Integration of Image Processing Algorithm and Path Planning for Search and Rescue Robot

by

Wong Chun Yew
19984

Dissertation submitted in partial fulfilment of
the requirements for the
Bachelor of Engineering (Hons)
(Electrical and Electronics Engineering)

JANUARY 2017

Universiti Teknologi PETRONAS,
32610, Bandar Seri Iskandar,
Perak Darul Ridzuan.

CERTIFICATION OF APPROVAL

Integration of Image Processing Algorithm and Path Planning for Search and Rescue Robot

by

Wong Chun Yew
19984

A project dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
BACHELOR OF ENGINEERING (Hons)
(ELECTRICAL & ELECTRONICS ENGINEERING)

Approved by,

_____

(AP Dr. Tun Zainal Azni bin Zulkifli)

UNIVERSITI TEKNOLOGI PETRONAS
BANDAR SERI ISKANDAR, PERAK
January 2017

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

_____

WONG CHUN YEW

ABSTRACT

The focus of this project was to explore algorithms and techniques used in motion detection, object recognition and facial recognition, path finding as well as obstacle avoidance. To apply these algorithms, OpenCV was used. In terms of hardware, Raspberry Pi were used to perform image processing and robot movement. To perform image processing various OpenCV commands, such as cv2.GreyScale followed by cv2.GaussianBlur and cv2.DetectContour were combined. Path finding and obstacle avoidance were done by integrating an ultrasonic sensor into the system. Path finding is done by utilizing the coordinates of the bounding box. As a result, the robot turned around, 90 degree each time, to have a view of each of the four directions, in search of the target. Once motion was detected, the robot would stop at that direction and approach until the ultrasonic detected something. The robot would then run a scan on the target using facial recognition to determine whether it is human. If the result turned out to be negative, the robot would move towards its right side, and began scanning for motion again. In a nutshell, with the integration of image processing, robots had certainly increased their efficiency as well as accuracy in carrying out designated tasks. With the integration of image processing, the robot was set to perform search and rescue missions with higher reliability and accuracy.

ACKNOWLEDGEMENTS

I would like to express my great appreciation for those who had provided assistance and advices to me throughout the completion of this dissertation. First of all, I would like to express my deepest gratitude and appreciation to my Final Year Project supervisor, AP Dr. Tun Zainal Azni bin Zulkifli for his continuous guidance throughout the process. I have also embedded Dr.Tun's advice and comments into this project.

Besides, I would like to acknowledge the effort of Final Year Project Coordinator for Electrical & Electronics Engineering Department, Dr. Norashikin bt Yahya for her constant coordination throughout the entire course and provided us with proper guidelines in completing this dissertation.

Last but not least, my utmost appreciation goes to my family and friends who have been continuously giving support and motivation throughout my final year project journey.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Background Study

In 1941 and 1942, the Three Laws of Robotics were created by Isaac Asimov. Following these laws, George Devol, in 1954, came up with the first programmable robot. From this point onwards, technology in robotic industry has escalated to what we see in modern days.

To completely take human out of the equation in search and rescue missions, the robot, first of all, has to be autonomous. Autonomous, a word describing the ability and freedom to act independently, is used with associations with robot (Bourdieu, 2001). An autonomous robot implies its capability to carry out specific tasks without the guidance of a person. Based on algorithms, an autonomous robot is capable of making decisions, as long as human permits. In order to acquire information, an autonomous robot has to be equipped with sensors, such as cameras, ultrasonic sensors, touch sensors, light sensors, etc., just like how humans receive stimulations from the surrounding through our sensory organ.

The most reliable sensory organ of humans is our eyes. For that reason, it could be of great advantage if robots could share the same vision. With the surface of image processing, which gave rise to computer vision, has provided autonomous robots with a whole new dimension of sensing ability. Computer vision, a field which allows images or videos to be understood by the computer, just like the humans do (Szeliski, 2011). Methods for extracting, processing, analyzing and gaining intellect of digital images to provide sufficient knowledge and information in making decisions are part of the computer vision field. The word understanding in this context, gives the meaning of changing pixels and data into meaningful information, instead of just numbers, to aid in the process of delivering desired output.

Scientifically, computer vision is concerned with the theory behind artificial systems that extract information from images. Videos, images, real-time footage, scanners and sensors are all one of many forms of image data capable of supplying the

information needed in the field of computer vision. Technologically, computer vision seeks to apply its theories and models for the construction of computer vision systems. Sub-domains of computer vision include scene reconstruction, event detection, video tracking, object recognition, object pose estimation, learning, indexing, motion estimation, and image restoration.

Object detection is the process of identifying real-world objects. Examples are faces, bicycles, buildings, trees, chairs, tables and fruits in images or videos. Object detection has its own specifically constructed algorithms. These algorithms are used to extract features (Amit, 2002). Other than harvesting information from images or videos, learning algorithms are applied to decide which category of object the targeted object falls into. Some of the common applications are image retrieval, security system, surveillance camera, and automated vehicle parking systems.

1.2 Problem Statement

In high-risk operations such as a search and rescue mission, whereby human lives are highly at stake, autonomous robots are the more favorable options as they are expendable. The amount of stress a human has to handle in these operations is immense. When humans work under stress, they have a higher likelihood to make the wrong decision (Van Heugten, 2011). Robots, on the other hand, have no emotions. They operate based on programmed logics. It seems to be robots are better than humans working under stressful conditions. However, the question is, are autonomous robots are as good as humans while performing search and rescue missions, in which these missions require the ability to extract sufficient information from the surrounding to successfully complete the task? The answer to that depends on whether or not the following problems can be solved:

1) Will an autonomous robot be able to accurately identify targeted objects?
2) How effective can an autonomous robot maneuver around a field with obstacles?
3) Can an autonomous robot extract the target back to safety after locating it?
4) How does the robot differentiate between passage and the wall?

5) What technique is used to identify the targeted object?

Frankly speaking, using image processing alone without the help of any other sensors to accomplish search and rescue task might not be the best solution. However, with minimal aid from other sensors, other than cameras, provides the opportunity to explore the limitations of image processing and how computer vision can be improved. It also opens the doorway to explore the efficiency of 2D and 3D image processing.

1.3 Objective

The main objectives of this project are:

1. To design a Search and Rescue Robot (SeRaBot) capable of identifying targets using motion detection technique as its main target detection method.
2. To differentiate between human and object using facial recognition.
3. To implement path finding and obstacle avoidance through image processing.

1.4 Scope of Study

The focus of this project is to explore the capabilities of image processing. Functions such as thresholding, contouring, edge detection, object detection and facial recognition will be some, but not all of the algorithms used. Prior to doing all the mentioned above, a certain level of familiarization towards cv2 commands is required. Hence, study of cv2 commands and their functions is within the scope of study. Other than that, testing these commands out and using them in relation to each other to generate image processing techniques should also be one of the focus of the project. Related algorithms will be exploited in order to meet the objectives of this project, which are to maneuver across the field without bumping into obstacles while only using image processing, identify targets using facial recognition, object detection and motion detection as well as successfully carrying out search and rescue mission.

1.5 Relevancy and Feasibility

For image processing to be implemented on a robot, the controller used has to be Raspberry Pi because clearly Arduino does not have that level of processing capability. The Raspberry Pi has to be installed with OpenCV software, which is where most of the image processing algorithms are pre-installed. OpenCV is an open source software which can be easily obtained through downloading online. Motion detection is done by comparing one frame with another frame through thresholding. By applying the thresholding function, the pixel value difference between the two frames will be identified, hence, theoretically, motion is detected. Some readily available facial recognition algorithms could be found on the Internet. By understanding and fine tuning the code, facial recognition will not be a problem as long as a camera is present.

CHAPTER 2
LITERATURE REVIEW

2.1 Introduction to Search and Rescue Robot

The ability to work in harsh environment is the main focus of search and rescue mission robots because that is the most likely situation and scenario search and rescue robots will face once they are out in the field. Undebatably, the main criterion in a search and rescue robot design is to be able to avoid obstacles during its maneuver. The environment that requires this type of robots are the ones that may be difficult for humans or dogs to search (Sam et al., 2009).

In post-catastrophic events such as earthquake, tsunami, volcano eruption, flood, typhoon, and the list goes on, search and rescue robots are the more reliable and humane option to carry out missions on fields whereby the number of life-taking uncertainties are tremendous. Accidents such as chemical leakage where humans are prone to the exposure of hazardous chemicals which might result in long term harm, search and rescue robots will prove their values. With proper engineering, design and planning, robots built can meet the objectives of missions by carrying out investigations and inspections under collapsed structures, or maybe access places inaccessible by humans (Search and Rescue Robot: What is it all about?, n.d.). Other than their contribution in any rescue missions that involves harsh environment, the advancements of robotics industry nowadays had contributed towards all kinds of aid and act as a helping hand in the heavy industry.

2.2 Sensors

Vision is the most used sense for detection of human presence. It has made its proof with humans, so it is one of the most effective (S. Burion, 2004). Therefore, the advancement of technology in camera to provide vision to machines and computers has brought to us many variations of cameras. As listed in the table above, there are 12 types of sensors and a third of them are cameras. The cheapest vision sensor of all, is the liner camera, but of course, given its price, there is no high expectation of its capability in detecting human presence. Color camera itself has a couple of versions. USB cameras equipped with CMOS sensors, due to the nature of the sensor, are the cheaper ones while cameras with CCD sensors are, as expected, more expensive. Some of the advantages of the pricy cameras are higher sensitivity and efficiency in detecting presence of human. Unfortunately, there is a big disadvantage, which is high-level image processing skills are required in order to put this type of camera into action.

Stereo vision utilizes two color cameras. The difference between images can be extracted to provide information on depth. Given the same properties as one camera, with similar advantages in detecting human presence, its price tag increment is due to the fact that it can provide additional information.

To detect human presence, infrared camera captures human body heat. This may be the best solution. As the human body temperature is highly unlikely to be the same as his surrounding temperature, this method is theoretically and clinically proven to be efficient. Although infrared camera is the most expensive vision sensor, they seem to be essential to a robust and efficient solution for human finding.

