IR-DEPTH FACE DETECTION AND LIP LOCALIZATION USING KINECT V2

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Katherine KaYan Fong

June 2015

COMMITTEE MEMBERSHIP

TITLE:	IR-Depth Face Detection and Lip Localization Using Kinect V2
AUTHOR:	Katherine KaYan Fong
DATE SUBMITTED:	June 2015
COMMITTEE CHAIR:	Xiaozheng (Jane) Zhang, Ph.D. Professor of Electrical Engineering, Adviser
COMMITTEE MEMBER:	Wayne Pilkington, Ph.D. Associate Professor of Electrical Engineering
COMMITTEE MEMBER:	John Saghri, Ph.D. Professor of Electrical Engineering

ABSTRACT

IR-Depth Face Detection and Lip Localization Using Kinect V2

Katherine KaYan Fong

Face recognition and lip localization are two main building blocks in the development of audio visual automatic speech recognition systems (AV-ASR). In many earlier works, face recognition and lip localization were conducted in uniform lighting conditions with simple backgrounds. However, such conditions are seldom the case in real world applications. In this paper, we present an approach to face recognition and lip localization that is invariant to lighting conditions. This is done by employing infrared and depth images captured by the Kinect V2 device. First we present the use of infrared images for face detection. Second, we use the face's inherent depth information to reduce the search area for the lips by developing a nose point detection. Third, we further reduce the search area by using a depth segmentation algorithm to separate the face from its background. Finally, with the reduced search range, we present a method for lip localization based on depth gradients. Experimental results demonstrated an accuracy of 100% for face detection, and 96% for lip localization.

Keywords: Face detection, lip localization, infrared (IR), Audio-visual automatic speech recognition, data fusion, depth information, Microsoft Kinect

ACKNOWLEDGMENTS

Completion of this thesis highlights my academic career in Cal Poly, San Luis Obispo. I would like to thank all my professors, friends and classmates for their influence and encouragements.

I would like to especially thank Dr. Jane Zhang for her endless support and dedication to this thesis. I would also like to thank Dr. Wayne Pilkington and Dr. John Saghri for serving on my thesis committee. Additionally, I would like to thank the entire Electrical Engineering faculty for teaching valuable knowledge and skills that I can use to tackle real world problems.

I would like to thank my mom and dad for their constant support, and for giving me this opportunity to continue to pursue my dreams for higher education. To my sisters, Carmen and Kelly, I would like to thank you for being my editors, tutors and advisers from elementary school to college. To my best friend Kenneth, thank you for teaching me numerous life lessons. To all my friends and family, thank you for the constant support and unwavering dedication that you have for me.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

1. Introduction

1.1. Background

Automatic Speech Recognition (ASR) plays a pivotal role in human-computer interfaces. Applications of this topic can range from education, entertainment and communication and beyond [1]. For instance, ASR can serve as an educational tool to aid children in language development [2] and as a communication tool for people with hearing impairment. Alternatively, ASR can be utilized in entertainment by controlling video games using voice commands. Undoubtedly, such applications can improve people's quality of life, however, before any of these applications can work properly, an accurate and reliable ASR system is required.

Traditionally, ASR relies on acoustic-only information. Yet, this type of system suffers from performance degradation due to environmental noise [3]. To remedy this, visual and audio modalities of the speech are combined to create a bimodal solution called audiovisual automatic speech recognizer (AV-ASR) [4]. In a typical AV-ASR system shown in Figure 1.1, features are extracted from their respective inputs before fusion. While the inclusion of visual information adds accuracy to the ASR system, this opens up a new challenge of building a robust visual front end.

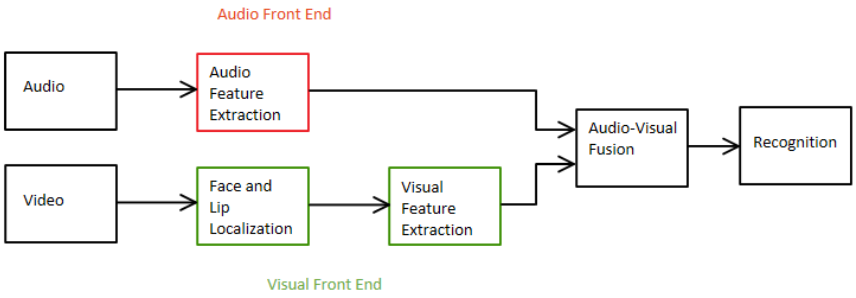


Figure 1.1: AV-ASR System Overview

1.2. Related Work: Visual Feature Extraction

In general, research on visual feature extraction can be classified into two main categories: appearance based, and shaped based. Appearance based method assumes all pixels within a region of interest (ROI) have information about the speech [4]. Usually, this ROI contains image pixels of the mouth. Once a ROI has been established, linear transformation such as Principal Component Analysis [5], or Discrete Cosine Transform (DCT) [6] [7] is used to reduce feature dimensionality. Shaped based method on the other hand, assumes that most of the speech information is contained in the contours of the lips. In such methods, geometric parameters such as the width, height [8], perimeter and area [9] of the lips are utilized. However, accurately extracting lip contours and the associated geometric parameters is challenging especially under varying lighting conditions. Despite the different approach in feature extraction, there is currently no consensus as to which method performs better. Both methods are employed by several ongoing research work [10] [11].

1.3. Related Work: Face and Lip Localization

Before visual features can be extracted from an input video, a robust face and lip detection is required. Galatas et al. [4] proposes the use of Viola Jones Algorithm to detect the face and another Viola Jones Algorithm pass to localize the mouth region within the face. However, its limitation includes a constant distance between the speaker and the recording device, controlled illumination and a simple background. Similarly, Navarathna et al. [12] also uses the Viola Jones method to detect both the face and lips, but instead of utilizing the entire face image, the lower half of the face was used. Unlike

the use of a simple background from Galatas et al, Navarathna et al. used images that were recorded within a car environment. Navarathna et al. achieved a detection rate of 96.92% and 92.36% for the face and mouth respectively. Navarathna et al. reported a false detection rate of 0.94% and 26.3% for the face and mouth respectively, where the rate was computed as a ratio of the sum of incorrect detections over the sum of all detections.

1.4. Challenges

While research in face detection has been expanding, challenges due to varying lighting conditions, presence of facial features such as beards, and glasses remain [13]. One way to solve such challenges is to use light invariant imaging methods. Amongst the various illumination invariant face detections, active near infrared (NIR) imaging technique [14] can be used. NIR is a source of electromagnetic radiation within the beginning of the infrared spectrum range, and borders the visible light spectrum. Its advantages include the ability to be reflected by objects, penetrate glass, and serve as an active illumination source [15]. Such NIR imaging technique can be found on the Kinect V2, a motion sensing device from Microsoft that also provides depth data from the same IR sensor and color from an additional image sensor. While most face detection utilizes color images as its input, illumination variance remains a challenge because illumination can alter the color intensity in an image. This change stems from the introduction of various shades and shadows to the face when the illumination source comes from light fixtures or sunlight. Especially with low light conditions, some faces may have minimal intensity contrast with the dark background, thus, causing face detection to fail. IR

images on the contrary remains the same despite changes in illumination. Even in the dark, IR images can capture distinct details of the face. To alleviate the challenges of face detection from various lighting conditions, this paper will deviate from the traditional color video input and instead use infrared (IR) video.

While many research studies on face detection have been traditionally conducted using color images, there has been a steady flow of research incorporating the use of NIR images in recent years. Of the numerous researches, there are several notable works worth mentioning. Li et al. [14] proposes extracting local binary pattern (LBP) features from NIR images for face detection. In a series of experiments, Li et al. achieve a recognition rate of 91.9 percent by incorporating Adaboost with LBP, 32 percent for NIR image with PCA and 62.4 percent for NIR image with LDA. Based on these results, Li et al. concluded that the use of learning-based methods in conjunction with the NIR images offer a good solution for highly accurate face recognition. The biggest constraint reported was that this method is only suitable for indoor applications because NIR images degrade in the sunlight. In other works, Socolinsky et al. [16] conducted face recognition using the PCA algorithm on both NIR and color images as a function of light level. The illumination ranges from bright lighting to near complete darkness. As expected, color image face recognition performed poorly in low light conditions compared to the NIR images, but both perform well for bright light conditions. Hizem et al. [17] also conducted a performance comparison between color and NIR images, and concluded that NIR images perform better when it comes to illumination changes.

1.5. Thesis Organization

The goal of this paper is to develop a robust face and lip localization algorithm independent of lighting conditions. This algorithm is developed as the visual front end of an AV-ASR system. The general system overview is shown in Figure 1.2. To obtain input videos that are lighting invariant, the Microsoft Kinect V2 is used to capture IR and depth data. Chapter 2 presents an introduction to the database and the collection process along with description of how data is represented within the IR and depth data. Chapter 3 discusses how face detection is applied and analyzed from the IR video input stream. In Chapter 4, the lip localization algorithm is introduced and analyzed. Finally, the paper is summarized and concluded in Chapter 5.

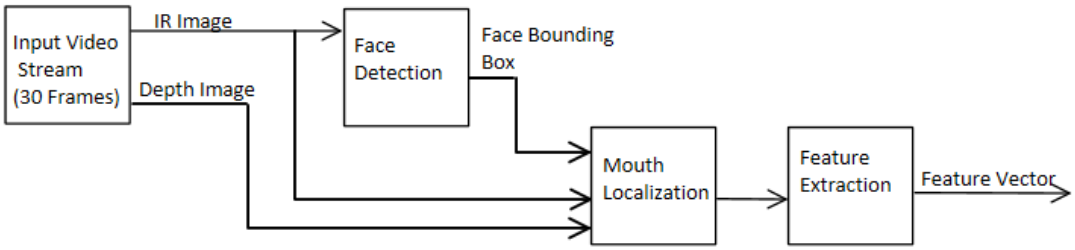


Figure 1.2: System Overview

2. Input Video Stream

2.1 Database

To test the hypothesis that depth and infrared information are useful for face detection, a database was collected. This database consists of two male and two female English speakers. Each subject participated in two to three sessions with various background and lighting conditions as summarized in Table 2.1. In each session, the subject was asked to recite the digits from 0-9 in English in front of the Kinect V2. For each digit, 30 frames of color, depth and IR data were captured simultaneously at 30 frames per second (fps). Additionally, each person faces directly at the Kinect within 0.5-4.5 meters from the Kinect. Sample color images from the database are shown in Figure 2.1.

Figure 2.1: Sample Color Images Captured from Kinect V2

Table 2.1: Database Summary, where P# Denotes the Session Number

P#	Gender	Extra Items on Facial Area	Surroundings	Lighting Conditions
1	Female	Glasses, Hair	Indoor	Sunlight
2	Female	None	Indoor	Artificial Lighting At Night
3	Female	Glasses	Indoor	Sunlight
4	Female	None	Indoor	Artificial Lighting At Night
5	Female	None	Outdoor	Sunlight
6	Male	None	Indoor	Sunlight
7	Male	None	Indoor	Sunlight
8	Male	Beard	Indoor	Sunlight
9	Male	Beard	Indoor	Sunlight

2.1.1 Depth Data

In each recording session, the raw depth data for each of the 30 frames are captured onto individual text files. The depth data, shown in the z axis in Figure 2.2, represents the distance from the Kinect V2 to the object of interest using the time of flight (ToF) technology. Time of Flight technology in imaging devices computes distances by measuring the time it takes for a light signal to travel between the device and the object.

Figure 2.2: Kinect Coordinate System [18]

To measure depth, the Kinect sends a light pulse to the object, and measures how long it takes for the light pulse to come back to the Kinect, as shown in Figure 2.3.

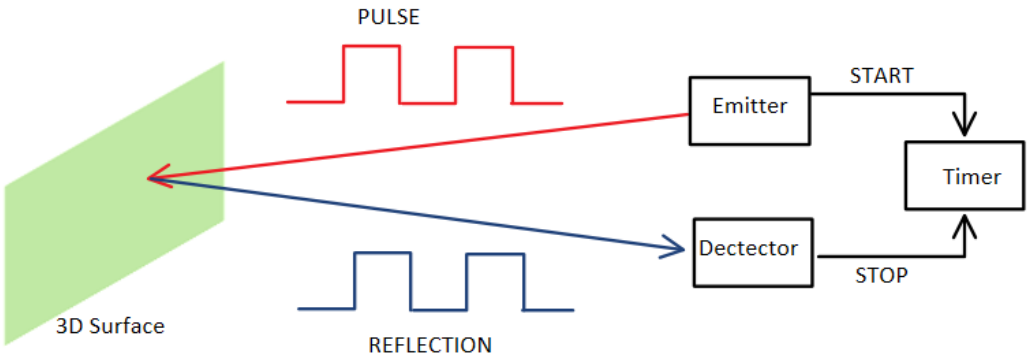


Figure 2.3: Illustration of Time of Flight Technology [19]

Since the speed of light is known, and the time duration is measured, the Kinect can now find the distance to an object. Each depth value is represented as 16 bits and is in the units of millimeters (mm). Once the depth values are recorded onto the text files, they are imported into MATLAB and are converted into depth images using the reshape MATLAB function. The resulting depth image, such as the depth image in Figure 2.4 has a resolution of 512 by 424.

Figure 2.4: Sample Depth Image with a Depth Range [500 2200]

One of the constraints of Kinect is that accurate depth range is limited from 500 to 4500mm. Other pixels that fall out of this range are disregarded. For example, the depth image shown in Figure 2.4 shows all pixel values that are within the range of 500 to 4500. In actuality, all the accurate depth values in Figure 2.3 ranges from 500 to 2200mm, thus histogram stretching is used to visualize the depth image in more detail. Another limitation of the Kinect is inaccurate depth information for reflective surfaces. For instance, the two black clusters appearing in the face area of the subject in Figure 2.4 were due to the glasses' surface reflectivity.

2.1.2 IR Data

Similar to the depth data, the 16 bit IR data for each of the 30 frames are captured onto individual text files. After the conversion to MATLAB, the IR images have a resolution of 512 by 424. This IR data is captured using the new active infrared capabilities of the Kinect. What this new capability allows the Kinect to do is to capture

images independent of light conditions. This means that the Kinect can capture IR images in the dark. An example of an IR image is shown in Figure 2.5.

Figure 2.5: Sample IR Image

2.1.3 Coordinate System

The XY coordinate systems are the same for IR and Depth [18]. X represents the column while Y represents the row. The origin is at the top left of the corner, and Y increases downwards while X increase rightwards as shown in Figure 2.6.

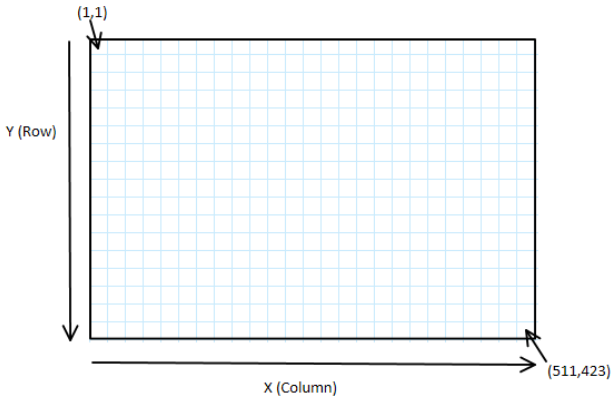


Figure 2.6: XY Coordinate System

The coordinate Z increases out in the direction that the Kinect is facing as shown in Figure 2.7.

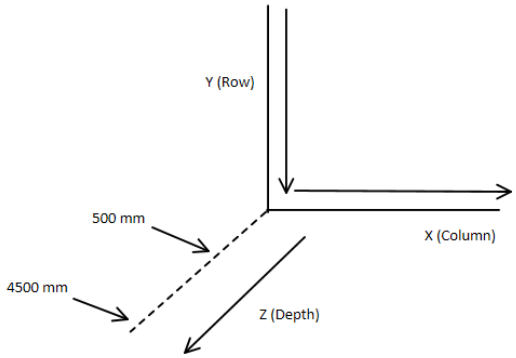


Figure 2.7: XYZ Coordinate System

2.2 Video Data Capture

For each video data capture session, the time for speaking one digit is recorded. Since IR and depth both have a frame rate of 30 fps, each stream recorded 30 consecutive frames. Once the data were captured, the images were stitched together using a video editor; with the duration amounts trimmed to 1 second for each stream. The video data captures were all completed using Visual Studio Express for Desktop 2013 in conjunction with the Kinect V2.

2.2.1 Visual Studio

The programming language used in Visual Studio was C sharp. Since the focus of this thesis is on image processing, the explanation of how the code was created will not be discussed. Instead, the code will be included in the appendix.

3. Face Detection

3.1 Viola Jones Algorithm: Overview

The Viola Jones framework serves as the face detection algorithm for our design. We use this algorithm because the Viola Jones algorithm provides rapid and accurate detection, with accuracy rates that are well above 90% [20]. The keys to the successes of this algorithm are its three main contributions: features from integral images, a variant of the Adaboost algorithm and cascade classifiers [21]. Each contribution will be briefly explained in the following subsections. The limitations of this algorithm are discussed in section 3.2.

3.1.1 Features

Detection within the Viola Jones algorithm begins with computation of simple rectangular features [21] that serve as simple classifiers. The features used in the algorithm are rectangular filters that resembles the Haar wavelets. These features are computed using integral images. Each pixel of the integral image I(x,y) is represented by a sum from above and left of the pixel location (x,y, inclusive) from the original image i(x,y) in Figure 3.1:

$$I(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{3.1}$$

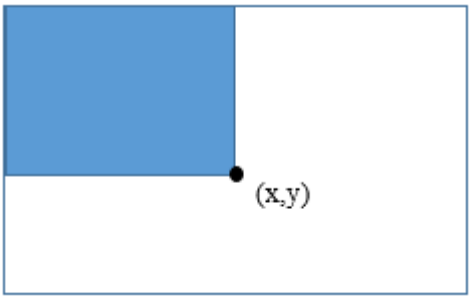


Figure 3.1: Integral Image at Point (x,y) [21]

Once the integral images are computed, different types of rectangular features can be utilized as shown in Figure 3.2. A two-rectangle feature is computed by taking the difference between the pixel value sums in two rectangular regions, while a three-rectangle feature is computed by the difference between the sum of the two outer rectangles with the inside rectangle. Lastly, a four-rectangle feature is computed by the difference between the pixel sums of the diagonal pairs of rectangles.

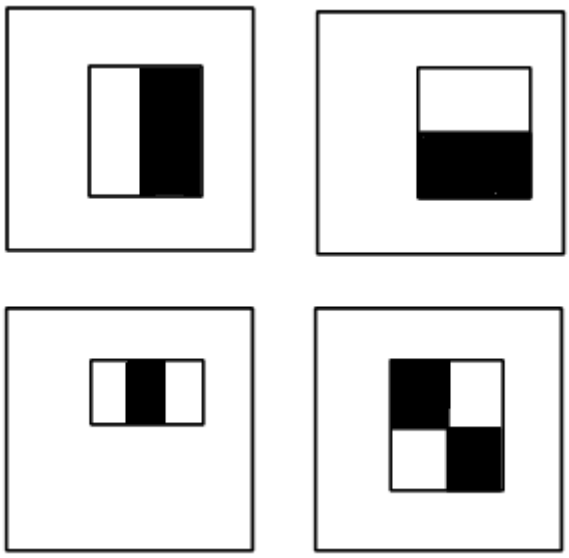


Figure 3.2: Example Rectangle Features: The Sum of the Pixels that Lie Within the White Rectangles are Subtracted from the Sum of the Pixels in the Black Rectangles [21]

3.1.2 Adaboost Algorithm

With so many variations of rectangle features available, an algorithm is required to choose the features that yield the best results. With that in mind, Viola et al. chose a variant of Adaboost to select features and to train a classifier [21]. Adaboost is a machine learning algorithm that trains a set of weak classifiers to develop a strong linear classifier. The process begins with one weak classifier. After detection, the weights of the misclassified objects change before entering the next iteration of a weak classifier. This change in weight allows the next classifier to pay attention to the misclassified objects. From experimentation, Viola et al. found that increasing the amount of rectangular features used with the Adaboost algorithm did improve detection rates. However, additional features increased computation time.

3.1.3 Cascaded Classifier

To reduce computation time, a cascade of Adaboost classifiers were implemented. While each stage of the cascade contains different amount of classifiers, the goal of each stage is to remove false detections. As such, the detection framework remains computationally efficient while maintaining accurate recognitions.

3.2 Viola Jones Algorithm: Limitations

The Viola Jones Algorithm has three main constraints. First, the training of the algorithm was performed on frontal, and upright faces. For that reason, the algorithm is not reliable for face orientation beyond the initial facial orientation and rotation. Second,

detection performance degrades when illumination varies. For instance, the algorithm failed to detect the correct face when the face is darker than the background. Lastly, faces that are significantly occluded by other objects can also cause failed detections.

Since this paper focuses on using the Viola Jones face detection algorithm, we use the first limitation to our advantage for lip localization in Chapter 4. For instance, since the faces are frontal and upright, we can assume the nose will have the smallest distance to the Kinect with respect to the face. Once we find the nose location, we can use this information to reduce the search area for the lips. The second limitation is addressed by using infrared images that are illumination invariant. Finally, the third limitation will not be addressed in this paper.

3.3 Viola Jones Face Detection: Implementation

To implement the Viola Jones Face Detection algorithm, the constructor, vision.CascadeObjectDetector, which resides within MATLAB's Computer Vision System Toolbox, was used. This constructor creates an object detector (based on Viola Jones) that detects certain objects defined by the classification model. Each model that MATLAB incorporates is trained for that specific object model. These models include: face, eyes, mouth, nose and etc. In this case, we choose the FrontalFaceCART classification model with the assumption that the built-in training set is sufficient to detect the faces within the input images. To set up the constructor, we use FrontalFaceCART as the input of vision.CascadeObjectDetector( ). Afterwards, the step( ) function from MATLAB is used to perform the detection. There are two inputs in the step( ) function, the detector and the input image. Once implemented, the step( ) function

returns an M x 4 matrix that contains bounding boxes of M detected objects. Each row contains a separate bounding box parameter set that consists of the upper left corner pixel location and the size of the box. To visually see all the detected face regions, the function insertObjectAnnotation( ) from MATLAB is used. The inputs of the insertObjectAnnotation() function consist of the image, shape, bounding box and the label. Once implemented, it outputs an annotated shape, along with the label onto the original image. For two of the inputs, we use the input image and its detected bounding boxes. For the other two inputs, we used rectangle as the shape and "face" as our label. Once the function is implemented, insertObjectAnnotation() inserts rectangles on all the detected objects of the input image.

3.4 Viola Jones Face Detection: IR Image and Color Image Experiment and Result

In this section, we conduct a Viola Jones Face Detection experiment between IR image and color images to validate why we use IR images instead of color images in this thesis. A separate set of images were used instead of the database included in this report.

