lcnf;
	float SbjCnf[100][2];		// This array will hold the confidence
values for up to 100 subjects [score][times] => average=score/times
	float HighestCnf=0;
	float HstAvgCnf;		// Calculated by dividing the highest
confidence over the number of times the face images were recognized as this
same person
	int SvdFcCntr=0;
	int index;			// This points to the person with the
highest confidence
	int NmbOfSbj;
	int Fprd,Eprd,Lprd;
	Mat Face,GrayFace;
	string Person;
	string FaceNameF,FaceNameE,FaceNameL;
	Ptr<FaceRecognizer>  Fisher = createFisherFaceRecognizer();
	Ptr<FaceRecognizer>  Eigen = createEigenFaceRecognizer();
	Ptr<FaceRecognizer>  LBPH = createLBPHFaceRecognizer();
	vector<int> FcLblsVct;
	vector<Mat> FcImgsVct;


// Approaching Related
	bool FndTrkFc=false; // A flag, when true it means a face was found
during face tracking
	bool Fresh;
	bool Arrived=false;		// A flag that becomes 'true' when the robot
arrives at the subject
	float x_ratio,y_ratio;
	int cols,rows;
	int FrmCnt=0;			// Frame Counter. This is used to number the
images saved
	int LFFW;			// Last Found-Face Width
	int RMPL;			// Robot Motors Power Level. Used to
gradually increase the speed of the robot
	int WAAngle=90;
	int PrvWAAngle=90;		// The current angle of the servo motor
rotating the Wide Angle camera
```

```
	Mat WAFTimg;				// Wide Angle Face Tracking image
	Rect WAFrect;				// Wide Angle Face rectangle
	Rect WAFTRg;				// Wide Angle Face Tracking Region


// Identity Verification
	string Password="hi";		// The password that should be entered by
the stranger
	int PSW;					// The password that will be entered by the
stranger
	int DIPsw;


//// Timing Related
	float FrontCamStartTime=clock()-4*CLOCKS_PER_SEC;
	float FrontCamElapsedTime;


// Others
	HWND hDesktopWnd=GetDesktopWindow();
	HWND handle = GetConsoleWindow();
	int TmpVal;				// a temporary variable used to calculate
values in lengthy calculations
	int TmpAng[6];			// a temporary variable used to hold the
angle values of the FHD cam
	Mat tmpFrame;			// A temporary image used to display
whatever image is to be displayed
	Rect Zero(0,0,0,0);


//############################ INITIALIZATIONS ##############################


	CMDhandle=IntializeWelcome();

	MoveWindow(CMDhandle, 685,400,700,320,TRUE);

	printf("\n\nStopping the robot");
	RobotMotors(100,100);

	printf("\n\nResetting the cameras to center position");
	CenterFHDcam();
	Sleep(750);
	CenterWAcam();

	Status.create(Size(700,200),1);


//=============== Fill in the previous frames ==============//

	printf("\n\nShall I check the camera directions and initialize PrvFrm
?\n");
	cin >> Ans;

	if(Ans=="Y" || Ans=="y")
	{

		printf("\n\nNow initializing the first few frames.\n");

		//// Use this line instead of the camera-reading lines if you
want to use a video file
```

```
		//Cam[4]=VideoCapture ("D:\\MTA\\UTM\\Research\\Master
Related\\Security Robot\\C++ Code\\Full App\\In the Lab - HD - 1.avi");


		// The followin lines will capture a frame for every camera in
order to fill the 'PrvFrame'
		// variable with the image of the frame previous to the one
being analyzed for motion
		for(int iii=0;iii<3;iii++)
		{

			Cam[iii]=VideoCapture (iii);
			Cam[iii].set(CV_CAP_PROP_FRAME_WIDTH,1920);
			Cam[iii].set(CV_CAP_PROP_FRAME_HEIGHT,1080);

			for(int tmp=0; tmp<5; tmp++)
				Cam[iii] >> PrvFrame[iii];  // The frame capturing
is repeated in order to get a good, stable image


			while(PrvFrame[iii].rows==0 || PrvFrame[iii].cols==0)
			{
				printf("\n\nThis is the 'Previous Frame'
initialization procedure ...");
				printf("\nI'm releasing and recapturing camera %d
as it is not responding",iii);
				Cam[iii].release();
					Sleep(500);
				Cam[iii] = VideoCapture(iii);
					Sleep(500);
				Cam[iii] >> PrvFrame[iii];
			}

			PrvFrame[iii].copyTo(tmpFrame);

			if(iii==FrontCam)
				putText(tmpFrame,"Front
Cam",Point(10,40),1,3,CV_RGB(255,0,0),3);

			if(iii==RightCam)
				putText(tmpFrame,"Right
Cam",Point(10,40),1,3,CV_RGB(255,0,0),3);

			if(iii==LeftCam)
				putText(tmpFrame,"Left
Cam",Point(10,40),1,3,CV_RGB(255,0,0),3);

			Show2(tmpFrame,format("Camera
%d",iii),0,0,Zero,Zero,3000,1,630,360);

			Cam[iii].release();
			Sleep(500);
		}

	}	// End of camera direction and PrvFrm initialization


//=========== Loading the Face, Eye & Body Detectors ===========//

	printf("\n\nLoading the face detection cascades");
```

```
	for(int fc=0;fc<5;fc++)
	{
		FaceDetector[fc].load(FaceDetectorName[fc]);
		if (FaceDetector[fc].empty())
		{
			printf("\nCould not load face cascade [%d]. Press any key
to exit",fc);
			system("pause");
			return -1;
		}
	}

	for(int bc=0;bc<3;bc++)
	{
		Bdy[bc].load(BodyCascadeName[bc]);
		if (Bdy[bc].empty())
		{
			printf("\nCould not load body cascade [%d]. Press any key
to exit",bc);
			system("pause");
			return -1;
		}
	}


	SmallEyeCascade.load(SmallEyeCascadeName);
	BigEyeCascade.load(BigEyeCascadeName);
	OtherEyeCascade.load(OtherEyeCascadeName);
	if (SmallEyeCascade.empty() || BigEyeCascade.empty() ||
OtherEyeCascade.empty())
	{
		printf("Could not load one of the eye detection haar cascades.
Press any key to exit");
		system("pause");
	}

//================ Face Recognizer Training ================//

	// Read the training face images
	cout << "\n\nNow reading the training images.";
	NmbOfSbj=ReadTrainingImages(FcImgsVct,FcLblsVct);


	//Train the Face Recognizers
	cout << "\n\nI'm now going to train the face recognizers; this may take
a few minutes...";
	cout << "\n\nTraining the Fisher face recognizer...";
	Fisher->train(FcImgsVct,FcLblsVct);

	cout << "\n\nTraining the Eigen faces recognizer...";
	Eigen->train(FcImgsVct,FcLblsVct);

	cout << "\n\nTraining the LBPH faces recognizer...";
	LBPH->train(FcImgsVct,FcLblsVct);



//################################ CALIBRATIONS ##############################

	printf("\n\n\n\t\t\tTESTING & CALIBRATION");
```

```
	printf("\n\nFirst: Please check that all the ultrasonic sensors are
operating correctly\n\n");
	system("pause");


	printf("\n\nNow Performing front & FHD camera calibration");
	printf("\n\nPress 'Enter' when your face (in a purple rectangle)
appears to be");
	printf(  "\nin the middle of the tracking region (red).");
	printf("\n\nIf not, then adjust the WA camera and press 'Space Bar' or
any other key");

	// Calibrate the direction of the Full HD camera
	FHDAngle[0]=90;
	FHDAngle[1]=90;

	// The Front camera will be captured temporarily to be used for
calibration
	Cam[FrontCam] = VideoCapture(FrontCam);
	Cam[FrontCam].set(CV_CAP_PROP_FRAME_WIDTH,1920);
	Cam[FrontCam].set(CV_CAP_PROP_FRAME_HEIGHT,1080);

	while(1)
	{
//------------------ Capturing a WA frame ----------------------

		// Capture three frames to get a good image
		Cam[FrontCam] >> WAFrame;
		Cam[FrontCam] >> WAFrame;
		Cam[FrontCam] >> WAFrame;

		while(!WAFrame.rows || !WAFrame.cols)
		{
			printf("\n\nCalibration Procedure: retrying to capture WA
cam");
			Cam[FrontCam].release();
				Sleep(250);
			Cam[FrontCam] = VideoCapture(FrontCam);
				Sleep(250);
			Cam[FrontCam].set(CV_CAP_PROP_FRAME_WIDTH,1920);
			Cam[FrontCam].set(CV_CAP_PROP_FRAME_HEIGHT,1080);
			Cam[FrontCam] >> WAFrame;
		}



//------------------ Face Detection in WA Frame ----------------------

		// First we try to find the face of the subject using a maximum
of four haar cascade classifiers
		LSRg=FindLrgstFace3(FaceDetector[0],WAFrame,90,50,30,0);

		WAFrame.copyTo(tmpFrame);
		if(LSRg.width>0)
		{

	putText(tmpFrame,format("Size=%d",LSRg.width),Point(LSRg.x,LSRg.y-
20),1,3,CV_RGB(255,0,255),3);
			Show2(tmpFrame,"Front Camera",0,0,Zero,LSRg,10,0,640,360);
		}
		else
		{
```

```
                        Show2(tmpFrame,"Front Camera",0,0,Zero,LSRg,10,0,640,360);
                        continue;      // If no faces are found, then grab a new
frame
                }


//------------------ Face Found = Aim FHD Cam ----------------------

                // 0.3 Backup FHDAngle as it may change during 'AimFHDcam5_2'
function call
                CopyArray(FHDAngle,0,1,TmpAng);

                // 1.0 Aiming the FHD cam at the subject using the coordinates
from the WA cam
                AimFHDcam5_2(WAFrame,LSRg,120,FHDAngle,0);

                // 1.1 Check whether an angle is beyond the limits
                        //printf("\n\nAiming angle problem at cell number %d,
value=%d\n",jjj,FHDAngle[jjj]);
                if(FHDAngle[2]!=0)
                        printf("\n\nUnrealistic horizontal angle adjustment;
value=%d\n",FHDAngle[2]);

                if(FHDAngle[3]!=0)
                        printf("\n\nUnrealistic vertical angle adjustment;
value=%d\n",FHDAngle[3]);

                if(FHDAngle[4]!=0)
                        printf("\n\nUnrealistic horizontal angle;
value=%d\n",FHDAngle[4]);

                if(FHDAngle[5]!=0)
                        printf("\n\nUnrealistic vertical angle;
value=%d\n",FHDAngle[5]);

                for(int iii=2;iii<6;iii++)
                {
                        if(FHDAngle[iii]!=0)
                                AngleError=true;
                }


                // 1.2 If an angle is beyond the limits, then the camera was not
aimed, so no face detection is required
                // and therefore we should restore FHDAngle values and return to
motion detection.
                if(AngleError)
                {
                        CopyArray(TmpAng,0,1,FHDAngle);    // restore FHDAngle
                        AngleError=false;                        // Reset
AngleError
                        continue;                                        // Go
back to motion detection
                }



//------------------ Capturing a FHD frame ----------------------

                // This line will release the front camera and capture the Full
HD Camera causing a few milliseconds delay
```

```
		// which is necessary to allow the FHD cam to stabilize after
aiming.
		Cam[FrontCam].release();
		Cam[FHDcam] = VideoCapture(FHDcam);
		Cam[FHDcam].set(CV_CAP_PROP_FRAME_WIDTH,1920);
		Cam[FHDcam].set(CV_CAP_PROP_FRAME_HEIGHT,1080);

		// Capture three frames to get a good image
		Cam[FHDcam] >> FHDframe;
		Cam[FHDcam] >> FHDframe;
		Cam[FHDcam] >> FHDframe;
		while(!FHDframe.rows || !FHDframe.cols)
		{
			printf("\n\nCalibration Procedure: retrying to capture FHD
cam");
			Cam[FHDcam].release();
				Sleep(250);
			Cam[FHDcam] = VideoCapture(FHDcam);
				Sleep(250);
			Cam[FHDcam].set(CV_CAP_PROP_FRAME_WIDTH,1920);
			Cam[FHDcam].set(CV_CAP_PROP_FRAME_HEIGHT,1080);
			Cam[FHDcam] >> FHDframe;
		}


//------------------ Face Detection in FHD Frame ----------------------

		// The above is based on face location, the below is based on
screen center
		FHDFSRgn.x=max(0,960-LSRg.width*4);		// Middle of
screen - Face Width * 3
		FHDFSRgn.y=max(0,540-LSRg.height*3);	// Middle of screen -
Face Height * 2
		FHDFSRgn.width=min(WAFrame.cols-FHDFSRgn.x-1,9*LSRg.width);
		FHDFSRgn.height=min(WAFrame.rows-FHDFSRgn.y-1,7*LSRg.height);


// This debug line serves to show the area inside which this section is going
to look for a face (in red)
// as well as a rectangle (in purple) showing the face search window that will
be used
Show2(FHDframe,"FHD Cam - Mapping WA Frame",650,0,FHDFSRgn,LSRg,10,0,640,360);


		// Extract the section that's expected to contain the face
		FHDframe(FHDFSRgn).copyTo(FHDFSimg);

		for (fd=0;fd<2;fd++)
		{

	FHDLFRg=FindLrgstFace3(FaceDetector[fd],FHDFSimg,LSRg.width,30,35,0);
			if(FHDLFRg.width>0)
				break; // This breaks from the inner 'for' loop
		}


//--------------------------- Face Found --------------------------------

		// 3.2.2 If a face is found, then add it to the array of faces
to be used later by the face rec. section
		if(FHDLFRg.width>0)
		{
```

```
                        FHDLFRg.x=FHDLFRg.x+FHDFSRgn.x;
                        FHDLFRg.y=FHDLFRg.y+FHDFSRgn.y;

                        // Show the face area and the tracking area in the FHD
frame image
                        Show2(FHDframe,"FHD Cam - Face
Capturing",0,360,FHDFSRgn,FHDLFRg,10,0,640,360);
                }


                if(waitKey(0)==13 && FHDLFRg.width>0)
                        break;
                else
                {
                        // Reset the Full HD camera angles
                        CenterFHDcam();
                        Cam[FHDcam].release();
                        FHDAngle[0]=90;
                        FHDAngle[1]=90;