Table 2.1: Summary of available sensors (Burion, 2004)

	Technology	Feature detected	External size	Cost	human/non human distinction	strengths	Weakness
Linear camera	CCD/CMOS EM 0.4 – 1.1 µm	vision	-	-	-	price	low resolution
USB camera	CCD/CMOS EM 0.4 – 1.1 µm	vision	+	+	++	cost/ performance	resolution
Stereo vision	CCD/CMOS EM 0.4 – 1.1 µm	vision/ distance	++	++	++	Vision + distance info.	computation ally expensive
Infrared camera	CCD/CMOS EM 7 – 14 µm	heat	++	+++	+++	human distinction	price
Pyroelectric	crystalline sensor EM 7 – 14 µm	body heat	-	-	++	price, human distinction	only motion detection
Thermopile	thermocouple EM 5.5 – 13 µm -25°C – 100 °C	heat	-	-	+	price	only average temperature
Microphone	membrane SW 100Hz – 16 kHz	Sound	-	-	+	price	noise sensitivity
Laser rangefinder	time of flight/triangul. EM 620 - 820 nm	distance	++	+++	-	precision of measure	price
Ultrasonic sensor	membrane SW 130 – 290 kHz	distance	-	-	-	price	echo sensitivity
Radar	time of flight EM 5 – 24 GHz	distance	+	+++	-	precision with big range	price
CO_2 sensor	Electro-chemical	gas	++	++	++	human distinction	too directional
SpO2	light absorption (650nm and 805nm)	blood oxygen/pul se rate	-	N/A	+++	human distinction	not available for robotics

EM = electromagnetic waves; SW = sound waves

2.3 Methods of facial detection

2.3.1 Correlation

This is the simplest classification method. Based on the closest pixel, an image is used to be tested on. The perfect match is indicated when there is zero mean and variance after comparison, which is normalization. However, also, due to the normalization process, light source intensity is independent and there is a gain control which is automatically control based on the normalization process.

Other than advantages, this method has a few disadvantages. First of all, images in test set cannot be set to have a big range of lighting difference. This is to prevent sample

points to have a vast distribution. Secondly, it is expensive. In order to carry out recognition, there must be a correlation between the test image and the sample images. Expensive VLSI hardware are used to reduce time computationally. Lastly, learning or sample set takes a lot of space in the storage.

2.3.2 Eigenfaces

Principal components analysis (PCA), also known as Karhunen-Loeve methods, uses reduced dimensionality to optimize scatter of all samples. If *N* sample images $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ have values in a *n*-dimensional image space, assuming all images fall under one of the c-class $\{X_1, X_2, \ldots, X_c\}$. Considering also a linear transformation, which maps the original *n*-dimensional image space into an *m*-dimensional feature space, where $m < n$. $\mathbf{y}_k \in \mathbb{R}^m$, new feature vectors, are defined by the following linear transformation:

$$\mathbf{y}_k = W^T \mathbf{x}_k \qquad k = 1, 2, \ldots, N \tag{1}$$

where $W \in \mathbb{R}^{n \times m}$ (matrix with orthonormal columns). $S_T = \sum_{k=1}^{N} (\mathbf{x}_k - \boldsymbol{\mu})(\mathbf{x}_k - \boldsymbol{\mu})^T$ defines the total scatter matrix, where *n* is the number of sample images, and $\boldsymbol{\mu} \in \mathbb{R}^n$ is the mean image of all samples. upon application of linear transformation W^T, the scatter of the transformed feature vectors $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\}$ is $W^T S_T W$. In PCA, the projection W_{opt} is chosen to maximize the determinant of the total scatter matrix of the projected samples, i.e.,

$$\begin{aligned} W_{opt} &= \arg\max_W |W^T S_T W| \\ &= [\mathbf{w}_1 \quad \mathbf{w}_2 \quad \ldots \quad \mathbf{w}_m] \end{aligned} \tag{2}$$

where $\{\mathbf{w}_i \mid i = 1, 2, \ldots, m\}$ set of *n*-dimensional eigenvectors of *ST* corresponding to the *m* largest eigenvalues.

If m is equal to N, Eigenface method is then equal to the correlation method. One of the down side of this method is when a cluster is formed, it might include unwanted pixels as well. Recall the comment by Moses et al. [9]: Much of the variation from one image to the next is due to illumination changes. Therefore, if variation of lighting is put into the equation for PCA, the projection matrix, W_{opt} will contain principal components lighting-caused variation. Consequently, bad classification will occur. Lighting variation effects can be reduced by removing three main components. However, it is not utterly likely that the first several principal components correspond

solely to variation in lighting, which as a consequence, useful information might be lost (Turk & Pentland, 1991).

2.3.3 Linear Subspaces

Correlation and Eigenface method are not good at handling lighting direction variation. The reason they fail is both of them did not exploit an observation. This observation is the images of a face which exists in 3D linear subspace for a Lambertian surface without shadowing. For example, a Lambertian surface with point p shined on by a light source. Let $\mathbf{s} \in \mathbb{R}^3$ be a column vector representing the product of the light source intensity with the unit vector for the light source direction. When the surface is viewed by a camera, the resulting image intensity of the point *p* is given by

$$E(p) = a(p)\mathbf{n}(p)^T\mathbf{s} \tag{3}$$

where n(p) is the unit inward normal vector to the surface at the point *p*, and *a*(*p*) is the albedo of the surface at *p*. This shows that the image intensity of the point *p* is linear on $\mathbf{s} \in \mathbb{R}^3$. Therefore, in the absence of shadowing, given three images of Lambertian surface from the same viewpoint taken under three known, linearly independent light source directions, the albedo and surface normal can be recovered; this is the well-known method of photometric stereo. Alternatively, one can reconstruct the image of the surface under an arbitrary lighting direction by a linear combination of the three original images. For classification, this fact has great importance. It shows that, for a fixed viewpoint, the images of a Lambertian surface lie in a 3D linear subspace of the high-dimensional imagespace. This observation suggests a simple classification algorithm to recognize Lambertian surfaces—insensitive to a wide range of lighting conditions. For each face, use three or more images taken under different lighting directions to construct a 3D basis for the linear subspace. Note that the three basis vectors have the same dimensionality as the training images and can be thought of as basis images (Cheng et. Al, 1991).

To perform recognition, we simply compute the distance of a new image to each linear subspace and choose the face corresponding to the shortest distance. We call this recognition scheme the Linear Subspace method. We should point out that this method is a variant of the photometric alignment method proposed, and is a special case of the more elaborate recognition method described. Subsequently, Nayar and Murase have

exploited the apparent linearity of lighting to augment their appearance manifold. If there is no noise or shadowing, the Linear Subspace algorithm would achieve error free classification under *any* lighting conditions, provided the surfaces obey the Lambertian reflectance model.

Nevertheless, there are several compelling reasons to look elsewhere. First, due to self-shadowing, specularities, and facial expressions, some regions in images of the face have variability that does not agree with the linear subspace model. Given enough images of faces, we should be able to learn which regions are good for recognition and which regions are not. Second, to recognize a test image, we must measure the distance to the linear subspace for each person. While this is an improvement over a correlation scheme that needs a large number of images to represent the variability of each class, it is computationally expensive. Finally, from a storage standpoint, the Linear Subspace algorithm must keep three images in memory for every person.

4. Fisherfaces

The previous algorithm takes advantage of the fact that, under admittedly idealized conditions, the variation within class lies in a linear subspace of the image space. Hence, the classes are convex, and, therefore, linearly separable. One can perform dimensionality reduction using linear projection and still preserve linear separability. This is a strong argument in favor of using linear methods for dimensionality reduction in the face recognition problem, at least when one seeks insensitivity to lighting conditions. Since the learning set is labeled, it makes sense to use this information to build a more reliable method for reducing the dimensionality of the feature space. Here we argue that using class specific linear methods for dimensionality reduction and simple classifiers in the reduced feature space, one may get better recognition rates than with either the Linear Subspace method or the Eigenface method. Fisher's Linear Discriminant (FLD) is an example of a *class specific method*, in the sense that it tries to "shape" the scatter in order to make it more reliable for classification. This method selects W in [1] in such a way that the ratio of the between-class scatter and the within-class scatter is maximized. Let the between-class scatter matrix be defined as

$$S_B = \sum_{i=1}^{c} N_i (\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^T \tag{4}$$

and the within-class scatter matrix be defined as

$$S_W = \sum_{i=1}^{c} \sum_{\mathbf{x}_k \in X_i} (\mathbf{x}_k - \boldsymbol{\mu}_i)(\mathbf{x}_k - \boldsymbol{\mu}_i)^T \tag{5}$$

where μ_i is the mean image of class X_i , and N_i is the number of samples in class X_i . If S_W is nonsingular, the optimal projection W_{opt} is chosen as the matrix with orthonormal columns which maximizes the ratio of the determinant of the between-class scatter matrix of the projected samples to the determinant of the within-class scatter matrix of the projected samples, i.e.,

$$\begin{aligned} W_{opt} &= \arg\max_W \frac{|W^T S_B W|}{|W^T S_W W|} \\ &= [\mathbf{w}_1 \quad \mathbf{w}_2 \quad \dots \quad \mathbf{w}_m] \end{aligned} \tag{6}$$

where $\{\mathbf{w}_i \mid i = 1, 2, \dots, m\}$ is the set of generalized eigenvectors of S_B and S_W corresponding to the m largest generalized eigenvalues $\{\lambda_i \mid i = 1, 2, \dots, m\}$, i.e.,

$$S_B \mathbf{w}_i = \lambda_i S_W \mathbf{w}_i, \qquad i = 1, 2, \dots, m \tag{7}$$

Note that there are at most c - 1 nonzero generalized eigenvalues, and so an upper bound on m is c - 1, where c is the number of classes.