3.4.1 IR and Color Image Test Set

Similar to the video database capture, each person faces directly at the Kinect within 0.5-4.5 meters from the Kinect. The subjects were asked to speak in front of the Kinect while individual images were captured. Additionally, a coordinate map is captured along with color, IR and depth. This test set consist of 108 images taken in various backgrounds and lighting conditions from dark to light.

3.4.2 Coordinate Map

The coordinate map used for this application takes on the same dimension as color image: 1920 x 1080 pixels. Because the color sensor and the sensor that creates depth and IR data are not located in the same location, the corresponding images have different viewpoints. The coordinate map serves as a roadmap to align the two sensors together by generating a roadmap that maps depth space onto the color space. More detail of how this was implemented can be seen in the appendix, under the Visual Studio Code.

3.4.3 Color and IR Image Alignment Implementation

In this section, we talk about the implementation briefly, as more details can be found in the MATLAB code located in the appendix. Before we begin the image alignment, we want the size of both color and IR images to be as similar as possible. To accomplish this, the MATLAB function, downsample( ), was applied to the coordinate map and the color image. The resulting size became 640 x 360, which is similar to the IR image size of 640 x 424. Since the values within the coordinate map corresponds to the pixel location of the IR map location, we map the IR value onto the coordinate map to create a new IR image that aligns with the color image. Lastly, to allow for fair comparison, the new IR image was re-quantized from 16 bits to 8 bits because the color image intensity are represented by 8 bits.

3.4.4 Face Detection Performance Comparison between Color and IR Image

Post image alignment, we implement the Viola Jones algorithm twice, once for color and another for the IR image based on section 3.3. In this experiment, we want to compare performance results between color and IR face detection in varying illumination conditions from light to dark. True positives denotes a bounding box that was correctly placed on the face, while false positive denotes incorrectly placed boxes. In addition, images with no bounding box correctly placed on the face were regarded as false negative. A summary of the test results is shown in Table 3.1. The results from Table 3.1 indicates the total number of occurrence of true positive, false positive, and false negative amongst all 108 images tested for each stream. Sample images of true positives for both streams are shown in Figure 3.3 and Figure 3.4. The results from both Figure 3.3 and Figure 3.4 indicate instances where that both streams perform correct face detection in bright and low light conditions. The face detection results in Figure 3.5 show both false and true positive results within the color image, while the IR image detects one true positive in dimly lit backgrounds. Of the 108 color images, 3 were reported as false negative, while there were 0 for all 108 IR images. To further investigate which lighting conditions yield the 3 false negatives, the results were divided into two lighting conditions: low and bright light in Table 3.2. Within the 108 images examined, 87 images were classified into bright lighting conditions and 21 images were classified into low lighting conditions. From Table 3.2, two false negatives from the color images originated from low light conditions, while one originated from bright light conditions. Sample images of false negative for the color stream are shown in in Figure 3.6 and Figure 3.7. In Figure 3.6, the low contrast between the face and the background

within the color image in low light conditions was the reason for the false negative detection. For Figure 3.7, the shadow on the face within the color image in bright light condition was the reason for false negative detection. However, in both cases, the IR equivalent in Figure 3.6 and Figure 3.7 were able to detect a true positive despite low light conditions or the additional shadows on the face. The results in Table 3.1 reveal that the IR image was capable of detecting the face in varying light conditions. For backgrounds with bright lighting conditions, both color and IR were able to obtain the correct face detection (Figure 3.6). Although the false positive for IR image was around 9%, we find a way to reduce this rate in Section 3.6.

Table 3.1: Viola Jones Face Detection Results between Color and IR Images

	Color			**IR**		
	False Positive	True Positive	False Negative	False Positive	True Positive	False Negative
Total	7	105	3	10	108	0
Percentage %	6.48%	97.22%	2.78%	9.26%	100.00%	0.00%

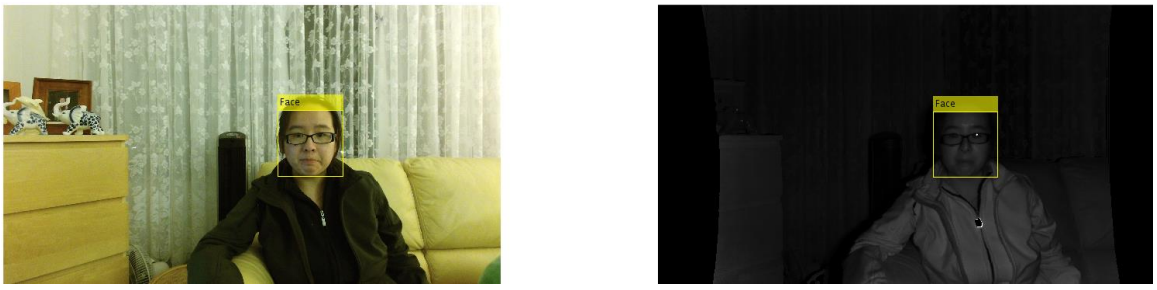


Figure 3.3: Viola Face Detection for (Left) Color Image and (Right) IR Image in Bright Lighting Condition with One True Positive Detected for Each

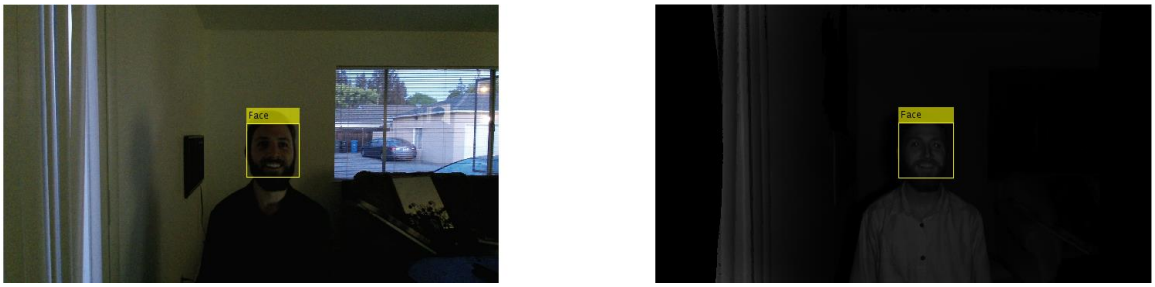


Figure 3.4: Viola Face Detection Results for (Left) Color Image and (Right) IR Image in Low Lighting Condition with One True Positive Detected for Each

Figure 3.5: Viola Face Detection for (Left) Color Image and (Right) IR Image in Low Lighting Condition with One True and False Positive on the Color Image, and One True Positive on the IR Image

Table 3.2: Viola Jones Face Detection Results between Color and IR Images Separated by Lighting Conditions

	Color			**IR**		
	False Positive	True Positive	False Negative	False Positive	True Positive	False Negative
Face with Bright Lighting Conditions						
Total	5	86	1	8	87	0
Percentage %	5.75%	98.85%	1.14%	9.20%	100.00%	0.00%
Face with Low Lighting Conditions						
Total	2	19	2	2	21	0
Percentage %	9.52%	90.48%	9.52%	9.52%	100.00%	0.00%

Figure 3.6: Viola Face Detection Results for (Left) Color Image and (Right) IR Image in Low Lighting Condition with One False Negative on the Color Image, and One True Positive on the IR Image

Figure 3.7: Viola Face Detection Results for (Left) Color Image and (Right) IR Image in Bright Lighting Condition with One False Negative on the Color Image, and One True Positive on the IR Image

3.5 Viola Jones Face Detection: IR Video Test Results Analysis

In this section, we use our original video database for testing. The results for the Viola Jones Face Detection are shown in Table 3.3. The P# corresponds to the session number described in Table 2.1. For this test, each P# consists of 300 IR images (30 IR images for each of the 10 digits). The criteria for true positive, false positive and false negative are the same as Section 3.4.4. A summary of the test results is shown in Table 3.3. The results from Table 3.3 indicates the total number of occurrence of true positive and false positive for each session (P#) and the total with all sessions combined. There

were no false negatives detected from the 2700 IR images that were used. Sample images of true positives are shown in Figure 3.8. For all the images in Figure 3.8, only one true positive result were detected in each image. Sample images of false positives are shown in Figure 3.9, where false positives occurred on different body locations while maintaining a true positive detection on the face. While this test set comprises only of IR images, the result of 100% recognition rate aligns with our results from Section 3.4.4. This also indicates that using IR images with Viola Jones face detection may improve accuracy despite illumination variance. Also, it should be noted that P5 was captured in an outdoor environment (subject was under a roof), which could explain why there were more false positives than the rest of the data.

Table 3.3: Viola Jones Face Detection Results for IR Images

	False Positive	**True Positive**
P#		
1	9	300
2	12	300
3	0	300
4	13	300
5	67	300
6	15	300
7	0	300
8	1	300
9	0	300
Total	117	2700
Percentage %	4.33%	100.00%

Figure 3.8: Viola Jones Face Detection with IR images Showing True Positive Recognition

Figure 3.9: Viola Jones Face Detection with Images Showing False and True Positive Recognition

3.6 Multiple Bounding Box Reduction Algorithm

Although there is a relatively small set of false positives from the Viola Jones IR face detection, further reduction can be achieved. Each input video sequence comprises of 30 images. With the false/true positive ratio as 117/2700=4% from Table 3.3, an assumption is made that most images within the video sequence will have no false detection. Additionally, since the video sequence contains 30 consecutive frames, the face bounding box size (and location) should be relatively similar since the person within one second cannot move much. With that in mind, for the images that have multiple faces detected, the final bounding box will resemble the median bounding box size of the video sequence.

To implement this algorithm, a *medianBoxSize* variable is created. For initialization, the *medianBoxSize* is set to the size of the facial bounding box from the first frame. If the initial frame contains multiple bounding boxes, the *medianBoxSize* will take on the value of the biggest box size. After each frame, the *medianBoxSize* is computed by taking the median of the current *medianBoxSize* value with the final facial bounding box value of the current frame. For image frames that have one facial bounding box candidate, that candidate becomes the final facial bounding box. For image frames that have multiple facial bounding boxes, the candidate that has a box size within threshold value of the *medianBoxSize* is used as the final facial bounding box. Through experimentation, using 10 pixels as the threshold gave the best result.

3.6.1 Multiple Bounding Box Reduction Algorithm Test Results

In this section, the results for the Multiple Face Bounding Algorithm compared with the Viola Jones Face Detection are shown in Table 3.4. False positives and true positives are judged using the same criteria from the previous section. The results from Table 3.4 indicates the total number of occurrence of true positive and false positive for each session (P#) and the total with all sessions combined. Sample images of true positives are shown in Figure 3.10, where column A contains the original IR image with multiple face detections and column B contains one true positive result for each image. The results shown in Table 3.4 indicates that the usage of prior knowledge can effectively reduce false positives.

Table 3.4: Viola Jones Face Detection and Multiple Face Bounding Box Results

	Viola Jones IR Face Detection		**Multiple Face Bounding Algorithm**	
	false positive	true positive	false positive	true positive
P#				
1	9	300	0	300
2	12	300	0	300
3	0	300	0	300
4	13	300	0	300
5	67	300	0	300
6	15	300	0	300
7	0	300	0	300
8	1	300	0	300
9	0	300	0	300
Total	117	2700	0	2700
Percentage %	4.33%	100.00%	0.00%	100.00%

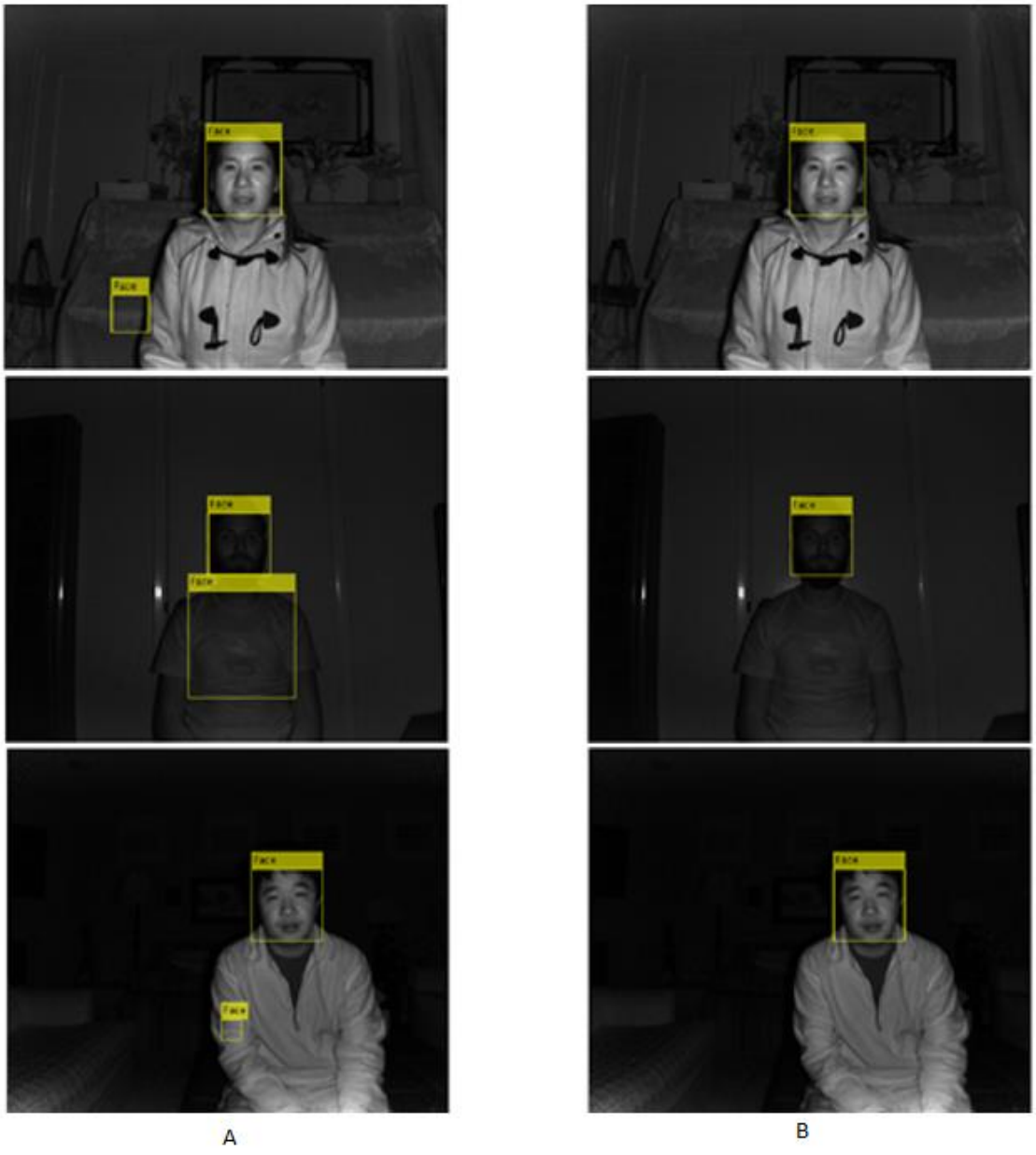


Figure 3.10: Column A Shows the Output of the Viola Jones IR Face Detection while Column B Shows the Output from the Multiple Bounding Box Algorithm

4. Feature Extraction: Lip Localization

4.1 Viola Jones Lip Detection using IR Image: Limitations

Given the excellent face detection results using the Viola Jones algorithm, it was used again for mouth detection from the detected IR face image. The implementation of the mouth detection was similar to Section 3.3, except the classification model used here is Mouth, and the corresponding label is "Mouth". After implementation, the detection results were not as desirable as its face predecessor. Table 4.1 lists the detection result using 300 facial IR images from P2. True positive denotes a mouth that is fully enclosed within the bounding box, and false positive denotes any bounding box that does not enclose the mouth. In addition, false positive also included bounding boxes that include partial mouth and nose. Images where no bounding box completely encloses the mouth are regarded as false negative. Sample images of false negatives within the IR face images are shown in Figure 4.1. The results from Figure 4.1 indicates how the Viola Jones Mouth Detection failed to detect a complete mouth when the subject has their mouth open. Sample images of true positives within the IR face image are shown in Figure 4.2. In each image, one true positive was detected on the subject's mouth. In addition, sample images of false positives are shown in the center and rightmost images of Figure 4.2, where most false positives occur on the eye area. While the results from Table 4.1 list a true positive detection rate of 85%, the false detection rate is above 100%. In addition, the false negative rate of 14.67% was mostly caused by subjects with open mouths. Some speculation on why there is a high false detection rate may stem from how similar the shape of the eyes are with regards to the mouth. In addition, the IR data are represented by 16 bits, while the training data for the mouth was most likely represented

by 8 bit. The wider range of value may allow for more combinations that mimics the recognition pattern of the trained mouth. From the results in Table 4.1, the ratio between false/true positive was 160/85.33=187.5%. With a ratio above 100%, the method of using multiple bounding boxes to reduce false detection will not work here. The Viola Jones mouth detection can detect a mouth from its facial IR image, but at a price of multiple false detections (Figure 4.2). Another method must be used to decrease the false detections.

Table 4.1: Viola Jones Mouth Detection Results for P2

	False Positive	**True Positive**	**False Negative**
Total	480	256	44
Percentage	160.00%	85.33%	14.67%

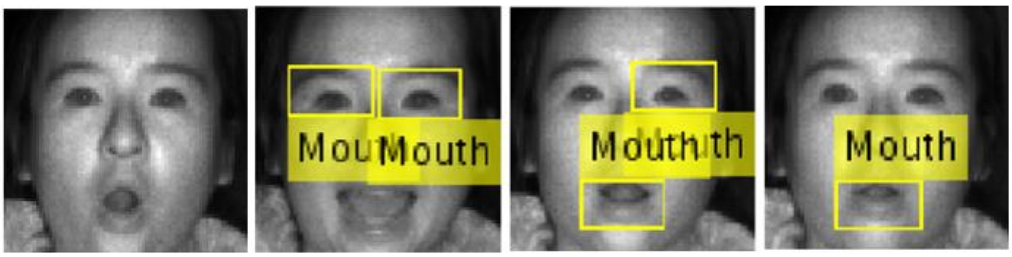


Figure 4.1: False Negative Detection Results from Viola Jones Mouth Detection Given IR Face Input in P2

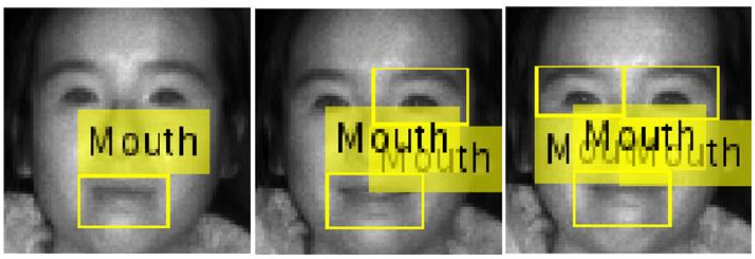


Figure 4.2: True Positive and False Positive Detection Results from Viola Jones Mouth Detection Given IR Face Input in P2

4.2 Depth Patterns among Various Facial Region

Since Viola Jones mouth detection does not work well with IR facial images, the use of depths images are explored next. Because both the IR data and depth data are created from the same IR sensor, both streams have the same coordinate system. Therefore, the location of the detected IR face can be directly applied to the depth image to create a facial depth image as shown in Figure 4.3.

Figure 4.3: Facial Depth Image

In section 3.2, we mentioned the idea that given the faces had to be frontal and upright for the Viola Jones algorithm, we can assume the following:

1. The nose is the closest point to the Kinect.
2. The nose lies in approximately the center of the face image.

From assumption 1, we can assume that the nose has the smallest depth value within the facial depth image. From assumption 2, the nose should never be located close to the border. With this assumption, we can remove potential nose point candidates that may be caused by hair and hat from the border. Such ideas are further explored in Section 4.4.

Once a nose point is found, its row location can be used to reduce the facial image to the lower facial area for lip localization. From a lower facial image, more assumptions for the ideal face are made:

3. At nose point, the nose tip has a higher elevation than the bottom of the nose.
4. There are sudden incline and decline of slope within the mouth depth region as shown in Figure 4.4. However, there are no sudden change of incline and decline from either side of the mouth to the edge of the face (towards the chin area).
5. From the bottom of the lip to the edge of the face, there are no sudden change of incline or decline.

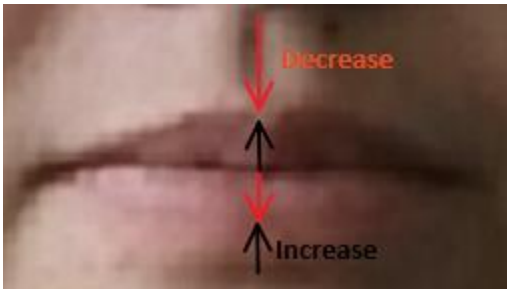


Figure 4.4: Depiction of How the Mouth Should Decrease and Increase in Depth Value in Certain Areas of the Mouth Region

Based on assumption 3, the depth value along the column of the nose point to the bottom of the nose should always decrease. From assumption 4, since there are no sudden incline or decline, there should be a gradual increase of depth to the edge of the face. The gradual increase of depth is caused by the TOF sensor. Likewise, from assumption 5, there should also be a decrease of depth from the bottom of the lip to the edge of the face. The following assumptions are shown as blue arrows in Figure 4.5.

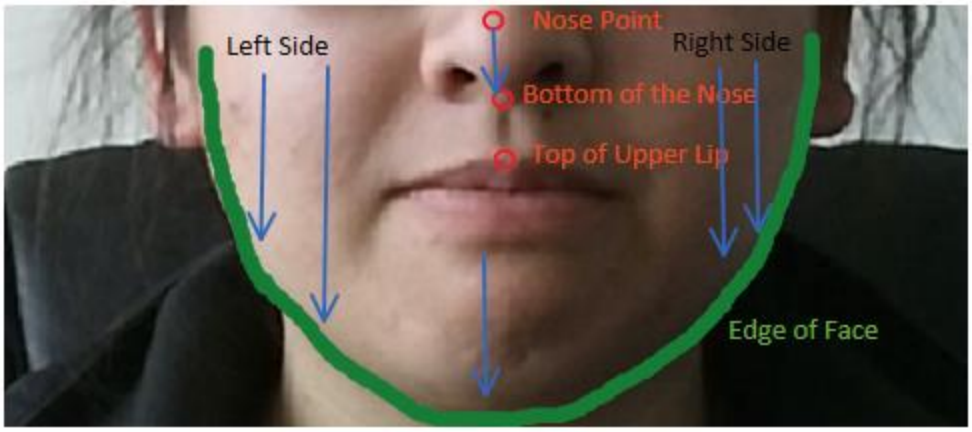


Figure 4.5: Depth Face Patterns for the Lower Facial Area

4.3 Undesirable Depth Pixels Effects

Before processing the depth images, several undesirable pixel occurrences are discussed. First, “Flying pixels” is an effect from TOF that occurs around the edges of objects when the depth level changes [22]. Second, “Zero pixels” occurs when the depth is out of range or when the light is incomprehensible due to highly reflective materials such as windows or edges. Lastly, “Jumpy pixels” occurs when some pixels “jump forward or back” each frames because of the difference in reflectivity in the material [23]. While the first two problems do not affect pixels with real depth value (500 to 4500mm), the latter “Jumpy pixels” do. To address the first two problems, depth data that are out the range of 500 to 4500mm are discarded. However, for the "Jumpy pixels” that may cause false detection, further processing is required.