                        // Re-captur the front wide angle camera
                        Cam[FrontCam] = VideoCapture(FrontCam);
                        Cam[FrontCam].set(CV_CAP_PROP_FRAME_WIDTH,1920);
                        Cam[FrontCam].set(CV_CAP_PROP_FRAME_HEIGHT,1080);

                }
        }

        CenterFHDcam();
        Cam[FrontCam].release();
        Cam[FHDcam].release();

        destroyWindow("Front Camera");
        destroyWindow("FHD Cam - Face Capturing");
        destroyWindow("FHD Cam - Mapping WA Frame");

//############################ MAIN OPERATIONS ##############################//

        printf("\n\nSurveillance operations have started.");

        while (1)
        {
//=========================== CAMERA SWITCHING ==========================//

                // First, we will release the FHD cam, just in case it's still
captured (you see, only one-cam-at-a-time is allowed)
                Cam[FHDcam].release();

                // This line calculates the time that has passed since the robot
rotated towards a suspected
                // intruder or the time elapsed since the front cam detected a
motion.
                // In such cases, it is necessary too keep the front cam active
to verify the source of motion
                FrontCamElapsedTime=(clock()-FrontCamStartTime)/CLOCKS_PER_SEC;


//-- Switch or Keep Current Cam ------------------------------------------

                // If more than 4 seconds have passed since rotating the robot
towards the subject or since
```

```
		if (FrontCamElapsedTime>4)                                        //
detecting motion using the front camera
		{// then resume normal 360 degree surveillance (camera juggling)
			Cam[CrCm].release();

			PrvCm=CrCm;
			CrCm++;
			if(CrCm>2)    CrCm=0;

			Cam[CrCm] = VideoCapture(CrCm);
		}



//-- Capture Frame from Current Cam ---------------------------------------

		Cam[CrCm] >> WAFrame;
		Cam[CrCm].set(CV_CAP_PROP_FRAME_WIDTH,1920);
		Cam[CrCm].set(CV_CAP_PROP_FRAME_HEIGHT,1080);
		Cam[CrCm] >> WAFrame;
		Cam[CrCm] >> WAFrame;

		// Verify that the captured frame does not have zero rows or
columns
		while(!WAFrame.cols || !WAFrame.rows)
		{
			printf("\n\nI'm releasing and recapturing camera %d as it
is not responding",CrCm);
			Cam[CrCm].release();
				Sleep(250);
			Cam[CrCm] = VideoCapture (CrCm);
				Sleep(250);
			Cam[CrCm] >> WAFrame;
		}



//============================ MOTION DETECTION ============================//

//Print(Status,1,"In Motion Detection");
//printf("\n\nIn Motion Detection. Camera %d",CrCm);

//-- COMPARE FRAMES & DETECT MOTION ---------------------------------------
		NMR=LMX3(Motion,WAFrame,PrvFrame[CrCm],15,0,10);
		WAFrame.copyTo(PrvFrame[CrCm]);


//-- Show Detected Motion -------------------------------------------------

		// Set the largest motion region width to zero. Later we will
check, if the largest motion region is big enough, then a face search will be
conducted
		LMW=0;

		for(int Rgn=0; Rgn<NMR;Rgn++)
		{

			// If a motion area's side is smaller than 12 pixels, then
it's not worth searching for faces as the face will not be clear
			if(Motion[Rgn].width<12 || Motion[Rgn].height<12)
					continue;
```

```
                        // find the largest motion region
                        if(Motion[Rgn].width>LMW)
                        {
                                LMW=Motion[Rgn].width;
                                LMR=Rgn;
                        }

                        // Draw a rectangle around the detected motion and display
the camera feed
                        // The rectangle is drawn in a tmpframe to keep the
'CurrentFrame' intact
                        WAFrame.copyTo(tmpFrame);
                        destroyWindow(format("Motion Detection - Camera
%d",PrvCm));  // this ensures showing only the current active camera

        putText(tmpFrame,format("Camera[%d]",CrCm),Point(5,40),1,3,CV_RGB(255,0
,0),3);
                        Show2(tmpFrame,format("Motion Detection - Camera
%d",CrCm),0,0,Motion[Rgn],Zero,10,0,640,360);

                        // A motion area larger than 200,000 sq. pixels (500x400)
is probably a false motion area caused by some pixel fluctuations

        if(Motion[Rgn].area()>=(0.75*WAFrame.cols*0.75*WAFrame.rows))
                        {
                                printf("\nIgnoring Motion Area, because its too big
= %d",Motion[Rgn].area());
                                continue;
                        }
                }


                // If a motion area's width is smaller than 12 pixels, then it's
not worth searching for faces as the face will not be clear
                if(LMW<12)
                        continue;


                // MOTION has been DETECTED, so we will search for a face in the
search area



//======================== FACE & BODY DETECTION ========================//

                // This counter is used to give the saved images a number so as
not to overwrite one another
                ImgCounter++;

//Print(Status,1,"Now doing face and body detection");
printf("\nNow doing face and body detection");

                // Set a flag in case the front camera detected the motion. This
will be useful to continue checking
                // using the front camera for a few seconds, so that if
something is found, the robot will do the
                // required action, and if nothing is found, then we will go
back to juggling the cameras
                if(CrCm==FrontCam)
                        FrontCamStartTime=clock();
```

```
		// Extract the image where motion has occured in order to search
it
		WAFrame(Motion[LMR]).copyTo(SrchRgnImg);
		WAFrame.copyTo(tmpFrame);

		// Calcualte a suitable face-search window size (between 12 and
30 pixels)
		FaceWin=0.25 * (min(SrchRgnImg.cols,SrchRgnImg.rows));
		FaceWin=max(12,FaceWin);	// The face search window should not
be smaller than 12 pixels
		//FaceWin=min(30,FaceWin);  // and not bigger than 35 pixels as
the faces appear smaller in the WA cam

		// First we try to find the face of the subject using a maximum
of four haar cascade classifiers
		for (fd=0;fd<5;fd++)
		{
//printf("\nfd=%d",fd);

	LFRg=FindLrgstFace3(FaceDetector[fd],SrchRgnImg,FaceWin,50,30,0);
			if(LFRg.width>0)
			{
				putText(tmpFrame,format("FaceDetector[%d] found the
face. Skin>=50, Sharpness<=30",fd),Point(5,40),1,3,CV_RGB(255,0,0),3);
				imwrite(format("The Face Detector that found the
Face %d.png",ImgCounter),tmpFrame);
				break;
			}
		}

		// If no faces are found in the motion area, then check for a
body of a subject
		if(LFRg.area()==0)
		{
printf("\nFace Not found, checking for presence of a body");

	LBRg=FindLrgstBody(SrchRgnImg,Bdy[0],Bdy[1],Bdy[2],FaceWin);

			// If also no body is found, then go back to motion
detection
			if(LBRg.area()==0)
			{
printf("\nBody not found, returning to motion detection");
				continue;
			}
		}


		// Now, LSRg will either become LFRg or LBRg depending on which
one was found (detected)
		if(LFRg.width>0)
		{
			LSRg=LFRg;
			DtctdRgn="Face Region";
		}
		else
		{
			LSRg=LBRg;
			DtctdRgn="Body Region";
			//rectangle(tmpFrame,LSRg,CV_RGB(255,0,255),2);
```

```
                    //putText(tmpFrame,"Body Found",Point(5,tmpFrame.rows-
10),1,1,CV_RGB(255,0,0),1);
              }

              // Drawing a Rectangle around the face
              LSRg.x+=Motion[LMR].x;
              LSRg.y+=Motion[LMR].y;

              putText(tmpFrame,"Motion
Region",Point(Motion[LMR].x,Motion[LMR].y-10),1,1.5,CV_RGB(255,0,0),2);
              putText(tmpFrame,DtctdRgn,Point(LSRg.x,LSRg.y-
10),1,2,CV_RGB(255,0,255),2);

              Show2(tmpFrame,format("Motion Detection - Camera
%d",CrCm),0,0,Motion[LMR],LSRg,10,0,640,360);
//imwrite(format("10.0 Detected Faces %d.png",ImgCounter),tmpFrame);

printf("\nFace Found\n");


//================= ROTATE THE ROBOT TO FACE THE PERSON ==================//

// In this section, if a person is detected and he/she is not infront of the
robot, then the robot
// is rotated to face the person.



              if(CrCm != FrontCam)        // If a face is detected in either
camera 0 or 1, then ...
              {
                    Beep(1000,750);
printf("\nNow rotating the robot");
                    if(CrCm==RightCam)
                    {
                          Say("Rotating Right");
                          RobotMotors(20,180); // If the camera number is 0
(right camera), then rotate right
                          Sleep(2000);
                          RobotMotors(100,100);       // If the camera number
is 0 (right camera), then rotate right
                    }
                    //Sleep(1000);
                    else
                    {
                          Say("Rotating Lefft");
                          RobotMotors(180,20); // Otherwise, the camera
number is 1 (left camera), so rotate left
                          Sleep(2000);
                          RobotMotors(100,100);       // If the camera number
is 0 (right camera), then rotate right
                    }

                    FndTrkFc=true;                    // Turn face tracking
ON, as the robot is currently (supposedly) facing the subject
                    Cam[CrCm].release(); // Release the current camera (0 or
1)
              //    de`stroyWindow(format("Camera %d motion detection",CrCm));

                    CrCm=FrontCam;                                  // Change
to camera 2 (the front wide angle camera)
```

```
                    Cam[CrCm] = VideoCapture(CrCm);    // Start capture the
feed from camera 2

                    Cam[CrCm].set(CV_CAP_PROP_FRAME_WIDTH,1920);
                    Cam[CrCm].set(CV_CAP_PROP_FRAME_HEIGHT,1080);

                    Cam[CrCm] >> PrvFrame[CrCm];       // Capture a frame and
put it in 'PrevFrame' in order to start motion detection

                    FrontCamStartTime=clock();
                    continue;

               }



//=================== ACQUIRE THE SUBJECT'S FACE IMAGE ===================//


printf("\nAcquiring the person's face image with the Full HD camera");

               Say("STOP");

               CenterFHDcam();                         // Reset the Full HD
Camera to center position

               // This line will delay for a few milliseconds while Camera 4
(the Full HD Camera) is being captured and aimed forward
               Cam[FrontCam].release();
               Cam[FHDcam] = VideoCapture(FHDcam);
               Cam[FHDcam].set(CV_CAP_PROP_FRAME_WIDTH,1920);
               Cam[FHDcam].set(CV_CAP_PROP_FRAME_HEIGHT,1080);

               FHDAngle[0]=90;
               FHDAngle[1]=90;

               // The face tracking process is as follows:
               // 1- Aim the FHD camera using the WA cam coordinates.
               // 2- Capture two or three frames
               // 3- Try to find the face in the captured frame(s)
               // 4- If a face is not found then go back to motion detection.
               // 5- If a face is found, then store its images and pass them to
the face recognition section

               // 0.0 Necessary reinitializations
               // 0.1 Reset the full HD face counter to zero
               FHDFcCntr=0;

               // 0.2 This variable counts how many times a face was not found
when looking for it.
               // In this version the maximum is three.
               FaceNotFound=0;

               // 0.3 Backup FHDAngle as it may change during 'AimFHDcam5_2'
function call
               CopyArray(FHDAngle,0,1,TmpAng);


//// These debug lines show the scenery of the Full HD before aiming.
//// This is done in order to compare this scenery with the scenery after
aiming
```

```
//Cam[FHDcam] >> FHDframe;
//Cam[FHDcam] >> FHDframe;
//Show2(FHDframe,"FHD Cam befor aiming",0,370,Zero,Zero,1,0,640,360);


		// 1.0 Aiming the FHD cam at the subject using the coordinates
from the WA cam
		AimFHDcam5_2(WAFrame,LSRg,120,FHDAngle,30);

		Say("Please wait while I verify your identity");

		// 1.1 Check whether an angle is beyond the limits
			//printf("\n\nAiming angle problem at cell number %d,
value=%d\n",jjj,FHDAngle[jjj]);
		if(FHDAngle[2]!=0)
			printf("\n\nUnrealistic horizontal angle adjustment;
value=%d\n",FHDAngle[2]);

		if(FHDAngle[3]!=0)
			printf("\n\nUnrealistic vertical angle adjustment;
value=%d\n",FHDAngle[3]);

		if(FHDAngle[4]!=0)
			printf("\n\nUnrealistic horizontal angle;
value=%d\n",FHDAngle[4]);

		if(FHDAngle[5]!=0)
			printf("\n\nUnrealistic vertical angle;
value=%d\n",FHDAngle[5]);

		for(int iii=2;iii<6;iii++)
		{
			if(FHDAngle[iii]!=0)
				AngleError=true;
		}


		// 1.2 If an angle is beyond the limits, then the camera was not
aimed, so no face detection is required
		// and therefore we should restore FHDAngle values and return to
motion detection.
		if(AngleError)
		{
			CopyArray(TmpAng,0,1,FHDAngle);	// restore FHDAngle
			AngleError=false;			// Reset
AngleError
			printf("\nGoing back to motion detection");
			Say("Please continue");
			Say("Have a good day");
			continue;				// Go
back to motion detection
		}


		// This line will delay for a few milliseconds while the camera
completes its aiming and stabilizes
		Sleep(25*abs(FHDAngle[0])+10);


		// 2.0 Capture a frame or two using the Full HD camera
		Cam[FHDcam] >> FHDframe;
		Cam[FHDcam] >> FHDframe;
```

```
		Cam[FHDcam] >> FHDframe;

		// 2.1 If the frame is corrupted after five tries, then restart
by going back to motion detection
		if(CheckFrame(Cam[FHDcam],FHDframe,5)==false)
		{
			printf("\nCouldn't capture a frame from the FHD cam. Going
back to Motion Detection\n");
			Say("Please continue");
			Say("Have a good day");
			continue;
		}


		// 3.0 Search for faces inside the frame
		// The following will try to capture five face images

		// 3.1 Designate a face-search window that is large, but focused
towards the person (see the 'Notes' document)
		//FHDFSRgn.x=max(0,LSRg.x-LSRg.width*5);  // WA Face Location x -
Face Width * 5
		//FHDFSRgn.y=max(0,LSRg.y-LSRg.height*3); // WA Face Location y -
Face Width * 3
		//FHDFSRgn.width=min(WAFrame.cols-FHDFSRgn.x-1,11*LSRg.width);
		//FHDFSRgn.height=min(WAFrame.rows-FHDFSRgn.y-1,7*LSRg.height);