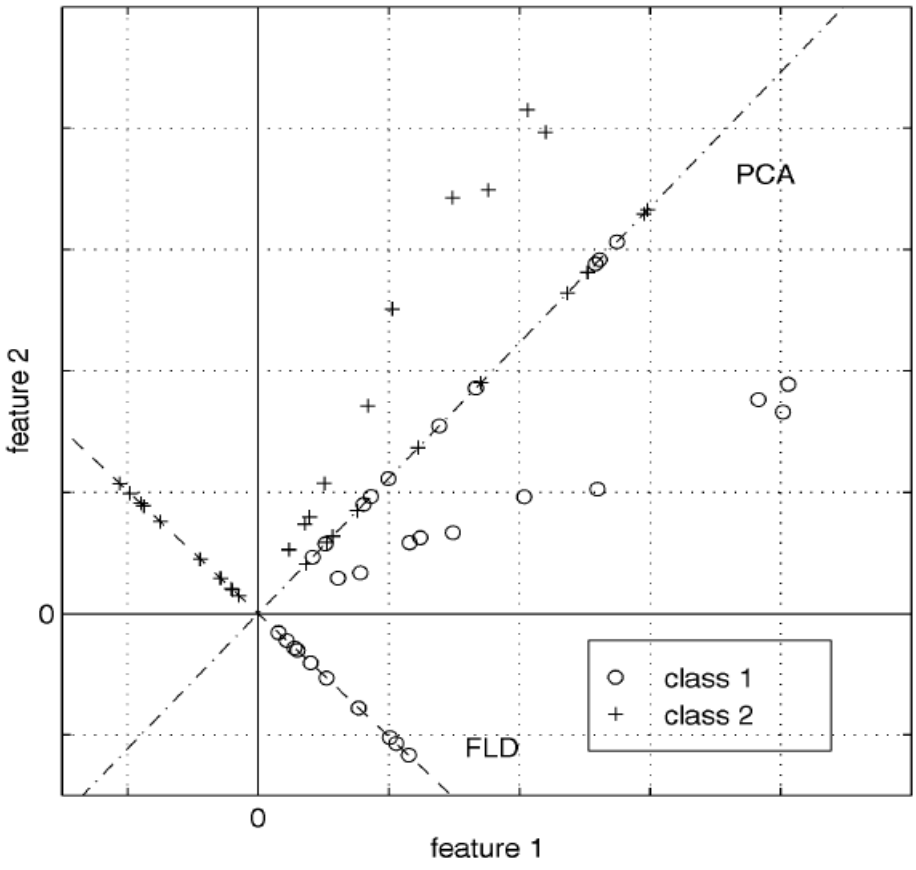

Figure 2.1: Comparison of PCA and LFD (Pentland, 1991)

To illustrate the benefits of class specific linear projection, we constructed a low dimensional analogue to the classification problem in which the samples from each class lie near a linear subspace. Fig. 1 is a comparison of PCA and FLD for a two-class problem in which the samples from each class are randomly perturbed in a direction perpendicular to a linear subspace. For this example, $N = 20$, $n = 2$, and $m = 1$. So, the samples from each class lie near a line passing through the origin in the 2D feature space. Both PCA and FLD have been used to project the points from 2D down to 1D. Comparing the two projections in the figure, *PCA actually smears the classes together* so that they are no longer linearly separable in the projected space. It is clear that, although PCA achieves larger total scatter, FLD achieves greater between-class scatter, and, consequently, classification is simplified. In the face recognition problem, one is confronted with the difficulty that the within-class scatter matrix $S_W \in \mathbb{R}^{n \times n}$ is always singular. This stems from the fact that the rank of S_W is at most $N - c$, and, in general, the number of images in the learning set N is much smaller than the number of pixels in each image n. This means that it is possible to choose the matrix W such that the within-class scatter of the projected samples can be made exactly zero. In order to overcome the complication of a singular S_W, we propose an alternative. This method, which we call Fisherfaces, avoids this problem by projecting the image set to a lower dimensional space so that the resulting within-class scatter matrix S_W is nonsingular. This is achieved by using PCA to reduce the dimension of the feature space to $N - c$, and then applying the standard FLD defined to reduce the dimension to $c - 1$. More formally, W_{opt} is given by

$$W_{opt}^T = W_{fld}^T W_{pca}^T$$

$$\begin{aligned} W_{pca} &= \arg\max_W \left| W^T S_T W \right| \\ W_{fld} &= \arg\max_W \frac{\left| W^T W_{pca}^T S_B W_{pca} W \right|}{\left| W^T W_{pca}^T S_W W_{pca} W \right|} \end{aligned} \tag{8}$$

Note that the optimization for W_{pca} is performed over $n \times (N - c)$ matrices with orthonormal columns, while the optimization for W_{fld} is performed over $(N - c) \times m$ matrices with orthonormal columns. In computing W_{pca}, we have thrown away only the smallest $c - 1$ principal components. There are certainly other ways of reducing the withinclass scatter while preserving between-class scatter. For example, a second method which we are currently investigating chooses W to maximize the between-

class scatter of the projected samples after having first reduced the withinclass scatter. Taken to an extreme, we can maximize the between-

class scatter of the projected samples after having first reduced the withinclass scatter. Taken to an extreme, we can maximize the between-class scatter of the projected samples subject to the constraint that the within-class scatter is zero, i.e.,

$$W_{opt} = \arg \max_{W \in \mathcal{W}} \left| W^T S_B W \right| \quad \text{---(8)}$$

where $\mathcal{W}$ is the set of n x m matrices with orthonormal columns contained in the kernel of S_W (Fisher, 1936).

2.3.5 Cascade Classifier

In a paper by Paul Viola and Michael Jones, the advantages of using a feature-based system over a pixel-based system were discussed. The main advantages are efficiency and accuracy. Three different type of features are used: difference in sum of pixels in two rectangular regions horizontally or vertically, difference between the middle rectangle with the two rectangular regions on both sides, and four rectangles with none of their neighboring regions having the same value with themselves. Rectangle features are comparatively better than steerable features, with higher sensitivity to edges and basic image structures, but relatively coarser, and less flexible. Training a cascade of classifiers requires more computational time, while achieving higher accuracy and vice versa. There is a trade-off and getting the optimum balance between the two are extremely difficult in practice. Hence, a simple framework is utilized to come up with an efficient classifier. This classifier is then eventually used to detect object in images. Accuracy is up to 95%.

CHAPTER 3

METHODOLOGY

3.1 Process Flow of Project

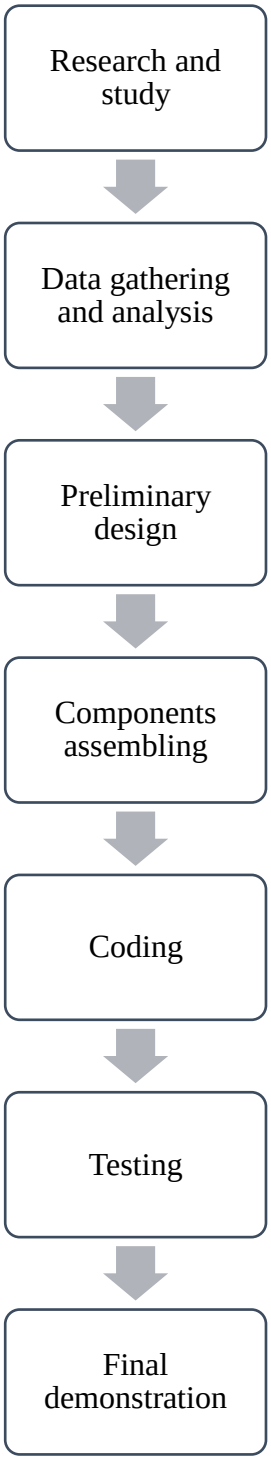

Figure 3.1: Process flow of project.

3.1.1 Research and study

The key element in this project is the robot's ability to make use of its vision (camera) through image processing. In depth research and study has been conducted in order to learn how to program the Raspberry Pi (for image processing) as well as the Arduino (for maneuvering). This exploration also allows the goal of this project to be set, whether or not it will be too ambitious, based on the image processing knowledge acquired. In order to execute image processing in the simplest way, OpenCV software is a must have. It comes with commands which run algorithms to easily process images and videos with their parameters set. Thresholding, contouring, edge detection, center detection, facial recognition and color manipulation are just some of the skills used in image processing to achieve bigger objectives.

3.1.2 Data gathering and analysis

Data will be gathered through trials and errors. As much as image processing is concerned, 2D image processing is not capable of sensing depth of an image, leading to the incapability to determine distance of object from the autonomous robot. However, through tests, the size of the object in the image reflects its actual distance from the point the image is taken. Hence, distance can be estimated. Under different lighting conditions, thresholding requires different pixel values as well. This is utterly important as contouring and edge detection are based on thresholding. Error in thresholding will result in utter failure.

3.1.3 Preliminary design

The robot design includes:

1) Raspberry Pi as controller
2) Platform, to install Raspberry Pi
3) Wheels, to maneuver around the field
4) Camera, to provide vision for the robot
5) Ultrasonic sensor, to detect distance with object

The design is just a simple 4-wheel autonomous robot with a gripper. It is true that wheels are not the best in maneuvering around real-life post-catastrophe surroundings,

but it is irrelevant in this case to go beyond this simple design as a replica of the real-life post-catastrophe field is too costly.

3.1.4 Components assembling

All the components are brought together to be assembled, namely Raspberry Pi, platform with wheels, ultrasonic sensor, and camera.

3.1.5 Coding

Firstly, the code is written in Python programming language. It involves OpenCV as well. At the start of the code, various libraries which provide support for image processing will be imported. Next up, the code will access the camera, which provides a real-time footage. Thresholding will be done to identify the walls through color identification to make decision on the direction of movement. This will then send signal to the motors to maneuver across the field once a movement is noticed. Once the target has been identified through facial recognition, signal will be sent out. The coding is targeted to be able to perform the operation mentioned above. Variables include size of the object which reflects its distance from the robot and the surrounding lighting conditions need to be taken into account while setting up the parameters during coding.