4.4 Nose Point Detection

For a simple nose point detection algorithm shown in block diagram form in Figure 4.6, three main blocks are needed. Out of Range Filtering removes all indices with values that are out the range of 500 to 4500 mm. Next, the min( ) function from MATLAB is used to find the smallest depth value of the filtered depth face image. Once the minimum depth is found, the find( ) function from MATLAB is used to locate pixel location of the minimum depth value from the depth face image.

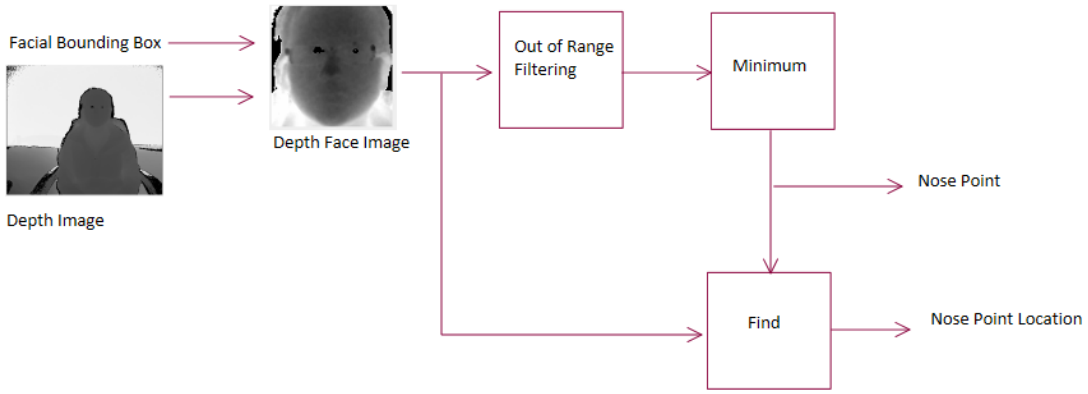


Figure 4.6: Simple Nose Point Detection Block Diagram

This simple nose point detection can accurately detect numerous nose points, however, false nose points detection occasionally occurs in the eyeglasses area and the hair. In the following subsections, we will address these problems by implementing various methods to combat such problems.

4.4.1 Median Filtered Nose Point Detection

From surveying the depth pixels surrounding the false nose points of the glasses area, it is observed that small groupings of sudden change in depth value occurred. Because depth value within the face area should not suddenly change in value, the culprit that caused such undesirable depth pixel effect was most likely the “jumpy pixel”. Unlike other undesirable pixel effects that can be filtered out by discarding depth value out of the range of 500 to 4500 mm, “Jumpy pixels” require extra steps to discard. Since such pixels act similarly to impulses, a median filter can be used to eliminate these sudden depth value changes. While the use of median filter can remove potential impulses, such filter can also remove true nose points. For that reason, additional steps are included to reintroduce any points that could be a true nose point after the use of the median filter. To implement a 2 dimensional median filter, the medfilt2( ) function from MATLAB was used. Its inputs include the matrix and a filter size. While the input matrix is the depth face image, the proper filter size remains unknown. Through experimentation of using filter sizes 1,3,5,7, and 9, results from Table 4.2 indicate that a filter size of 7 yields the best results (Figure 4.7). Therefore, a 2 dimensional median filter with a size of 7 is applied on the depth face image within the Median Filter Nose Point Detection in Figure 4.8.

Table 4.2: True Positive Nose Point Detection Results for Various 2D Median Filter Sizes

	Nose Points (recognition)				
	2D Median Filter Size				
Database	1	3	5	7	9
P2	300	300	300	300	279
P3	209	237	276	292	300
Total	509	537	576	592	579
	84.83%	89.50%	96.00%	98.67%	96.50%

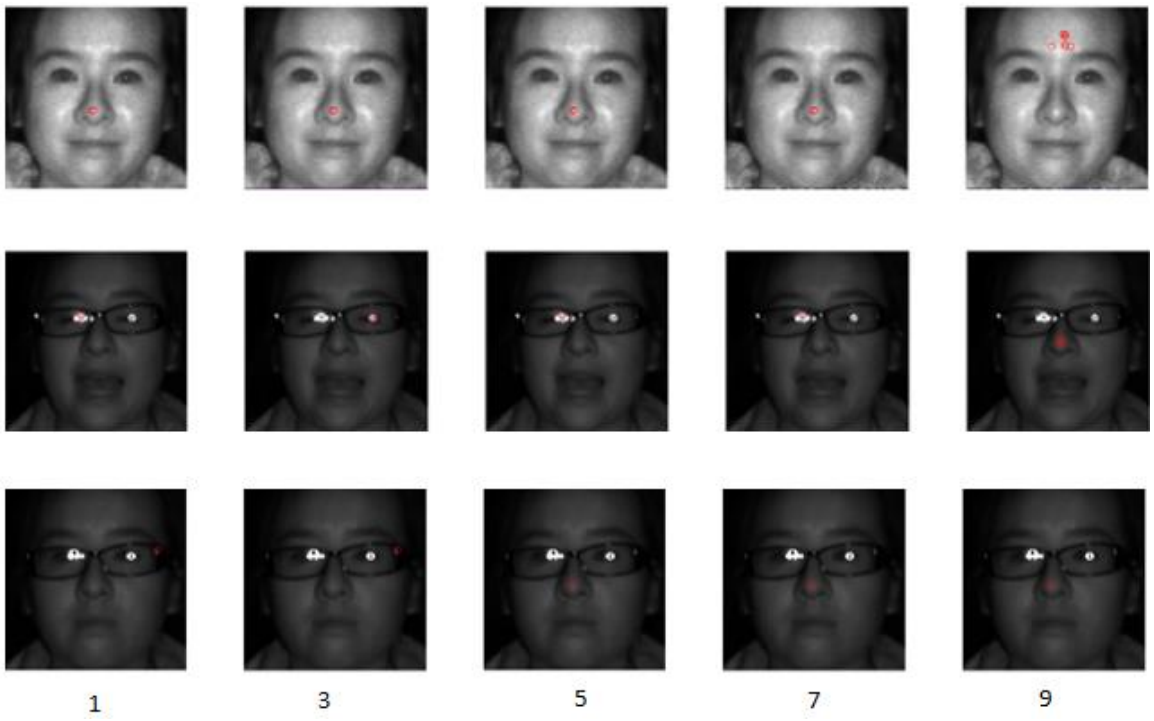


Figure 4.7: Nose Point Detection Results with 2D Median Filter of Various Filter Sizes of 1,3,5,7 and 9

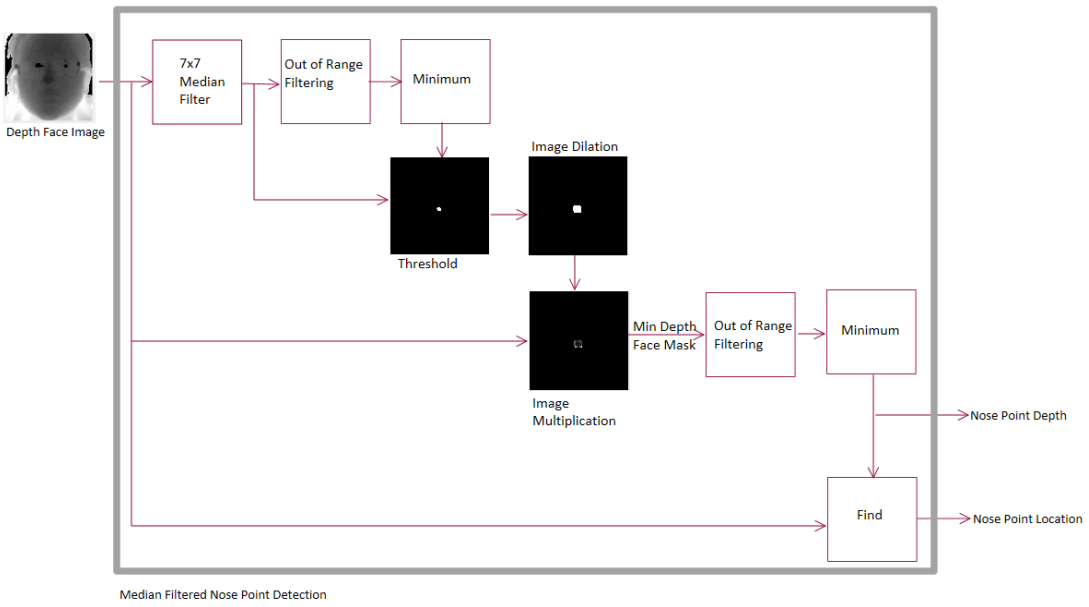


Figure 4.8: Median Filtered Nose Point Detection Block Diagram

Once filtered, the image goes through the out of range filtering, and finds the minimum value. To ensure that the original minimum depth value is not discarded from the median filter pass, the neighbors surrounding the minimum depth value are considered. First, a binary image was created from the median filter depth image with the minimum depth value as the threshold. Afterwards, a square shape structuring element with the same size as the median filter was used to dilate the image. The structuring element can be implemented by using the strel() function from MATLAB and specifying the shape and the size. Afterwards, imdilate() from MATLAB is used, with its input being the face depth image and the structuring element. The resulting dilated binary image is then multiplied against the original depth face image, element by element, to obtain an expanded search area for the nose point candidate. Finally, the out of the range filtering

minimum and find() functions are used to locate the new nose point and its corresponding pixel location.

4.4.2 Clear Border Nose Point Detection

While the median filtered nose point detection algorithm reduced false detection occurrences on glasses, another undesirable location occurs around the outer facial area: hair. To remedy this, the function Clear Border Nose Point Detection (Figure 4.9) was created to remove any minimum depth value that is within a certain border proximity threshold.

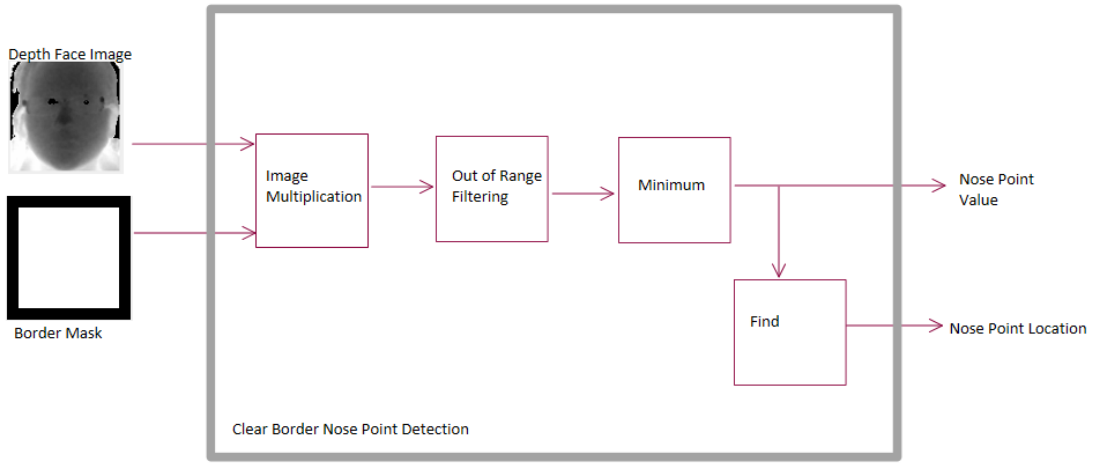


Figure 4.9: Clear Border Nose Point Detection Block Diagram

First, a Border Mask was created with the desired amount of border pixels to clear from each side of the image. Next, the border mask is multiplied to the depth face image, element by element. With the borders of the depth face image cleared out, the out of

range filtering, minimum and find blocks are used to locate the nose point depth value and location.

4.4.3 Final Nose Point Detection

While both detection algorithms work separately from each other, the algorithm in Figure 4.10 incorporates both blocks. The depth face image first passes through the median filtered nose point detection. From there, a minimum depth mask is made, where all nose point locations are declared as 1. Afterwards, the border mask is multiplied with the minimum in depth mask. If the sum of the resulting matrix is greater than (for cases with multiple nose point detections) or equal to 1, then the original nose point from the median filtered nose point detection is indeed the true nose point. If the sum was 0, then we assume that the nose pointed detected was around the border, and it would go through the clear border nose point detection to find the new nose point depth and location.

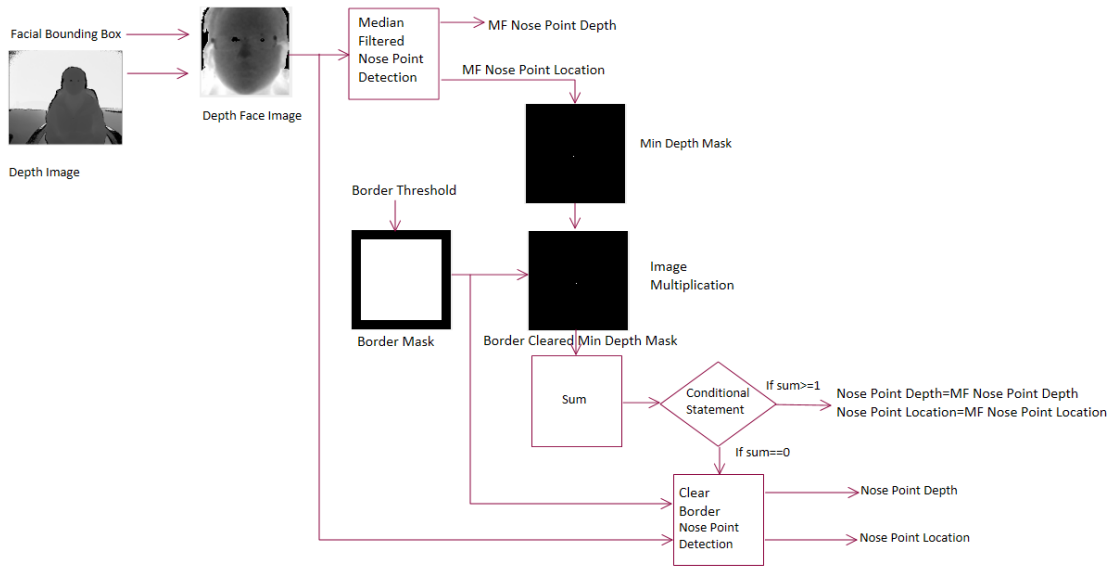


Figure 4.10: Final Nose Point Detection Block Diagram

4.5 Nose Point Detection Results

In this section, the results for the nose point detection algorithm are summarized in Table 4.3. The result from Table 4.3 indicates the total number of occurrence of true positive and false positive for each session (P#) and the total when all the sessions are combined. True positive denotes any red points that touches the nose, while false positive denotes red points were not located in the nose area. Sample images of false positives are shown in Figure 4.11, where all the failed nose point detection resides in the glasses area of the subject. While most false nose detection from the glasses area were removed, several cases occurred where the median nose point algorithm failed to remove these false detections. Sample images of true positives are shown in Figure 4.12. Although the images in Figure 4.12 contain multiple nose points, they are all considered true positives as long the detection results are located within the nose area.

Table 4.3: Nose Point Detection Result

	Nose Point Detection	
	Depth	Depth
	False Positive	True Positive
P#		
1	0	300
2	0	300
3	8	292
4	0	300
5	0	300
6	0	300
7	0	300
Total	8	2092
Percentage	0.38%	99.62%

As expected, the nose point is indeed the smallest depth value with respect to the depth face area. Although there were instances where the minimum depth value were not on the nose, this was due to inaccuracies of the depth pixels. For the most part, the median filter helped to remove most of the inaccurate minimum depth pixels.

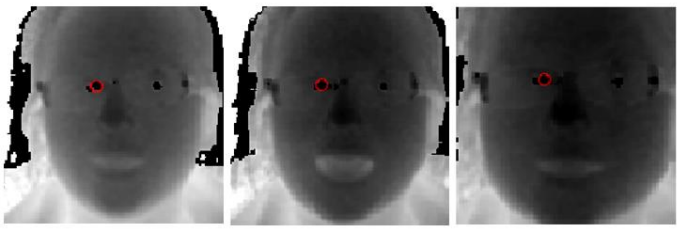


Figure 4.11: False Positive Detection Results Using the Final Nose Point Detection Algorithm

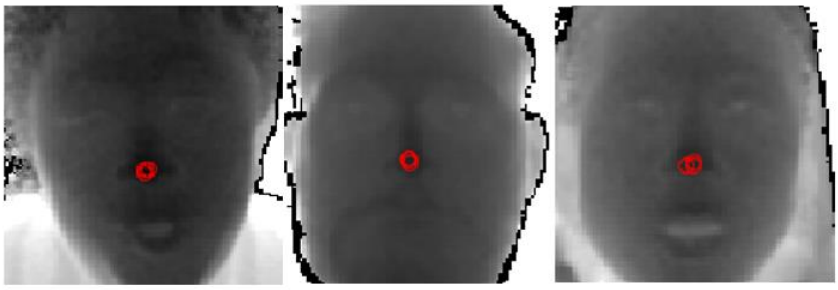


Figure 4.12: True Positive Detection Results Using the Final Nose Point Detection Algorithm

4.6 Depth Normalization

Before we implement algorithms on the facial depth image, we normalize the depth image such that the normalized depth map is independent of the distance between the subject and the Kinect. The normalization procedure shown in Figure 4.13 begins with applying the out of range filtering on the facial depth image. Afterwards, the detected nose point value is subtracted from the filtered result. This computation results in shifting the overall depth data range to begin at 0 (Figure 4.14), where 0 corresponds to the nose point. In creating this normalization algorithm, the dependency between the distance of the Kinect and subject is removed.

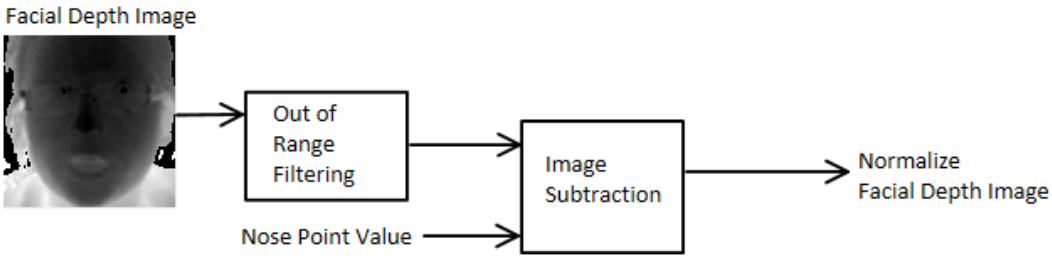


Figure 4.13: Normalized Depth Face Block Diagram

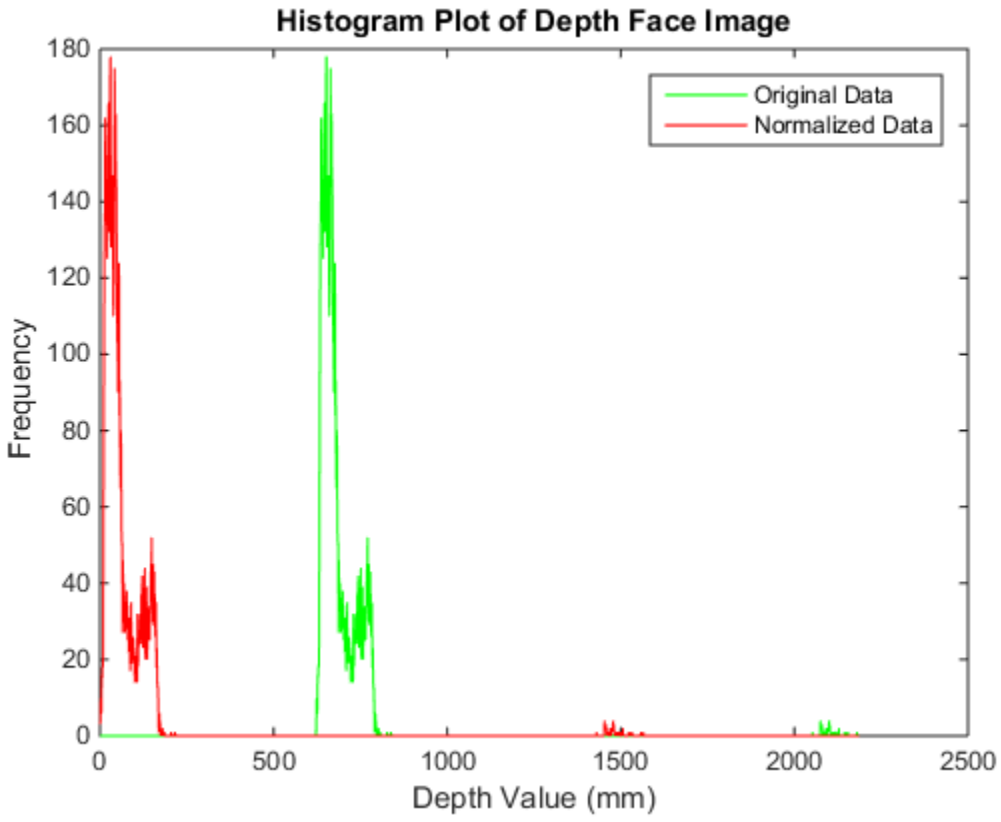


Figure 4.14: Histogram Plot of Depth Face Image

4.7 Lower Facial Bounding Box

Following the depth normalization, the next step is to reduce the current face ROI to a region below the nose. This ROI reduction process is shown as a block diagram in Figure 4.15. First, we use the row corresponding to the nose point to discard the upper half of the original face ROI. In the case where there are multiple detected nose points, we used the max() function from MATLAB to find the row with the highest value. Once the upper portion is discarded, what remains of this current ROI is called the lower facial region.