		// The above is based on face location, the below is based on
screen center
		FHDFSRgn.x=max(0,960-LSRg.width*3);		// Middle of
screen - Face Width * 3
		FHDFSRgn.y=max(0,540-LSRg.height*2);	// Middle of screen -
Face Height * 2
		FHDFSRgn.width=min(WAFrame.cols-FHDFSRgn.x-1,7*LSRg.width);
		FHDFSRgn.height=min(WAFrame.rows-FHDFSRgn.y-1,5*LSRg.height);


// This debug line serves to show the area inside which this section is going
to look for a face (in red)
// as well as a rectangle (in purple) showing the face search window that will
be used
Show2(FHDframe,"FHD Cam - Face Capturing",650,0,FHDFSRgn,LSRg,10,0,640,360);


		// 3.2 Try to find the face of the subject using two haar
cascade classifiers (for speed, otherwise I would use 5)
		do
		{
			// 3.3 Extract the section that's expected to contain the
face
			FHDframe(FHDFSRgn).copyTo(FHDFSimg);

//-------------------------- Face Detection Loop ----------------------------
			for (fd=0;fd<2;fd++)
			{

	FHDLFRg=FindLrgstFace3(FaceDetector[fd],FHDFSimg,LSRg.width,30,35,0);
				if(FHDLFRg.width>0)
				{
					printf("\nFaceDetector[%d] found the
face",fd);
					break; // This breaks from the inner 'for'
loop
```

```
                        }
                }
//---------------------- End of Face Detection Loop ------------------------


                // If a face is found, then add it to the array of faces
to be used later by the face rec. section
                if(FHDLFRg.width>0)
                {
                        FHDLFRg.x=FHDLFRg.x+FHDFSRgn.x;
                        FHDLFRg.y=FHDLFRg.y+FHDFSRgn.y;

                        // If a face is found, then store it's image.
                        FHDframe(FHDLFRg).copyTo(FHDFace[FHDFcCntr]);

                        // If 5 images are captured so far then exit the
'do-while' loop.
                        FHDFcCntr++;
                        FaceNotFound=0;
                }
                else
                        FaceNotFound++;

                if(FaceNotFound>=3)
                        break;
                else
                        Sleep(20);



        }while(FHDFcCntr<=7);

//------------------ End of 'do' loop which searches for faces --------------


        // 4- If a face is not found (counter is still zero) then go
back to motion detection.
        if(FHDFcCntr==0)
        {
                printf("\nNo face was found by the FHD cam. Going back to
motion detection\n");

// This debug line serves to show the area inside which this section looked
for a face (in red)
// as well as a rectangle (in purple) showing the face search window that was
used
//Show2(FHDframe,"FHD Cam - Face Capturing",650,0,FHDFSRgn,LSRg,0,1,640,360);

                Say("Please continue");
                Say("Have a good day");

                destroyWindow("FHD Cam - Face Capturing");

                continue;                               // go back to
motion detection
        }
        else    // otherwise
        {
                // Show the captured face images during face tracking (up
to 5 images)
                for(int iii=0;iii<FHDFcCntr;iii++)
                {
```

```
                        resize(FHDFace[iii],tmpFrame,Size(250,250));
                        imshow(format("Face %d",iii),tmpFrame);

                        moveWindow(format("Face %d",iii),iii*250,400);
                }

                // This line shows the face images and waits for two
seconds before erasing them from the screen
                if(!SnW(100)) return 0;

                for(int iii=0;iii<FHDFcCntr;iii++)
                        destroyWindow(format("Face %d",iii));

        }


        // Remove the FHD window before moving on to Face Recognition
        destroyWindow("FHD Cam - Face Capturing");



//============================ FACE RECOGNITION ============================//


        // Reset the confidence values for each subject for the current
image
        for(int df=1; df<=NmbOfSbj; df++)
        {
                SbjCnf[df][0]=0;
                SbjCnf[df][1]=0;
        }



//-- The Recognition and Prediction Process --------------------------------

        // Check all the captured faces by the FHD camera
        for (int fr=0;fr<FHDFcCntr;fr++)
        {
                // We start with an "Unknown" identity
                Person=FaceNameF=FaceNameE=FaceNameL="Unknown";

                HighestCnf=0; // The highest confidence value for the
captured face image

                // Preparations for face recognition operations
                cvtColor(FHDFace[fr],GrayFace,CV_RGB2GRAY);
                resize(GrayFace,GrayFace,Size(100,120));
                equalizeHist(GrayFace,GrayFace);   // I found that this
negetively affects recognition

                /////////// FACE RECOGNITION OPERATIONS //////////
                // FISHER ////
                Fprd=-1;
                Fcnf=0.0;
                Fisher->predict(GrayFace,Fprd,Fcnf);

                // Convert the confidence to be between 0 & 100 %.
                Fcnf=Ffactor/Fcnf;
                Fcnf=min(Fcnf,100.0);
                Fcnf=max(Fcnf,0.0);
```

```
                    if(Fprd>-1)   // Add the confidence value to the person's
score (the cell in the 'SbjCnf' array)
                    {
                           FaceNameF=Names[Fprd];
                           SbjCnf[Fprd][0]=SbjCnf[Fprd][0]+Fcnf;
                           SbjCnf[Fprd][1]++;
                    }



                    //// EIGEN ////
                    Eprd=-1;
                    Ecnf=0.0;
                    Eigen->predict(GrayFace,Eprd,Ecnf);

                    // Convert the confidence to be between 0 & 100 %.
                    Ecnf=Efactor/Ecnf;
                    Ecnf=min(Ecnf,100.0);
                    Ecnf=max(Ecnf,0.0);

                    if(Eprd>-1)   // Add the confidence value to the person's
score (the cell in the 'SbjCnf' array)
                    {
                           FaceNameE=Names[Eprd];
                           SbjCnf[Eprd][0]=SbjCnf[Eprd][0]+Ecnf;
                           SbjCnf[Eprd][1]++;
                    }



                    //// LBPH ////
                    Lprd=-1;
                    Lcnf=0.0;
                    LBPH->predict(GrayFace,Lprd,Lcnf);

                    // Convert the confidence to be between 0 & 100 %.
                    Lcnf=Lfactor/Lcnf;
                    Lcnf=min(Lcnf,100.0);
                    Lcnf=max(Lcnf,0.0);

                    if(Lprd>-1)   // Add the confidence value to the person's
score (the cell in the 'SbjCnf' array)
                    {
                           FaceNameL=Names[Lprd];
                           SbjCnf[Lprd][0]=SbjCnf[Lprd][0]+Lcnf;
                           SbjCnf[Lprd][1]++;
                    }



                    // Determine the name by which the current face image
should be stored
                    if(Fcnf>=Ecnf && Fcnf>=Lcnf)
                    {
                           Person=FaceNameF;
                           HighestCnf=Fcnf;
                    }

                    if(Ecnf>=Fcnf && Ecnf>=Lcnf)
                    {
                           Person=FaceNameE;
                           HighestCnf=Ecnf;
```

```
                        }

                        if(Lcnf>=Fcnf && Lcnf>=Ecnf)
                        {
                                Person=FaceNameL;
                                HighestCnf=Lcnf;
                        }

                        // Save the image with the name of the highest confidence
                        if(HighestCnf<50)
                                Person=format("Unknown %d %d
",SvdFcCntr,fr)+Person+format(" %2.2f.png",HighestCnf);
                        else
                                Person=format("%d %d
",SvdFcCntr,fr)+Person+format(" %2.2f.png",HighestCnf);

                        imwrite(Person,FHDFace[fr]);

                }


                // Reset the variables used in the recognition and prediction
process
                HighestCnf=0; // The highest confidence value for the captured
face image
                index=-1;            // Points to the name of the subject if
known, otherwise -1 means "unknown"
                SvdFcCntr++;  // Increment the counter for saved face images

        //////// Calculate (by accumulation) the highest confidence for this
face
                for(int df=1; df<=NmbOfSbj; df++)
                {
                        // find the person with the highest confidence
                        if(SbjCnf[df][0]>HighestCnf)
                        {
                                HighestCnf=SbjCnf[df][0];
                                index=df;
                                Person=Names[df];
                        }
                }


                HstAvgCnf=SbjCnf[index][0]/SbjCnf[index][1];


//================ Deciding whether to Approach the Subject ================//

                if(HstAvgCnf<50)
                {
                        Beep(750,1500);      //Alarm(1);          // Raises the
first alarm
                        printf("\nThe person in the image is unknown");
                        printf("\n\nAn unrecognized person has been detected");
                        Sleep(500);          // This delay is important in order
to separate the beep from the next speech output
                }
                else
                {
                        printf("\nThe person in the image could be : %s with a
confidence of %2.2f",Person.c_str(),HstAvgCnf);
```

```
                    if(FcImgsVct.empty() || FcLblsVct.empty())
                           printf("\nEither the image or the label vector is
empty");

                    // If the total average confidence of this subject is
above or equal 50% and LBPH confidence is lower
                    // than 90%, then update the LBPH face recognizer since
the face is somewhat unfamiliar to LBPH
                    if( Lcnf<90 && (FaceNameL==Person) )
                    {
                           // Check all the captured faces by the FHD camera
                           for (int fr=0;fr<FHDFcCntr;fr++)
                           {
                                  // Preparations for face recognition
operations
                                  cvtColor(FHDFace[fr],GrayFace,CV_RGB2GRAY);
                                  resize(GrayFace,GrayFace,Size(100,120));
                                  equalizeHist(GrayFace,GrayFace);

                                  ////////// FACE RECOGNITION OPERATIONS
//////////
                                  //// LBPH ////
                                  Lprd=-1;
                                  Lcnf=0.0;
                                  LBPH->predict(GrayFace,Lprd,Lcnf);

                                  // Convert the confidence to be between 0 &
100 %.
                                  Lcnf=Lfactor/Lcnf;
                                  Lcnf=min(Lcnf,100.0);
                                  Lcnf=max(Lcnf,0.0);

                                  // Only update LBPH if the confidence in
this image is 50 - 75 %
                                  if( Lprd>-1 && (Lcnf<75 && Lcnf>50) )    //
Add the confidence value to the person's score (the cell in the 'SbjCnf'
array)
                                  {
                                         // This line pushes the image into
the images vector in order to be used by the LBPH to update its database
                                         FcImgsVct.push_back(GrayFace);

                                         // This line saves the subject labe
to be used by LBPH to update its database if required
                                         FcLblsVct.push_back(index);

                                  }
                           }

                           LBPH->update(FcImgsVct,FcLblsVct);
                    }

                    Say("Please continue");
                    Say("Have a good day");
                    continue;                                  // go back to
motion detection
             }


//======================== APPROACH THE PERSON ==========================//

//Print(Status,1,"Now approaching the person");
```

```
printf("\nNow approaching the person");

                Say("DON'T MOVE");

                // Release the full HD camera and Re-captur the front wide angle
camera
                Cam[FHDcam].release();
                Cam[FrontCam] = VideoCapture(FrontCam);
                Cam[FrontCam].set(CV_CAP_PROP_FRAME_WIDTH,1920);
                Cam[FrontCam].set(CV_CAP_PROP_FRAME_HEIGHT,1080);

//-- Mapping & Reinitialization -----------------------------------------------
//==== Firstly, we map the location of the face/subject from the Full HD to
the VGA resolution

                // 1- Calculate the ratio of the 'face tracking frame' to the
'motion detection frame'
                cols=WAFrame.cols;
                rows=WAFrame.rows;
                x_ratio=(float)WAFrame.cols/640;
                y_ratio=(float)WAFrame.rows/360;
                resize(WAFrame,WAFrame,Size(640,360));

                // 2- Adjust the face region's coordinates and size according to
the 'face tracking frame'
                WALFRg.x=LSRg.x/x_ratio;
                WALFRg.y=LSRg.y/y_ratio;
                WALFRg.width=LSRg.width/x_ratio;
                WALFRg.height=LSRg.height/y_ratio;

                // 3- Adjust the face region's coordinates and size according
based on the new FoV (=100)
                if(WALFRg.x>320+WALFRg.width) // if the face is on the right
side of the frame image
                        WALFRg.x=WALFRg.x*1.12;

                if(WALFRg.x<320-WALFRg.width) // if the face is on the left side
of the frame image
                        WALFRg.x=WALFRg.x/1.12;

                if(WALFRg.y>240+WALFRg.height) // if the face is on the lower
side of the frame image
                        //WALFRg.y=WALFRg.y/1.12;
                        WALFRg.y=WALFRg.y/(1.12*480/360);  // accomodate the
change in FoV as well as the change from 360p to 480p

                if(WALFRg.y<240-WALFRg.height) // if the face is on the higher
side of the frame image
                        //WALFRg.y=WALFRg.y*1.12;
                        WALFRg.y=WALFRg.y*(1.12*480/360);


//// These debug lines show the face/body region based on the WAFrame as well
as the face and tracking regions
//Show2(WAFrame,"The WAFTRg (red) & WALFRg (purple)",650,0, WAFTRg, WALFRg,
0,0,640,360);

//==== Secondly, we reinitialize some variables that will be used in the
tracking activity

                // Set the 'Found Tracked Face' flag to false. This flag will be
set back to true if a face is found
```

```
		int FaceNotFoundCounter=0;
		Fresh=true;

		// The Last Found Face Width
		LFFW=WALFRg.width;

		// Reset the Robot Motor Power Level to 10%. This counter is
useful in making the robot speed up gradually
		RMPL=10;

		// Set the tracking camera resolution to 640x480 (VGA)
		Cam[FrontCam].set(CV_CAP_PROP_FRAME_HEIGHT,480);
		Cam[FrontCam].set(CV_CAP_PROP_FRAME_WIDTH,640);
		// And capture two frames to empty the camera's buffer so that
we get new frames when we enter the tracking loop
		Cam[FrontCam] >> WAFrame;
		Cam[FrontCam] >> WAFrame;

		// Aim the WA cam and acquire its angle
		PrvWAAngle=WAAngle;
		WAAngle=AimWAcam3(WAFrame,WALFRg,130,PrvWAAngle);
		Sleep(25*abs(WAAngle-PrvWAAngle)+10);