Prior to any coding, some installations need to be done on the Raspberry Pi to enable OpenCV. The list of commands can be found in Appendix A.

3.1.6 Testing

Testing is done to gain information on the suitable size of the object in the image as well as the proper pixel value range to increase accuracy in processing the image. Minor adjustments on the robot's movement are also necessary.

3.2 Motion Detection

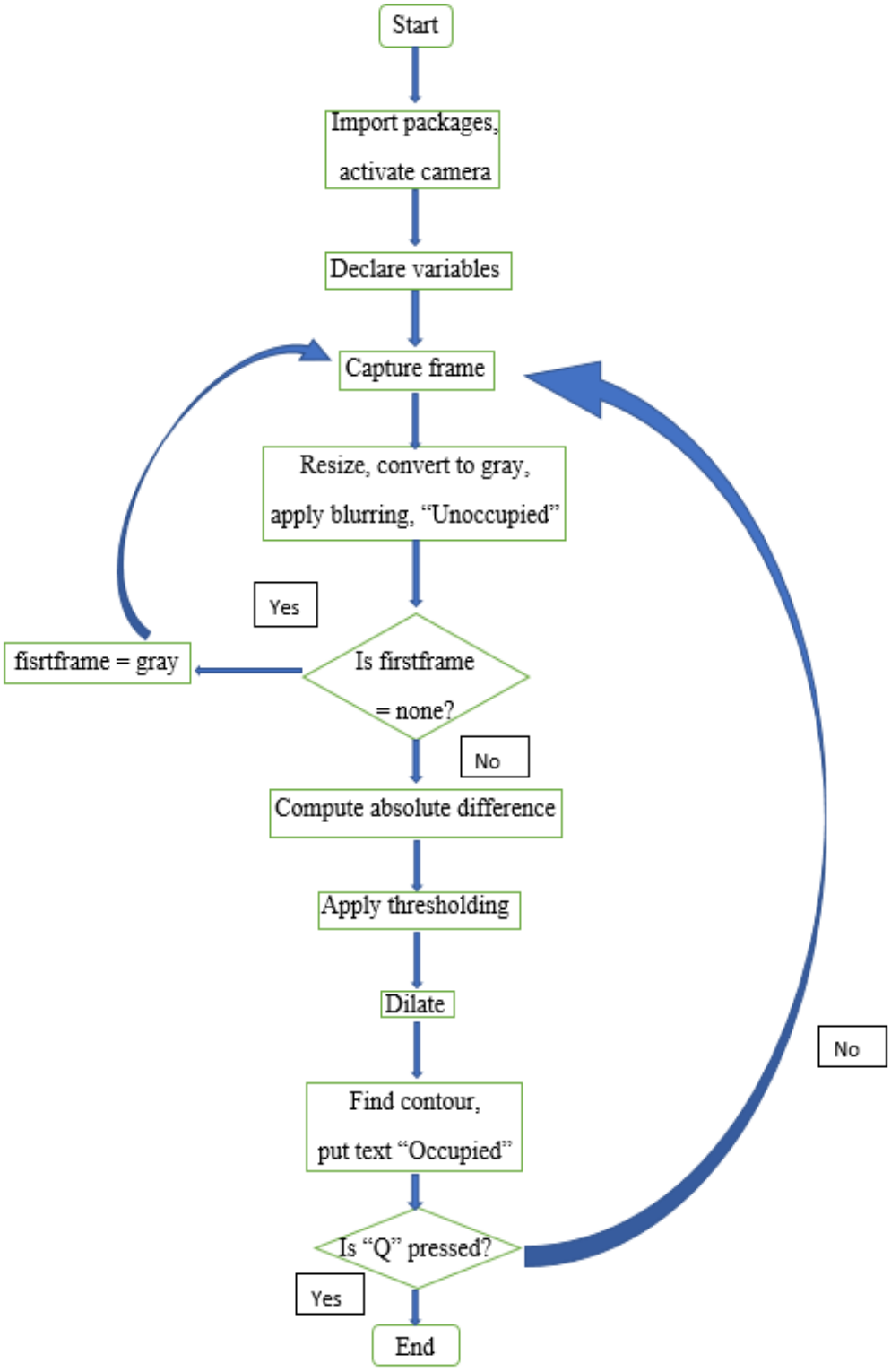

Figure 3.2: Flowchart of motion detection

Below is the code used for motion detection, broken down into multiple parts for explanation purpose. The full code is available in Appendix B.

```
import argparse
import datetime
import imutils
import time
import cv2

ap = argparse.ArgumentParser()
ap.add_argument("-a",      "--min-area",      type=int,
default=500, help="minimum area size")
args = vars(ap.parse_args())


camera = cv2.VideoCapture(0)
time.sleep(0.25)
firstFrame = None
```

In the initial part of the code, some packages are imported. Next, setup argument parser, activate the camera, and declare a variable firstFrame. Noted that the sleep time for the camera is 0.25 second to allow the camera to warm up before any frame is captured.

```
while True:
     (grabbed, frame) = camera.read()
     text = "Unoccupied"

     frame = imutils.resize(frame, width=500)
     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
     gray = cv2.GaussianBlur(gray, (21, 21), 0)

     if firstFrame is None:
          firstFrame = gray
          continue

     frameDelta = cv2.absdiff(firstFrame, gray)
     thresh   =   cv2.threshold(frameDelta,   25,   255,
cv2.THRESH_BINARY)[1]

     thresh = cv2.dilate(thresh, None, iterations=2)
     (cnts,    _)    =    cv2.findContours(thresh.copy(),
cv2.RETR_EXTERNAL,
          cv2.CHAIN_APPROX_SIMPLE)

     for c in cnts:
          if cv2.contourArea(c) < args["min_area"]:
               continue
```

Firstly, grab a frame and show the text to be unoccupied (first frame captured is always unoccupied). Next, resize the frame, convert the color from RGB to Gray to apply GaussianBlur function. Noted mentioned above, to apply GaussianBlur function, the image has to be in Gray. Subsequently, compute the absolute difference between pixels in two frames and apply thresholding. From the results of thresholding, compute the contour and draw it out. Put the text as occupied if contour is present.

```
        cv2.imshow("Security Feed", frame)
        cv2.imshow("Thresh", thresh)
        cv2.imshow("Frame Delta", frameDelta)
        key = cv2.waitKey(1) & 0xFF

        if key == ord("q"):
            break

camera.release()
cv2.destroyAllWindows()
```

Lastly, show the results in separate windows. Stop code execution if "q" key is pressed and destroy all user-open-windows.

3.3 Facial Recognition

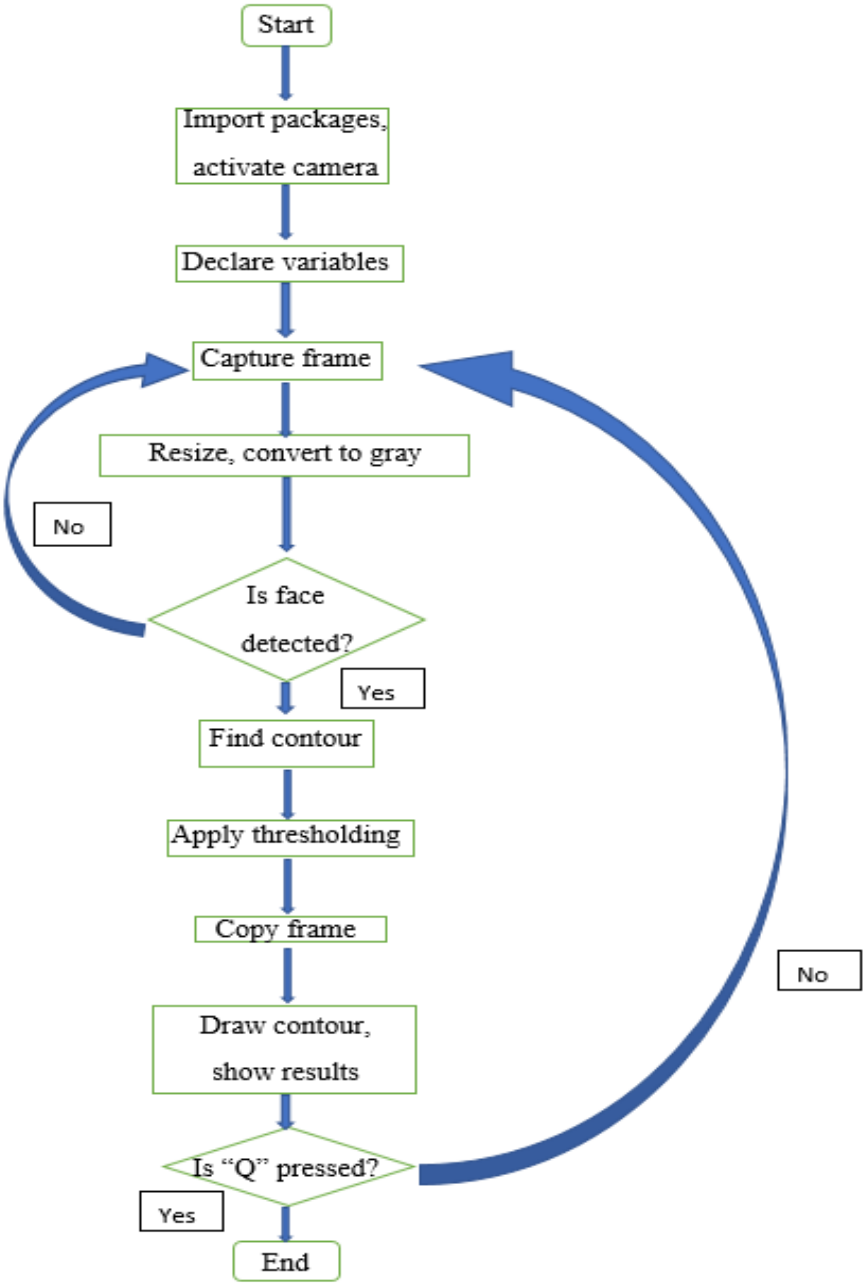

Figure 3.3: Flowchart of facial recognition

Below is the code used for motion detection, broken down into multiple parts for explanation purpose. The full code is available in Appendix C.

```
cascades/haarcascade_frontalface_default.xml

from pyimagesearch.facedetector import FaceDetector
from pyimagesearch import imutils
import argparse
import cv2

ap = argparse.ArgumentParser()
ap.add_argument("-f", "--face", required = True,
      help = "path to where the face cascade resides")
args = vars(ap.parse_args())

fd = FaceDetector(args["face"])

camera = cv2.VideoCapture(0)
```