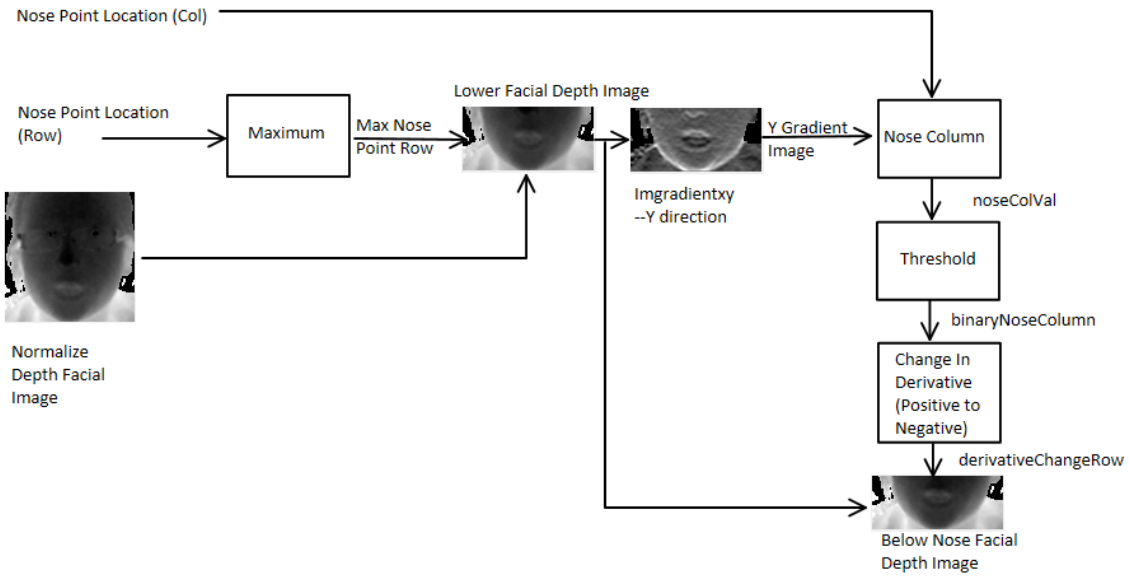


Figure 4.15: Lower Facial Bounding Box Block Diagram

With the current ROI beginning at the row corresponding to the nose point, the next reduction stems from assumption 3: nose point has a higher elevation than the bottom of the nose. In terms of depth, this translate to a positive slope. Furthermore, from the bottom of nose to the top of the upper lip, a negative slope should be observed since the

depth is decreasing. Based on the previous observations, removing the rows corresponding to the first positive slope should remove the nose area from the ROI. Thus, the second ROI reduction begins with taking first derivative of the current ROI in the y direction using imgradientxy() from MATLAB. To find the first change in derivative from positive to negative, the nose column, *noseColVal* corresponding to the nose point location of the Y gradient image is observed. Afterwards, a threshold is applied to the *noseColVal* using the following equation:

$$g(x,y) = \begin{cases} 0, & f(x,y) > 0 \\ 1, & f(x,y) \leq 0 \end{cases} \tag{4.1}$$

where f(x,y) represents the *noseColVal*, and g(x,y) represents the resulting *binaryNoseColumn*. From the *binaryNoseColumn*, the first row value change from 0 to 1 is denoted as the *derivativeChangeRow*. Finally, the *derivativeChangeRow* value rows are discarded from the top of the current ROI to create the below nose facial region.

4.8 Depth Based Segmentation

From the previous section, the ROI, with the goal of feature extraction of the lips, has been successfully reduced to the facial area below the nose. However, within the ROI, partial backgrounds and non-facial objects exist. The goal of this section is to limit the ROI to only the facial area. To accomplish this, image segmentation is used.

There are many approaches to image segmentations. These techniques include: edge detection, threshold-based, and etc. [24]. Edge detection is based on discontinuities in intensity. Such discontinuities can be found by taking the first and second order derivatives of the image [25]. Threshold-based segmentation uses one or more thresholds

to divide an image [24]. Such thresholds are usually chosen based on the image's corresponding histogram.

Within the depth based images, their corresponding histogram have distinct peaks, thus threshold-base segmentation is chosen as the focus of this section. In the previous section, the normalized depth image begins at the depth value of 0, where 0 is the nose point. Based on that knowledge, the first histogram peak will include depth within the facial area. Thus, the first local min after the first local max (peak) of the histogram will serve as the threshold value. The local min and local max are computed based on the findpeaks() function from MATLAB, where its only input is the data, and the output is the peaks and their corresponding data. First, we convert the depth base image into a vector. Afterwards, we insert this vector as the input data of the findpeaks() function to find the local max. To find the local min, we take the negative version of the vector and insert that into the findpeaks() function.

Although the human eye can see several distinct peaks on the histogram in Figure 4.16, the computer see a lot more peaks. While it may not be apparent about why so many peaks are detected in the histogram in Figure 4.16, the zoomed in version in Figure 4.17 shows how noisy the histogram is.

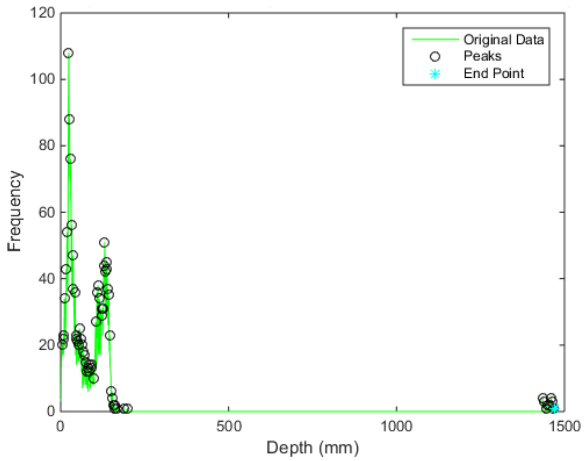


Figure 4.16: Histogram Based Depth Segmentation: Below Nose Facial Depth Image

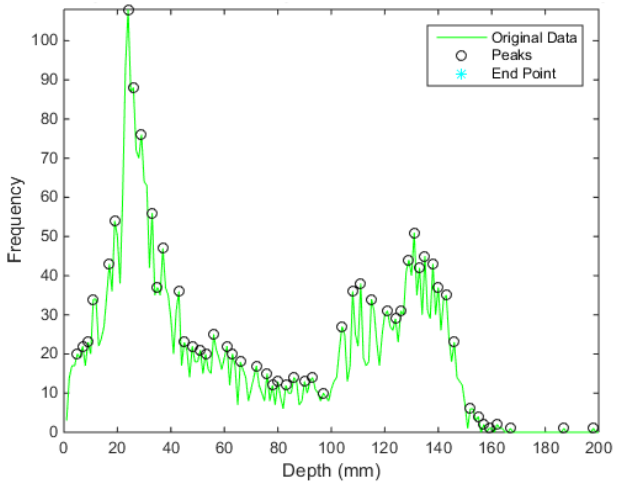


Figure 4.17: Zoomed- In Histogram Based Depth Segmentation: Below Nose Facial Depth Image

To remedy this, a smoothing filter, smooth( ) from MATLAB is used. The smooth( ) function includes two inputs: the data and the span. After the implementation of a smoothing filter, less peaks are observed, however the little spikes along the histogram curve in Figure 4.18 remain a culprit of numerous peaks. These little spikes can be resolved by passing the averaged curve into a MATLAB median filter function called

medfilt1( ). Upon adding the median filter, vast improvements can be seen in Figure 4.19. Local minimums are also included in Figure 4.19.

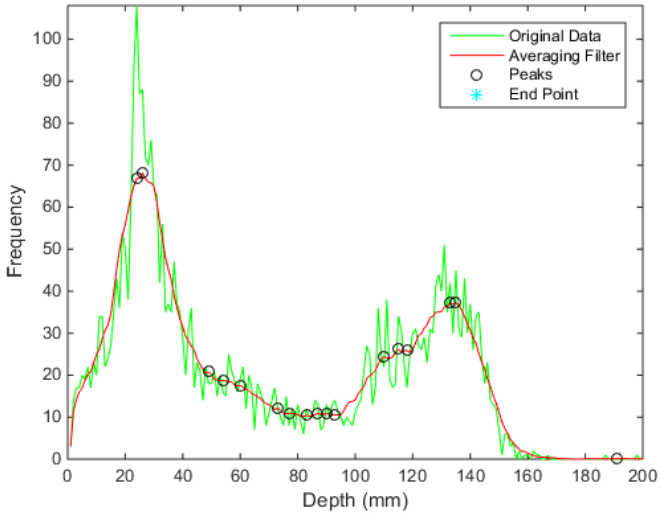


Figure 4.18: Histogram Based Depth Segmentation: Below Nose Facial Depth Image with Smoothing Filter Span of 15

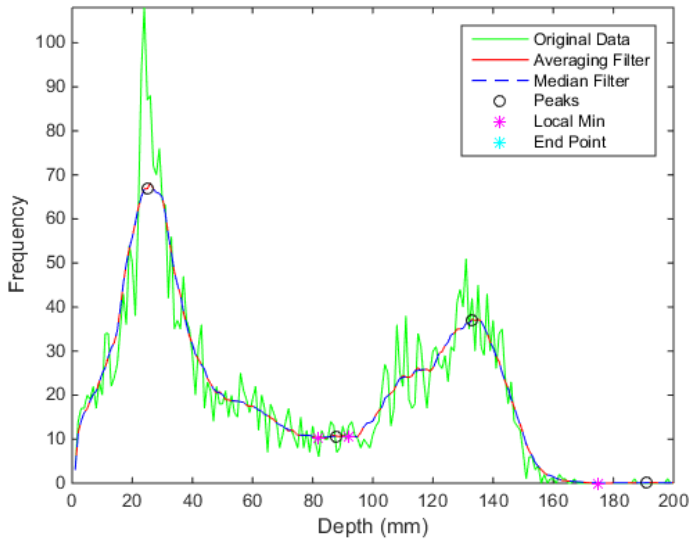


Figure 4.19: Histogram Based Depth Segmentation: Below Nose Facial Depth Image with Smoothing Filter Span of 15 and Median Filter Size of 5

Upon experimentation, a smoothing span of 15, and a median filter size of 5 were implemented to de-noise the histogram curve. After successful filtering of data, the first local min depth value is used as the threshold.

$$g(x,y) = \begin{cases} 0, & f(x,y) > \text{1st Local Min Depth Value} \\ 1, & f(x,y) \leq \text{1st Local Min Depth Value} \end{cases} \quad (4.2)$$

However, with a fixed size for both filters and varying sizes of facial depth images, a safeguard is included into the function to ensure that most of the face is included in the segmentation. This precaution ensures that at least 40% of the image pixels should be segmented as a face. Starting with k=1, the next kth local min depth value will be used until this condition is met. The new threshold now takes on the following equation:

$$g(x,y) = \begin{cases} 0, & f(x,y) > \text{kth Local Min Depth Value} \\ 1, & f(x,y) \leq \text{kth Local Min Depth Value} \end{cases} \quad (4.3)$$

where f(x,y) represents the below nose facial depth image, and g(x,y) represents the binary segmented depth mask. The percentage is calculated by summing up the 1s in the binary image and dividing it with the number of pixels in binary image. Once the conditional statement is satisfied, the following equation is used to create the final segmented below nose facial depth image:

$$g(x,y) = \begin{cases} \text{NaN}, & f(x,y) > \text{kth Local Min Depth Value} \\ f(x,y), & f(x,y) \leq \text{kth Local Min Depth Value} \end{cases} \quad (4.4)$$

where f(x,y) represents the below nose facial depth image, and g(x,y) represents the segmented below nose facial depth image. NaN is applied to background pixels to ensure that the background pixels are not included for the gradient computation in the next

section. A block diagram in Figure 4.20 illustrates the progress of the depth segmentation process.

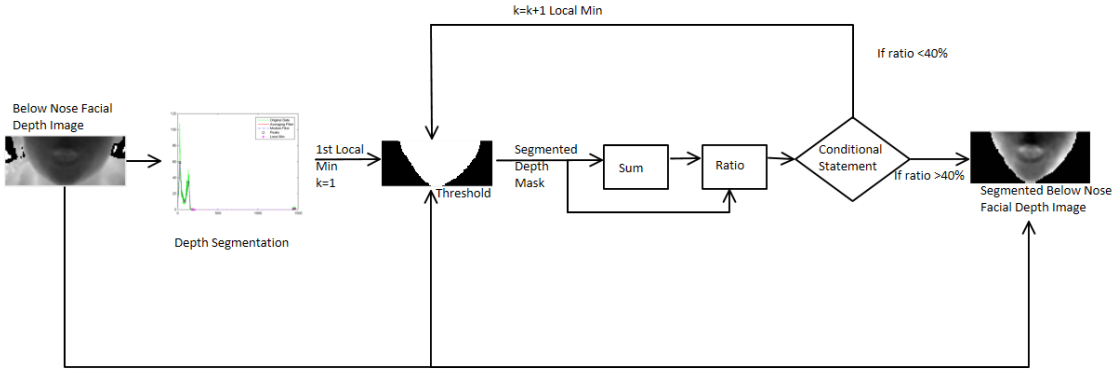


Figure 4.20: Depth Segmentation Block Diagram

After the depth segmentation process, we evaluate the performance by observing several segmented images. As mentioned in the beginning of this section, the goal of the segmentation process is to limit the ROI to the face. Therefore, positive segmentation results would include only the face region in the image, while negative results would include additional regions other than the face. Sample images of positive segmentation results are shown in Figure 4.21. In these images, the depth segmentation algorithm successfully reduce the ROI to the face. Sample images of negative segmentation results are shown in Figure 4.22, where hair and the collar of a jacket were segmented as part of the face region. While the face in these images were successfully included, other non-facial objects were included because they belong in the same depth range as where the face. Although numerous images obtained positive segmentation results, these negative segmentation results can cause false detection for the lip localization algorithm. More

filtering techniques will be required to reduce the number of false detection from the negative depth segmentation process.

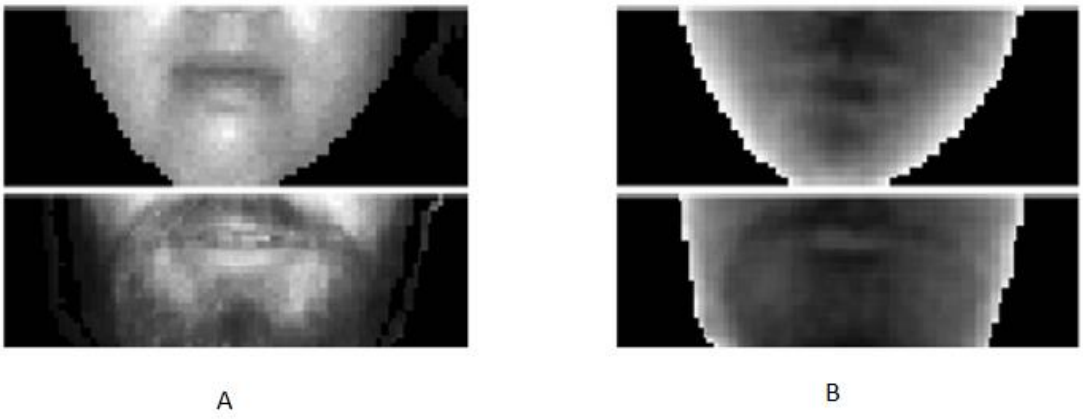


Figure 4.21: Positive Segmentation Results Shown in A) IR Image, and B) Depth Image

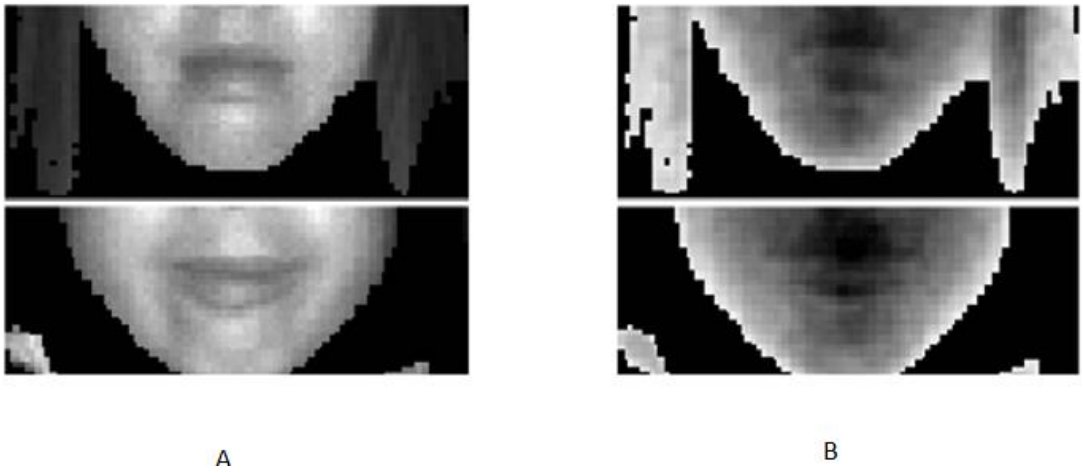


Figure 4.22: Negative Segmentation Results Shown in A) IR Image, and B) Depth Image

4.9 Lip Localization Algorithm

From the depth segmentation process, the corresponding output contains a below nose facial depth image that ends at the edge of the face. With the given edge, assumption 4 can be implemented as an algorithm. Assumption 4 states that while there are sudden inclines and declines within the lip area, such change should not exist from the side of mouth to the edge of the face. To observe inclines and declines of the depth data surrounding the current ROI, the gradient with respect to the y direction is calculated. The y directional gradient image can be implemented by using imgradientxy() function from MATLAB. Afterwards, a threshold using the equation below is applied to y gradient, f(x,y) image to obtain a binary image: Y Gradient Mouth to Chin Binary Image, g(x,y). Pixels within f(x,y) that are equal to NaN represents the background pixels from the segmentation process.

$$g(x,y) = \begin{cases} 0, & f(x,y) \geq 0 \\ & f(x,y) = \text{NaN} \\ 1, & f(x,y) < 0 \end{cases} \tag{4.5}$$

From the binary image, the search algorithm, MouthColFinder, begins by summing up individual columns starting from the nose point column. Afterwards, the search continue with the respective direction until a pattern where one column has a sum greater than 0, follow by two consecutive columns with the sum of 0 are found.

$$\text{sum}(\text{column}) > 0$$

$$\text{sum}(\text{next column}) = 0$$

$$\text{sum}(\text{next next column}) = 0$$

For example, in Figure 4.23, while searching for left side of the lip, if the column *a* contains a sum greater than 1, and then the next two columns, *a*-1, and *a*-2 gives a sum of 0 for their respective column, then column *a* is denoted as the left side of the mouth, *leftMouthEndCol*. For right side, if the sum of column *b* is greater than 0, and the sum of column *b*+1 and *b*+2 are 0, then, *b* is denoted as *rightMouthEndCol*.

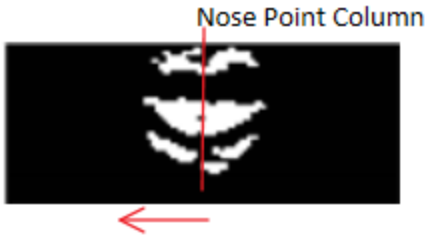


Figure 4.23: Y Gradient Mouth to Chin Binary Image

Once the side of the lips is found, the end of the lips row can be found by implementing assumption 5: there are no sudden change of incline or decline from the bottom of the lip to the edge of the face. The algorithm, MouthRowFinder begins by extracting the nose point column from the binary image. Starting from the bottom row and ascending, the search continues until the row value pattern is found:

$$\text{row} = 0$$

$$\text{row above} = 1$$

Once this pattern is found, the row is declared as bottomMouthEndRow. With the three values, and the original segmented below nose facial depth image, the ROI can now be reduced to the mouth. The block diagram for this mouth feature extraction is shown in Figure 4.24.

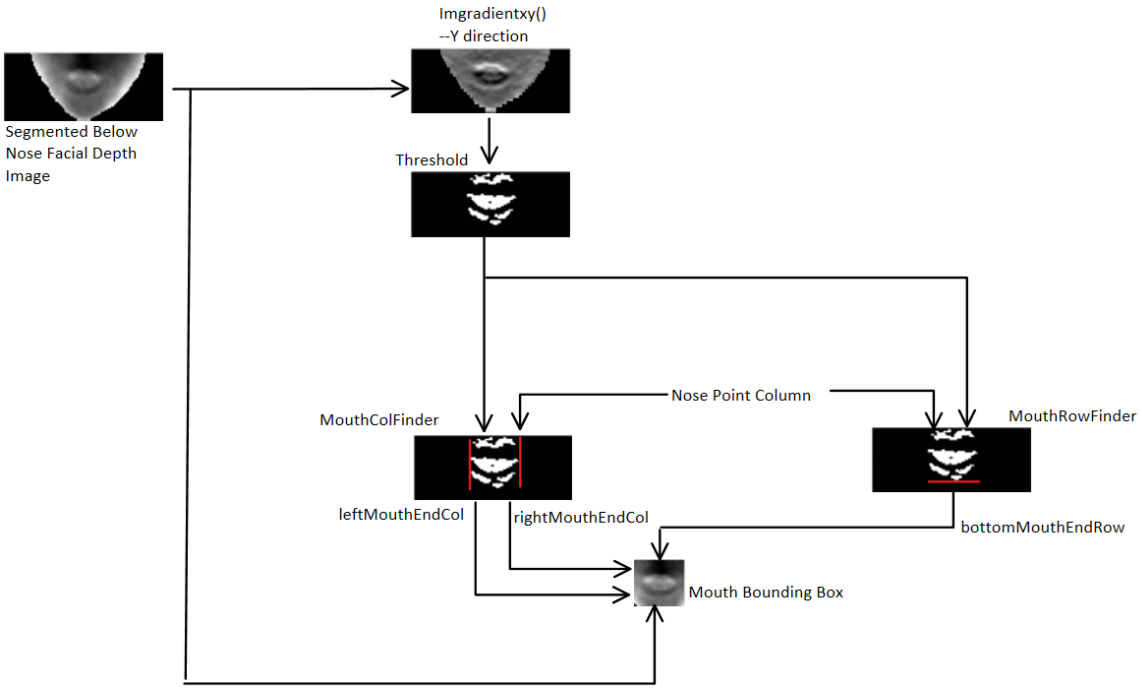


Figure 4.24: Lip Localization Algorithm

The Lip Localization algorithm in Figure 4.24 relies on having a clean binary image that clearly depicts the mouth. Both the MouthColFinder and MouthRowFinder assumes that only a certain lip area appears as 1 in the binary image, while all non-mouth pixels appear as 0. However, not all binary images computed from the segmented below nose facial depth images exhibit these ideal conditions. Sometimes, the resulting binary image may contain clusters that do not belong to the mouth, as shown in the sample images in Figure 4.25. The little clusters surrounding the mouth in Figure 4.25 are most likely due to varying faces and illumination conditions. In addition, hair that are included in the depth segmentation process can cause false positives. In the following subsections, we will attempt to filter out non lip objects from the binary image.

Figure 4.25*:* Sample Y Gradient Mouth to Chin Binary Images with Multiple Non-Mouth Clusters

4.9.1 Hair Filtering

As mentioned in the previous section, sometimes hair depths are within the same depth range as where the face is located. When this happens, the segmented results include hair, and thus the binary image contains unwanted hair pixels that have the binary pixel value as 1. An example of this scenario is shown in Figure 4.26. To remedy this, we assumed hair that are included in the segmented result are located on the side of the face and the mouth is located in the middle of the face. Based on these assumptions, we can create an algorithm to remove any binary hair pixels that are connected to the border within the hair border threshold.