//-- The Tracking Loop ---------------------------------------------------
//==== Thirdly, we now TRACK THE FACE
		while(WALFRg.width>0)
		{
Print(Status,1,"I'm now inside Approach's while loop");
Print(Status,2,format("Power Level = %d %%",RMPL));


//>>>>>>>>>> 1.0 Capture an image using the wide angle camera
			// Three frames are captured because the camera may buffer
a frame which is the same as the previous one.
			Cam[FrontCam] >> WAFrame;
			Cam[FrontCam] >> WAFrame;
			Cam[FrontCam] >> WAFrame;


			// 1.1 Go back to motion detection if frames cannot be
captured
			if(CheckFrame(Cam[FrontCam],WAFrame,3)==false)
			{
				printf("\nCouldn't capture an image from the Wide
Angle camera. Going back to motion detection.");
				RobotMotors(100,100);
				//Say("Please continue");
				//Say("Have a good day");
				break;
			}
			//else // this is used when using the hd or full hd video
rather than the camera
			//	resize(WAFrame,WAFrame,Size(640,480));


			// 2.0 Save images of the environment during approach in
case the administrator needs to see it
			// 2.1 Increment the saved frame image counter
			FrmCnt++;

			// 2.2 And save a picture of the environment (frame image)
```

```
                    //imwrite(format("WAFrame %d.png",FrmCnt),WAFrame);


                    // 3.0 Find the face in the Wide Angle frame image
                    // 3.1 Prepare the portion of the frame image that is
expected to contain the face

                    // 3.1.1 Calculate a face search region which is nine
times (3x3) the size of the currently found face.
                    if(FaceNotFoundCounter==0)  // This 'if' statement ensures
that WAFTRg will only be calculated if a face has
                    {                                                       // been
previously found, otherwise, the dimensions (100,100,440,240) will be used

                            // These lines locate the tracking region at the
center of the WA frame image. WAFTRg Size=6*Face.w,4*Face.h
                            WAFTRg.x=max(0,320-WALFRg.width*4);      // 320 -
Face Width * 3
                            WAFTRg.y=max(0,240-WALFRg.height*2);     // 240 -
Face Height * 2
                            WAFTRg.width=min(WAFrame.cols-
WAFTRg.x,9*WALFRg.width);
                            WAFTRg.height=min(WAFrame.rows-
WAFTRg.y,5*WALFRg.height);

                    }


                    // 3.1.2 Extract the portion of WAFrame that is enclosed
by WAFTRg which is thought to contain the face
                    WAFrame(WAFTRg).copyTo(WAFTimg);

//// These debug lines show the face-tracking portion of the frame image.
//if(WAFTimg.cols && WAFTimg.rows)
//     imshow("WAFTimg",WAFTimg);

                    // 3.2 Set the size of the face search window to 0.65 the
size of the previously found face (to make sure we detect it)
                    FaceWin=0.65*WALFRg.width;


Rect FcWn(WAFTRg.x,WAFTRg.y,FaceWin,FaceWin);
Show2(WAFrame,"Approaching - Wide Angle Camera",650,0, WAFTRg, FcWn, 10,0);

//-- Face Search-------------------------------------------------------------
                    // 3.3 Search for faces inside the search area of the
frame ( WideAngleFrameTracking image (WAFTimg) )
                    // The following small loop will try to capture a new
frame and search it in case the current one doesn't contain a face
                    for(int iii=1;iii<3;iii++)
                    {
                            // We try to find the face of the subject using a
maximum of four haar cascade classifiers
                            for (fd=0;fd<5;fd++)
                            {

      WALFRg=FindLrgstFace3(FaceDetector[fd],WAFTimg,FaceWin,30,35,0);
                                    if(WALFRg.width>0)
                                    {
                                            Fresh=false;  // Since a face is
found, the loop is no longer fresh
```

```
                                        printf("\nFaceDetector[%d] found the
face",fd);
                                        break;
                                }
                        }

                        // If a face is found, exit the face-search loop
                        if(WALFRg.width>0)
                                break;

                        // Otherwise capture two frames (to skip camera
buffering) and search again
                        Cam[FrontCam] >> WAFrame;
                        Cam[FrontCam] >> WAFrame;

                        if(!WAFrame.cols || !WAFrame.rows)
                        {
                                printf("\nCouldn't capture a correct frame
from the WA Cam. Exiting Face-Search");
                                break;
                        }

                }

Show2(WAFrame,"Approaching - Wide Angle Camera",650,0, WAFTRg, WALFRg, 10,0);
//Print(Status,3,format("detected face size= %d",WALFRg.width));


//-- No Face Found -------------------------------------------------------
                // 4.0 If no faces are found, then try to expand the
tracking area and decrease the face size
                // If that's no use, then exit the while loop and go back
to motion detection
                if(WALFRg.width==0)
                {
                        // Reduce power to the robot motors by 50% to stop
gradually if a face is not found again, and
                        RMPL=RMPL/2;         // pick up quickly when a face
is re-found.

                        if(FaceNotFoundCounter==0)
                        {
                                //RobotMotors(100,100);     // Stop the
robot
                                destroyWindow("Wide Angle Camera Tracking
Frame"); // Remove the tracking window

                                // Set the new and expanded tracking region
                                WAFTRg.x=max(0,320-LFFW*5);
                                WAFTRg.width=min(LFFW*10,640-WAFTRg.x);

                                WAFTRg.y=max(0,240-LFFW*3);
                                WAFTRg.height=min(LFFW*6,480-WAFTRg.y);

                        }
                        else
                        {       // Gradually expand the height of the
searching area to cover more space

                                // This line results in an increase of one
face in width each time the face is not found
                                TmpVal=5+FaceNotFoundCounter*0.5;
```

```
                                        WAFTRg.x=max(0,320-LFFW*TmpVal);
                                        WAFTRg.width=min(LFFW*(2*TmpVal),640-
WAFTRg.x);

                                        // This line results in an increase of half
a face in height each time the face is not found
                                        TmpVal=3+FaceNotFoundCounter*0.25;
                                        WAFTRg.y=max(0,240-LFFW*TmpVal);
                                        WAFTRg.height=min(LFFW*(2*TmpVal),480-
WAFTRg.y);
                                }

                                FaceNotFoundCounter++;

                                if(FaceNotFoundCounter>10)
                                        break;

                                WALFRg.width=max(12.0,0.9*LFFW);

                                continue;
                        }
                        else
                                FaceNotFoundCounter=0;


//-- Face Found -----------------------------------------------------------

                        // Hold the value of the found face as we will need it in
searching for the face in case the face is lost
                        LFFW=WALFRg.width;

                        // 5.0 Display the frame image while highlighting the face
and search (tracking) regions on it
                        // 5.1 Make a temporary copy of the frame image so we
don't ruin the frame image by drawing the rectangles
                        WAFrame.copyTo(tmpFrame);

                        // 5.2 Calculate the face region's global coordinate in
order to draw a rectangle around the face
                        WALFRg.x=WALFRg.x+WAFTRg.x;
                        WALFRg.y=WALFRg.y+WAFTRg.y;

                        // 5.3 Draw a rectangle around the search (tracking) on
the temporary frame image
                        putText(tmpFrame,"Tracking
Region",Point(WAFTRg.x,WAFTRg.y-10),1,1.5,CV_RGB(255,0,255),2);
                        rectangle(tmpFrame,WAFTRg,CV_RGB(255,0,255),2);

                        // 5.4 Draw a rectangle around the found face on the
temporary frame image
                        putText(tmpFrame,"Face",Point(WALFRg.x,WALFRg.y-
10),1,1.5,CV_RGB(255,0,0),2);
                        rectangle(tmpFrame,WALFRg,CV_RGB(255,0,0),2);

                        // 5.5 Display the frame image
                        imshow("Wide Angle Camera Tracking Frame",tmpFrame);
                        moveWindow("Wide Angle Camera Tracking Frame",650,0);
                        if(!SnW(10)) return 0;

                        // 6.0 Re-Aim the camera at the subject region (face or
body) for two reasons:
                        // 1) To prepare for the next round of the loop
```

```
                        // 2) To drive the robot towards the subject by following
the angle
                        PrvWAAngle=WAAngle;
                        WAAngle=AimWAcam3(WAFrame,WALFRg,90,PrvWAAngle);


                        // 7.0 Drive the robot towards the subject by following
the angle
                        Approach7wi(WAAngle,RMPL,tmpFrame,FrmCnt);

                        // 7.1 Increase RMPL (Robot Motor Power Level) in order to
gradually speed up the robot
                        if(RMPL<100)
                                RMPL=RMPL+10;


                        // 8.0 If the face size is large than 70 pixels, then that
means the robot has arrived at the subject, so exit this 'while' loop
                        if(WALFRg.width>65)
                        {
                                Arrived=true;
                                break;
                        }


                        // The robot has not reached the subject yet, so let's go
for another round of this 'while' loop
                }       // End of the 'face tracking' loop:
'while(WALFRg.width>0)'

                //// This delay is necessary to allow for a smooth camera
movement (due to low power supply by the
                //// system, only one camera operation can happen at a time. The
last operation was aiming the WA cam)
                //Sleep(500);

                // It's important to re-center the Wide Angle Camera before
going back to motion detection
                CenterWAcam();


                if(Arrived)
                {
                        printf("\n\nI have arrived at the subject\n");

                        // Stop the robot
                        RobotMotors(100,100);

                        // Ask the person to enter a password
                        Say("Please enter the password");
                        printf("\n\nPlease enter the password : ");
                        SetForegroundWindow(CMDhandle);
                        cin >> PSW;

                        DIPsw=ReadDIP();

                        if(PSW!=DIPsw)
                        {
                                printf("\n\nWrong password has been entered");

                                //Alarm(2);          // Raises the second alarm
                                Beep(600,250);
```

```
                        Beep(500,750);

                        //printf("\n\nI don't have instructions to block
your way.");
                        //Sleep(2000);
                        printf("\n\nPlease report to the administrator.");
                        Sleep(2000);
                        printf("\n\nHave a good day.");
                        Sleep(2000);


                        Cam[FrontCam] >> tmpFrame;
                        Cam[FrontCam] >> tmpFrame;
                        imwrite(format("Unidentified person Front Cam
%d.png",ImgCounter),tmpFrame);

                        Cam[FHDcam] >> tmpFrame;
                        Cam[FHDcam] >> tmpFrame;
                        imwrite(format("Unidentified person FHD Cam
%d.png",ImgCounter),tmpFrame);
                }
                else
                {
                        Say("Please continue");
                        Say("Have a good day");
                }

                // Remove the face-tracking camera-windows
                destroyWindow("Wide Angle Camera Tracking Frame");
                destroyWindow("Approaching - Wide Angle Camera");

                continue;
        }
        else
        {
                printf("\n\nLost track of the person");

                // Stop the robot
                RobotMotors(100,100);

                // Raises the third alarm
                for(int iii=0;iii<10;iii++)
                {
                        Beep(1000,500);
                        Beep(1500,500);
                }

                // Remove the face-tracking camera-windows
                destroyWindow("Wide Angle Camera Tracking Frame");
                destroyWindow("Approaching - Wide Angle Camera");

                printf("\n\nNothing to do, but to go back to motion
detection ...");
                continue;
        }



    } // end of the major program loop; the (while(1)) which runs the
motion detetion, face detection and recognition operations
```

```
 } // end of main()


//------------------------------------ END OF MAIN --------------------
//============================================================================


////////////////////////////////////////////////////////////////////////////
// This function is just an introduction to the program
////////////////////////////////////////////////////////////////////////////
HWND IntializeWelcome(void)
{
	HWND CMDhandle = GetConsoleWindow();
	if (CMDhandle == 0)
	{
		printf("\n\nCouldn't get a console window handle\n");
		system("pause");
		return 0;
	}

	MoveWindow(CMDhandle, 0,260,700,300,TRUE);

	printf("\n\nWelcome to the Indoor Surveillance Robot program");
	printf("\nThis program was written by Mohammad Tahir Ahmad for the
Master project");
	printf("\n'Authentication Based Security Robot Incorporating
Omnidirectional Vision'.\n");
	//Sleep (2000);

	return CMDhandle;
}


////////////////////////////////////////////////////////////////////////////
// This function reads the images and labes that will be used to recognize the
// subjects
////////////////////////////////////////////////////////////////////////////
int ReadTrainingImages(vector<Mat> &FcImgsVct,vector<int> &FcLblsVct)
{
	char ImgName[300];
	HANDLE hFind;
	int SubjectNumber=0;
	int TotalNmbSbj=0;
	LPCWSTR ImagesFolder;
	Mat FaceImage;
	string FaceName;                    // PersonLabel is the text version of
the PersonNumber with added zeros infront
	string A_Line,CurrentLine,ImagePath,ImgLabel,LineContent,FullPath;
	WIN32_FIND_DATA FindFileData;
	wstring wFullPath;


	ifstream ReadFoldersFile(PixFolder+"Folders.txt");

	while (getline(ReadFoldersFile,A_Line))
	{

		if(A_Line=="")
			continue;

		CurrentLine=A_Line;  // CurrentLine would be something like:
0001:Alaa
```

```
		stringstream LineContent(CurrentLine); // Copy the contents of
'CurrentLine' to 'LineContent'
		getline(LineContent,ImgLabel,':'); // ImgLabel would now be 0001
(or some other label)
		getline(LineContent,FaceName);		// CurrentLine will
contain the string "Alaa"
		//FaceName=CurrentLine;

		FullPath=PixFolder+FaceName+"\\*.png";	// CurrentLine would be
something like: 'Alaa'
		wFullPath=wstring(FullPath.begin(),FullPath.end());
		ImagesFolder=(LPCWSTR)wFullPath.c_str(); // Copy the contents of
'CurrentLine' to 'folder'

		if((hFind = FindFirstFile(ImagesFolder, &FindFileData)) !=
INVALID_HANDLE_VALUE)
		{
			do
			{
				for (int iii=0;iii<300;iii++)
				{
					ImgName[iii]=FindFileData.cFileName[iii];
					if(ImgName[iii]==0)
						break;
				}

				ImagePath=PixFolder+FaceName+"\\"+ImgName;

				FaceImage = imread(ImagePath,0);

