Master Thesis report

Robust Hand Tracking And Gesture Recognition With Kinect Camera

Author
Abhijit Vyas
Mtr. # 2769947
Information and Automation Engineering (IAE)
Universität Bremen, Germany
vyasabhijit21@gmail.com

Supervisor
Prof. Axel Gräser
Institute of Automation(IAT)
Universität Bremen

Dr.-Ing. Adrian Leu
Institute of Automation(IAT)
Universität Bremen

October 17, 2015

Abstract

This thesis task is based on finding hand motion with color as well as depth information provided by kinect camera and tries to understand the hand gesture from the detected 3D position of human hand. The idea is to have some interface between human and robot with visual aid that can provide the information about hand movements and later the robot can emulate human gestures. The task is developed for MeRoSy (*Mensch Roboter Synergie*) project at IAE (Information & Automation Engineering) institute of University Bremen. This task is mainly divided in to two parts, the first section is to find and track the human hand with skin color information with combination of depth data from kinect. The main challenge for this task was the execution must be in real time with robust detection. Histogram Back Projection, CAMShift and Kalman algorithms along with depth tracking are utilized for successful results.

The second task was to understand human movement and define those movements in to certain gestures which can be later used for assigning different tasks to robot. This part of thesis incorporates the functionality of machine learning algorithm such as SVM(Support Vector Machine) and Shape Descriptors for simple shapes detection. It includes successful detection of circle, triangle and square/rectangle shapes and digits from 0 to 9. This task was tested with different users for live and recorded video streams. An efficient program was designed for Kinect 1 & 2 systems with windows and Linux+ROS compatible environments. The task is completed with successful results and can be first step forward for MeRoSy system in the direction of human robot interface.

Acknowledgments

It's been great pleasure to work with young and dynamic team and supportive supervisors like Dr.-Ing. Adrian Leu and Dr .-Ing. Danijela Ristic-Durrant. I would also like to take this opportunity and show my gratitude towards the Institute of IAE and our Head of the Department Prof. Dr.-Ing. Axel Gräser who has provided such an interesting and exciting opportunity to work with all necessary support that a student can get. I would like to thank all of above personalities for being such a kind and patient, specially my supervisor Adrian with whom I had countless discussions and meetings where he make sure that I do not get distracted from the main task. I would like to extend my well wishes to other student of the team who are working in the lab tirelessly knowingly and unknowingly providing motivation and support for this task.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Robust hand detection and tracking is an important part of vision based gesture recognition for natural Human Robot Interaction(HRI). It is worth to say that the hands are vital organs for human being to communicate and carried out many an important tasks on a daily basis. Hand and other body parts tracking from digital images has revolutionized not only the scientific world but for entertainment like gaming industry and animation creation. It also took it's place in traditional hard work on factory automation, virtual reality, rehabilitation purposes & disability support, performance measurement and many more. Now, there are approaches where user can have unique hardware attached to it's body which can have good analysis of the joints and geometry of some body parts. But it is neither natural nor comfortable. In addition, it is relatively expensive to wear specific hardware designed for certain tasks. Therefore, vision based applications are preferred over those approaches for simplicity, cost effectiveness and natural interface experience.

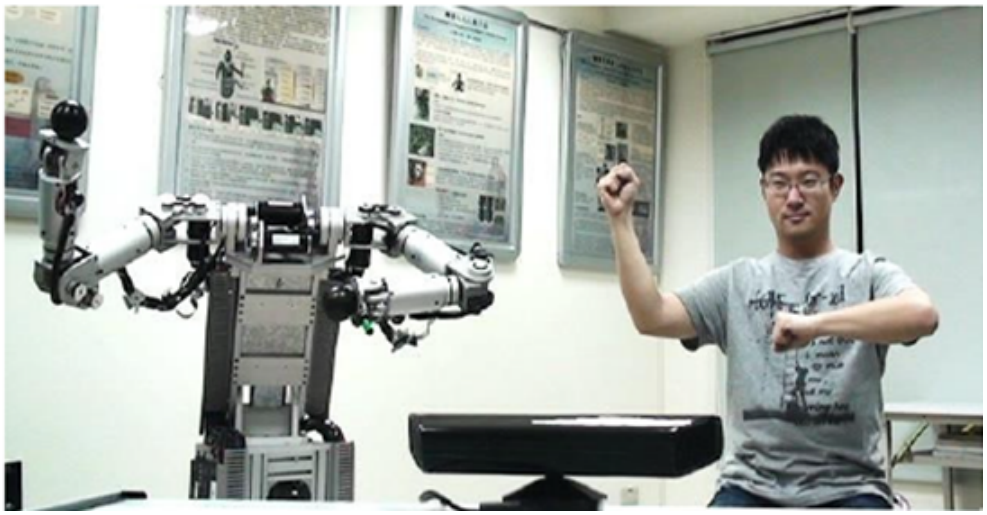


Figure 1.1: A robot is controlled by human demonstrations:TODO:REF:40

Conventional methods for hand detection includes color information from RGB cameras. But with development of RGB-D cameras, we can acquire depth informations along with RGB images which can help user to understand distance, shape and size of the object. These color and depth images are suitable for detecting and tracking an object

further more we can represent the positions and orientations in 3D world coordinates. In this task, we have used Microsoft Kinect camera for digital images and developed an algorithm for hand tracking in 3D space with skin color information along with depth tracking.