Figure 4.26: Depth Segmentation of the Below Nose Facial Region with Hair Included

Starting from the Y Gradient Mouth to Chin Binary Image, a hair border threshold is used to determine how many columns to clear (to 0) from both sides of the image. The hair border threshold has the same value as the border threshold for consistency. The resulting image is called Hair Border Mask. Afterwards, to ensure that the objects

touching the top and bottom of the image are preserved, the Hair Border Mask is padded with one row of 1s. Finally, imclearborder() function from MATLAB is used and the resulting image is reduced to the same size as the original Y Gradient Mouth to Chin Binary Image. This algorithm, hair border removal, is shown as a block diagram in Figure 4.27.

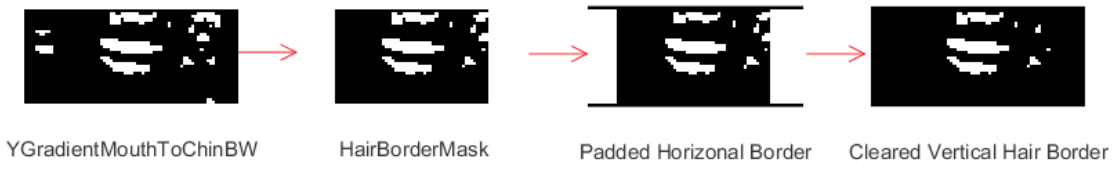


Figure 4.27: Hair Border Removal Block Diagram

4.9.2 Gradient Image Processing and Filtering

Up until now, we have used the Sobel operator for calculating the y directional gradients. The Sobel mask for the y direction, shown in Figure 4.28, have smoothing capabilities that can remove noisy depth pixels. However, it also have the capability to remove potential mouth pixels.

-1	-2	-1
0	0	0
1	2	1

Figure 4.28: Sobel Mask for the Y Direction

When such situations happen, it leads to false detections for the lip algorithm due to its inability to encompass the whole mouth. In addition, because we are focusing on the change of depth pixels value in the y direction, we consider the use of gradient operators

that uses only 1 dimensional mask, since 2 dimensional mask considers diagonal direction. Gradient methods that use 1 dimensional mask include the central difference gradient operator shown in equation (4.6) and the intermediate difference gradient operator shown in equation (4.7).

$$\frac{dI}{dy} = (I(y+1) - I(y-1))/2 \tag{4.6}$$

$$\frac{dI}{dy} = I(y+1) - I(y) \tag{4.7}$$

While the central difference gradient operator in equation (4.6) can remove depth pixel noise, the intermediate difference gradient operator in equation (4.7) is more accurate as it preserves more detail of the original image. Since the MouthColFinding method is based on summations of columns, the y gradient vector can afford the loss in detail, but the same cannot be said about the MouthRowFinding method. The MouthRowFinding method relies on more precise depth information, thus the intermediate difference gradient operator is more suitable. Therefore, two gradients operations are implemented respectively for the MouthColFinder and MouthRowFinder methods.

In the case of the MouthRowFinding method, once we implemented the intermediate difference gradient method, we use the following equation to threshold the resulting gradient image:

$$\mathrm{g(x,y)} = \begin{cases} 0, & \mathrm{f(x,y)} > -1 \\ & \mathrm{f(x,y)} = \mathrm{NaN} \\ 1, & \mathrm{f(x,y)} \leq -1 \end{cases} \tag{4.8}$$

Once converted into a binary image, this is used as the input for the MouthRowFinding method.

For the MouthColFinding method, more processing is required since small binary clusters can cause false detection. To minimize the possibilities of small clusters buildup that doesn't belong to the mouth, we first consider gradient pixels with strong edges. This can be done by applying the threshold from equation (4.8) to the central difference gradient image. Once we obtained the binary image, Y Gradient Mouth to Chin Binary Image, we used the hair border removal method from the previous section, followed by the binary object removal function, bwareaopen() from MATLAB to remove small objects. The remaining binary image, sideMouthBW, is cleared of any clusters that are not part of the lips. However, the actual mouth may also include weaker edges, thus, we reintroduce the neighboring weaker edges by using image dilation on the sideMouthBW. The image dilation is implemented using a square structuring element with the size of 5. The resulting dilated image was multiplied to another binary image called expandedSideMouthBW element by element. After the computation, the equation (4.9) was applied as the threshold to the expandedSideMouthBW:

$$g(x,y) = \begin{cases} 0, & f(x,y) \geq 0 \\ & f(x,y) = \text{NaN} \\ 1, & f(x,y) < 0 \end{cases} \tag{4.9}$$

An example of these filtering steps are shown in Figure 4.29.

Figure 4.29: 1)Y Gradient Mouth To Chin Binary Image 2)Hair Border Removal Result 3) Small Object Removal Result 4)Image Multiplication by Image Dilation Result

The final lip localization algorithm is shown in Figure 4.30. The final output of this algorithm is a mouth bounding box that includes the pixel location of the top left corner, and the corresponding box size.

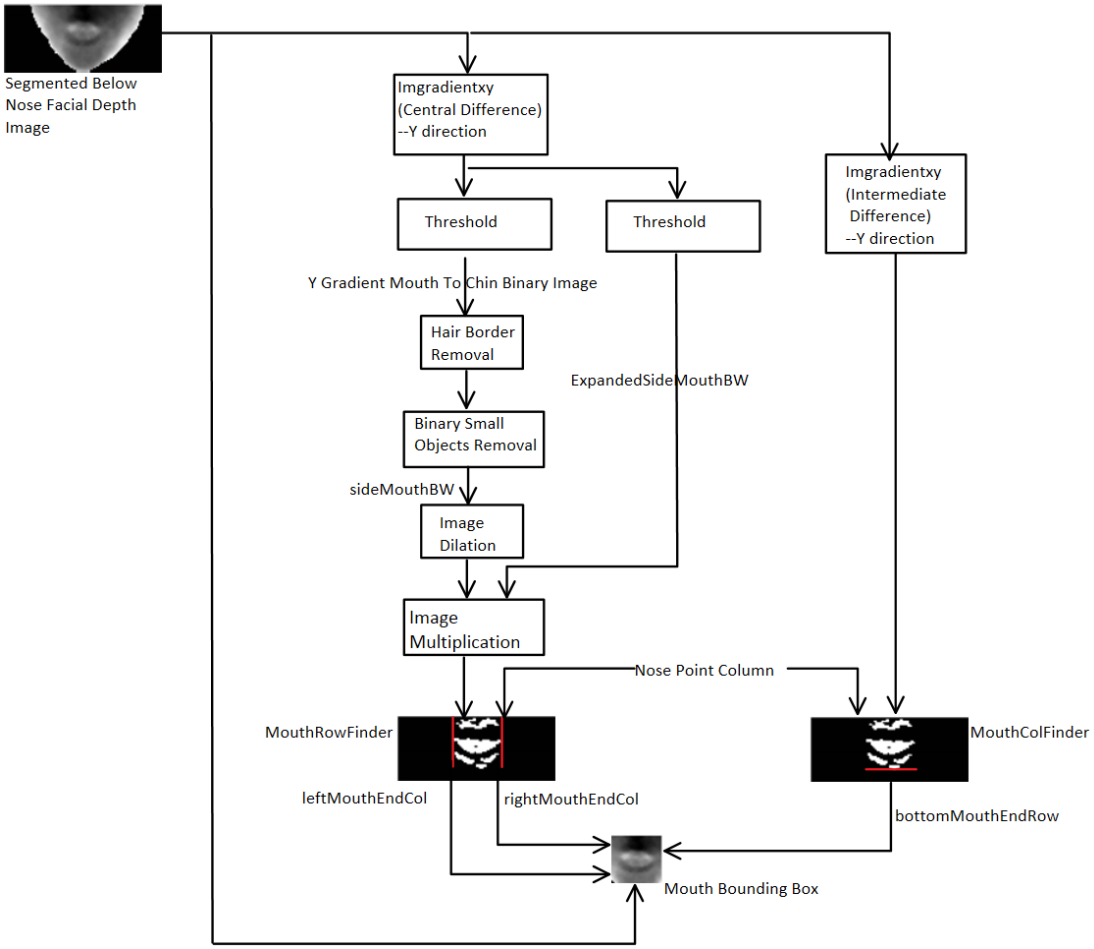


Figure 4.30: Final Lip Localization Algorithm

4.10 Lip Localization Algorithm Test Results

In this section, we evaluate the lip localization algorithm performance by conducting several experiments.

For the first experiment, we compare the immediate effects of incorporating depth into the Viola Jones Mouth Detection. Here, we use the Viola Jones mouth detection on the depth inspired Below Nose Face Region IR Image. More specifically, we use P2 as our test set, which consist of 300 IR images and 300 depth images. For comparison, we include the test results in using the full IR face image for mouth detection from Section 4.1. The criteria for true positive, false negative and false positive for mouth detection are the same as Section 4.1. True positive denotes a mouth that is fully enclosed within the bounding box, and false positive denotes any bounding box that does not enclose the mouth. In addition, false positive also included bounding boxes that include partial mouth and nose. For example, when the lips are detected with excessive surrounding pixels that include the nose pixels, the detection becomes false positive. Images where no bounding box completely encloses the mouth are regarded as false negative. The results are summarized in Table 4.4. Sample images of true positives within the below nose facial region IR image are shown in Figure 4.31. In each image, one true positive was detected on the subject's mouth. Sample images of false negatives within the below nose facial region IR image are shown in Figure 4.32. In each image within Figure 4.32, the Viola Jones Mouth Detection failed to detect a complete mouth when the subject has their mouth open. In addition, sample image of false positives are shown in the two rightmost images of Figure 4.32, where the false positives are the results of the failure to enclose the entire mouth within the bounding box.

Table 4.4: Facial IR Image and Below Nose Facial Region IR Image Using Viola Jones Mouth Algorithm Results

	Facial IR Image			Below Nose Facial Region IR Image		
	False Positive	True Positive	False Negative	False Positive	True Positive	False Negative
Total	480	256	44	36	258	42
Percentage %	160.00 %	85.33%	14.67%	12.00%	86.00%	14.00%

From results in Table 4.4, we see the improved performance following the use of depth to decrease the ROI: the false positive rate drops from 160% to 12% and false negative rate drops from 14.7% to 14%. Sample image revealing the decrease in false negative rate between the two different ROI image sizes are shown in Figure 4.33. The below nose facial region IR image in Figure 4.33 was able to obtain a true positive, while the original facial image obtained a false negative.

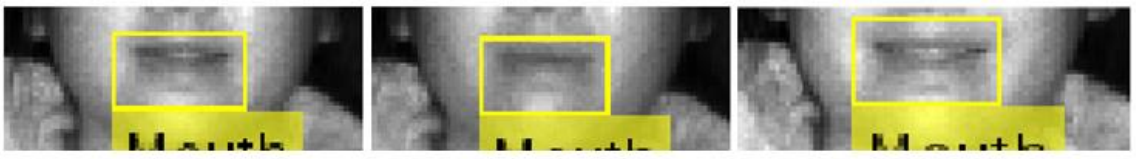


Figure 4.31: True Positive Detection Results from Viola Jones Mouth Detection Given Below Nose Facial Region IR Image Input in P2

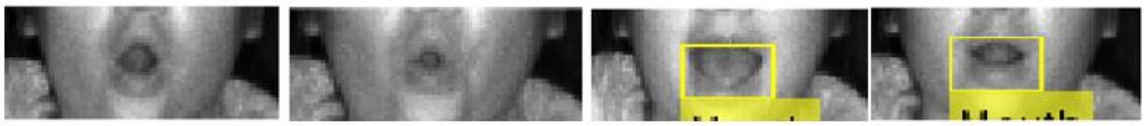


Figure 4.32: False Negative Detection Results (All Four Images) and False Positive Detection Results (Right Two Images) from Viola Jones Mouth Detection Given Below Nose Facial Region IR Image Input in P2

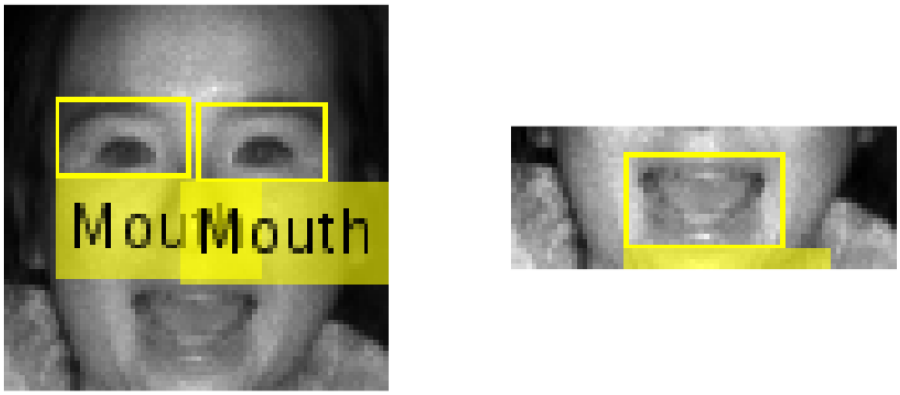


Figure 4.33: False Negative Detection Result from Viola Jones Mouth Detection Given IR Face Image Input(Left) and True Positive Detection Result Given Below Nose Facial Region IR Image

In Figure 4.1, we noticed that most of the false positive on the IR facial region occurs on the eye area. Given that the usage of depth information effectively reduce the search region to the lower half of the face, the eyes are no longer included in this region. With no eyes, there are no other objects that bare resemblance to the mouth, thus paving the way for the Viola Jones Mouth algorithm to obtain a lower false positive rate.

From the first experiment, we learned that the usage of depth allows for less detection error by reducing the region of interest. To see how depth affects the lip localization algorithm in reducing the ROI to the lips, we conduct another experiment on

how often the detected mouth bounding box includes the entire lip region. For the second experiment, we compare the lip inclusiveness of using the depth based lip localization algorithm with the Viola Jones Mouth Detection using below Nose Face Region of the IR Image. This is important because if the mouth bounding boxes cut off part of the mouth, important information that can benefit the visual feature extraction may be lost. The criteria for true positive, false negative and false positive for mouth detection are the same as Section 4.1. True positive denotes a mouth that is fully enclosed within the bounding box, and false positive denotes any bounding box that does not enclose the mouth. In addition, false positive also include bounding boxes with partially enclosed mouths as well as bounding boxes that include the nose. Images where no bounding box completely encloses the mouth are regarded as false negative. The overall results are shown in Table 4.5. In terms of overall coverage of the lip area, the depth based lip localization algorithm performs better than the IR based Viola Jones mouth algorithm. The higher false positive rate for the IR based Viola Jones mouth algorithm is due to the mouth shape. When the lips are closed (Figure 4.34), the IR based Viola Jones mouth algorithm does a good job including the whole mouth with the bounding box. However, the same does not apply when the mouth (Figure 4.35) is open.

Table 4.5: Overall Depth Based Mouth Compared with IR Based Viola Jones Mouth Results

P #	**Mouth-Viola Jones IR**			**Mouth-Depth Based**		
	False Positive	True Positive	False Negative	False Positive	True Positive	False Negative
1	26	274	26	29	272	28
2	25	270	30	21	279	21
3	99	208	92	26	273	27

4	71	227	73	0	300	0
5	64	236	64	21	279	21
6	72	228	72	2	298	2
7	0	291	9	0	300	0
Total	357	1734	366	29	2001	99
Percentage %	17.00%	82.57%	17.43%	4.71%	95.29%	4.71%

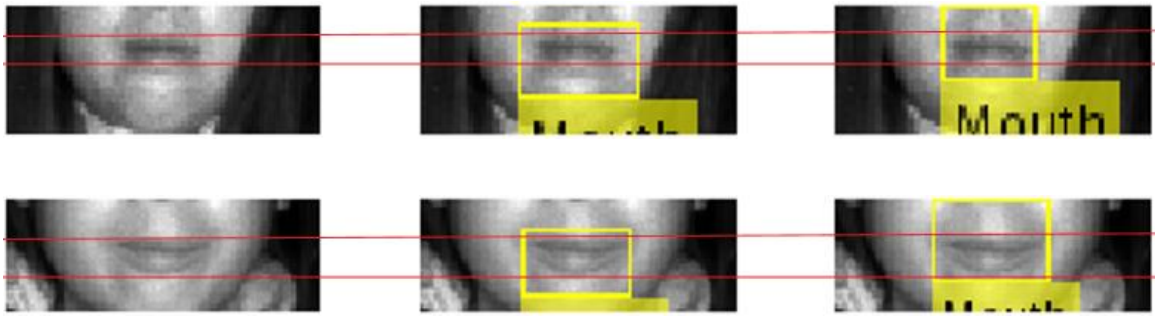
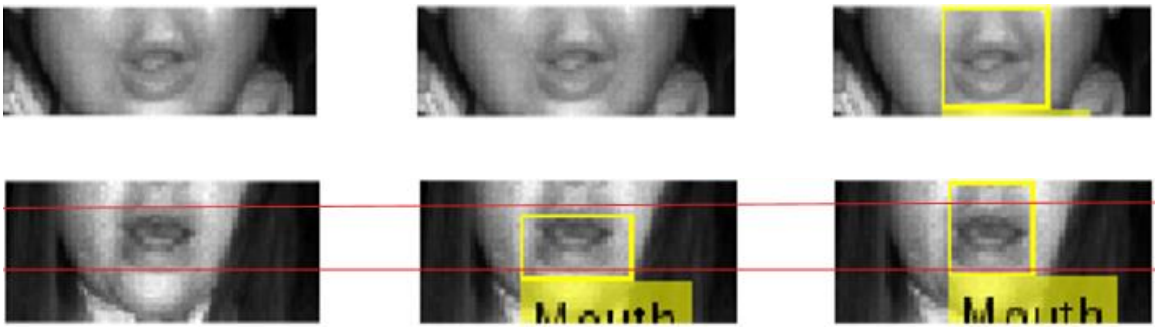


Figure 4.34: Detection of Closed Mouth: (Left) Original Below Nose IR Image, (Center) Viola Jones Mouth Detection Output, (Right) Depth Base Lip Localization Detection Output Projected onto the Original Below Nose IR Image



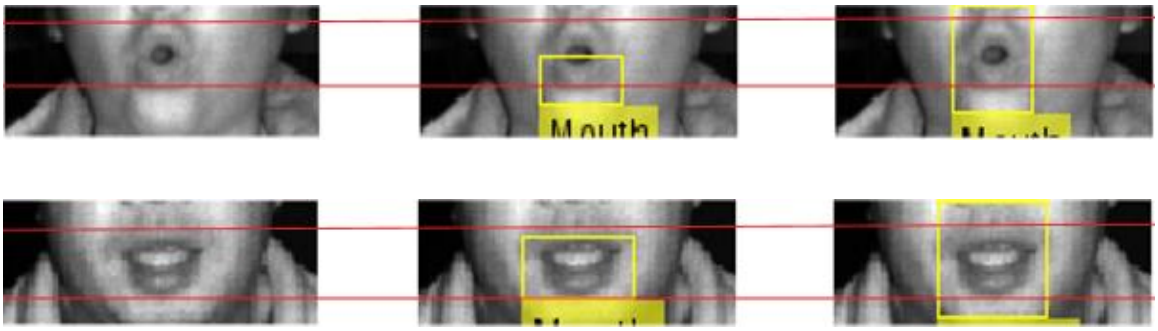


Figure 4.35: Detection of Open Mouth: (Left) Original Below Nose IR Image, (Center) Viola Jones Mouth Detection Output, (Right) Depth Base Lip Localization Detection Output Projected onto the Original Below Nose IR Image

To further evaluate the accuracy between open/close mouths for both algorithms, we organized the results of session P1 in terms of open mouths and closed mouths. Of the 300 captured frames from P1, 116 frames contains open mouths, and 184 frames contains closed mouths. The results are summarized in Table 4.6.

Table 4.6: Depth Based Mouth and IR Based Viola Jones Mouth Results for P1 with Respect to Open/Closed Mouths Based on Bounding Box Inclusivity

Closed Mouth				**Open Mouth**			
IR false positive	IR true positive	Depth false positive	Depth true positive	IR false positive	IR true positive	Depth false positive	Depth true positive
0	184	23	162	27	89	6	110
0.00%	100.00%	12.50%	88.04%	23.28%	76.72%	5.17%	94.83%

For closed mouths, the Viola Jones Mouth detection performed better in terms of including the whole mouth within the bounding box, however, the performance deteriorates for open mouths. For the Depth Base Lip Localization detection, the closed mouth performance was worst compared to the open mouths. The reason for this alludes

to the depth data's y gradient. When the mouth is open, there is a more distinct derivative change since more area is covered. When the mouth is closed, because the mouth covers less area, the derivative change is not as distinct. These distinct patterns can be seen when observing the y gradient mouth to chin binary image in Figure 4.36

Figure 4.36: Y Gradient Mouth to Chin Binary Image (Left) for a Closed mouth, (Center) Closed Mouth and (Right) Open Mouth

From the second experiment, we learned that the depth based lip localization algorithm performs well when it comes to using a bounding box that fully encompasses all mouth pixels. However, the detected bounding box can vary in size. While some bounding boxes include tight boundaries that only the mouth pixels, some may include expanded boundaries that include an abundance of face pixels surrounding the mouth pixels. Thus, in the third experiment, we evaluate the effectiveness of the depth localization algorithm by examining the lip localization boundaries. Ideally, tighter lip localization boundary is more effective since the goal of the depth based lip localization algorithm is to reduce the ROI to the lips. Additionally, tighter lip localization boundary means that the lip localization algorithm successfully filtered out all pixels that did not belong to the mouth. Meanwhile, expanded lip localization boundary alludes to less success in filtering out non mouth pixels within the lip localization algorithm. The digit "Zero" was used from each subject to examine a variety of conditions. The resulting lip

localization boundaries were examined for proximity with respect to the actual lip boundary. The vertical boundaries refer to the left and right side of the lip, while the horizontal boundaries refer to the top and bottom of the lip. Note that 150% (Figure 4.37) for the vertical dimension refers to the bounding box covering 50% more of the lip. In addition, 150% (Figure 4.38) for the horizontal dimension refers to the bounding box covering at least half a lip length or more on the bottom end of the box. The exception to this rule is if the nose is included. Table 4.7 summarizes the results.