				// This will check whether the face image is square
or rectangular
				if(FaceImage.cols==FaceImage.rows) // if it's
square, then it will crop it

	FaceImage=FixedFaceCrop(FaceImage,FaceImage.cols*10/12);

				// If the face image size is larger than 100x120
pixels, then it will be resized
				resize(FaceImage,FaceImage,Size(100,120));
				//equalizeHist(FaceImage,FaceImage);	// I
found that this negetively affects recognition
				FcImgsVct.push_back(FaceImage);

				SubjectNumber = atoi(ImgLabel.c_str());
				FcLblsVct.push_back(SubjectNumber);

				if(Names[SubjectNumber]=="")
					Names[SubjectNumber]=FaceName;


			}while(FindNextFile(hFind, &FindFileData));

			FindClose(hFind);
		}


		TotalNmbSbj++;
	}

	ReadFoldersFile.close();
```

```
	return TotalNmbSbj;
}



////////////////////////////////////////////////////////////////////////////
// This function shows a graphic window and waits (waitKey) for some time
////////////////////////////////////////////////////////////////////////////
int SnW(int w)
{
	int k=waitKey(w);

	if(k==27)
	{
		// 'Esc' key is pressed, so I'm stopping the robot motors and
exiting
		RobotMotors(100,100);
		return 0;
	}

	if(k==32)
	{
		// 'Space Bar' is pressed, so I'm stopping the robot motors and
waiting for another keypress
		RobotMotors(100,100);
		waitKey(0);
	}

	return 1;
}


////////////////////////////////////////////////////////////////////////////
// This function outputs the message as speech as well as display it on the
console window
////////////////////////////////////////////////////////////////////////////
bool Say(string saying)
{
	//cout << saying;

	wstring tmp = wstring(saying.begin(), saying.end());
	LPCWSTR sw = tmp.c_str();

	ISpVoice * pVoice = NULL;

    if (FAILED(::CoInitialize(NULL)))
        return FALSE;

    HRESULT hr = CoCreateInstance(CLSID_SpVoice, NULL, CLSCTX_ALL,
IID_ISpVoice, (void **)&pVoice);
    if( SUCCEEDED( hr ) )
    {
		hr=pVoice->SetRate(-2);                                  // from -
10 to 10; 0 is natural, -10 is the slowest
		hr=pVoice->SetVolume(750.0);
		//hr=pVoice->SetVoice();
		hr = pVoice->Speak(sw, 0, NULL);
        pVoice->Release();
        pVoice = NULL;
    }
```

```
    ::CoUninitialize();
    return TRUE;

}


//////////////////////////////////////////////////////////////////////////
// This function copies a part of an array to another array (both intger)
//////////////////////////////////////////////////////////////////////////
void CopyArray(int Src[], int From, int To, int Dst[])
{
       for(int iii=From;iii<=To;iii++)
              Dst[iii]=Src[iii];
}



/*************************************************************************

INTRODUCTION TO MAIN
This program does the following:
1- Conducts indoor monitoring using three cameras to cover 360 degrees (all
around it).
2- When a motion is detected, it will look for a face in the motion area.
3- When a face is found, a tracking camera is used to track the person and
capture a good image of the face.
4- When an image is acquired, the face familiarity is checked:
       3a- If the face is familiar, then no action is taken. Back to step 1.
       3b- If the face is not familiar, then two more images are taken.
              3b1- If two of the three images say the person is familiar, then
back to step 1.
              3b2- Otherwise raise alarm 1 and go to step 5.
5- Approach the person.
6- When the robot arrives at the person, it will ask him/her for a password.
7- Failure to approach the person in 30 seconds or entering a wrong password
results in raising the second alarm.


The robot may raise the following alarms:
ALARM 1: Raised when the robot cannot recognize the face (could be a stranger)
ALARM 2: Raised when the stranger has entered the wrong password
ALARM 3: Raised when the robot loses track of the stranger while trying to
approach him/her


Operation Notes:
The face recognition part of this program requires a text file by the name of
'Folders.txt' in a subfolder called 'images' (which is where the images are).
This folder should contain the labels of the subjects and their folder names.
An example is as follows:

              0001:Abdulla
              0002:Alaa
              0003:Mohammad

*************************************************************************/
```

```
// Function LMX3 ************************************************************
/****************************************************************************
This function detects motion by subtracting img2 from img1 and checking the
resulting difference.

If the two images have different sizes, then the larger one will be resized to
the smaller dimensions.

The two images can also have different number of channels (one gray and one
color). This is because both of them will be converted to gray (single
channel) before finding the difference between them.

The function returns an integer that represents the number of detected regions
where motion may have occurred.

An array containing the rectangles that represent the detected motion regions
is updated to contain the properties of all the detected motion regions.

This function also prints an error and returns a zero if the rows or columns
count of any  of the two images is zero.

This function allows the user to choose the threshold value of his choice.

PARAMETERS:
Rect: An array containing the rectangles that represent the motion regions
Mat    : First image of the two images that will be compared to detect motion.
Mat    : Second image of the two images that will be compared to detect
motion.
int    : The threshold value to be used
int    : If not zero, the function will return a larger motion area by 'Extra'
pixels on each side
int    : The separation (in Pixels) between adjacent motion detection regions.



RETURNS
This function returns a rectangle which contains the detected motion.
If no motion is detected, a 'Zero' (empty) rectangle is returned

****************************************************************************/

#include "Stdafx.h"                        // required by Visual Studio
#include <iostream>
#include "highgui\highgui.hpp"     // for VideoCapture & waitKey
#include <imgproc\imgproc.hpp>     // for blur and other image related stuff

using namespace cv;
using namespace std;


int LMX3(Rect MotionRegion[], Mat img1,Mat img2,int Threshold,int Extra,int
Separation)
{
//==================== DECLARATIONS ====================
//bool ImgShown=false;
       bool MwD=false;                     // Motion was detected
       float x_mag=img1.cols / 160;        // resizing ratio in the x-direction
       float y_mag=img1.rows / 120;        // resizing ratio in the y-direction
       int SepCounter=0;                   // Counts the number of pixels that
separates two motion areas
       int Region=0;                       // An index that points to the number
of the detected-motion area
```

```
	int P[160][2]={0,0};		// The Points matrix that holds the
coordinates of detected motion areas [x][y1,y2]
	Mat Diff;			// The difference image
	Rect Zero(0,0,0,0);		// A zero sized rectangle
	Scalar i;			// Pixel intensity

Mat Frame;

//==================== INITIALIZATIONS ====================

	// Reset all motion regions
	for(int iii=0;iii<50;iii++)
		MotionRegion[iii]=Zero;

	// Reset all the points matrix
	for(int iii=0;iii<160;iii++)
	{
		// These two lines fill the array with -1. This is used later to
distinguish filled from empty cells.
		P[iii][0]=-1;
		P[iii][1]=-1;
	}


//====================== PRECAUTIONS ======================

	// If the image has zero rows or zero columns, then there's no motion
to calculate
	if(img1.cols==0 || img1.rows==0 || img2.cols==0 || img2.rows==0)
	{
		cout << "\nImage 1 or Image 2 has zero rows or zero columns.";
		return 0;
	}

	// If the sizes of the two image do not match, then resize the larger
image to match the smaller one
	if(img1.cols != img2.cols || img1.rows != img2.rows)
	{
		cout << "\nThe dimensions of image 1 and image 2 is not equal.";
		if(img1.size > img2.size)
			resize(img1,img1,Size(img2.cols,img2.rows));
		else
			resize(img2,img2,Size(img1.cols,img1.rows));
	}



//==================== MAIN OPERATIONS ====================

	// Convert the images to grayscale and resize them to 160x120
	cvtColor( img1, img1, CV_BGR2GRAY );
	cvtColor( img2, img2, CV_BGR2GRAY );
	resize(img1,img1,Size(160,120));
	resize(img2,img2,Size(160,120));
	Diff.create(img1.size(),img1.type());

	// Subtract the images from each other to reveal the difference
	Diff=img2-img1;

	blur(Diff,Diff,Size(3,3));
```

```
	// Main body of the function: Finding the pixels of difference
	for (int xxx=0;xxx<Diff.cols-1;xxx++)
	{
		for (int yyy=0;yyy<Diff.rows-1;yyy++)
		{
			// Read the pixel
			i = Diff.at<uchar>(Point(xxx, yyy));
			if(i[0]>Threshold)
			{
				// Set a flag that Motion was Detected
				MwD=true;


				// Set Y[0] to the first detected motion pixel
				if(P[xxx][0]==-1)
					P[xxx][0]=yyy;

				// Set Y[1] to the furthest detected motion point
down this same column.
				P[xxx][1]=max(P[xxx][1],yyy);
			}
		}

	}

	// Checking to see if motion areas exist.
	if(MwD)		// if Motion was Detected
	{
		for(int xxx=0;xxx<Diff.cols-1;xxx++)
		{
			if(P[xxx][0]>-1)
			{
				// If the separation is larger than the set 'Sep
distance'
				// or if it's motion area 0 and the motion area's x
= 0 (still un-initialized)
				if(SepCounter>Separation || (Region==0 &&
MotionRegion[Region].x==0))
				{
					// Increment the Region counter if the
previous motion area's area was larger than zero
					if(MotionRegion[Region].area() > 0)
						Region++;

					// Set the x and y of the detected motion
region
					MotionRegion[Region].x=xxx;
					MotionRegion[Region].y=P[xxx][0];
				}


	MotionRegion[Region].x=min(MotionRegion[Region].x,xxx);		//
minimum x of this motion region

	MotionRegion[Region].y=min(MotionRegion[Region].y,P[xxx][0]); //
minimum y of this motion region
				MotionRegion[Region].width=xxx-
MotionRegion[Region].x;		// current x - previous x

				// The tallest height. This will be zero if only
one motion pixel was detected.
```

```
	MotionRegion[Region].height=max(MotionRegion[Region].height,P[xxx][1]-
MotionRegion[Region].y);

				// Reset the seperation counter
				SepCounter=0;
			}

			// If no pixel is found (no motion is detected in this
column), then increment the separation area counter.
			if(P[xxx][0]==-1)
				SepCounter++;

		}
	}

	Region++;

	for(int iii=0;iii<Region;iii++)
	{

		MotionRegion[iii].x=MotionRegion[iii].x*x_mag;
		MotionRegion[iii].y=MotionRegion[iii].y*y_mag;
		MotionRegion[iii].width=MotionRegion[iii].width*x_mag;
		MotionRegion[iii].height=MotionRegion[iii].height*y_mag;



	return Region;

}
```

```
// Function AimFHDcam5_2 ************************************************
/************************************************************************

IMPORTANT
For the security robot, I have decided to use this function with the Arduino
Uno as the Arduino Mega will be located at the base of the robot (near the
laptop) to be used for controlling the motors.


INTRODUCTION
This function aims the Full HD camera at a region specified by the calling
program (the location of the person closest to the camera) to capture his/her
face.


PARAMETERS:
Mat Frame     : The image of the frame containing the region to be aimed at.
Rect Face     : The region to be aimed at
int Fov       : The camera's field of view (in degrees, e.g. 30, 45, 90,
..etc.)
int FHDAngle  : An array containing the current horizontal and vertical servo
angle of the FHD camera.
       FHDAngle[0] is the horizontal angle
       FHDAngle[1] is the vertical angle
       FHDAngle[2] is the horizontal angle adjustment value if it exceeds a
certain value
       FHDAngle[3] is the vertical angle adjustment value if it exceeds a
certain value
       FHDAngle[4] is the horizontal angle value if it overshoots the limit
       FHDAngle[5] is the vertical angle value if it overshoots the limit



RETURNS
This function returns one of the following values in the array:
FHDAngle[0] The final horizontal angle that the servo is supposedly at (if the
servo fails to move for any reason, the program wouldn't know).
FHDAngle[1] The final vertical angle that the servo is supposedly at (if the
servo fails to move for any reason, the program wouldn't know).
FHDAngle[2] Either zero or the horizontal angle adjustment value if it
exceeded a certain value
FHDAngle[3] Either zero or the vertical angle adjustment value if it exceeded
a certain value
FHDAngle[4] Either zero or the horizontal angle value if it overshot the limit
FHDAngle[5] Either zero or the vertical angle value if it overshot the limit



NOTE
For the wide angle camera (Genius F100), the FoV=90 at resolution of 640x480
and FoV=120 at HD & Full HD resolutions.
For the Full HD camera, use FoV=60 for all modes.

************************************************************************/

#include "stdafx.h"        // required by Visual Studio

#include <Windows.h> // for organizing the different windows on the screen
#include "imgproc\imgproc.hpp"           // for cvtColor & equalizeHist
#include "highgui\highgui.hpp"
```

```
using namespace cv;


HANDLE SerialPort3(string Name,int Speed);              // This function opens
the requested serial port (int: Serial Port Number) and returns -1 if
unsuccessful.


void AimFHDcam5_2(Mat Frame, Rect Face, int FoV, int FHDAngle[6], float Inc)
{
//############################### DECLARATIONS ##############################
       bool Error=false;           // This flag is used to indicate an error
with the aiming angle of one of the servo's
       int W,H;                            // Half the width and height of the
frame image (half the number of pixels horizontally and vertically)
       int Cx,Cy;                          // Face's center point
       float HPosAdj,VPosAdj;      // Position adjustment in degrees
       float HDPP;                         // Horizontal Degree Per Pixel
(Camera's field of view / Screen width)
       float VDPP;                         // Vertical Degree Per Pixel
(Camera's field of view / Screen height)
       float CamFoV=FoV;           // This converts the FoV value to a floating
point

// Motor Control Related
       HANDLE FHDArduino;
       DWORD btsIO;
       char Header[2] = "!";       // The initial character that will be sent
to the serial port to initiate reception of the angle
       char Hrz[2] = "f";          // The first character that will be sent to
the serial port. The "f" means nothing
       char Vrt[2] = "s";          // The second character that will be sent to
the serial port. The "s" means nothing


//############################### PRECAUTION ################################

       if(Face.width==0)
       {
              printf("\nThis is the AimCamFHD 3.0 function.");
              printf("\nI have received a face size of zero, so nothing will
be done");
              //return *FHDAngle;
       }


//############################## INITIALIZATIONS ############################

       // Initialize the Arduino port to control the aiming servos
       FHDArduino=SerialPort3("30",9600); // define the port for the Arduino
board that controls the robot

       // Reset the last four cells of the FHDAngle array as they may be used
to hold irregular angle values
       for(int iii=2;iii<6;iii++)
              FHDAngle[iii]=0;

//############################ MAIN OPERATIONS ############################//

       // 1- Calculate the face's center coordinates
       Cx=Face.x+Face.width/2;
       Cy=Face.y+Face.height/2;
```

```
	//printf("\n\nMiddle point of the face is at location (%d,%d)",Cx,Cy);

//-------------------------------------------------------------------------

	// 2.0 Calculate whether the camera needs to turn left or right, up or
down. This is done by
	// finding the screen's midpoint, then comparing this with the location
of the face's center point.