In the initial part of the code, some packages are imported. Next, setup argument parser and activate the camera. Declare variable fd to utilize the imported package FaceDetector.

```
while True:
      (grabbed, frame) = camera.read()

      frame = imutils.resize(frame, width = 300)
      gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

      faceRects = fd.detect(gray, scaleFactor = 1.1,
minNeighbors = 5,
            minSize = (30, 30))
      frameClone = frame.copy()

      for (fX, fY, fW, fH) in faceRects:
            cv2.rectangle(frameClone, (fX, fY), (fX + fW, fY
+ fH), (0, 255, 0), 2)
```

Firstly, resize the frame. Next, convert frame color from RGB to gray. Using the gray frame, detect face. When a face is detected, clone the frame. Find contour on the frame and draw the contour out.

```
      cv2.imshow("Face", frameClone)

      if cv2.waitKey(1) & 0xFF == ord("q"):
            break

camera.release()
cv2.destroyAllWindows()
```

Show the results in a window called “Face”. If the “Q” key is pressed, destroy all

windows opened, in this case, window “Face”, deactivate the camera and stop code execution.

3.4 Integrated Image Processing

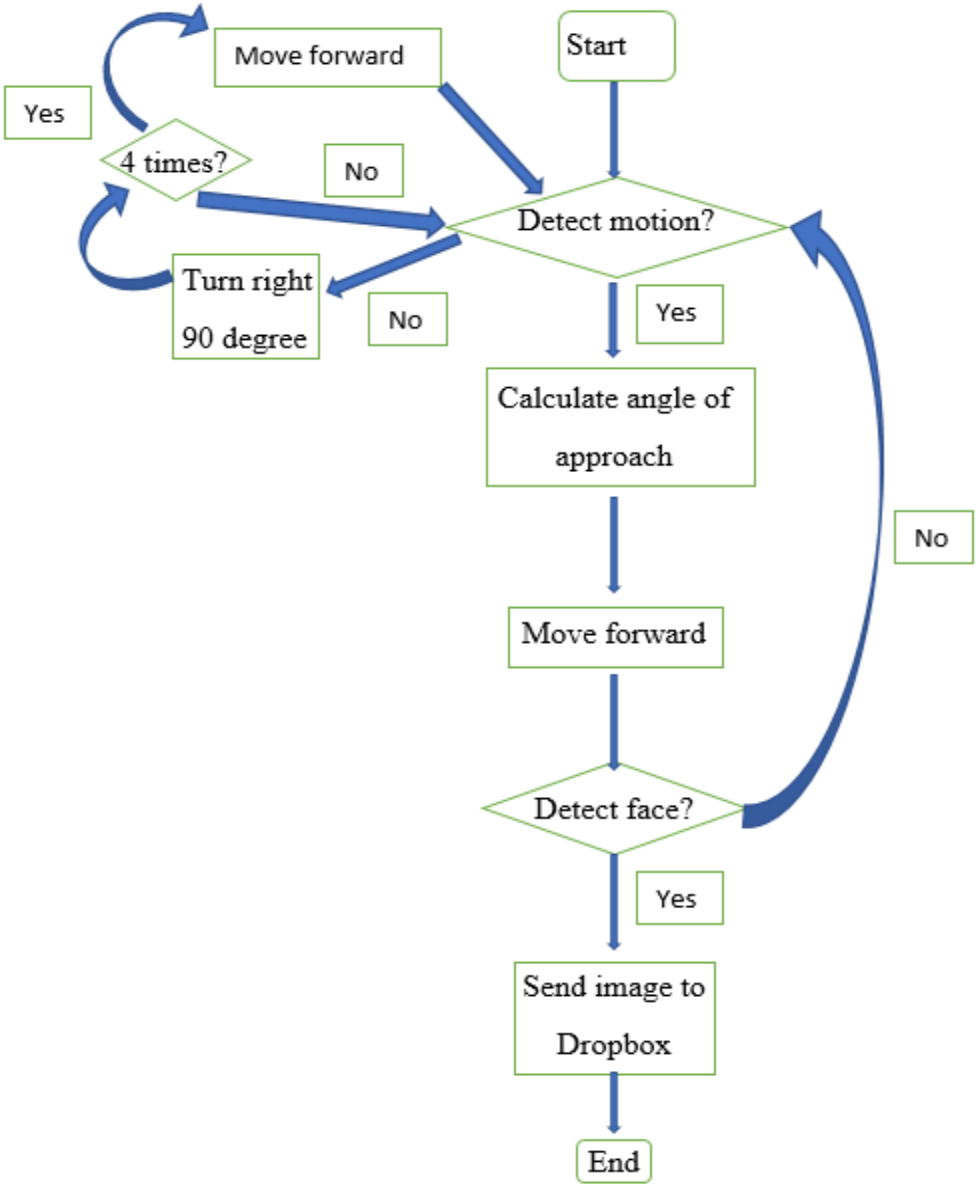

Figure 3.4: Flowchart of integrated image processing

Table 3.1: List of cv2 commands used and their functions

No.	Command	Function
1	Cv2.VideoCapture	To capture a video. Its argument can be either the device index or the name of a video file. Device index is just the number to specify which camera. `cv2.VideoCapture(0)`
2	Cv2.cvtColor	Converts an image from one color space to another. In case of a transformation to-from RGB color space, the order of the channels should be specified explicitly (RGB or BGR). Note that the default color format in OpenCV is often referred to as RGB but it is actually BGR (the bytes are reversed). So the first byte in a standard (24-bit) color image will be an 8-bit Blue component, the second byte will be Green, and the third byte will be Red. The fourth, fifth, and sixth bytes would then be the second pixel (Blue, then Green, then Red), and so on. `cv2. cvtColor(frame, cv2.COLOR_BGR2HSV)`
3	Cv2.GaussianBlur	Blurs an image using a Gaussian filter. specify the width and height of kernel which should be positive and odd. User also should specify the standard deviation in X and Y direction, sigmaX and sigmaY respectively. If only sigmaX is specified, sigmaY is taken as same as sigmaX. If both are given as zeros, they are calculated from kernel size. Gaussian

		blurring is highly effective in removing gaussian noise from the image. `cv2.GaussianBlur(img,(5,5),0)`
4	Cv2.absdiff	Calculates the per-element absolute difference between two arrays or between an array and a scalar.
5	Cv2.threshold	If pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black). First argument is the source image, which should be a grayscale image. Second argument is the threshold value which is used to classify the pixel values. Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value.
6	Cv2.FindCountours	To find a contour. there are three arguments in this function, first one is source image, second is contour retrieval mode, third is contour approximation method. And it outputs the contours and hierarchy. contours is a Python list of all the contours in the image. Each individual contour is a Numpy array of (x,y) coordinates of boundary points of the object.
7	Cv2.boundingRect	Create bounding box. It is a straight rectangle, it doesn't consider the rotation of the object. So area of the bounding rectangle won't be minimum.

8	Cv2.countourArea	Calculates a contour area.
9	Cv2.imshow	To display an image in a window. The window automatically fits to the image size. First argument is a window name which is a string. Second argument is the image. User can create as many windows as he wishes, but with different window names. `cv2.imshow('image',img)`
10	Cv2.putText	Draws a text string.
11	Cv2.waitKey	A keyboard binding function. Its argument is the time in milliseconds. The function waits for specified milliseconds for any keyboard event. If user presses any key in that time, the program continues. If **0** is passed, it waits indefinitely for a key stroke. It can also be set to detect specific key strokes like, if key *a* is pressed, etc. It also processes many other GUI events, so it is necessary to have this function to display image. `cv2.waitKey(0)`
12	Cv2.imwrite	To save an image. First argument is the file name, second argument is the image you want to save. `cv2.imwrite('messigray.png',img)`
13	Cv2.destroyAllWindows	Destroys all windows created. There is a special case where user can already create a window and load image to it later. In that case, user can specify whether window is

		resizable or not. It is done with the function cv2.namedWindow(). By default, the flag is cv2.WINDOW_AUTOSIZE. But if user specifies flag to be cv2.WINDOW_NORMAL, he can resize window. It will be helpful when image is too large in dimension and adding track bar to windows. `cv2.destroyAllWindows()`
14	Cv2.dilate	Dilates an image by using a specific structuring element.