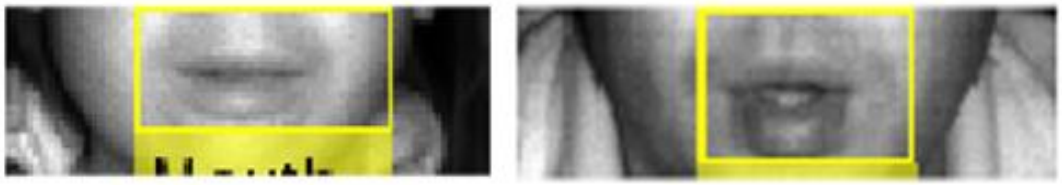


Figure 4.37: Lip Localization Results Projected onto IR Images for Clarity: Results with Vertical Boundaries> 150% and Horizontal Boundaries< 150%

Figure 4.38: Lip Localization Results Projected onto IR Images for Clarity: Results with Vertical Boundaries< 150% and Horizontal Boundaries>150%

Table 4.7: Lip Localization Algorithm Results for the Digit "Zero"

	Vertical			**Horizontal**		
Database	>50%	>100%	>150%	>50%	>100%	>150%
1	30	30	1	30	30	0
2	30	30	0	30	30	0
3	30	29	1	30	27	4
4	30	30	0	30	30	2
5	30	30	2	30	30	9
6	30	30	14	30	30	8
7	30	30	3	30	30	0
8	30	30	30	30	30	30
9	30	30	30	30	30	30
Total	270	269	81	270	267	83
Percentage %	100.00%	99.63%	30.00%	100.00%	98.89%	30.74%

For the most part, the results indicate that the vertical boundaries enclose the lips slightly better than the horizontal boundaries. Furthermore, the vertical boundaries perform better at containing less excess area than the horizontal counterparts. To further examine why both vertical and horizontal boundaries contain excess area respectively, we divide the results into no facial hair, and facial hair categories in Table 4.8.

Table 4.8: Lip Localization Algorithm Results for the Digit "Zero" Comparison between Facial Hair and No Facial Hair

No Facial Hair						
	Vertical			Horizontal		
P	>50%	>100%	>125%	>50%	>100%	>150%
1	30	30	1	30	30	0
2	30	30	0	30	30	0
3	30	29	1	30	27	4
4	30	30	0	30	30	2
5	30	30	2	30	30	9
6	30	30	14	30	30	8
7	30	30	3	30	30	0
Total	210	209	21	210	207	23
%	100.00%	99.52%	10.00%	100.00%	98.57%	10.95%
Facial Hair						
8	30	30	30	30	30	30
9	30	30	30	30	30	30
Total	60	60	60	60	60	60
Percentage %	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%

Here, the results indicate that for the subject with facial hair, the lip localization algorithm had a higher rate of using excessive boundaries to detect the lip. Because facial hair such as Figure 4.39 introduces additional depth to the facial area, all horizontal and vertical boundaries were recorded to have excessive boundaries. While these results indicate that the lip boundary detection algorithm can apply to subjects with facial hair, this is done at a cost of excessive bounding box. For the case with no facial hair, both directional boundaries reported a reduced percentage for excess area. Less excessive area detections occur in non-facial hair subjects occur because there are less obstructions. In Figure 4.39, the large boundary box occurs because our algorithm is based largely on the 1st derivative of the depth facial image, where sudden changes of slopes would cause a stop in ROI reduction. In the case of non-facial hair subjects in Figure 4.40, there are no

sudden changes in slopes, thus, such subjects are able to obtain detections where the lip localization successfully obtain a ROI that fully encases the lip without the covering excessive area of the lips.

Figure 4.39: Lip Localization Results Projected onto IR Images for Clarity: Results for Subject with Facial Hair with Vertical Boundaries and Horizontal Boundaries>150%

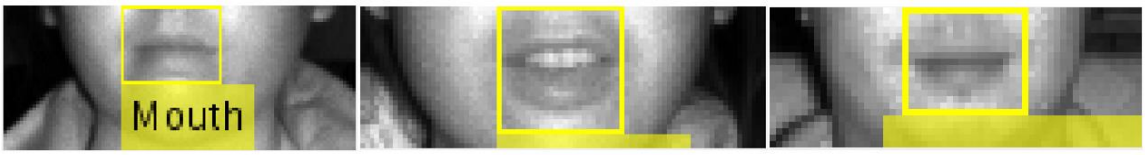


Figure 4.40: Lip Localization Results Projected onto IR Images for Clarity: Results for Subject Without Facial Hair with Vertical Boundaries and Horizontal Boundaries<150%

Although non-facial hair subjects attain better results with tighter boundaries, the noise or error in the depth image prevents the algorithm from attaining more accurate results. In Table 4.9, we divide the results in terms of the lighting condition. While sunlight allows for less excessive boundaries, artificial light allows for more complete mouth enclosure. While artificial light were reported to have a higher percentage the vertical boundary above 150%, sunlight had more bounding boxes with excessive horizontal boundaries. Since each have its disadvantage and advantage, the best form of lighting cannot be concluded. Further investigation of lip localization accuracy regarding the lighting conditions is required.

Table 4.9: Lip Localization Algorithm Results for the Digit “Zero” Comparison between Various Lighting Conditions for Subjects with No Facial Hair

Sunlight						
	Vertical			Horizontal		
P	>50%	>100%	>150%	>50%	>100%	>150%
1	30	30	1	30	30	0
3	30	29	1	30	27	4
5	30	30	2	30	30	9
6	30	30	14	30	30	8
7	30	30	3	30	30	0
Total	150	149	21	150	147	21
%	100.00%	99.33%	14.00%	100.00%	98.00%	14.00%
Artificial Light						
2	30	30	0	30	30	0
4	30	30	12	30	30	7
Total	60	60	12	60	60	7
Percentage %	100.00%	100.00%	20.00%	100.00%	100.00%	11.67%

5. System Performance, Conclusion and Future Work

5.1 Overall System Performance

In this section, we investigate the overall system performance. The results are shown in Table 5.1. The abbreviation FP, represents false positives, while TP represents true positives. For our face detection system, we obtain 100% accuracy despite varying illumination. Such results prove that the use of IR images can aid in improved face detection. The word, true from the column corresponding to the inaccurate below nose face image in Table 5.1 represents occurrences where such image contains the nose pixels such as Figure 5.1 or any other face pixels above the nose. This allows us to investigate how accurate our ROI reduction was from the Lower Facial Bounding Box algorithm. Although only 8 out of 2700 nose points were inaccurately detected, 24 frames with accurate nose point failed to reduce the facial image to below the nose region. This equates to 24/2692=0.89% false detection. Hence, our first attempt to reduce the ROI using depth gradients was a success.

Table 5.1: Overall Performance Results

	Multiple Face Bounding Box		**Nose Point Detection**		**Lip Localization**		**Inaccurate Below Nose Face Image**
	IR	IR	Depth	Depth	Depth	Depth	Depth
P#	FP	TP	FP	TP	FP	TP	TRUE
1	0	300	0	300	29	272	2
2	0	300	0	300	21	279	5
3	0	300	8	292	26	273	17
4	0	300	0	300	0	300	0
5	0	300	0	300	21	279	8
6	0	300	0	300	2	298	0
7	0	300	0	300	0	300	0
8	0	300	0	300	0	300	0
9	0	300	0	300	0	300	0
Total	0	2700	8	2692	99	2601	32
Perc. %	0.00%	100.00%	0.30%	99.70%	3.67%	96.33%	1.19%

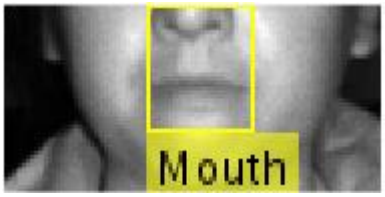


Figure 5.1: Inaccurate Below Nose Face Image

Additionally, in the last experiment of Section 4.10, we attempted to investigate the effect of lighting conditions on the lip localization accuracy of non-facial hair subjects. Here, we conduct a similar investigation by dividing the results into different types of lighting conditions, and focus on the cases with non-facial hair subjects. The results from Table 5.2 indicate that the lip localization performs better with artificial lighting.

Table 5.2: System Performance Comparison between Different Lighting Conditions for Non-Facial Hair Subjects

	Multiple Face Bounding Box		**Nose Point Detection**		**Lip Localization**		**Inaccurate Below Face Image**
	IR	IR	Depth	Depth	Depth	Depth	Depth
Sunlight							
P#	FP	TP	FP	TP	FP	TP	TRUE
1	0	300	0	300	29	272	2
3	0	300	8	292	26	273	17
5	0	300	0	300	21	279	8
6	0	300	0	300	2	298	0
7	0	300	0	300	0	300	0
Total	0	1500	8	1492	78	1422	27
Percentage %	0.00%	100.00%	0.53%	99.47%	5.20%	94.80%	1.80%
Artificial Lighting							
2	0	300	0	300	21	279	5
4	0	300	0	300	0	300	0
Total	0	600	0	600	21	579	5
Percentage %	0.00%	100.00%	0.00%	100.00%	3.50%	96.50%	0.83%

Finally, the time duration for various functions to process 30 frames on a Lenovo computer with an Intel Core i5 1.80GHz 64 bit processor are shown in Table 5.3. For the face detection function, this includes the Viola Jones Face Detection and Multiple Face Bounding Box Algorithm. For Lip Localization function, this includes the final lip localization algorithm. Depth Segmentation function includes only the histogram base depth segmentation. Nose Point Function includes the Final Nose Point Detection Algorithm. Lastly, the Viola Jones Mouth function includes only the Viola Jones Mouth Detection, with its input being the below nose face IR image. From Table 5.3, the Lip

Localization function uses less processing time than the Viola Jones Mouth Detection. The function that took the longest was the face detection, where it took the Viola Jones Face Detection 9.057 seconds to process 30 frames.

Table 5.3: Time Duration to Process One Video Clip (30 Frames)

Functions	Duration(Seconds)
Face Detection	9.090
Lip Localization	0.256
Depth Segmentation	0.281
Nose Point Detection	0.343
Viola Jones Mouth(Below Nose Facial IR Image)	2.093

5.2 Conclusion

In this thesis, we investigate the use of depth and IR to see if the use of these two streams are plausible for face detection and lip localization algorithms. From the lighting perspective, we see an advantage of using the IR image as compared to color image for places that has low level of light. When given the IR face bounding box coordinates, we took advantage of how the IR and depth stream share the same coordinate system, and compare the two streams for mouth detection. Given the depth face image, we were able to reduce the ROI for the mouth based on depth gradients. From the result of depth based lip localization, it proved that using depth did indeed increase the detection performance. Based on the results presented in this thesis, we believe that the inclusion of depth based gradient and the use of IR images can further improve the accuracy of lip feature extraction.

5.3 Limitations and Future Work

In the general sense, we see an increased detection performance by incorporating depth and IR image for lip localization. However, some limitations from this thesis includes a small database size. In this work, 4 subjects were involved, and all but one session were captured indoor. One session was captured outdoor with a roof above the subject. The system proposed here provides a proof-of-concept and should be tested against a larger data set to study its effectiveness in face detection and lip localization. Additionally, the current system required that the subject directly faces the Kinect to ensure that the nose is the closest point to the sensor. If the subject looks away from the Kinect while talking, the algorithm will not work. Lastly, if the subject applies foreign objects onto the face that leads to other objects being closer to the Kinect than the nose, this algorithm will not work. Once an accurate and robust visual front end is built, the next step is to develop strategies to extract useful visual speech features followed by audio-visual integration to perform automatic speech recognition.

REFERENCES

[1] Potamianos, A.; Narayanan, S., "Robust recognition of children's speech," Speech and Audio Processing, IEEE Transactions on , vol.11, no.6, pp.603,616, Nov. 2003

[2] Russell, M.; Brown, C.; Skilling, A.; Series, R.; Wallace, J.; Bonham, B.; Barker, P., "Applications of automatic speech recognition to speech and language development in young children," Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on , vol.1, no., pp.176,179 vol.1, 3-6 Oct 1996

[3] R. Stern, A. Acero, F.-H. Liu, and Y. Ohshima, "Signal processing for robust speech recognition," in Automatic Speech and Speaker Recognition. Advanced Topics, C.-H. Lee, F. K. Soong, and Y. Ohshima, Eds. Norwell, MA: Kluwer, 1997, ch. 15, pp. 357–384.

[4] Galatas, G.; Potamianos, G.; Makedon, F., "Audio-visual speech recognition incorporating facial depth information captured by the Kinect," Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European , vol., no., pp.2714,2717, 27-31 Aug. 2012

[5] Samoil, S.; Lai, K.; Yanushkevich, S., "Multispectral Hand Biometrics," Emerging Security Technologies (EST), 2014 Fifth International Conference on , vol., no., pp.24,29, 10-12 Sept. 2014

[6] Potamianos, G.; Neti, C.; Gravier, G.; Garg, A.; Senior, A.W., "Recent advances in the automatic recognition of audiovisual speech," Proceedings of the IEEE , vol.91, no.9, pp.1306,1326, Sept. 2003

[7] Neti, C.; Potamianos, G.; Luettin, J.; Matthews, I.; Glotin, H.; Vergyri, D., "Large-vocabulary audio-visual speech recognition: a summary of the Johns Hopkins Summer 2000 Workshop,"Multimedia Signal Processing, 2001 IEEE Fourth Workshop on , vol., no., pp.619,624, 2001

[8] Wang, S.L.; Lau, W.H.; Leung, S.H.; Yan, H., "A real-time automatic lipreading system," Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on , vol.2, no., pp.II,101-4 Vol.2, 23-26 May 2004

[9] Ibrahim, M.Z.; Mulvaney, D.J., "A lip geometry approach for feature-fusion based audio-visual speech recognition," Communications, Control and Signal Processing (ISCCSP), 2014 6th International Symposium on , vol., no., pp.644,647, 21-23 May 2014

[10] Nguyen Thien Chuong; Chaloupka, J., "Visual feature extraction for isolated word visual only speech recognition of Vietnamese," *Telecommunications and Signal Processing (TSP), 2013 36th International Conference on* , vol., no., pp.459,463, 2-4 July 2013

[11] Chalamala, S.R.; Gudla, B.; Yegnanarayana, B.; Anitha, S.K., "Improved lip contour extraction for visual speech recognition," *Consumer Electronics (ICCE), 2015 IEEE International Conference on* , vol., no., pp.459,462, 9-12 Jan. 2015 doi: 10.1109/ICCE.2015.7066486

[12] Navarathna, R.; Lucey, P.; Dean, D.; Fookes, C.; Sridharan, S., "Lip detection for audio-visual speech recognition in-car environment," *Information Sciences Signal Processing and their Applications (ISSPA), 2010 10th International Conference on* , vol., no., pp.598,601, 10-13 May 2010

[13] Ming-Hsuan Yang; Kriegman, D.; Ahuja, N., "Detecting faces in images: a survey," Pattern Analysis and Machine Intelligence, IEEE Transactions on , vol.24, no.1, pp.34,58, Jan 2002

[14] Li, S.Z.; RuFeng Chu; Shengcai Liao; Lun Zhang, "Illumination Invariant Face Recognition Using Near-Infrared Images," Pattern Analysis and Machine Intelligence, IEEE Transactions on , vol.29, no.4, pp.627,639, April 2007

[15] Xuan Zou,; Kittler, J.; Messer, K., "Illumination Invariant Face Recognition: A Survey," Biometrics: Theory, Applications, and Systems, 2007. BTAS 2007. First IEEE International Conference on , vol., no., pp.1,8, 27-29 Sept. 2007

[16] Socolinsky, D.A.; Wolff, L.B.; Lundberg, A.J., "Image Intensification for Low-Light Face Recognition," *Computer Vision and Pattern Recognition Workshop, 2006. CVPRW '06. Conference on* , vol., no., pp.41,41, 17-22 June 2006

[17] Hizem, W.; Allano, L.; Mellakh, A.; Dorizzi, B., "Face recognition from synchronised visible and near-infrared images," *Signal Processing, IET* , vol.3, no.4, pp.282,288, July 2009

[18] "Kinect for Windows SDK 2.0" Microsoft, 2015 [Online]. Available: https://msdn.microsoft.com/en-us/library/dn785530.aspx

[19] Castaneda,V.;Navab,N. (2011) "Time-of-Flight and Kinect Imaging" [Online]. Available: http://campar.in.tum.de/twiki/pub/Chair/TeachingSs11Kinect/2011-DSensors_LabCourse_Kinect.pdf

[20] Viola, P.; Jones, M., "Rapid object detection using a boosted cascade of simple features," Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on , vol.1, no., pp.I-511,I-518 vol.1, 2001

[21] Paul Viola, Michael J. Jones, Robust Real-Time Face Detection, International Journal of Computer Vision 57(2), 2004.

[22] Alexander Sabov , Jörg Krüger, Identification and correction of flying pixels in range camera data, Proceedings of the 24th Spring Conference on Computer Graphics, April 21-23, 2008, Budmerice, Slovakia

[23] "Programing Kinect for Windows V2: (07) Advance Topics: Skeletal Tracking and Depth Filtering" Microsoft, 2015 [Video]. Available: http://channel9.msdn.com/series/Programming-Kinect-for-Windows-v2/07

[24] Gonalez,R , "Image Segmentation" in Digital Image Processing, Third ed. Upper Saddle River, NJ, USA: Prentice Hall, 2008, ch 10, sec 1-3, p. 689-761

[25] Raut, S.; Raghuvanshi, M.; Dharaskar, R.; Raut, A., "Image Segmentation – A State-Of-Art Survey for Prediction," Advanced Computer Control, 2009. ICACC '09. International Conference on , vol., no., pp.420,424, 22-24 Jan. 2009

[26] Chalamala, S.R.; Gudla, B.; Yegnanarayana, B.; Anitha, S.K., "Improved lip contour extraction for visual speech recognition," *Consumer Electronics (ICCE), 2015 IEEE International Conference on* , vol., no., pp.459,462, 9-12 Jan. 2015

APPENDICES

APPENDIX A: Video Capture Visual Studio Code

A.1: Video Capture Code: For Main Database

MainWindow.xaml.cs:

```
/**
This code records 30 consecutive frames of IR and Depth images and saves the data in .txt files,
additionally, 30 consecutive frames of color images  are saved in .png format
This code was build upon a tutorial from Vangos Pterneas [1]

References:
[1]Pterneas,Vangos (2014,February 20). "Kinect for Windows version2: Color,depth and infrared streams". [Online]
Available:http://pterneas.com/2014/02/20/kinect-for-windows-version-2-color-depth-and-infrared-streams/
[Accessed: November 10, 2014]

The MIT License (MIT)
Copyright (c) 2014 Vangos Pterneas
Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THESOFTWARE.
```

```
 **/

using System;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Microsoft.Kinect;
using System.IO;
using System.Diagnostics;


namespace VideoRecorderRGBDIR
{
   /// <summary>
   /// Interaction logic for MainWindow.xaml
   /// </summary>
   public partial class MainWindow : Window
   {
      //Variable Initializations
      private KinectSensor kinect = null;
      private MultiSourceFrameReader frameReader = null;

          private readonly int bytePerPixel = (PixelFormats.Bgr32.BitsPerPixel + 7) / 8;

      //ColorData
      private byte[] colorPixels = null;
      private WriteableBitmap colorBitmap = null;
      private WriteableBitmap[] arrayColorStorage = null;

      //DepthData
      private ushort[] depthData = null;
      private byte[] depthPixels = null;
      private WriteableBitmap depthBitmap = null;
      private ushort[][] depthDataArray = null;//jagged array
      private ushort[] tempDepthData = null;

      //IRData
```

```
private ushort[] irData = null;
private byte[] irPixels = null;
private WriteableBitmap irBitmap = null;
private ushort[][] irDataArray = null;
private ushort[] tempIRData = null;

//For the record button
private Boolean recordstatus = false;

//Timer
private Stopwatch time = null;
private int counter = 0;


public MainWindow()
{
   this.InitializeComponent();

   //Activate Sensor, and connect
   this.kinect = KinectSensor.GetDefault();

   if (kinect == null) return;


            //open the streams
            frameReader =
            kinect.OpenMultiSourceFrameReader(FrameSourceTypes.Color |
            FrameSourceTypes.Depth | FrameSourceTypes.Infrared);

            //event handler when the next frame is ready
            frameReader.MultiSourceFrameArrived +=
            multiSourceFrameReader_MultiSourceFrameArrived;

            //opens the kinect sensor
            this.kinect.Open();

            //temporary storage arrays
            arrayColorStorage = new WriteableBitmap[30];
            depthDataArray = new ushort[30][];
            irDataArray = new ushort[30][];

            //Creates a new stopwatch
            time = new Stopwatch();
            TextBlock Timer = new TextBlock();
            TextBlock Title = new TextBlock();
}
```

```
//when the user cicks on record, the timer and counter restarts to 0
void Record_Click(object sender, RoutedEventArgs e)
{
   recordstatus = true;
   counter = 0;
   time.Restart();
}

//This event happens everytime a frame is ready
            private void multiSourceFrameReader_MultiSourceFrameArrived(object
            sender, MultiSourceFrameArrivedEventArgs e)
{
   var reference = e.FrameReference.AcquireFrame();
   if (reference == null) return;

      // Open color frame reader
      using (var frame = reference.ColorFrameReference.AcquireFrame())
      {
         if (frame != null) colorView.Source = ToImageBitmap(frame);
      }

      // Open depth frame reader
      using (var frame = reference.DepthFrameReference.AcquireFrame())
      {
         if (frame != null)
         {
            tempDepthData = dataExtraction(frame);
         }

      }

      // Open IR frame reader
            using (var frame = reference.InfraredFrameReference.AcquireFrame())
      {
         if (frame != null)
         {
            irView.Source = ToImageBitmap(frame);
            tempIRData=dataExtraction(frame);
         }
      }

      //Records frame data if counter is less than 30
      if (recordstatus == true && counter < 30)
      {
         arrayColorStorage[counter] = colorBitmap;
```

```
            depthDataArray[counter] = tempDepthData;
            irDataArray[counter] = tempIRData;
            counter++;
        }
        //stops recording after 30 frames have been reached
        if (recordstatus == true && counter == 30)
        {
            Timer.Text = time.Elapsed.ToString();
            String title = Title.Text.ToString();
            recordstatus = false;
            SaveImages(arrayColorStorage, title);
            SaveDatas(depthDataArray, title + "Depth");
            SaveDatas(irDataArray, title+"IR");
        }
}

//Saving color images to bitmap images
private void SaveImages(WriteableBitmap[] Bitmap, string name)
{
    for (int i = 0; i < 30; i++)
    {
        SaveImage(Bitmap[i], name + i);
    }
}

//saving depth or IR data
private void SaveDatas(ushort[][] DataArray, string name)
{

    for (int i = 0; i < 30; i++)
    {
        ushort[] tempDataAraray = DataArray[i];
        SaveData(tempDataAraray, name + i);
   }

}

//Convert color data to color image
private ImageSource ToImageBitmap(ColorFrame frame)
{
    //array of pixel RGB values
            colorPixels = new byte[frame.FrameDescription.Height *
            frame.FrameDescription.Width * bytePerPixel];

    //create new writeable bitmap
```

```
            colorBitmap = new WriteableBitmap(frame.FrameDescription.Width,
            frame.FrameDescription.Height, 96, 96, PixelFormats.Bgr32, null);

    if (frame.RawColorImageFormat == ColorImageFormat.Bgra)
    {
       frame.CopyRawFrameDataToArray(colorPixels);
    }
    else
    {
              frame.CopyConvertedFrameDataToArray(colorPixels,
            ColorImageFormat.Bgra);
    }

    //copy output to bitmap
            colorBitmap.WritePixels(new Int32Rect(0, 0,
            frame.FrameDescription.Width, frame.FrameDescription.Height),
            colorPixels, frame.FrameDescription.Width * bytePerPixel, 0);
    return colorBitmap;
}

//storing depth frame data into array
private ushort[] dataExtraction(DepthFrame frame)
{

            depthPixels = new byte[frame.FrameDescription.Height *
            frame.FrameDescription.Width * bytePerPixel];
          depthData = new ushort[frame.FrameDescription.Height *
          frame.FrameDescription.Width];
          depthBitmap = new WriteableBitmap(frame.FrameDescription.Width,
          frame.FrameDescription.Height, 96, 96, PixelFormats.Bgr32, null);

    //copy depth frames
    frame.CopyFrameDataToArray(depthData);
    return depthData;
}

//storing IR frame data into array
private ushort[] dataExtraction(InfraredFrame frame)
{
            irPixels = new byte[frame.FrameDescription.Height *
            frame.FrameDescription.Width * bytePerPixel];
          irData = new ushort[frame.FrameDescription.Height *
          frame.FrameDescription.Width];
          irBitmap = new WriteableBitmap(frame.FrameDescription.Width,
          frame.FrameDescription.Height, 96, 96, PixelFormats.Bgr32, null);
```

```
    // Copy data
    frame.CopyFrameDataToArray(irData);
    return irData;
}

//Convert IR data to IR image
private ImageSource ToImageBitmap(InfraredFrame frame)
{
            irPixels = new byte[frame.FrameDescription.Height *
            frame.FrameDescription.Width * bytePerPixel];
            irData = new ushort[frame.FrameDescription.Height *
            frame.FrameDescription.Width];
            irBitmap = new WriteableBitmap(frame.FrameDescription.Width,
            frame.FrameDescription.Height, 96, 96, PixelFormats.Bgr32, null);


    // Copy data
    frame.CopyFrameDataToArray(irData);

    int colorPixelIndex = 0;

    for (int i = 0; i < irData.Length; ++i)
    {
        // Get infrared value
        ushort ir = irData[i];

        // Bitshift
        byte intensity = (byte)(ir >> 8);

        // Assign infrared intensity
        irPixels[colorPixelIndex++] = intensity;
        irPixels[colorPixelIndex++] = intensity;
        irPixels[colorPixelIndex++] = intensity;

        ++colorPixelIndex;
    }

    // Copy output to bitmap
            irBitmap.WritePixels(new Int32Rect(0, 0, frame.FrameDescription.Width,
            frame.FrameDescription.Height), irPixels, frame.FrameDescription.Width
            * bytePerPixel, 0);
    return irBitmap;
}

//Saving the image onto the computer
private void SaveImage(WriteableBitmap Bitmap, string name)
```

```
        {
            if (Bitmap != null)
            {

            string path =
            System.IO.Path.Combine(@"C:\Users\Katherine\Desktop\Thesis\VideoDemo",
            name + ".png");

                // create a png bitmap encoder which knows how to save a .png file
                BitmapEncoder encoder = new PngBitmapEncoder();

                // create frame from the writable bitmap and add to encoder
                encoder.Frames.Add(BitmapFrame.Create(Bitmap));


                FileStream fs = new FileStream(path, FileMode.Create);
                encoder.Save(fs);

                fs.Close();
            }
        }

        //Saving the .txt data onto the computer
        private void SaveData(ushort[] Data, string name)
        {
                    string path =
                    System.IO.Path.Combine(@"C:\Users\Katherine\Desktop\Thesis\VideoDe
                    mo", name + ".txt");
                    //initialize a StreamWriter
                     StreamWriter sw = new StreamWriter(path);
                    //temp=ushort(Data);

                     //search the data and add it to the file
                    for (int i = 0; i < Data.Length; i++)
                    {
                            sw.WriteLine(Data[i] + "\n"); //\n for a new line
                    }

            //dispose of sw
            sw.Close();
        }
    }
}
```