	// 2.1 Find the screen's mid point
	W=Frame.cols/2;
	H=Frame.rows/2;

	// 2b- Find out how many degrees is equal to one pixel.
	HDPP=CamFoV/Frame.cols;			// This is done by deviding the
screen width by the camera's field of view
	VDPP=CamFoV/Frame.rows;			// and also the screen's height by
the camera's field of view
	HPosAdj=(W-Cx)*HDPP;		// (W-Center point) x Horizontal Degrees Per
Pixel
	VPosAdj=(Cy-H)*VDPP;		// (H-Center point) x Vertical Degrees Per
Pixel

//-------------------------------------------------------------------------

	// 3.0 Adjust the aiming angle if the camera is not at the default
(90,90) position
	if(FHDAngle[0]!=90)
		HPosAdj=HPosAdj-FHDAngle[0];	// Add the previous horizontal
angle to accurately aim the camera

	if(FHDAngle[1]!=90)
		VPosAdj=VPosAdj-FHDAngle[1];	// Add the previous vertical
angle to accurately aim the camera

//-------------------------------------------------------------------------

	// 4.0 Check for special angle and angle adjustment values
	// 4.1 Ignor minor adjustments that are less than 3 degrees
horizontally or 2 degrees vertically.
	if(HPosAdj<3 && HPosAdj>-3 && VPosAdj<2  && VPosAdj>-2)
		Error=true;

	// 4.2 If the angle adjustment is more than 60 degrees horizontally or
30 degrees vertically,
	// then return and inform the calling program that this is not
realistic.
	if(HPosAdj>60 || HPosAdj<-60)
	{
		Error=true;
		FHDAngle[2]=HPosAdj;
	}

	if(VPosAdj>30 || VPosAdj<-30)
	{
		Error=true;
		FHDAngle[3]=VPosAdj;
	}


//-------------------------------------------------------------------------
```

```
	// 5.0 If an error flag is raised, then return without moving the
servos
	if(Error)
	{
		CloseHandle(FHDArduino);
		return;
	}

//--------------------------------------------------------------------------

	// Increase (or decrease) the horizontal angle to accomodate the
subject's movement
	HPosAdj=HPosAdj+HPosAdj*(Inc/100);

	FHDAngle[0]=HPosAdj;
	FHDAngle[1]=VPosAdj;

//--------------------------------------------------------------------------

	// 5- Send the position adjustments to the servo motors
	// 5.1- Send the Header to denote the start of the motor control batch
	WriteFile(FHDArduino, Header, strlen(Header), &btsIO, NULL);

	// 5.2- Prepare and send the horizontal angle adjustment value
	Hrz[0]=100+HPosAdj;  // Since we can't send negetive numbers, we add 100
here and deduct 100 at the Arduino side
	WriteFile(FHDArduino, Hrz, strlen(Hrz), &btsIO, NULL);

	// 5.4- Prepare and send the vertical angle adjustment value
	Vrt[0]=100+VPosAdj;  // Also for the second value, we add 100 here and
deduct 100 at the Arduino side
	WriteFile(FHDArduino, Vrt, strlen(Vrt), &btsIO, NULL);


//printf("\nAdjusted the camera by %d degrees horizontally and %d degrees
vertically",HAngle,VAngle);

//--------------------------------------------------------------------------

	// 6.0 Close port and return to the calling program
	CloseHandle(FHDArduino);
	return;

//--------------------------------------------------------------------------
}


//WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
//MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM


// This function resets the Camera to the default (middle) position (90
degrees)
void CenterFHDcam(void)
{
//############################## DECLARATIONS ################################

	char Header[2] = "%";	// The initial character that will be sent
to the serial port to initiate reception of the angles
	char Hrz[2] = "f";		// The first character that will be sent to
the serial port. The "f" means nothing
```

```
	char Vrt[2] = "s";		// The second character that will be sent to
the serial port. The "s" means nothing
	DWORD btsIO;


// Motor Control Related
	HANDLE FHDArduino;


//############################## INITIALIZATIONS ##############################

	FHDArduino=SerialPort3("30",9600); // define the port for the Arduino
board that controls the robot

//############################ MAIN OPERATIONS ############################//

	// 1- Send the header
	WriteFile(FHDArduino, Header, strlen(Header), &btsIO, NULL);

	// 2- Send the default angles.
	Hrz[0]=90;		// Send horizontal angle 90 to return the
horizontal servo
	Vrt[0]=90;		// Send vertical angle 90 to return the vertical
servo
	WriteFile(FHDArduino, Hrz, strlen(Hrz), &btsIO, NULL);
	WriteFile(FHDArduino, Vrt, strlen(Vrt), &btsIO, NULL);

	CloseHandle(FHDArduino);
	return;
}
```

```
// Function Approach7wi ************************************************
/*********************************************************************

INTRODUCTION
This function controls the robot motors based on angle of the subject with
respect to the robot (face location in the frame image).

This version of the function provides a percentage of power to the motors
rather than the full power. This is useful when requiring to increase the
robot speed gradually.


PARAMETERS:
int CamAngle: The angle to which the robot must turn
int Percentage: The percentage of power to be used (100 = full power)


RETURNS
This function returns nothing.

*********************************************************************/


#include "stdafx.h"  // required by Visual Studio

#include <Windows.h> // for organizing the different windows on the screen
#include <string>    // for handling strings ((names and the like))
#include <highgui\highgui.hpp>

using namespace std;
using namespace cv;


HANDLE SerialPort3(string Name,int Speed);



//=====================================================================
//------------------------- START OF FUNCTION BODY --------------------------

void Approach7wi(int CamAngle, int Percentage, Mat FrmImg,int FrmCounter)
{
//################################ DECLARATIONS ##############################
       bool Extreme=false;  // Extreme right or left flag
       char Header[2] = "&";       // The starting sequence character that will
be sent to the Arduino instructing it to receive two bytes
       int LMotor[2];              // The first character that will be sent to
the Arduino (serial port). The "c" is just an arbitrary character
       int RMotor[2];              // The second character that will be
received from the serial port. The "d" is just an arbitrary character
       float Diff;                 // The difference between the current camera
angle and the center (90 degrees)
       float RMPL;                 // Right Motor Power Level
       float LMPL;                 // Left Motor Power Level
       DWORD btsIO;

// Motor Control Related
       HANDLE MotorArduino;



//############################## INITIALIZATIONS #############################
```

```
	printf("\nInside Approach-7 function");
	LMotor[1]=Header[1];
	RMotor[1]=Header[1];

	MotorArduino=SerialPort3("49",9600);	// define the port for the
Arduino board that controls the robot

//################################ PRECAUTIONS ################################

	if (CamAngle<10)
		CamAngle=10;

	if (CamAngle>170)
		CamAngle=170;

	if (Percentage>100)
		Percentage=100;

	if (Percentage<0)
		Percentage=0;

//############################ MAIN OPERATIONS ##############################//

	// 2- Calculate the face's location in the frame image, and prepare an
appropriate
	// drive command based on that in order to drive the robot's motors.


	// 2a- If the face is to the far right of the robot
	if(CamAngle<=40)
	{	// Then rotate clockwise
		LMPL=50;
		RMPL=-50;
		Extreme=true;

		RMotor[0]=RMPL;
		LMotor[0]=LMPL;
		printf("\nRotating clockwise");
	}



	// 2b- If the face is to the right of the robot, then turn right
	if(CamAngle<85 && CamAngle>40)
	{
		LMPL=100;
		Diff=(90-CamAngle)*3;
		RMPL=100-Diff;

		RMotor[0]=RMPL;
		LMotor[0]=LMPL;
		printf("\nTurning right");
	}



	// 2c- If the face is to the far left of the robot
	if(CamAngle>=130)
	{	// Then rotate counter-clockwise
		LMPL=-50;
		RMPL=50;
```

```
                Extreme=true;

                RMotor[0]=RMPL;
                LMotor[0]=LMPL;
                printf("\nRotating counter-clockwise");
        }



        // 2d- If the face is to the left of the robot, then turn left
        if(CamAngle>95 && CamAngle<130)
        {
                RMPL=100;
                Diff=(CamAngle-90)*3;
                LMPL=100-Diff;

                RMotor[0]=RMPL;
                LMotor[0]=LMPL;
                printf("\nTurning left");
        }

        // 2c- If the face is somewhat in the center of the frame, then drive
forward
        if(CamAngle>=85 && CamAngle<=95)
        {
                RMPL=100;
                LMPL=100;

                RMotor[0]=RMPL;
                LMotor[0]=LMPL;
                printf("\nGoing Forward.");
        }



////// 3- Send the command to the Arduino board to drive the robot
        // A- Send the character '&' to denote the start of the motor control
batch
        WriteFile(MotorArduino, Header, 1, &btsIO, NULL);

        // For extreme right or left, do not use power factor adjustments. i.e.
Do not use gradual increase in power
        if(Extreme)
        {

                // C- Send the left motor power level
                LMotor[0]=100+LMotor[0];          // Full Power=200 (100 will be
deducted at the Arduino side)
                WriteFile(MotorArduino, LMotor, 1, &btsIO, NULL);


                // B- Send the right motor power level
                RMotor[0]=100+RMotor[0];          // Full Power=200 (100 will be
deducted at the Arduino side)
                WriteFile(MotorArduino, RMotor, 1, &btsIO, NULL);

        }
        else   // Othersize (in not-extreme cases), use gradual power increment
(soft starts)
        {

                // C- Send the left motor power level
```

```
		if(LMotor[0]==0)
			LMotor[0]=100;			// Zero Power=100 (100 will be
deducted at the Arduino side)
		else
			LMotor[0]=100+((LMotor[0]*Percentage)/100);			//
Full Power=200 (100 will be deducted at the Arduino side)

		WriteFile(MotorArduino, LMotor, 1, &btsIO, NULL);


		// B- Send the right motor power level
		if(RMotor[0]==0)
			RMotor[0]=100;			// Zero Power=100 (100 will be
deducted at the Arduino side)
		else
			RMotor[0]=100+((RMotor[0]*Percentage)/100);			//
Full Power=200 (100 will be deducted at the Arduino side)

		WriteFile(MotorArduino, RMotor, 1, &btsIO, NULL);

	}

	putText(FrmImg,format("Angle=%d, RMotor=%d,
LMotor=%d",CamAngle,RMotor[0],LMotor[0]),Point(5,470),1,1.25,CV_RGB(255,0,0),1
);
	imwrite(format("Env %d.png",FrmCounter),FrmImg);

//	printf("\nRMotor[0]=%i",RMotor[0]);
//	printf("\nLMotor[0]=%i",LMotor[0]);
//	printf("\n");
	CloseHandle(MotorArduino);

}



/*************************************************************************

NOTES
Three bytes are sent to the Arduino:

	1- '&' : Sending an ambersand '&' will tell the Arduino that the next
two bytes are a the motor power levels.

	2- Left motor power level (100-200):

	3- Right motor power level (100-200):

200 means 100% power, while 100 means 0% power. This is because the Arduino
will deduct 100 from the received value.

This method (adding 100 here and deducting 100 at the Arduino side) is used in
order to be able to send zero as well as negative values of motor power.
Negative values represent power level in the reverse direction.


*************************************************************************/
```

```
// Function RobotMotors ***************************************************
/*************************************************************************

INTRODUCTION
This function moves the robot in a particular direction for a specific period
of time, or stops it.


PARAMETERS:
string XtrnCmnd : The External Command can be anyone of the following
commands:
                    "Left"        : Rotate the robot to the left.
                    "Right"       : Rotate the robot to the right.
                    "Forward"     : Moves the robot to forward.
                    "Stop"        : Stops the robot.
                    "Reverse"     : Moves the robot in the reverse direction.

int Period : The period in milliseconds that the command should be conducted.
The "Stop" and "Approach" commands will ignore the 'Period'.



RETURNS
This function returns nothing.


*************************************************************************/

#include "stdafx.h"         // required by Visual Studio

#include <Windows.h>        // for organizing the different windows on the
screen
#include "imgproc\imgproc.hpp"     // for cvtColor & equalizeHist
#include <string>           // for handling strings ((names and the like))
#include <iostream>


using namespace cv;
using namespace std;


HANDLE SerialPort3(string Name,int Speed);// This function opens the requested
serial port (int: Serial Port Number) and returns -1 if unsuccessful.



void RobotMotors(int LeftMotor,int RightMotor)
{
//############################## DECLARATIONS ################################
       char Header[2] = "&";       // The starting sequence character that will
be sent to the Arduino instructing it to receive two bytes
       int LMotor[2];       // The first character that will be sent to the
Arduino (serial port). The "c" is just an arbitrary character
       int RMotor[2];       // The second character that will be received from
the serial port. The "d" is just an arbitrary character
       DWORD btsIO;

// Motor Control Related
       HANDLE MotorArduino;


//############################ INITIALIZATIONS ###############################
```

```
//     printf("\nInside RobotMotors function");

       MotorArduino=SerialPort3("49",9600);      // define the port for the
Arduino board that controls the robot
//     printf("\n");                             // Start a new line for
printing the text of this function

       LMotor[1]=Header[1];
       RMotor[1]=Header[1];

//############################ MAIN OPERATIONS ##############################//

              if(RightMotor>0 && RightMotor<201)
                     RMotor[0]=RightMotor;
              else
                     RMotor[0]=100;

              if(LeftMotor>0 && LeftMotor<201)
                     LMotor[0]=LeftMotor;
              else
                     LMotor[0]=100;



       // A- Send the character '&' to denote the start of the motor control
batch
       WriteFile(MotorArduino, Header, strlen(Header), &btsIO, NULL);

//     printf("\nRMotor[0]=%i",RMotor[0]);
       WriteFile(MotorArduino, RMotor, 1, &btsIO, NULL);

//     printf("\nLMotor[0]=%i",LMotor[0]);
       WriteFile(MotorArduino, LMotor, 1, &btsIO, NULL);

       CloseHandle(MotorArduino);

}


/*****************************************************************************

This program works as follows:
       1- It calculates the center of the face.
       2- It finds the face's location in the frame image, and prepares an
appropriate drive command based on that.
              The face location is determined based on the following frame
portions (45%,10%,45%) -> (left, center or right)
              The center part is so narrow, because the camera used is a
fisheye camera and the center part is quite narrow in it.
       3- It sends a signal to the Arduino board to drive one or both motors
based on the face's location:
              3a- If the face is to the right by more than 10% of the frame's
width, then only the left motor will be driven.
              3b- If the face is to the left by more than 10% of the frame's
width, then only the right motor will be driven.
              3c- Otherwise both motors will be driven.