Source code explanation

Below is the code used for motion detection, broken down into multiple parts for explanation purpose. The full code is available in Appendix D.

```
import argparse
import datetime
import imutils
import cv2
import time
import numpy as np
import RPi.GPIO as gpio
from pyimagesearch.facedetector import FaceDetector
from pyimagesearch.tempimage import TempImage
from dropbox.client import DropboxOAuth2FlowNoRedirect
from dropbox.client import DropboxClient
```

This part of the code imports necessary packages. All packages are pretty straight forward except imutils, which is a set of convenience functions to make basic image processing tasks easier, and FaceDetector, which is the file containing algorithms to perform facial recognition.

```
ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video", help="path to the video
file")
ap.add_argument("-a",      "--min-area",      type=int,
default=3000, help="minimum area size")
ap.add_argument("-f", "--face", required = True,
```

The path to FaceDetector file is required upon activation of the code, hence the setting of argument parse for it. Minimum area is defined --min-area, which is the minimum size (in pixels) for a region of an image to be considered actual “motion”. Small regions of an image that have changed substantially happens quite often, likely due to noise or changes in lighting conditions. In reality, these small regions are not actual motion at all. Therefore, a minimum size of a region is defined to combat and filter out these false-positives.

```
gpio.setmode(gpio.BOARD)
gpio.setwarnings(False)
gpio.setup(7, gpio.OUT, initial=0)
gpio.setup(11, gpio.OUT, initial=0)
gpio.setup(13, gpio.OUT, initial=0)
gpio.setup(15, gpio.IN)
camera = cv2.VideoCapture(0)
time.sleep(0.25)
timer = 0
firstFrame = None
fd = FaceDetector(args["face"])
```

Setting up the GPIO pins to follow BOARD numbering. Declaration of the parameters and variables used in the following code as well as activating the camera.

```
while True:
        gpio.output(7, 0)
        gpio.output(11, 0)
        gpio.output(13, 1)
        if gpio.input(15):
                gpio.output(13, 0)
                while timer<10:
                        timer+= 0.25
                        (grabbed, frame) = camera.read()
                        text = "Unoccupied"
                        frame  =  imutils.resize(frame,
width=500)
                        gray   =   cv2.cvtColor(frame,
cv2.COLOR_BGR2GRAY)
                        gray  =  cv2.GaussianBlur(gray,
(21, 21), 0)

                        if firstFrame is None:
                                firstFrame = gray
                                continue
```

In the body of the code, first start an infinite loop which continues until the letter “q” is pressed. This part of the code captures a frame as make it as the reference. After

converting the color of the frame from RGB to GRAY, blurring is applied. If previously no frame was taken as reference, which in this case is true, the filtered frame named “gray” will be the reference frame. At this point, the robot is stationary.

```
                        frameDelta                        =
cv2.absdiff(firstFrame, gray)
                        thresh                            =
cv2.threshold(frameDelta, 25, 255, cv2.THRESH_BINARY)[1]

                        thresh    =    cv2.dilate(thresh,
None, iterations=2)
                        (_,       cnts,       _)          =
cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,

cv2.CHAIN_APPROX_SIMPLE)

                        for c in cnts:
                                if cv2.contourArea(c) <
args["min_area"]:
                                        continue


                                (x,    y,    w,    h)    =
cv2.boundingRect(c)
                                cv2.rectangle(frame,
(x, y), (x + w, y + h), (0, 255, 0), 2)
                                text = "Occupied"
                                gpio.output(7, 1)
                                gpio.output(11, 0)
                                gpio.output(13, 1)
                                gpio.output(13, 0)
```

Continue to filter the frame and apply thresholding to compare the pixel offset for the following frames. If any offset is detected and the area covered is larger than the default minimum area, it will be regarded as motion. A green bounding box will appear on that area. Text “Occupied” will be updated and the robot will move forward.

```
frame = imutils.resize(frame, width = 300)
                                gray                      =
cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
                                faceRects                 =
fd.detect(gray, scaleFactor = 1.1, minNeighbors = 5,
                                        minSize  =  (30,
30))
```

```
                                    frameClone              =
frame.copy()

                                    for (fX, fY, fW, fH) in
faceRects:

cv2.rectangle(frameClone, (fX, fY), (fX + fW, fY + fH),
(0, 255, 0), 2)

cv2.imshow("Face", frameClone)

                            cv2.putText(frame, "Room Status:
{}".format(text), (10, 20),

cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
                            cv2.putText(frame,
datetime.datetime.now().strftime("%A %d %B %Y %I:%M:%S%
p"),
                                    (10,  frame.shape[0]  -
10), cv2.FONT_HERSHEY_SIMPLEX, 0.35, (0, 0, 255), 1)

                            cv2.imshow("Security     Feed",
frame)
                            cv2.imshow("Frame        Delta",
frameDelta)

                            if  cv2.waitKey(1)  &  0xFF  ==
ord("q"):
                                    break

                if cv2.waitKey(1) & 0xFF == ord("q"):
                            break
```

This is where the facial recognition kicks in. While the motion is detected, the robot will start to detect for faces, once a face is detected, a new pop up window will appear to showcase the detected face with a green bounding box around it.

```
if text == "Occupied":
     if (timestamp - lastUploaded).seconds >=
conf["min_upload_seconds"]:
          motionCounter += 1

          if motionCounter >= conf["min_motion_frames"]:
               if conf["use_dropbox"]:
                    t = TempImage()
                    cv2.imwrite(t.path, frame)

                    print "[UPLOAD] {}".format(ts)
```

```
                        path =
"{base_path}/{timestamp}.jpg".format(
                        base_path=conf["dropbox_base_path"],
timestamp=ts)
                        client.put_file(path, open(t.path,
"rb"))
                        t.cleanup()

                        lastUploaded = timestamp
                        motionCounter = 0

else:
     motionCounter = 0
```

When a face is detected, the image will be automatically saved into a directory named TempImage. Once the frame is done, it will be uploaded onto Dropbox.

```
                firstFrame = None
                gpio.output(7, 0)
                gpio.output(11, 1)
                gpio.output(13, 1)
                time.sleep(0.5)
                gpio.output(13, 0)
                time.sleep(2)


                timer = 0
        else:
                gpio.output(13, 1)

camera.release()
cv2.destroyAllWindows()
gpio.cleanup()
```

If all the above happened without any motion detected, the robot will turn right and a new reference frame will be initiated. If there's no motion detected after 4 times of checking, the robot will move forward and repeat the cycle. All windows will be cleansed if the button "q" is pressed.

3.5 Key Milestone

Table 3.2: Project key milestone.

Period (Date)	Description
5/9/16	Selection of FYP title.
12/9/16 – 14/10/16	Completion of extended proposal and pre-submission to supervisor for evaluation.
15/10/16 – 27/10/16	Correction of extended proposal report.
28/10/16	Submission of extended proposal report to supervisor.
29/10/16 – 10/11/16	Completion and presentation for proposal defense.
14/11/16 – 2/12/16	Completion of interim report.
2/1/17 – 22/1/17	Starting of preliminary work: OpenCV familiarization
1/3/17	Submission of progress report.
10/3/17	Completion of motion detection.
21/3/17	Submission of poster to supervisor for checking.
22/3/17	Pre-Sedex (Poster Presentation)
27/3/17	Completion of facial recognition, prototype aseembly
29/3/17	Submission of draft report to supervisor.
30/3/17 – 3/4/17	Correction for draft report.
17/4/17	Submission of soft bound dissertation.
17/4/17	Submission of technical paper to supervisor for checking.
20/4/17	Completion of presentation slides.
25/4/17	Viva Oral Presentation.
2/5/17	Submission of hard bound dissertation.

3.6 Project Gantt Chart

Table 3.3: Project Gantt Chart for FYP I.

Task/Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Meeting with supervisor														
Write an abstract of the study														
Perform background study														
Identify problem statement, objectives and scope of study														
Completion of literature review														
Completion of methodology														
Submission of Extended Proposal														
Preparation for Proposal Defense														
Proposal Defense														
Completion of Interim Report														

Table 3.4: Project Gantt Chart for FYP II.

Activity	Week													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Improve image processing accuracy	✓													
Acquire suitable path planning algorithms		✓												
Explore advanced algorithms in image processing			✓	✓	✓									
Combine motion detection and facial recognition algorithms						✓	✓	✓	✓					
Setup appropriate testing field										✓				
Testing, troubleshooting and improving											✓	✓	✓	✓

CHAPTER 4

RESULTS AND DISCUSSION

The final results were captured using the integrated image processing code, which consists of motion detection, facial recognition and communication with Dropbox. In this chapter, the results for were captured and discussed.

4.1 Motion Detection

Figure 4.1: First frame captured

Figure 4.2: Room status unoccupied even with motion

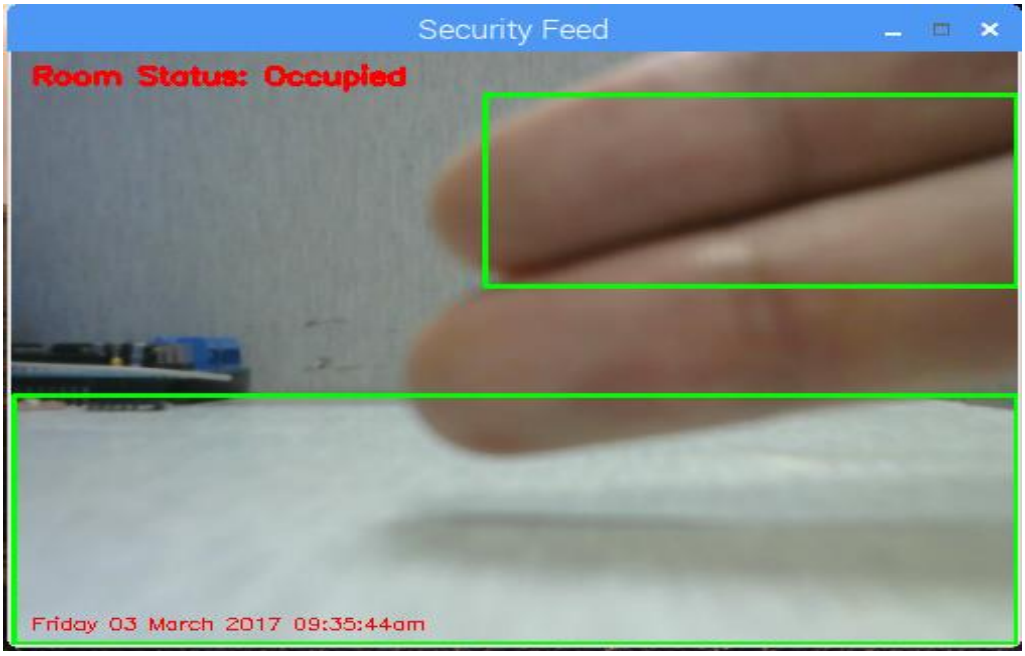

Figure 4.3: Room status occupied

When the camera was first activated, it captured a frame (Figure 4.1) and set it as the reference frame. Any succeeding frame would be threshold and compared with this frame. However, in Figure 4.2, it was clear that there was a difference as compared to Figure 4.1, but the text was still “Unoccupied”. The reason was, the pixel difference in Figure 4.2 did not cover an area which exceeded the minimum area declared. Hence, it was considered as no motion. In this case, people might argue that defining a minimum area affects the accuracy of the result. It is utterly true, but in a positive way. Defining a minimum area to be considered as motion filters small changes in a frame, which may be caused by spikes in the hardware, or a slight change in ambient light. However, the down side of using a pre-defined minimum area is when the target is too far away, causing the motion detected to be smaller than the defined minimum area.