MainWindow.xaml:

```
<Window x:Class="VideoRecorderRGBDIR.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="526.119" Width="1197.015">
    <Grid Margin="0,0,0,15">
        <Image x:Name="colorView" Margin="720,0,0,132"/>
        <Image x:Name="depthView" Margin="360,0,472,132"/>
        <Image x:Name="irView" Margin="0,0,832,132"/>
        <Button x:Name="Record" Content="Record" HorizontalAlignment="Left"
Height="21" Margin="524,438,0,0" VerticalAlignment="Top" Width="75"
Click="Record_Click"/>
        <TextBox x:Name="Timer" HorizontalAlignment="Left" Height="18"
Margin="720,438,0,0" TextWrapping="Wrap" Text="TextBox"
VerticalAlignment="Top" Width="96"/>
        <TextBox x:Name="Title" HorizontalAlignment="Left" Height="19"
Margin="120,434,0,0" TextWrapping="Wrap" Text="EnterTitleHere"
VerticalAlignment="Top" Width="96"/>
        <Label x:Name="Time_Elasped_" Content="Time_Elasped:"
HorizontalAlignment="Left" Margin="637,434,0,0" VerticalAlignment="Top"
Width="92" Height="33"/>
        <Label x:Name="Person" Content="Title:" HorizontalAlignment="Left"
Height="25" Margin="81,428,0,0" VerticalAlignment="Top" Width="77"/>

    </Grid>
</Window>
```

A.2: Image Capture Code with Coordinate Mapping: For Color vs. IR Face Detection

MainWindow.xaml.cs:

```
/**
This code records 1 frame of IR and Depth and saves the data in .txt files,
additionally, 1 color, depth and IR image  are saved in .png format
The coordinate map between depth and color space is saved in .txt file
This code was build upon a tutorial from Vangos Pterneas [1]

References:
[1]Pterneas,Vangos (2014,February 20). "Kinect for Windows version2: Color,depth and
infrared streams". [Online]
Avaiable:http://pterneas.com/2014/02/20/kinect-for-windows-version-2-color-depth-and-
infrared-streams/
[Accessed: November 10, 2014]


The MIT License (MIT)
```

```
 **/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Microsoft.Kinect;
using System.IO;

namespace Version1
{
```

```
/// <summary>
/// Interaction logic for MainWindow.xaml
/// </summary>
public partial class MainWindow : Window
{
    //Intializing variables
    private KinectSensor kinect = null;

    //color variables
    //each color has 8 bit per pixel
    private readonly int bytePerPixel = (PixelFormats.Bgr32.BitsPerPixel + 7) / 8;
    private ColorFrameReader colorReader = null;
    //array of color pixel
    private byte[] colorPixels = null;
    private WriteableBitmap colorBitmap = null;

    //depth variables
    private DepthFrameReader depthReader = null;
    private ushort[] depthData = null;
    private byte[] depthPixels = null;
    private WriteableBitmap depthBitmap = null;

    //Infrared variables
    private InfraredFrameReader irReader = null;
    private ushort[] irData = null;
    private byte[] irPixels = null;
    private WriteableBitmap irBitmap = null;

    //Coordinate Mapping
    private DepthSpacePoint[] depthMapToColor = null;
    private CoordinateMapper coordinateMapper = null;
    private float[] xcoordinate = null;
    private float[] ycoordinate = null;

    //Image Naming Number
    private int counter = 120;

    public MainWindow()
    {
        //loads the compiled page of a component
        InitializeComponent();
        InitializeKinect();
    }

    //This method initializes any functions that are required to initialize the kinect
    private void InitializeKinect()
```

```
{
    kinect = KinectSensor.GetDefault();
    kinect.Open();
    if (kinect == null) return;

    InitializeDepth();
    IntializeColor();
    InitializeIR();
}

//Open Depth stream
private void InitializeDepth()
{
    if (kinect == null) return;

    //get frame description for depth output
    FrameDescription frameDescription =
    kinect.DepthFrameSource.FrameDescription;
    //get frame reader for depth
    depthReader = kinect.DepthFrameSource.OpenReader();

    //allocate pixel array
    depthData = new ushort[frameDescription.Width * frameDescription.Height];
    depthPixels = new byte[frameDescription.Width * frameDescription.Height *
    bytePerPixel];

    depthBitmap = new WriteableBitmap(frameDescription.Width,
    frameDescription.Height, 96, 96, PixelFormats.Bgr32, null);

    DepthImage.Source = depthBitmap;
    depthReader.FrameArrived += depthReader_FrameArrived;

}

//Events triggers every time a depth frame arrives
void depthReader_FrameArrived(object sender, DepthFrameArrivedEventArgs e)
{
    DepthFrameReference reference = e.FrameReference;

    if (reference == null) return;
    DepthFrame depthFrame = reference.AcquireFrame();

    if (depthFrame == null) return;

    using (depthFrame)
    {
```

```
        FrameDescription depthframeDescription =
        depthFrame.FrameDescription;

                if (((depthframeDescription.Width *
                depthframeDescription.Height) == depthData.Length) &&
                (depthframeDescription.Width == depthBitmap.PixelWidth))
            {
        //copy depth frames
        depthFrame.CopyFrameDataToArray(depthData);

        // Get min & max depth
        ushort minDepth = depthFrame.DepthMinReliableDistance;
        ushort maxDepth = depthFrame.DepthMaxReliableDistance;

        // Adjust visualisation
        int colorPixelIndex = 0;
        for (int i = 0; i < depthData.Length; ++i)
        {
            // Get depth value
            ushort depth = depthData[i];

            if (depth == 0)
            {
                depthPixels[colorPixelIndex++] = 0;
                depthPixels[colorPixelIndex++] = 0;
                depthPixels[colorPixelIndex++] = 0;
            }
            else if (depth < minDepth || depth > maxDepth / 2)
            {
                depthPixels[colorPixelIndex++] = 0;
                depthPixels[colorPixelIndex++] = 0;
                depthPixels[colorPixelIndex++] = 0;
            }
            else
            {
                //double gray = (depth);
                depthPixels[colorPixelIndex++] = (byte)depth;
                depthPixels[colorPixelIndex++] = (byte)depth;
                depthPixels[colorPixelIndex++] = (byte)depth;
            }

            // Increment
            ++colorPixelIndex;
        }
    }
```

```
            depthBitmap.WritePixels(new Int32Rect(0, 0,
            depthframeDescription.Width, depthframeDescription.Height),
            depthPixels, depthframeDescription.Width * bytePerPixel, 0);
    }
}

//Open Color stream
private void IntializeColor()
{
    if (kinect == null) return;

    //get frame description for color output
    //info about the image
    FrameDescription colorFrameDescription =
    kinect.ColorFrameSource.FrameDescription;

    //get the frame reader for color
    colorReader = kinect.ColorFrameSource.OpenReader();

    //allocate pixel array
    colorPixels = new byte[colorFrameDescription.Height *
    colorFrameDescription.Width * bytePerPixel];

    //create new writeable bitmap
    colorBitmap = new WriteableBitmap(colorFrameDescription.Width,
    colorFrameDescription.Height, 96, 96, PixelFormats.Bgr32, null);

    CameraImage.Source = colorBitmap;

    //coordinate mapping
            xcoordinate = new float[colorFrameDescription.Height *
            colorFrameDescription.Width];
            ycoordinate = new float[colorFrameDescription.Height *
            colorFrameDescription.Width];
            this.coordinateMapper = this.kinect.CoordinateMapper;
            this.depthMapToColor = new
            DepthSpacePoint[colorFrameDescription.Height *
            colorFrameDescription.Width];

            colorReader.FrameArrived += colorReader_FrameArrived;

}

//Events triggers every time a color frame arrives
void colorReader_FrameArrived(object sender, ColorFrameArrivedEventArgs e)
{
```

```
    ColorFrameReference colorRef = e.FrameReference;

    if (colorRef == null) return;

    ColorFrame colorFrame = colorRef.AcquireFrame();
    if (colorFrame == null) return;

    using (colorFrame)
    {
       FrameDescription frameDesciption = colorFrame.FrameDescription;

                if (frameDesciption.Width == colorBitmap.PixelWidth &&
                colorFrame.FrameDescription.Height == colorBitmap.PixelHeight)
       {
          if (colorFrame.RawColorImageFormat == ColorImageFormat.Bgra)
          {
             colorFrame.CopyRawFrameDataToArray(colorPixels);
          }
          else
          {

             colorFrame.CopyConvertedFrameDataToArray(colorPixels,
             ColorImageFormat.Bgra);
          }

          //copy output to bitmap
                      colorBitmap.WritePixels(new Int32Rect(0, 0,
                      frameDesciption.Width, frameDesciption.Height), colorPixels,
                      frameDesciption.Width * bytePerPixel, 0);

       }
    }
}



//Open IR stream
private void InitializeIR()
{
    if (kinect == null) return;

    //get frame description for depth output
              FrameDescription frameDescription =
              kinect.InfraredFrameSource.FrameDescription;

    //get frame reader for depth
```

```
    irReader = kinect.InfraredFrameSource.OpenReader();

    //allocate pixel array
            irData = new ushort[frameDescription.Width * frameDescription.Height];
            irPixels = new byte[frameDescription.Width * frameDescription.Height *
            bytePerPixel];


          irBitmap = new WriteableBitmap(frameDescription.Width,
          frameDescription.Height, 96, 96, PixelFormats.Bgr32, null);

    IRImage.Source = irBitmap;
    irReader.FrameArrived += irReader_FrameArrived;

}

//Events triggers every time a IR frame arrives
void irReader_FrameArrived(object sender, InfraredFrameArrivedEventArgs e)
{
    // Reference to infrared frame
    InfraredFrameReference reference = e.FrameReference;

    if (reference == null) return;

    // Get infrared frame
    InfraredFrame irFrame = reference.AcquireFrame();

    if (irFrame == null) return;

    // Process it
    using (irFrame)
    {
        // Get the description
        FrameDescription frameDescription = irFrame.FrameDescription;

                if (((frameDescription.Width * frameDescription.Height) ==
                irData.Length) && (frameDescription.Width == irBitmap.PixelWidth)
                && (frameDescription.Height == irBitmap.PixelHeight))
        {
            // Copy data
            irFrame.CopyFrameDataToArray(irData);

            int colorPixelIndex = 0;

            for (int i = 0; i < irData.Length; ++i)
            {
```

```
            // Get infrared value
            ushort ir = irData[i];

            // Bitshift
            byte intensity = (byte)(ir >> 8);

            // Assign infrared intensity
            irPixels[colorPixelIndex++] = intensity;
            irPixels[colorPixelIndex++] = intensity;
            irPixels[colorPixelIndex++] = intensity;

            ++colorPixelIndex;
        }

        // Copy output to bitmap
        irBitmap.WritePixels(
                        new Int32Rect(0, 0, frameDescription.Width,
                        frameDescription.Height),
            irPixels,
            frameDescription.Width * bytePerPixel,
            0);
    }
  }
}

//Image is capture once the user presses the "Capture" Button

private void Button_Click(object sender, RoutedEventArgs e)
{

    SaveData(this.depthData, "_depth" + counter);
    SaveData(this.irData, "_ir" + counter);
    SaveImage(this.colorBitmap, "_color" + counter);
    SaveImage(this.depthBitmap, "_depth" + counter);
    SaveImage(this.irBitmap, "_infrared" + counter);
    SaveCoordinateData(this.depthData, this.depthMapToColor);
    counter++;// for naming the image

}


//Saving the coordinate data into .txt file
  private void SaveCoordinateData(ushort[] Data, DepthSpacePoint[]
  depthSpacePoint)
{
```

```
    int colorPointCount = depthSpacePoint.Length;
    coordinateMapper.MapColorFrameToDepthSpace(Data, depthSpacePoint);
    for (int colorIndex = 0; colorIndex < colorPointCount; ++colorIndex)
    {

        xcoordinate[colorIndex] = depthSpacePoint[colorIndex].X;
        ycoordinate[colorIndex] = depthSpacePoint[colorIndex].Y;
    }
            SaveData(negativeInfinityConverter(this.xcoordinate), "xcoord" +
            counter);
            SaveData(negativeInfinityConverter(this.ycoordinate), "ycoord" +
            counter);

}

//Converts any value that is negative infinity to -1
private float[] negativeInfinityConverter(float[] floatArray)
{

    for (int i = 0; i < floatArray.Length; i++)
    {
        if (float.IsNegativeInfinity(floatArray[i]))
        {
            floatArray[i] = -1;
        }
        else
        {
            floatArray[i] = floatArray[i];
        }
    }

    return floatArray;
}

//Saving ushort type data into .txt files
private void SaveData(ushort[] Data, string name)
{
            string path =
            System.IO.Path.Combine(@"C:\Users\Katherine\Desktop\Thesis\Thesis
            Code\FaceDatabase", name + ".txt");
    //initialize a StreamWriter
    StreamWriter sw = new StreamWriter(path);

    //search the data and add it to the file
    for (int i = 0; i < Data.Length; i++)
    {
```

```
        sw.WriteLine(Data[i] + "\n"); //\n for a new line
    }

    //dispose of sw
    sw.Close();
}

//Saving float type data into .txt files
private void SaveData(float[] Data, string name)
{
    string path =
    System.IO.Path.Combine(@"C:\Users\Katherine\Desktop\Thesis\Thesis
    Code\FaceDatabase", name + ".txt");
    //initialize a StreamWriter
    StreamWriter sw = new StreamWriter(path);

    //search the data and add it to the file
    for (int i = 0; i < Data.Length; i++)
    {
        sw.WriteLine(Data[i] + "\n"); //\n for a new line
    }

    //dispose of sw
    sw.Close();
}

//Saving depth data into .txt file
private void SaveDepthData(ushort[] depthData)
{
    //initialize a StreamWriter
            StreamWriter sw = new
            StreamWriter(@"C:\Users\Katherine\Desktop\Thesis\Thesis
            Code\FaceDatabase\Depthdata.txt");

    //search the depth data and add it to the file
    for (int i = 0; i < depthData.Length; i++)
    {
        sw.WriteLine(depthData[i] + "\n"); //\n for a new line
    }

    //dispose of sw
    sw.Close();
}

//Saving images into .png file
private void SaveImage(WriteableBitmap Bitmap, string name)
```

```
        {
            if (Bitmap != null)
            {

                // create a png bitmap encoder which knows how to save a .png file
                BitmapEncoder encoder = new PngBitmapEncoder();

                // create frame from the writable bitmap and add to encoder
                encoder.Frames.Add(BitmapFrame.Create(Bitmap));

                        string path =
                        System.IO.Path.Combine(@"C:\Users\Katherine\Desktop\Thesis\Thesis
                        Code\FaceDatabase", name + ".png");


                FileStream fs = new FileStream(path, FileMode.Create);
                encoder.Save(fs);
            }
        }
    }
}
```

MainWindow.xaml:

```
<Window x:Class="Version1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="450" Width="1200">
    <Grid>
        <Image x:Name="CameraImage" Margin="720,0,0,44"/>
        <Image x:Name="DepthImage" Margin="360,0,472,59"/>
        <Image x:Name="IRImage" Margin="0,0,832,59"/>
        <Button Content="Capture" HorizontalAlignment="Left" Height="31"
Margin="529,365,0,0" VerticalAlignment="Top" Width="68" Click="Button_Click"/>

    </Grid>
</Window>
```

APPENDIX B: MATLAB Code

B.1: Face and Mouth Detection Code

```
%Face/Mouth Detector
%This is the main MATLAB file that processes the face and mouth detection
%for each video sequence that consist of 30 frames

%Folder Path of the Database and person
folderPath='C:\Users\Katherine\Desktop\Thesis\Digits Video Database\P8';

%Digit Selection
digit='Zero';

fileNameIR=strcat(digit,'IR');
fileNameDepth=strcat(digit,'Depth');

%initializations
depthImages=zeros(424,512,30);
irImages=zeros(424,512,30);
noseRow=zeros(30,1);
noseCol=zeros(30,1);
nosePointValue=zeros(30,1);
medianBoxSize=0;
boxSizeThreshold=10;
height = 424;

%For loop is used to process all 30 frames of the digit
for i=1:30

   %frames start at 0
   num=i-1;

   %raw data extraction
   depthData=importData(folderPath,strcat(fileNameDepth,...
         num2str(num),'.txt'),'i');
   irData=importData(folderPath,strcat(fileNameIR,...
      num2str(num),'.txt'),'i');

   %converting data to image
   depthImages(:,:,i)=reshape(depthData,[],height)';
   irImages(:,:,i)=reshape(irData,[],height)';

   %1. Face Detection
      [finalFaceBoundingBox,medianBoxSize] = faceDetection(...
            irImages(:,:,i),i,medianBoxSize,boxSizeThreshold);
```

```
    %Obtaining the face ROI for both IR and depth images
    x=finalFaceBoundingBox(1,1);
    y=finalFaceBoundingBox(1,2);
    col=finalFaceBoundingBox(1,3);
    row=finalFaceBoundingBox(1,4);

    %Extracting the IR and Depth Face Images from the facial
    % bounding box
    irFace=uint16(irImages(y:y+row,x:x+col,i));
    depthFace=depthImages(y:y+row,x:x+col,i);

    [faceRow, faceCol]=size(depthFace);


    %BorderThreshold
    borderThreshold=ceil(faceRow/8);

%2.Nose Point Detection
[nosePointValue,noseRow,noseCol]=nosePointDetection(depthFace,...
    borderThreshold,7);

%3. Depth Based Normalization
  [normalizedDepthFace]=DepthFaceNormalization(depthFace, ...


   nosePointValue);

    %Converts any possible negative values from normalized facial
    % depth image to NaN
    normalizedDepthFace(normalizedDepthFace<0)=NaN;

%4.Find Below Nose Facial Depth Image
[belowNoseFacialDepthImage,belowNoseFacialIRImage]...
     =belowNoseFaceRegion(noseRow,noseCol, ...


     normalizedDepthFace,irFace);

    %Vectorizes Below Nose Facial Depth Image, and removes any
    %negative values from normalized facial depth image
    depthArray=belowNoseFacialDepthImage(:);
    depthArray(isnan(depthArray)==1) = [];

%5.Segmentation
[maxPeaks,maxPeakLoc,minPeaks,minPeakLoc]...
        =depthValueSegmentation(depthArray,15,5,'n');
```

```
%intializations
segmentedBelowNoseFacialIRImage=belowNoseFacialIRImage;
segmentedBelowNoseFacialDepthImage=belowNoseFacialDepthImage;

%kth local min
for k=1:length(minPeakLoc)
   localMinLoc=k;

   %threshold
   segmentFaceMask=ones(size(belowNoseFacialIRImage));
   segmentFaceMask(belowNoseFacialDepthImage>...
      minPeakLoc(localMinLoc))=0;
   segmentFaceMask(isnan(belowNoseFacialDepthImage)==1)=0;

   %sum
   sumSegmentFaceMask=sum(sum(segmentFaceMask));

         %number of pixels within the %segmentedBelowNoseFacialDepthImage
         totalLowerFaceRegionPixel=size(belowNoseFacialIRImage,1)...
         *size(belowNoseFacialIRImage,2);

   %ratio
   ratioSegmentedFrace=sumSegmentFaceMask/...
      totalLowerFaceRegionPixel;

   if(ratioSegmentedFrace>=.40)
      break;
   end

end

   %Segmented Below Nose Facial Depth Image and corresponding IR %image
segmentedBelowNoseFacialDepthImage(belowNoseFacialDepthImage...
   >minPeakLoc(localMinLoc))=NaN;
segmentedBelowNoseFacialIRImage(belowNoseFacialDepthImage...
   >minPeakLoc(localMinLoc))=NaN;

%6. Lip Localization Algorithm
   [depthMouthBox] = lipLocalization(...
                  segmentedBelowNoseFacialDepthImage,...
                        borderThreshold,noseCol);

   %Inserts the mouth boudning box into the IR image for easy
   % viewing purposes
   dMouth = insertObjectAnnotation(...
```

```
                    mat2gray(belowNoseFacialIRImage)...
                , 'rectangle', depthMouthBox, 'Mouth');

    % 7.Find Mouth from the Mouth Region using Viola Jones mouth %detection
    [mbboxes_I]=ViolaJonesIRmouth(belowNoseFacialIRImage);
    VJMouth = insertObjectAnnotation(...
      mat2gray(belowNoseFacialIRImage), 'rectangle',...
      mbboxes_I, 'Mouth');

    %Image Comparision Between Viola Jones IR Mouth Detection and
    % Lip Localization Algorithm
    figure,
    subplot(1,3,1)
    imshow(mat2gray(belowNoseFacialIRImage))
    subplot(1,3,2)
    imshow(VJMouth)
    xlabel('Viola Jones')
    subplot(1,3,3)
    imshow(dMouth)
    xlabel('Lip Localization')
end
```