       4- The program in the Arduino ultimately decides whether to move the
right, left or both motors depending on the obstacles it faces.
```

```
ADDITION
This version may or may not prints whatever messages the Arduino sends it
using the 'ArdPrint' function


DIFFERENCES
This version is different to 'Approach' version 5 by sending two bytes to the
Arduino (after the ambersand) rather than one.


NOTES
Three bytes are sent to the Arduino (  & n1 n2  ):

       1- Sending an ambersand '&' will tell the Arduino that the next two
bytes are the command and period.

       2- N1: The movement command:
              100 = Drive = Go forward
              103 = turn right (clockwise) for a specific period of time
              106 = turn left (couterclockwise) for a specific period of time
              109 = Stop
              112 = Go reverse (no obstacle avoidance here)

       3- N2: Is the time length (in milliseconds) that the command should be
carried out for:
              Example: 500 : this means half a second.


       4-     I tried using the following code to read and print text from the
Arduino, but it didn't work well.
              Sometimes it works, when there's something already at the
receive buffer, but if there isn't then it will add the same character to the
string many times (because it is reading the same character over and over
again). Here is the code:

       do
       {
              ReadFile(Arduino, Text, strlen(Text), &btsIO, NULL);   // Read
the data
              Received+=Text;
              if(Text[0]==13)
                     break;
       }while(Text[0]>31 && Text[0]<127);

       // 2- Check whether the byte is a '{'
       if(Received.length()>0)     // if the byte recieved was not '{' then
return
              cout << endl << Received << endl;

       return;


       5- I tried the code currently mentioned (using the '{' & '}' , but
still it didn't work. It seems that the PC is reading the serial port too fast
for the Arduino's sending.

*******************************************************************************/
```

```
// Function SerialPort3 ***************************************************
/**************************************************************************
INTRODUCTION
This function opens the requested serial port (int: Serial Port Number).

PARAMETERS:
string Port   : The port 'handle' of the Arduino (or any other) board that
controls the servos.

RETURNS
If successful, this function returns a 'HANDLE' which can be used to send data
to the device attached to the serial port.
If unsuccessful, this function returns 'INVALID_HANDLE_VALUE' or -1 ((error)).

**************************************************************************/

#include <stdafx.h>

#include <string>    // for handling strings ((names and the like))
#include <Windows.h> // for organizing the different windows on the Firstreen
as well as for serial port communication with the Arduino board

using namespace std;

HANDLE SerialPort3(string Port, int bps)
{
       string sPort="\\\\.\\COM"+Port;          // convert Port to a string
       wstring wPort=wstring(sPort.begin(),sPort.end());      // convert sPort
to a wide string
       LPCWSTR lPort= (LPCWSTR)wPort.c_str();    // convert wPort to an LPCWSTR
type string to use it in getting the handle ( complicated isn't it :) )

       // Setup serial port connection and needed variables. Port 29 is the
current Arduino port. Double check the port number
       HANDLE hSerial = CreateFile(lPort, GENERIC_READ | GENERIC_WRITE, 0, 0,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);

       if (hSerial !=INVALID_HANDLE_VALUE)
       {
    //    printf("\n\nPort opened! \n");

              DCB dcbSerialParams;
              GetCommState(hSerial,&dcbSerialParams);

              dcbSerialParams.BaudRate = bps;

              dcbSerialParams.ByteSize = 8;
              dcbSerialParams.Parity = NOPARITY;
              dcbSerialParams.StopBits = ONESTOPBIT;

              SetCommState(hSerial, &dcbSerialParams);
       }
       else
       {
              if (GetLastError() == ERROR_FILE_NOT_FOUND)
                     printf("\n\nSerial port doesn't exist! \n");

              printf("\n\nError while setting up serial port %s
\n",Port.c_str());
       }

return hSerial;      }
```

```
// Function AimWAcam3 ******************************************************
/*****************************************************************************

IMPORTANT
For the security robot, I have decided to use this function with the Arduino
Uno as the Arduino Mega will be located at the base of the robot (near the
laptop) to be used for controlling the motors.


INTRODUCTION
This function aims the Full HD camera at a region specified by the calling
program (the location of the person closest to the camera) to capture his/her
face.


PARAMETERS:
Mat Frame     : The image of the frame containing the region to be aimed at.
Rect Face     : The region to be aimed at
int Fov       : The camera's field of view (in degrees, e.g. 30, 45, 90,
..etc.)
int FHDAngle  : An array containing the current horizontal and vertical servo
angle of the FHD camera.
         FHDAngle[0] is the horizontal angle
         FHDAngle[1] is the vertical angle
         FHDAngle[2] is the horizontal angle adjustment value if it exceeds a
certain value
         FHDAngle[3] is the vertical angle adjustment value if it exceeds a
certain value
         FHDAngle[4] is the horizontal angle value if it overshoots the limit
         FHDAngle[5] is the vertical angle value if it overshoots the limit


RETURNS
This function returns one of the following values in the array:
       FHDAngle[0] The final horizontal angle that the servo is supposedly at
(if the servo fails to move for any reason, the program wouldn't know).
       FHDAngle[1] The final vertical angle that the servo is supposedly at
(if the servo fails to move for any reason, the program wouldn't know).
       FHDAngle[2] Either zero or the horizontal angle adjustment value if it
exceeded a certain value
       FHDAngle[3] Either zero or the vertical angle adjustment value if it
exceeded a certain value
       FHDAngle[4] Either zero or the horizontal angle value if it overshot
the limit
       FHDAngle[5] Either zero or the vertical angle value if it overshot the
limit


NOTE
For the wide angle camera (Genius F100), the FoV=90 at resolution of 640x480
and FoV=120 at HD & Full HD resolutions.  For the Full HD camera, use FoV=60
for all modes.


*****************************************************************************/

#include "stdafx.h"         // required by Visual Studio

#include <Windows.h> // for organizing the different windows on the screen
#include "imgproc\imgproc.hpp"            // for cvtColor & equalizeHist
#include "highgui\highgui.hpp"
```

```
using namespace cv;


HANDLE SerialPort3(string Name,int Speed);             // This function opens
the requested serial port (int: Serial Port Number) and returns -1 if
unsuccessful.


int AimWAcam3(Mat Frame, Rect Face, int FoV, int PrvAng)
{
//############################## DECLARATIONS ############################
       bool Error=false;           // This flag is used to indicate an error
with the aiming angle of one of the servo's
       int W,H;                            // Half the width and height of the
frame image (half the number of pixels horizontally and vertically)
       int Cx,Cy;                          // Face's center point
       int HPosAdj,VPosAdj; // Position adjustment in degrees
       float HDPP;                         // Horizontal Degree Per Pixel
(Camera's field of view / Screen width)
       float VDPP;                         // Vertical Degree Per Pixel
(Camera's field of view / Screen height)
       float CamFoV=FoV;           // This converts the FoV value to a floating
point

// Motor Control Related
       HANDLE FHDArduino;
       DWORD btsIO;
       char Header[2] = "#";       // The initial character that will be sent
to the serial port to initiate reception of the angle
       char Hrz[2] = "f";          // The first character that will be sent to
the serial port. The "f" means nothing
       char Vrt[2] = "s";          // The second character that will be sent to
the serial port. The "s" means nothing


//############################### PRECAUTION ##############################

       if(Face.width==0)
       {
              printf("\nThis is the AimWACam 3.0 function.");
              printf("\nI have received a face size of zero, so nothing will
be done");
              return PrvAng;
       }


//############################## INITIALIZATIONS ##########################

       // Initialize the Arduino port to control the aiming servos
       FHDArduino=SerialPort3("30",9600); // define the port for the Arduino
board that controls the robot

//     VDPP=CamFoV/Frame.rows;             // and also the screen's height by
the camera's field of view

//############################## MAIN OPERATIONS ##########################//

       // 1- Calculate the face's center coordinates
       Cx=Face.x+Face.width/2;
       Cy=Face.y+Face.height/2;
       //printf("\n\nMiddle point of the face is at location (%d,%d)",Cx,Cy);
```

```
//---------------------------------------------------------------------------

	// 2.0 Calculate whether the camera needs to turn left or right, up or
down. This is done by
	// finding the screen's midpoint, then comparing this with the location
of the face's center point.

	// 2.1 Find the screen's mid point
	W=Frame.cols/2;
	H=Frame.rows/2;

	// 2.2 Find out how many degrees is equal to one pixel.
	HDPP=CamFoV/Frame.cols;		// This is done by deviding the
screen width by the camera's field of view
	VDPP=CamFoV/Frame.rows;		// and also the screen's height by
the camera's field of view
	HPosAdj=(W-Cx)*HDPP;	// (W-Center point) x Horizontal Degrees Per
Pixel
	VPosAdj=(Cy-H)*VDPP;	// (H-Center point) x Vertical Degrees Per
Pixel

//---------------------------------------------------------------------------

	// 2.3 Ignor minor adjustments that are less than 3 degrees
horizontally or 2 degrees vertically.
	if(HPosAdj<3 && HPosAdj>-3 && VPosAdj<2  && VPosAdj>-2)
	{
		CloseHandle(FHDArduino);
		return PrvAng;
	}

	// If the angle adjustment is more than 60 degrees horizontally or 30
degrees vertically,
	// then return, because this is not realistic.
	if(HPosAdj>60 || HPosAdj<-60)
	{
		CloseHandle(FHDArduino);
		return -1;
	}

	if(VPosAdj>30 || VPosAdj<-30)
	{
		CloseHandle(FHDArduino);
		return -1;
	}


	// 3.0- Send the position adjustments to the servo motors
	// 3.1- Prepare the angle values
	Hrz[0]=100+HPosAdj;  // Since we can't send negetive numbers, we add 100
here and deduct 100 at the Arduino side
	Vrt[0]=100+VPosAdj;  // Also for the second value, add 100 here and
deduct 100 at the Arduino side




//---------------------------------------------------------------------------
```

```
        // 4.0- Send the position adjustments to the servo motors
        // 4.1- Send the Header to denote the start of the motor control batch
        WriteFile(FHDArduino, Header, strlen(Header), &btsIO, NULL);

        // 4.2- Prepare and send the horizontal angle adjustment value
        Hrz[0]=100+HPosAdj;  // Since we can't send negetive numbers, we add 100
here and deduct 100 at the Arduino side
        WriteFile(FHDArduino, Hrz, strlen(Hrz), &btsIO, NULL);

        // 4.3- Prepare and send the vertical angle adjustment value
        Vrt[0]=100+VPosAdj;  // Also for the second value, we add 100 here and
deduct 100 at the Arduino side
        WriteFile(FHDArduino, Vrt, strlen(Vrt), &btsIO, NULL);


//printf("\nAdjusted the camera by %d degrees horizontally and %d degrees
vertically",HAngle,VAngle);

//----------------------------------------------------------------------------

        // 5.0 Close port and return to the calling program
        CloseHandle(FHDArduino);
        return HPosAdj+PrvAng;

//----------------------------------------------------------------------------
}



//WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
//MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM



// This function resets the Camera to the default (middle) position (90
degrees)
int CenterWAcam(void)
{
//############################### DECLARATIONS ##############################

        char Header[2] = "$";       // The initial character that will be sent
to the serial port to initiate reception of the angles
        char Hrz[2] = "f";          // The first character that will be sent to
the serial port. The "f" means nothing
        char Vrt[2] = "s";          // The second character that will be sent to
the serial port. The "s" means nothing
        DWORD btsIO;


// Motor Control Related
        HANDLE FHDArduino;


//############################# INITIALIZATIONS #############################

        FHDArduino=SerialPort3("30",9600); // define the port for the Arduino
board that controls the robot

//############################# MAIN OPERATIONS ############################//

        // 1- Send the header
        WriteFile(FHDArduino, Header, strlen(Header), &btsIO, NULL);
```

```
	// 2- Send the default angles.
	Hrz[0]=90;		// Send horizontal angle 90 to return the
horizontal servo
	WriteFile(FHDArduino, Hrz, strlen(Hrz), &btsIO, NULL);

	Vrt[0]=90;	// Also for the second value, add 100 here and deduct 100
at the Arduino side
	WriteFile(FHDArduino, Vrt, strlen(Vrt), &btsIO, NULL);

	CloseHandle(FHDArduino);

	return 90;
}
```

```
// Function FindLrgstFace3 ************************************************
/**************************************************************************
This function finds the largest face in the image passed to it by the calling
program using the haar cascades (Viola Jones) method.


PARAMETERS:
	CascadeClassifier: The Detector to be used (face detector, body
detector ... etc.)
	Mat Image     : The image to be searched for faces.
	int FaceSize: The face size to look for (searches for faces of this
size or larger)
	float SkinCheck       : The percentage of Skin-Color-Content (0 will
nullify this filter as all image have a skin color content of 0 or more).
	float SharpnessCheck: The percentage of sharpness to check for (100
will nullify this filter as all images have a sharpness of less than 100%)


RETURNS
This function returns a Rectangle containing the largest face.
If no face is detected, a 'Zero' rectangle is returned.