In Figure 4.3, motion was detected, as indicated by the green bounding box, and the text has changed from “Unoccupied” to “Occupied”. Looking at the green bounding box in Figure 4.3, the lower half of the image was totally bounded while the only significant difference between Figure 4.3 and 4.1 was the presence of two fingers. This was because their presence has caused the focus of the camera to change, consequently changing the ambient light intensity captured. Remember the function calculates absolute difference in pixel value. When the surrounding lighting changes, the pixel value captured would definitely be affected as well. Hence, resulting in the result presented in Figure 4.3.

One problem with this motion detection technique is, once the camera is activated, the robot captures the first frame as its background. Any changes to the frame will result in motion detected, including movement of the robot itself. Hence, the robot has to be stationary upon camera activation, or the camera has to be turned off prior to robot movement. Failure to do so will result in the whole frame be detected as motion.

4.2 Facial Recognition

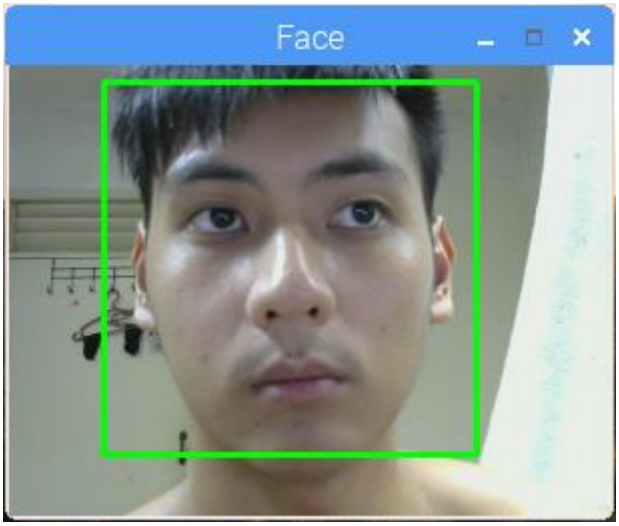

Figure 4.4: Straight and still face

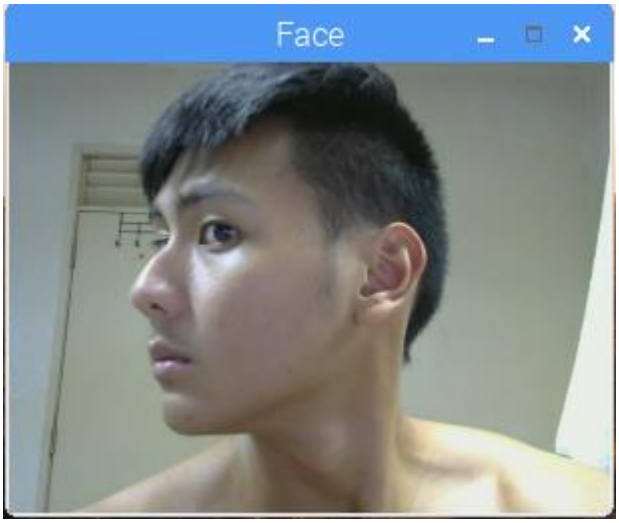

Figure 4.5: Side face

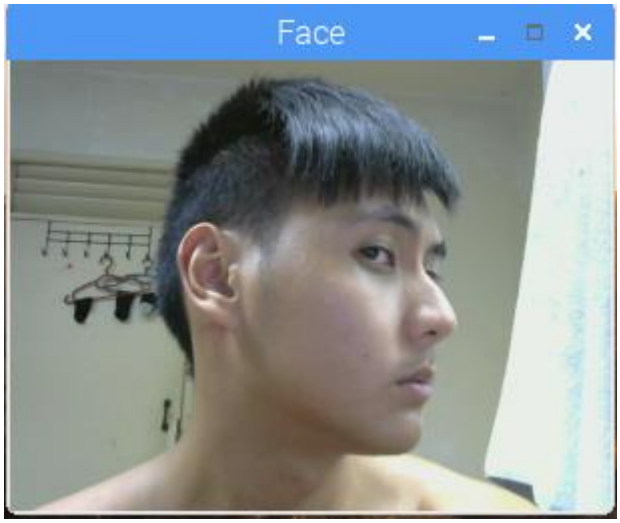

Figure 4.6: Side face

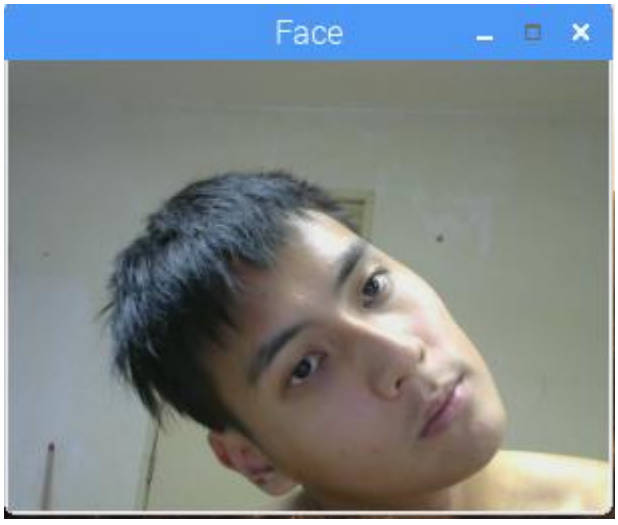

Figure 4.7: Tilted head

In Figure 4.4, a green bounding box was drawn around the face, indicating a face has been detected. The facial recognition technique used here detects any object which resembles a human face. It was not restricted to only certain individual's face.

However, even though the same face was recorded in Figure 4.5, 4.6 and 4.7, the results were different. No green bounding box around the face. This was because in Figure 4.5 and 4.6, the face was captured sideways whereas in Figure 4.7, the head was tilted. This means that the subject has to be captured with a straight face in order to be recognized by the robot. The reason behind this is the FaceDetector algorithm

uses samples presented the same way as Figure 4.4. The angle a face is captured will also cause failure to recognize.

CHAPTER 5

CONCLUSION AND RECOMMENDATION

The first objective of this project is to design a robot capable of using motion detection technique as its primary target detection method. As proven in the result, motion was detected via comparison of pixel values between frames captured. This motion detection method is also capable of filtering out minor motions, improving the accuracy of detection. The accuracy of the result was further enhanced by using facial recognition to detect faces.

Basic path finding was also integrated into the robot by using coordinates of bounding box as inputs to formula, to calculate the exact location of the target, instead of just the direction. The calculated values were then used to control the robot movement, turning it at a certain angle.

For future work, I would recommend continuing the quest to explore other motion detection and facial recognition techniques, as clearly the ones used have their own flaws as well. Hopefully in the near future, with exploration of new techniques, these flaws can be overcome. Apart from that, image processing algorithms as a whole should continue to be researched, improved and utilized. The limitations should continue to be pushed and boundaries would be extended.

CHAPTER 6

REFERENCES

[1] R. Chellappa, C. Wilson, and S. Sirohey, "Human and Machine Recognition of Faces: A Survey," *Proc. IEEE*, vol. 83, no. 5, pp. 705-740, 1995.

[2] A. Samal and P. Iyengar, "Automatic Recognition and Analysis of Faces and Facial Expressions: A Survey," *Pattern Recognition*, vol. 25, pp. 65-77, 1992.

[3] P. Viola and M. Jones, "Rapid Object Detection Using A Boosted Cascade of Simple Features," *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2001.

[4] A. Shashua, "Geometry and Photometry in 3D Visual Recognition," PhD thesis, Massachusetts Institute of Technology, 1992.

[5] R. Duda and P. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.

[6] R.A. Fisher, "The Use of Multiple Measures in Taxonomic Problems," *Ann. Eugenics*, vol. 7, pp. 179-188, 1936.

[7] L. Sirovitch and M. Kirby, "Low-Dimensional Procedure for the Characterization of Human Faces," *J. Optical Soc. of Am. A*, vol. 2, pp. 519-524, 1987.

[8] M. Turk and A. Pentland, "Eigenfaces for Recognition," *J. Cognitive Neuroscience*, vol. 3, no. 1, 1991.

[9] M. Turk and A. Pentland, "Face Recognition Using Eigenfaces," *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 1991, pp. 586-591.

[10] Y. Moses, Y. Adini, and S. Ullman, "Face Recognition: The Problem of Compensating for Changes in Illumination Direction," European Conf. Computer Vision, 1994, pp. 286-296.

[11] Y. Cheng, K. Liu, J. Yang, Y. Zhuang, and N. Gu, "Human Face Recognition Method Based on the Statistical Model of Small Sample Size," SPIE Proc. Intelligent Robots and Computer Vision X: Algorithms and Technology, 1991, pp. 85-95.

[12] S. Baker and S.K. Nayar, "Pattern Rejection," Proc. IEEE Conf. Computer Vision and Pattern Recognition, 1996, pp. 544-549.

[13] P.N. Belhumeur, J.P. Hespanha, and D.J. Kriegman, “Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection,” European Conf. Computer Vision, 1996, pp. 45-58.

[14] Y. Cui, D. Swets, and J. Weng, “Learning-Based Hand Sign Recognition Using SHOSLIF-M,” Int’l Conf. on Computer Vision, 1995, pp. 631-636.

[15] P. Hallinan, “A Low-Dimensional Representation of Human Faces for Arbitrary Lighting Conditions,” Proc. IEEE Conf. Computer Vision and Pattern Recognition, 1994, pp. 995-999.

[16] P. Hallinan, “A Deformable Model for Face Recognition Under Arbitrary Lighting Conditions,” PhD thesis, Harvard Univ., 1995.