B.2: Import Data Code

```
function [dataName] = importData(folderPath,fileName,type)
%This function allows for file importing of different types
%The main focus of this function is to import txt files that contains depth
%and IR data

filepath=strcat(folderPath,'\',fileName);

fileID1 = fopen(filepath);

if(type=='i')
   dataName=fscanf(fileID1, '%i'); % Data in decimal
elseif (type =='f')
   dataName=fscanf(fileID1, '%f'); % Data in float
else
   return;
end

fclose(fileID1);
```

B.3: Face Detection Code

```
function[finalFaceBoundingBox,medianBoxSize] = faceDetection(irImages,...
    i,medianBoxSize,boxSizeThreshold)
 %In this function, the IR image first passes through the Viola Jones
 %Detection, follow by the multiple face bounding box algorithm to
 % discard any possible false detections
 %i=the frame # in the video sequence
 %medianBoxSize=the running median Box Size Value
[faceboundingBox]=ViolaJonesIR(uint16(irImages));


      %1.1 Multiple Face Bounding Box Filtering
      if(i==1)
         %The first frame of a video sequency
         if(size(faceboundingBox,1)>1)
            medianBoxSize=max(faceboundingBox(:,4));
         else
           medianBoxSize=faceboundingBox(1,3);
         end
      end
      if(size(faceboundingBox,1)>1)
         for j=1:size(faceboundingBox,1)
            if(faceboundingBox(j,3)<=medianBoxSize+boxSizeThreshold...
                  && faceboundingBox(j,3)>=...
                  medianBoxSize-boxSizeThreshold)
               finalFaceBoundingBox=faceboundingBox(j,:);
            end
         end
      else
         %if there is only one bounding box for the face
         finalFaceBoundingBox=faceboundingBox;
         medianBoxSize=median(medianBoxSize,finalFaceBoundingBox(1,3));
      end

 end
```

B.4: Viola Jones Face Detection Code

```
function[bboxes_I] = ViolaJonesIR(image)

   % Viola Jones Face Detection
   % bboxes_I=the resulting face bounding box

   %Setting up the viola jones detection by seleting
```

```
    %the classification model
    faceDetector_ir = vision.CascadeObjectDetector('FrontalFaceCART');

    %Step function performs the Viola Jones Object
    %Detection based on the object specified from
    %the classification model
    bboxes_I = step(faceDetector_ir, image);

end
```

B.5: Final Nose Point Detection Code

```
function[nosePoint,noseR,noseC]=nosePointDetection(depthFace,...
        borderThreshold,medFiltSize)
% Final Depth Based Nose Point Detction
% borderThreshold=width # to clear off from the sides of the image
% medFiltSize=Median Filter Size
% nosePoint=the corresponding depth value of the nose Point
% noseR=the row of the nose Point
% noseC=the column of the nose Point



    %Median Filtered Nose Point Detection
    [medNoseValue,medNoseRow,medNoseCol]=...
        medianFilteredNosePointDetection(depthFace,medFiltSize);

    [depthRow,depthCol]=size(depthFace);

    %Border Mask
    %The Border Mask takes on the same size as the depth face image
    %1. First the BorderMask Matrix contains elements with all 1s
    %2. Each side of the mask are converted to 0, the width of each side is
    %specified by the borderThreshold
    borderMask=ones(depthRow,depthCol);
    borderMask(:,1:borderThreshold)=0;
    borderMask(:,depthCol-borderThreshold+1:depthCol)=0;
    borderMask(1:borderThreshold,:)=0;
    borderMask(depthRow-borderThreshold+1:depthRow,:)=0;

    %Minimum Depth Mask
    %Created to include all nose point values from
    %the median filtered nose point, all possible nose points are denoted
    %as 1 while all other pixel element location are denoted as 0
    minDepthMask=zeros(size(depthFace));
```

```
    for j=1:length(medNoseCol)
        minDepthMask(medNoseRow(j),medNoseCol(j))=1;
    end

    %Image Multiplicaiton
    %The resulting binary image contains potential nose
    %points that are notwithin the specified border width
    borderMinPixels=minDepthMask.*borderMask;

    %Conditional Statement for sum
    if(sum(borderMinPixels(:))==0)
        [nosePoint,noseR,noseC]=clearBorderNosePtDetection(...
            depthFace,borderMask);

    elseif(sum(borderMinPixels(:))>=1)
        %If sum is 0, then the clearBorderNosePoint Detection is used to
        %clear any potential borders and to find the true nose point
        [noseR,noseC]=find(borderMinPixels==1);
        nosePoint=medNoseValue;

    end

end
```

B.6: Median Filtered Nose Point Detection Code

```
function[nosePoint,noseR,noseC]=medianFilteredNosePointDetection(...
    depthFace,medFiltSize)

    %2 dimensional median filtered depth face image
    %A filter size of 3x3,5x5,7x7, and 9x9 were attempted, but from
    %experimentation, the filter size of 7x7 gave the best results for
    %correct nose point detection
    % medFiltSize=7;
    medianFiltDepth = medfilt2(depthFace, [medFiltSize medFiltSize]);

    %Valid Depth Value Range Specified by the specs from Kinect V2
    minDepthThres=500;
    maxDepthThres=4500;

    %Ommitting all out of range values from the median filtered depth
    %face image
    [medianFiltDepth,~]=outOfRangeFiltering(medianFiltDepth,...
        minDepthThres,maxDepthThres);
```

```
    %Find the minimum value within the median filtered depth image
    minMedFiltDepth=min(min(medianFiltDepth));

    %Binary image of the region that contains the min value within the
    %median filtered depth face image
    binaryMedFiltDepth=medianFiltDepth<=minMedFiltDepth &...
        medianFiltDepth>=0;

    %Image dilation is used to expan the search area of binaryMedFiltDepth
    %since median filter might had filtered away what was actually a real
    %min element pixel, such can be done with image dilation
    structElement = strel('square', medFiltSize);
    dilatedBinaryMedFiltDepth=imdilate(binaryMedFiltDepth,structElement);

    %Minimum Depth Face Mask
    %Out of Range filtering is used to make sure no depth value that are
    %out of range are included in the nose point search
    maskDepthFaceImage=depthFace.*dilatedBinaryMedFiltDepth;
    [maskDepthFaceImage,~]=outOfRangeFiltering(maskDepthFaceImage...
        ,minDepthThres,maxDepthThres);


    %Minimum Value of the median filtered nose point detection
    nosePoint=min(min(maskDepthFaceImage));
    [noseR,noseC]=find(maskDepthFaceImage==nosePoint);

End
```

B.7: Clear Border Nose Point Detection Code

```
function[nosePoint,noseR,noseC]=clearBorderNosePtDetection(...
    depthFace,borderMask)
% This clears out a number of columsn or rows based on the value specified
% from borderMask, this limits where the potential nose point can be
% located, aka, can't be located around the borders of the image
        clearedBorder=double(borderMask).*depthFace;
        clearedBorder(clearedBorder==0) = NaN;
        [clearedBorder,~]=outOfRangeFiltering(clearedBorder,500,4500);
        nosePoint=min(min(clearedBorder));
        [noseR,noseC]=find((depthFace.*double(borderMask))==nosePoint);

end
```

B.8: Depth Face Normalization Code

```
function[normalizedDepthFace]=DepthFaceNormalization(depthFace,nosePoint)
%This function returns the normalized Depth Face Image

   %Valid Depth Value Range Specified by the specs from Kinect V2
   minDepthThres=500;
   maxDepthThres=4500;

   %Ommitting all out of range values from the median filtered
   %depth face image
   [depthFace,~]=outOfRangeFiltering(depthFace,...
      minDepthThres,maxDepthThres);
   normalizedDepthFace=depthFace - nosePoint;

end
```

B.9: Out of Range Filtering Code

```
function[filteredData,outOfRangeIndex]=outOfRangeFiltering(data,min,max)
   %   Converts all elements from data that are out of the range of
   %   min(exclusive) and max(exclusive) to NaN.
   %   The resulting output is now called filteredData,
   %   and the corresponding indices are of out of range is
   %   called outofRangeIndex

   outOfRangeIndex=data<min|data>max;
   data(outOfRangeIndex)=NaN;
   filteredData=data;

end
```

B.10: Below Nose Face Region Code

```
function[mouthDepthegion,mouthIRregion]=belowNoseFaceRegion(...
   noseR,noseC,depthFace,irFace)
% This function returns the below Nose Face Region of both the IR image and
% the Depth image
% noseR:Nose Point Row
% noseC:Nose Point Column
```

```
% depthFace:depth face image
% irFace:ir face image


   noseC=floor(median(noseC));
   noseR=max(max(noseR));

   %Lower Facial Depth Image
   lowerFaceDepthImage=depthFace(noseR:size(depthFace,1),:);
   lowerFaceIRImage=irFace(noseR:size(irFace,1),:);

   %Image Gradient, taking the first derivative of the
   %LowerFacialDepth Image with respect to the y direction
   [~, Gy] = imgradientxy(lowerFaceDepthImage);
   noseColVal=Gy(:,noseC);
   noseColValues=noseColVal;

   %Threshold
   noseColValues(noseColVal>0)=0;
   noseColValues(noseColVal<=0)=1;

   %finding the first change in derivative
   for i=1:size(lowerFaceDepthImage,1)-1
      if(noseColValues(i)==0 && noseColValues(i+1)==1)
         endofNoseRow=i+1;
         break
      end
   end

   mouthDepthegion=lowerFaceDepthImage(...
      endofNoseRow:size(lowerFaceDepthImage,1),:);
   mouthIRregion=lowerFaceIRImage(...
      endofNoseRow:size(lowerFaceDepthImage,1),:);

end
```

B.11: Depth Segmentation Code

```
function[maxPeaks,maxPeakLoc,minPeaks,minPeakLoc]=...
   depthValueSegmentation(depthArray,averageFilterSize,...
   medianFilterSize,displayFigure)
% This function first forms a histogram based on the depthArray and then it
% smoothes the histogram based on the averageFilterSize follow by a median
% filter that with the medianFilterSize
% Afterwards, the max and min(local min and local max)
```

```
% peaks are located on the filtered histogram
% curve, the end peak is included within the min peak
   depthArray(isnan(depthArray)==1)=[];
   range=max(depthArray)-min(nonzeros(depthArray))+1;
   histogram=zeros(range,1);

   for i=1:range
      histogram(i) = sum(depthArray==(min(depthArray)+i-1));
   end

   xaxis=1:range;
   smoothedData=smooth(xaxis',histogram,averageFilterSize);
   y = medfilt1(smoothedData,medianFilterSize);


   [maxPeaks,maxPeakLoc] = findpeaks(y) ;
   %local min
   [minPeaks,minPeakLoc] = findpeaks(-y) ;

   %include the end point for minPeaks
   endPeak=y(range);
   endPeakLoc=range;

   if (displayFigure=='y')
      figure
      plot(xaxis',histogram,'g')
      hold on
      plot(xaxis',smoothedData,'r')
      hold on;
      plot(xaxis',y,'b--');
      hold on
      plot(maxPeakLoc, maxPeaks ,'ko')
      hold on
      plot(minPeakLoc, y(minPeakLoc) ,'m*')
      hold on
      plot(endPeakLoc, endPeak ,'c*')
      title('Histogram Based Depth Segmentation:Lower Facial Depth Image')
      xlabel('Depth (mm)')
      ylabel('Frequency')
      legend('Original Data','Averaging Filter','Median Filter','Peaks','Local Min','End
Point')
   end

   minPeakLoc(length(minPeakLoc)+1)=endPeakLoc;
   minPeaks(length(minPeaks)+1)=endPeak;
```

end

B.12: Lip Localization Code

```
function[depthMouthBox] = lipLocalization(...
    segmentedBelowNoseFacialDepthImage,borderThreshold,noseCol)
% This function finds the corresponding Mouth Boundinb box based on the
% segmentedBelowNoseFacialDepthImage, the borderThreshold is used to clear
% some potential hair clusters, and the noseCol that corresponds to the
% nose point is used to search for the the boundaries of the mouth

    [~, YGradCentDiff] = imgradientxy(...
       segmentedBelowNoseFacialDepthImage,'CentralDifference');

    %Y Gradient Mouth To Chin Binary Image
    %We know that there should be some abrupt changes around the mouth
    %area, so we use the threshold of -1
    yGradMouth2ChinBW=YGradCentDiff<=-1;
    yGradMouth2ChinBW(isnan(yGradMouth2ChinBW)==1)=0;

    %Hair Border Removal Algorithm
    hairBW=hairBorderRemoval(yGradMouth2ChinBW,borderThreshold);
    sideMouthBW = bwareaopen(hairBW, 10);

    %Expansion for the Search Area
    expandedSideMouthBW=YGradCentDiff<0;
    structElement = strel('square', 5);
    dilatedBinary=imdilate(sideMouthBW,structElement);
    sideMouthBinaryMask=expandedSideMouthBW.*dilatedBinary;

    %Nose Point Column
    mouthCenterCol=ceil(median(noseCol));

    %MouthColFinder
    [rightMouthEndCol,leftMouthEndCol]=MouthColFinder(...
       sideMouthBinaryMask,mouthCenterCol);

    %Y Gradient Mask using Intermediate Difference
    [~, YGradIntDiff] = imgradientxy(...
       segmentedBelowNoseFacialDepthImage,...
       'IntermediateDifference');

    %Binary Image for finding the bottom of the lips
    YGradIntDiffBW=YGradIntDiff<=-1;
    YGradIntDiffBW(isnan(YGradIntDiffBW)==1)=0;
```

```
    [endofButtomLip]=buttomLipFinder(YGradIntDiffBW,noseCol);

    %Lip Localization Bounding Box Paramter Formation
    depthY=1;
    depthX=leftMouthEndCol;
    depthRow=endofButtomLip;
    depthCol=rightMouthEndCol-leftMouthEndCol+1;

    %Lip Boudning Box
    depthMouthBox=[ depthX, depthY, depthCol,depthRow];

end
```

B.13: Mouth Col Finder Code

```
function[rightMouthEndCol,leftMouthEndCol]=MouthColFinder(...
  mouthBW,mouthCenterCol)
  %Given the Binary Image that includes the mouth,the search begins in
  %the at the mouthCenterCol(nose point column) and goes from the center
  %to the ends of both sides
  for a=mouthCenterCol:size(mouthBW,2)-2
    if (sum(mouthBW(:,a))>0 && sum(mouthBW(:,a+1))==0 ...
        &&sum(mouthBW(:,a+2))==0 )
      rightMouthEndCol=a;
      break

    elseif(a==size(mouthBW,2)-2)
      rightMouthEndCol=1;
    end
  end

  for k=1:mouthCenterCol-2
    c=mouthCenterCol-k+1;
    if (sum(mouthBW(:,c))>0  && sum(mouthBW(:,c-1))==0 ...
        && sum(mouthBW(:,c-2))==0)
      leftMouthEndCol=c;
      break
    elseif(k==mouthCenterCol-2)
      leftMouthEndCol=size(mouthBW,2);
    end
  end

end
```

B.14: Mouth Row Finder Code

```
function[endofButtomLip]=buttomLipFinder(mouthBW,noseCol)
%This function finds the end of the lips by using find the first change
%in the binary image, the search is limited to the corresponding nose
%column of that binary image because we know that nose column is similar to
%the mouth center, this returns the row where the end of the lip is located
      endofButtomLip=NaN;
      colThreshold=0;
      noseColMin=min(noseCol)-colThreshold;
      noseColMax=max(noseCol)+colThreshold;

   for p=1:size(mouthBW,1)-1
      q=size(mouthBW,1)-p;
      if (sum(mouthBW(q+1,noseColMin:noseColMax))==0)
         if (sum(mouthBW(q,noseColMin:noseColMax))>0)
                  endofButtomLip=q;
                  break
         end
      end
   end

      if(isnan(endofButtomLip))
         return
      end

end
```

B.15: Viola Jones Mouth Detection Code

```
function[bboxes_I] = ViolaJonesIRmouth(image)

   % Viola Jones Mouth Detection
   % bboxes_I=the resulting mouth bounding box

   %setting up the viola jones detection by selecting
   %the classification model
   faceDetector_ir = vision.CascadeObjectDetector('Mouth');

   %Step function performs the Viola Jones Object
   %Detection based on the object specified from the classification model
   bboxes_I = step(faceDetector_ir, image);
```

```
end
```

B.16: IR and Color Image Reconstruction and Alignment with Face Detection

```
%Coordinate Map Reconstruction from Depth Space To Color Space
%Test Set Image Number
num=1;

%raw data extraction
depthData=importData(strcat('_depth',num2str(num),'.txt'),'i');
irData=importData(strcat('_ir',num2str(num),'.txt'),'i');
colorImage=imread(strcat('_color',num2str(num),'.png'));
xcoord=importData(strcat('xcoord',num2str(num),'.txt'),'f');
ycoord=importData(strcat('ycoord',num2str(num),'.txt'),'f');


%Default Kinect Dimensions for depth and color Image
dheight = 424;
dlength=640;
cheight = 1080;

%Data to Matrix conversion
depthMatrix=reshape(depthData,[],dheight)';
irMatrix=reshape(irData,[],dheight)';
xMatrix=reshape(xcoord,[],cheight)';
yMatrix=reshape(ycoord,[],cheight)';

%downsample x and y coordinate maps
x1=downsample(xMatrix,3);
x2=downsample(x1',3);
xCoordMap=x2';
y1=downsample(yMatrix,3);
y2=downsample(y1',3);
yCoordMap=y2';

%downsample color image
color1=downsample(colorImage,3);
color2(:,:,1)=downsample(color1(:,:,1)',3);
color2(:,:,2)=downsample(color1(:,:,2)',3);
color2(:,:,3)=downsample(color1(:,:,3)',3);
final(:,:,1)=color2(:,:,1)';
final(:,:,2)=color2(:,:,2)';
final(:,:,3)=color2(:,:,3)';

%round up coordinate map values
```

```
depth2colorCoordMap(:,1)=ceil(xCoordMap(:));
depth2colorCoordMap(:,2)=ceil(yCoordMap(:));


%searchAndRecover lost values from the coordinate map, all pixel elements
%that are -1 represents values that are not a number
%Certain values that are not a number on the coordinate map follows a
%pattern,for example, if the pattern is 1 _ _ _ 5,
%then the recovered values are 2 3 4.
[row, col]=size(depth2colorCoordMap);

for n=2:row-1
   before=depth2colorCoordMap(n-1,1);
   current=depth2colorCoordMap(n,1);
   future=depth2colorCoordMap(n+1,1);
   %pos neg pos
   %case where there is a -1 between two numbers that can be consecutive
   %ex: 2 -1 4, the -1 becomes 3
   if(before>0 && current<0 && future>0 && before==future ...
         && depth2colorCoordMap(n-1,2)==(depth2colorCoordMap(n+1,2)-2))
      depth2colorCoordMap(n,1)=before;
      depth2colorCoordMap(n,2)=depth2colorCoordMap(n-1,2)+1;
      %pos neg neg
      %for a series of -1 between two numbers that can become
      %consecutive
   elseif(before>0 && current<0 && future<0)
      counter=1;
      futureRange=depth2colorCoordMap(counter+n,1);

      while(futureRange<0  && (counter+n)~=row)
        counter=counter+1;
        futureRange=depth2colorCoordMap(counter+n,1);
      end

      if(before==futureRange  && (depth2colorCoordMap(counter+n,2)...
            -depth2colorCoordMap(n-1,2))==counter+1)
         depth2colorCoordMap(n:n-1+counter,1)=before;
         depth2colorCoordMap(n:n-1+counter,2)=...
            depth2colorCoordMap(n-1,2)+(1:counter);

      end
   end
end


coordinates(:,1)=depth2colorCoordMap(:,1);
```

```
coordinates(:,2)=depth2colorCoordMap(:,2);


subsize=size(xCoordMap);
numElements=length(xCoordMap(:));
newDepth=zeros(numElements,1);
newIR=newDepth;

%coordinate mapping from depth to color space
for i=1:numElements
   if(not(coordinates(i,1)==-1 || coordinates(i,2)==-1))
      y=coordinates(i,1);
      x=coordinates(i,2);
      newDepth(i)=depthMatrix(x,y);
      newIR(i)=irMatrix(x,y);

   end
end

%8 bit aligned color Image
alignedColorImage=final;

%aligned depth Image
alignedDepthImage=reshape(newDepth,[],subsize(2));

newIRImage=reshape(newIR,[],subsize(2));

%8 bit aligned IR image
alignedIRImage=uint8(uint16(newIRImage)./256);

%Viola Jones Face Detection: Color Vs. IR
[bboxes_color] = ViolaJonesIR(alignedColorImage);
[bboxes_I] = ViolaJonesIR(alignedIRImage);


VJcolor_face = insertObjectAnnotation(...
          alignedColorImage, 'rectangle',...
          bboxes_color, 'Face');

VJIR_face = insertObjectAnnotation(...
   mat2gray(alignedIRImage), 'rectangle',...
   bboxes_I, 'Face');

figure,
subplot(1,2,1)
imshow(VJcolor_face)
```

```
subplot(1,2,2)
imshow(VJIR_face)
```