**************************************************************************/

#include "Stdafx.h"               // required by Visual Studio
#include <iostream>
#include <ctime>
#include "highgui\highgui.hpp"    // for VideoCapture & waitKey
#include <imgproc\imgproc.hpp>    // for blur and other image related stuff
#include "objdetect\objdetect.hpp" // for cascade classifiers


using namespace cv;
using namespace std;


float SkinColorPcnt(Mat FaceImage);                                    //
Returns the percentage of skin color in the image
float HowSharp(Mat FaceImage);                                         //
Returns the sharpness of an image
Mat FixedFaceCrop(Mat FaceImg,int WidthToCropTo);      // Crops the face from
one or two sides



Rect FindLrgstFace3(CascadeClassifier Detector, Mat Image, int FaceSize, float
ReqSkinClrCnt, float ReqSharpness, bool Save)
{

////// DECLARATIONS //////
	int LFSz=0;                        // The Largest Face Size is the size
of the larges face found in the frame
	int time=clock();
	time=time/1000;
	float Sharpness=0;          // A Temporary measure of face image
sharpness (<5 is very blurry)
	float Skin=0;               // Skin color percentage of the face image
	Mat UncroppedFace;          // The image of the captured face
	Mat CroppedFace;
	Mat OriginalFace;           // The face with the original size
	Mat tmpImg;
	Rect FaceRegion;
```

```
	Rect RoLF(0,0,0,0);		// Region of the largest face.
	Rect Zero(0,0,0,0);
	vector <Rect> DetectedFaces;		// Detected Faces

//############################## INITIALIZATIONS ##############################

	Image.copyTo(tmpImg);

//############################## MAIN OPERATIONS ############################//

	// Search for faces inside the image
	Detector.detectMultiScale(Image,DetectedFaces,1.2,2,0|CV_HAAR_SCALE_IMA
GE,Size(FaceSize,FaceSize));


	//////// VERIFYING DETECTED FACES & LOOK FOR THE LARGEST FACE
	for (int jjj=0;jjj<DetectedFaces.size();jjj++)
	{
		FaceRegion = DetectedFaces[jjj];

		// Extract the face image from the frame image
		Image(FaceRegion).copyTo(OriginalFace);

		// Firstly, we will enlarge and crop the face to 100x120 to
calculate its sharpness
		resize(OriginalFace,UncroppedFace,Size(120,120));	// First
resize the original face to 120x120
		CroppedFace=FixedFaceCrop(UncroppedFace,100);		// then
crop it to 100x120 and put it in 'CroppedFace'

		// Secondly, we calculate the Skin Color Content and Sharpness
if required to do so
		if(ReqSkinClrCnt>0)
			Skin=SkinColorPcnt(CroppedFace);

		if(ReqSharpness>0)
		Sharpness=HowSharp(CroppedFace);

		// Thirdly, let's write the values found above the face region
		rectangle(tmpImg,FaceRegion,CV_RGB(255,0,0),1);
		putText(tmpImg,format("Skin=%2.2f,
Sharpness=%2.2f",Skin,Sharpness),Point(FaceRegion.x,FaceRegion.y-
5),1,1,CV_RGB(255,0,0),1);


		// If Skin Color Content is required and the SCC of the image
was found to be less than the Required SCC
		if(Skin<ReqSkinClrCnt)
			continue;	// then ignore this face

		// If Sharpness is required and the Sharpness of the image was
found to be zero
		// or larger than the Required Sharpness
					//if(ReqSharpness!=0 && (Sharpness==0 ||
Sharpness > ReqSharpness))  // this one didn't work quite well
		if(ReqSharpness!=0 && Sharpness > ReqSharpness )
			continue;	// then ignore this face


		//if(EyesCheck)
		//{
```

```
                //
        SmallEyeCascade.detectMultiScale(CroppedFace,DetectedEyes,1.1,1,0|CV_HA
AR_SCALE_IMAGE,Size(CroppedFace.cols/10,CroppedFace.rows/10));
                //
                //      if(!DetectedEyes.size())
                //
        BigEyeCascade.detectMultiScale(CroppedFace,DetectedEyes,1.1,1,0|CV_HAAR
_SCALE_IMAGE,Size(CroppedFace.cols/10,CroppedFace.rows/10));

                //      if(!DetectedEyes.size())
                //
        OtherEyeCascade.detectMultiScale(CroppedFace,DetectedEyes,1.1,1,0|CV_HA
AR_SCALE_IMAGE,Size(CroppedFace.cols/10,CroppedFace.rows/10));
                //}


                // Now, if the face is larger than the previous (all others)
then hold its information for later use
                if(LFSz<OriginalFace.cols)
                {
                        LFSz=OriginalFace.cols;
                        RoLF=FaceRegion;
                }


        }

        if(LFSz==0)
                return Zero;
        else
        {
                if(Save)
                {
                        imwrite(format("FindLrgstFace-The Received Image
%d.png",time),Image);
                        imwrite(format("FindLrgstFace-The Faces
%d.png",time),tmpImg);
                }
                return RoLF;
        }

}
```

```
// Function FindLrgstBody ************************************************
/*************************************************************************
This function finds the largest body in the image passed to it by the calling
program using the haar cascades (Viola Jones) method.


PARAMETERS:
	Mat Image		: The image to be searched for bodies.
	CascadeClassifier	: The Detector to be used (A number of different
body detectors)



RETURNS
This function returns a Rectangle containing the largest face.
If no face is detected, a 'Zero' rectangle is returned.

*************************************************************************/

#include "Stdafx.h"			// required by Visual Studio
//#include <iostream>
#include "highgui\highgui.hpp"	// for VideoCapture & waitKey
#include <imgproc\imgproc.hpp>	// for blur and other image related stuff
#include "objdetect\objdetect.hpp" // for cascade classifiers


using namespace cv;
using namespace std;



Rect FindLrgstBody(Mat Image, CascadeClassifier BodyCascade1,CascadeClassifier
BodyCascade2,CascadeClassifier BodyCascade3, int BdySz)
{

////// DECLARATIONS //////
	int LBSz;			// The Largest Body Size is the size
of the larges face found in the frame
	Mat OriginalFace;		// The face with the original size
	Rect FaceRegion;
	Rect RoLB(0,0,0,0);		// Region of the largest body.
	Rect Zero(0,0,0,0);
	vector <Rect> DetectedFaces;		// Detected Faces


//############################## MAIN OPERATIONS ############################//

	// Search for faces inside the image
	BodyCascade1.detectMultiScale(Image,DetectedFaces,1.2,2,0|CV_HAAR_SCALE
_IMAGE,Size(BdySz,BdySz));

	//////// VERIFYING DETECTED BODIES & LOOK FOR THE LARGEST BODY
	LBSz=0;
	for (int jjj=0;jjj<DetectedFaces.size();jjj++)
	{
		FaceRegion = DetectedFaces[jjj];

		// Extract the face image from the frame image
		Image(FaceRegion).copyTo(OriginalFace);
```

```
                // Now, if the face is larger than the previous (all others)
then hold its information for later use
                if(LBSz<OriginalFace.cols)
                {
                        RoLB=FaceRegion;
                        LBSz=OriginalFace.cols;
                }

        }


        if(LBSz==0)
                return Zero;
        else
                return RoLB;

}
```

```
// Function CheckFrame ***************************************************
/*************************************************************************
This function checks the frame image to see whether it has width and height or
not. This is because the frame captured from a camera may sometimes be
corrupted and contain only the width or the height.


PARAMETERS:
	VideoCapture Cam: The camera to obtain frames from if the current one
is corrupted.
	Mat Frame	: The currently captured frame image to be checked.
	int Iterations	: The number of times the camera should be queried
for a non-corrupt frame image.


RETURNS
This function returns a Rectangle containing the largest face.
If no face is detected, a 'Zero' rectangle is returned.

*************************************************************************/

#include "Stdafx.h"			// required by Visual Studio
//#include <iostream>
//#include <ctime>
#include "highgui\highgui.hpp"	// for VideoCapture & waitKey
//#include <imgproc\imgproc.hpp>	// for blur and other image related stuff
//#include "objdetect\objdetect.hpp"	// for cascade classifiers


using namespace cv;
//using namespace std;



bool CheckFrame(VideoCapture Camera, Mat Frame, int times)
{

	if(!Frame.cols || !Frame.rows)
	{
		int counter=0;

		do
		{
			Camera >> Frame;		// Maybe the FHD cam number is
'4', verify it.
			printf("\nNo rows or no columns from
CurrentFHDFrame=%i",counter);
			counter++;
			if (counter>5)
				break;
		}while(!Frame.cols || !Frame.rows);

		if (Frame.cols && Frame.rows)
		{
			printf("\n\nThe Camera is now OK ...");
			return true;
		}
		else
		{
			printf("\n\nUnable to get a proper feed from the camera,
skipping ...");
			return false;
```

```
                }
        }


        return true;
}
```

```
// Function Print ******************************************************
#include "stdafx.h"         // required by Visual Studio
#include "highgui\highgui.hpp"


using namespace cv;


void Print(Mat Image,int Line,String Text)
{
        if(Line==1)
                Image=Scalar::all(0);

        putText(Image,Text,Point(5,Line*20),1,1.5,Scalar(255),2);
        imshow("Status",Image);
        waitKey(1);
}
```

```
// Function Show2 ************************************************************
/*****************************************************************************

This function displays a graphic window containing one or two rectangles


	PARAMETERS:
	Mat Image	: The frame image to be displayed.
	Rect Rect1	: The first rectangle to be drawn on the frame image.
	Rect Rect2	: The second rectangle to be drawn on the frame image.
	int w		: The waiting period in milliseconds



RETURNS
This function returns nothing.

*****************************************************************************/

#include "Stdafx.h"                         // required by Visual Studio
#include "highgui\highgui.hpp"     // for VideoCapture & waitKey
#include <imgproc\imgproc.hpp>     // for blur and other image related stuff


using namespace cv;

int SnW(int Wait_Tim_Millisec);    // Show (graphic window) and wait (waitKey)
for some time



//============================== VERSION 1 ================================//

// Version 1: Without title and without resizing
void Show2(Mat Image,string Title, Rect Rect1, Rect Rect2, int w, bool
Destroy)
{
	Mat Frame;

	// Make a temporary copy of the frame image so we don't ruin it by
drawing the rectangles
	Image.copyTo(Frame);

	//putText(tmpFrame,"Face",Point(WALFRg.x,WALFRg.y-
10),1,2,CV_RGB(255,0,0),2);
	rectangle(Frame,Rect1,CV_RGB(255,0,0),3);

	//putText(tmpFrame,"Tracking Region",Point(WAFTRg.x,WAFTRg.y-
10),1,2,CV_RGB(255,0,255),2);
	rectangle(Frame,Rect2,CV_RGB(255,0,255),3);

	imshow(Title,Frame);

	if(!SnW(w)) return;

	if(Destroy)
	destroyWindow(Title);

}
```

```
//============================== VERSION 2 ==============================//

// Version 2: with title and without resizing
void Show2(Mat Image,string Title,int x, int y, Rect Rect1, Rect Rect2, int w,
bool Destroy)
{

	Mat Frame;

	// Make a temporary copy of the frame image so we don't ruin it by
drawing the rectangles
	Image.copyTo(Frame);

	//putText(tmpFrame,"Face",Point(WALFRg.x,WALFRg.y-
10),1,2,CV_RGB(255,0,0),2);
	rectangle(Frame,Rect1,CV_RGB(255,0,0),3);

	//putText(tmpFrame,"Tracking Region",Point(WAFTRg.x,WAFTRg.y-
10),1,2,CV_RGB(255,0,255),2);
	rectangle(Frame,Rect2,CV_RGB(255,0,255),3);

	imshow(Title,Frame);
	moveWindow(Title,x,y);

	if(!SnW(w)) return;

	if(Destroy)
	destroyWindow(Title);

}

//============================== VERSION 3 ==============================//

// Version 3: With title and resizing
void Show2(Mat Image,string Title,int x, int y, Rect Rect1, Rect Rect2, int w,
bool Destroy,int Rsz_W, int Rsz_H)
{
	Mat Frame;

	// Make a temporary copy of the frame image so we don't ruin it by
drawing the rectangles
	Image.copyTo(Frame);

	//putText(tmpFrame,"Face",Point(WALFRg.x,WALFRg.y-
10),1,2,CV_RGB(255,0,0),2);
	rectangle(Frame,Rect1,CV_RGB(255,0,0),3);

	//putText(tmpFrame,"Tracking Region",Point(WAFTRg.x,WAFTRg.y-
10),1,2,CV_RGB(255,0,255),2);
	rectangle(Frame,Rect2,CV_RGB(255,0,255),3);

	// Resize the window before displaying it
	resize(Frame,Frame,Size(Rsz_W,Rsz_H));

	imshow(Title,Frame);
	moveWindow(Title,x,y);

	if(!SnW(w)) return;

	if(Destroy)   destroyWindow(Title);
}
```

```
// Function ReadDIP ***********************************************************
/*****************************************************************************

IMPORTANT
For the correct operation of this program, a DIP switch must be present and
connected to the Arduino board that will be reading it.

INTRODUCTION
This function reads the DIP switch that is connected to the Arduino board.

PARAMETERS:
None.

RETURNS
This function returns an integer representing the decimal equivalent of the
binary value which is represented by the DIP switch keys.

*****************************************************************************/


#include "stdafx.h"  // required by Visual Studio
#include <Windows.h> // for organizing the different windows on the screen
#include <string>

using namespace std;


HANDLE SerialPort3(string Name,int Speed);// This function opens the requested
serial port (int: Serial Port Number) and returns -1 if unsuccessful.


int ReadDIP(void)
{
//################################ DECLARATIONS ##############################

// Arduino Related
	HANDLE DIPArduino;
	DWORD btsIO;
	char Dec[2] = "d";          // The decimal value that is sent by the
Arduino. The 'd' means nothing


//################################ INITIALIZATIONS ###########################

	// Initialize the Arduino port to control the aiming servos
	DIPArduino=SerialPort3("29",9600); // define the port for the Arduino
board that controls the robot

//############################## MAIN OPERATIONS ############################//

	ReadFile(DIPArduino, Dec, strlen(Dec), &btsIO, NULL);
	ReadFile(DIPArduino, Dec, strlen(Dec), &btsIO, NULL);
	ReadFile(DIPArduino, Dec, strlen(Dec), &btsIO, NULL);
	ReadFile(DIPArduino, Dec, strlen(Dec), &btsIO, NULL);


	// Close port and return to the calling program
	CloseHandle(DIPArduino);

	return Dec[0];
}
```