[17] R. Brunelli and T. Poggio, “Face Recognition: Features vs. Templates,” IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 15, no. 10, pp. 1,042-1,053, Oct. 1993.

[18] J.M. Gilbert and W. Yang, “A Real-Time Face Recognition System Using Custom VLSI Hardware,” Proc. IEEE Workshop on Computer Architectures for Machine Perception, 1993, pp. 58-66.

[19] B.K.P. Horn, Computer Vision. Cambridge, Mass.: MIT Press, 1986.

[20] W.M. Silver, Determining Shape and Reflectance Using Multiple Images, PhD thesis, Massachusetts Institute of Technology, 1980.

APPENDICES

Appendix A

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install build-essential cmake pkg-config
$ sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
$ sudo apt-get install libxvidcore-dev libx264-dev
$ sudo apt-get install libgtk2.0-dev
$ sudo apt-get install libatlas-base-dev gfortran
$ sudo apt-get install python2.7-dev python3-dev
$ cd ~
$ wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.1.0.zip
$ unzip opencv.zip
$ wget -O opencv_contrib.zip https://github.com/Itseez/opencv_contrib/archive/3.1.0.zip
$ unzip opencv_contrib.zip
$ wget https://bootstrap.pypa.io/get-pip.py
$ sudo python get-pip.py
$ sudo pip install virtualenv virtualenvwrapper
$ sudo rm -rf ~/.cache/pip
$ echo -e "\n# virtualenv and virtualenvwrapper" >> ~/.profile
$ echo "export WORKON_HOME=$HOME/.virtualenvs" >> ~/.profile
$ echo "source /usr/local/bin/virtualenvwrapper.sh" >> ~/.profile
$ source ~/.profile
$ mkvirtualenv cv -p python3
$ source ~/.profile
$ workon cv
$ pip install numpy
$ cd ~/opencv-3.1.0/
```

```
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D INSTALL_PYTHON_EXAMPLES=ON \
    -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.1.0/modules \
    -D BUILD_EXAMPLES=ON ..
$ make clean
$ make
$ sudo make install
$ sudo ldconfig
$ ls -l /usr/local/lib/python3.4/site-packages/
total 1852
-rw-r--r-- 1 root staff 1895932 Mar 20 21:51 cv2.cpython-34m.so
$ cd /usr/local/lib/python3.4/site-packages/
$ sudo mv cv2.cpython-34m.so cv2.so
$ cd ~/.virtualenvs/cv/lib/python3.4/site-packages/
$ ln -s /usr/local/lib/python3.4/site-packages/cv2.so cv2.so
$ source ~/.profile
$ workon cv
$ python
>>> import cv2
>>> cv2.__version__
'3.1.0'
>>>
$ rm -rf opencv-3.1.0 opencv_contrib-3.1.0
```

Appendix B

```
import argparse
import datetime
import imutils
import time
import cv2

ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video", help="path to the video file")
ap.add_argument("-a", "--min-area", type=int, default=500,
help="minimum area size")
args = vars(ap.parse_args())


camera = cv2.VideoCapture(0)
time.sleep(0.25)

firstFrame = None

while True:
      (grabbed, frame) = camera.read()
      text = "Unoccupied"

      frame = imutils.resize(frame, width=500)
      gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
      gray = cv2.GaussianBlur(gray, (21, 21), 0)

      if firstFrame is None:
            firstFrame = gray
            continue

      frameDelta = cv2.absdiff(firstFrame, gray)
      thresh = cv2.threshold(frameDelta, 25, 255, cv2.THRESH_BINARY)[1]

      thresh = cv2.dilate(thresh, None, iterations=2)
      (cnts, _) = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
            cv2.CHAIN_APPROX_SIMPLE)

      for c in cnts:
            if cv2.contourArea(c) < args["min_area"]:
                  continue

            (x, y, w, h) = cv2.boundingRect(c)
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0),
2)
            text = "Occupied"

      cv2.putText(frame, "Room Status: {}".format(text), (10, 20),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
```

```
      cv2.putText(frame, datetime.datetime.now().strftime("%A %d %B %Y
%I:%M:%S%p"),
            (10, frame.shape[0] - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.35,
(0, 0, 255), 1)

      cv2.imshow("Security Feed", frame)
      cv2.imshow("Thresh", thresh)
      cv2.imshow("Frame Delta", frameDelta)
      key = cv2.waitKey(1) & 0xFF

      if key == ord("q"):
            break

camera.release()
cv2.destroyAllWindows()
```

Appendix C

```
cascades/haarcascade_frontalface_default.xml

from pyimagesearch.facedetector import FaceDetector
from pyimagesearch import imutils
import argparse
import cv2

ap = argparse.ArgumentParser()
ap.add_argument("-f", "--face", required = True,
      help = "path to where the face cascade resides")
args = vars(ap.parse_args())

fd = FaceDetector(args["face"])

camera = cv2.VideoCapture(0)

while True:
      (grabbed, frame) = camera.read()

      if args.get("video") and not grabbed:
            break

      frame = imutils.resize(frame, width = 300)
      gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

      faceRects = fd.detect(gray, scaleFactor = 1.1, minNeighbors = 5,
            minSize = (30, 30))
      frameClone = frame.copy()

      for (fX, fY, fW, fH) in faceRects:
            cv2.rectangle(frameClone, (fX, fY), (fX + fW, fY + fH), (0,
255, 0), 2)

      cv2.imshow("Face", frameClone)

      if cv2.waitKey(1) & 0xFF == ord("q"):
            break

camera.release()
cv2.destroyAllWindows()
```

Appendix D

```
import argparse
import datetime
import imutils
import cv2
import time
import numpy as np
import RPi.GPIO as gpio
from pyimagesearch.facedetector import FaceDetector

ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video", help="path to the video file")
ap.add_argument("-a", "--min-area", type=int, default=3000,
help="minimum area size")
ap.add_argument("-f", "--face", required = True,
     help = "path to where the face cascade resides")
args = vars(ap.parse args())

gpio.setmode(gpio.BOARD)
gpio.setwarnings(False)

gpio.setup(7, gpio.OUT, initial=0)
gpio.setup(11, gpio.OUT, initial=0)
gpio.setup(13, gpio.OUT, initial=0)
gpio.setup(15, gpio.IN)

camera = cv2.VideoCapture(0)
time.sleep(0.25)
timer = 0

firstFrame = None
fd = FaceDetector(args["face"])

while True:
        gpio.output(7, 0)
        gpio.output(11, 0)
        gpio.output(13, 1)
        if gpio.input(15):
                gpio.output(13, 0)
                while timer<10:
                # grab the current frame and initialize the
occupied/unoccupied
                # text
                        timer+= 0.25
                        (grabbed, frame) = camera.read()
                        text = "Unoccupied"

                # resize the frame, convert it to grayscale, and blur
it
                        frame = imutils.resize(frame, width=500)
```

```
                        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
                        gray = cv2.GaussianBlur(gray, (21, 21), 0)

                # if the first frame is None, initialize it
                        if firstFrame is None:
                                firstFrame = gray
                                continue

                # compute the absolute difference between the current
frame and
                # first frame
                        frameDelta = cv2.absdiff(firstFrame, gray)
                        thresh = cv2.threshold(frameDelta, 25, 255,
cv2.THRESH BINARY)[1]

                # dilate the thresholded image to fill in holes, then
find contours
                # on thresholded image
                        thresh = cv2.dilate(thresh, None,
iterations=2)
                        (_, cnts, _) = cv2.findContours(thresh.copy(),
cv2.RETR EXTERNAL,
                                cv2.CHAIN_APPROX_SIMPLE)

                # loop over the contours
                        for c in cnts:
                 # if the contour is too small, ignore it
                                if cv2.contourArea(c) <
args["min_area"]:
                                        continue

                        # compute the bounding box for the contour,
draw it on the frame,
                        # and update the text
                                (x, y, w, h) = cv2.boundingRect(c)
                                cv2.rectangle(frame, (x, y), (x + w, y
+ h), (0, 255, 0), 2)
                                text = "Occupied"
                                gpio.output(7, 1)
                                gpio.output(11, 0)
                                gpio.output(13, 1)
                                # resize the frame and convert it to
grayscale

                                gpio.output(13, 0)
                                frame = imutils.resize(frame, width =
300)
                                gray = cv2.cvtColor(frame,
cv2.COLOR BGR2GRAY)

                                # detect faces in the image and then
clone the frame
```

```
                                # so that we can draw on it
                                faceRects = fd.detect(gray,
scaleFactor = 1.1, minNeighbors = 5,
                                        minSize = (30, 30))
                                frameClone = frame.copy()

                                # loop over the face bounding boxes
and draw them
                                for (fX, fY, fW, fH) in faceRects:
                                        cv2.rectangle(frameClone, (fX,
fY), (fX + fW, fY + fH), (0, 255, 0), 2)
                                        cv2.imshow("Face", frameClone)
                                # show our detected faces



                        # draw the text and timestamp on the frame
                        cv2.putText(frame, "Room Status:
{}".format(text), (10, 20),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0,
255), 2)
                        cv2.putText(frame,
datetime.datetime.now().strftime("%A %d %B %Y %I:%M:%S%p"),
                                (10, frame.shape[0] - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.35, (0, 0, 255), 1)

                # show the frame and record if the user presses a key
                        cv2.imshow("Security Feed", frame)
                        cv2.imshow("Frame Delta", frameDelta)

                # if the `q` key is pressed, break from the lop
                        if cv2.waitKey(1) & 0xFF == ord("q"):
                                break

                if cv2.waitKey(1) & 0xFF == ord("q"):
                        break

                firstFrame = None
                gpio.output(7, 0)
                gpio.output(11, 1)
                gpio.output(13, 1)
                time.sleep(0.5)
                gpio.output(13, 0)
                time.sleep(2)


                timer = 0
        else:
                gpio.output(13, 1)
```

```
camera.release()
cv2.destroyAllWindows()
gpio.cleanup()
```