People needs to use mouse, keypads and remotes to control and manipulate machines. This interface between devices and humans can certainly be improved in coming years. Several research has been going on to modify existing tools and components which combine human and machine interface. It can be though of by giving commands to a computer or robot with speech and/or some gestures. This is the way we humans conduct communication among ourselves and we can definitely extend this way of communications to machines as well! This tasks is the small attempt in that direction. Since we need to train and test PI4 robot for the MeRoSy(Mensch Roboter Synergie) project, we would like it to understand human actions and commands with gestures. It is an exiting task carried out by technical team of this project in IAT department at Universtiy of Bremen.

1.2 Motivation and Objective

Hand detection and tracking for Pi4 robot is the initial task for MeRoSy project. It can also be considered as in terms of human mimicking in this huge task for making the robot suitable for industry work. The idea is that, the robot should observe human hand movements and be able to understand at some point what is going on around it's surrounding. That's why accurate and robust real time hand movement detection is priority task for this project which can be the first step forward to proceed. Pi4 will eventually work side by side with humans in industry area where it has to understand hand gesture and be able to repeat at some point of time. It must have a very good insight about where humans hands are with respect to its world coordinates and also it must be able to detect an object for being operated upon. In this task, we are limiting our approach to human hands tracking without any objects in hand. It would be simpler approach where hand movements are being observed by Pi4 and with gesture or trajectory detection algorithm developed in the later part of this thesis would help Pi4 to understand that movement.

1.3 Structure of the Report

- Chapter-2: This chapter presents the *Hand Detection Algorithm*. In the beginning of this chapter we will introduce the basic skin detection modeling of human hand and then we will try to explain additional functionality related to detection which will eventually lead to optimization of our approach in real time.

- Chapter-3: Describes the *Hand Gesture Understanding* which includes the supervised classification of different gestures with machine learning algorithm: SVM. It will explain in depth the trajectory understanding and its limitation with SVM

as well. Additionally shape descriptor approach has also been developed for basic shape detection such as square/rectangle and triangles.

- Chapter-4: Briefly discuss the environment set Up with kinect 2 and ROS indigo and Pi4 robot and it will help user to visualize the application related to this task. This chapter will also include the difference of working with kinect 1 as kinect 2. The performance measurement with both the cameras will be explained in brief.
- Chapter-5: This chapter will be related to results and future works. It will be concluded with our thesis task.

Chapter 2

Hand Tracking

Hand motion tracking algorithm locates hands on frame by frame basis and continuously do so as they move in 3D space. Most tracking algorithms consider object's shape size and color in to account. But once tracking is missed or lost, algorithms fails to find hands and again needs to reinitialize. Skin mask is an important feature for hand detection which will be the indicator of hands and face in the scene. Without clear skin mask, depth is not helpful to track hands in this case. Although false skin segmentation vs tracking efficiency needs to be balanced perfectly; by putting strict thresholds would limit false detection but at the same time will have less information to process. Hence efficiency of tracking algorithm will suffer. On the other hand if we keep thresholds too big then the noise and false skin indicator will increase error in detection.

2.1 Background And Related Works

There are several attempts made for real time robust hand tracking. This problem is quite old and many approaches has been developed to detect and understand hand movement with many different devices. Earlier people used to wear gloves and some special hardware to detect hand postilion and used to interact with the world. TODO: Ref: paper 20,21. RGB camera was one of the simple solution for complex hardware problem where people can extract skin information from color images with different chrominance models. TODO: Ref:paper24.

State of the art papers include many vision based object identification and tracking approaches. Representation with modeling, specifying key features for detection, contours tracking, or by using classification techniques such as neural network(TODO:Ref paper22) and binary tree classification. TODO:Ref paper22. Although machine learning algorithms and classifiers needs robust training with lots of data only then they are able to segment hand from images. This approach needs lots of processing and hence computationally expensive. The more traditional method where we use kinect color and depth information and try to find hand from those images combined; is an ideal approach for this task.TODO: Ref: paper1..

Skin color detection can be useful information for detecting hand position under certain conditions, but it is highly dependent on illumination. There is lots of research has been done on dynamic adaptive skin model representation with varying lighting conditions.refere: paper2:. M. Soriano, B. Martinkauppi etal. uses chromaticity-based constraint to select training pixels from a scene for updating dynamic skin color model. They observe chrominance under various lighting conditions and different camera calibration

and develops model based on that experiments and called 'skin locus'. This research is mainly focus on face detection and is dedicated to single type of camera where they create a model for skin locus which is suitable for that camera only. They also mention in their paper that on-line images or other camera images can not be used for training phase.

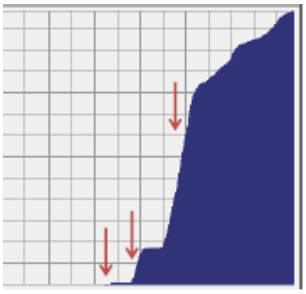
With complex background, one can observe skin model and additionally use assumption that the ROI is always in motion(Ref: paper4). There are cases where false skin detection under different lighting conditions in uncontrolled situations can occur. This can be identified with motion segmentation with finding difference between moving foreground and stationary background. This is brilliant idea for hand segmentation to reduce false skin detection and can certainly implementable. F. Dadgostar & A. Sarrafzadeh suggests two different approaches to track motion pixels; the first one is simple frame subtraction method. Although it has limitation as it's sensitive to camera noise, additionally hand or face movement will open and occlude regions in background and can contribute in some sort of false detection. The second method is more robust with Lucas and Kanade(1981) optical flow motion tracking, where they need to initialize feature points. We have tried this approach with opencv with our case and experience suggests that it is good for face motion where you can define multiple tracking points. But for hand, since it has smaller size compare to it's background which include face, this method is not that much helpful. Also it will place limitation of detection algorithm, since ROI must be in motion.

Figure 2.1: Skin color detection with similar background:TODO:REF4

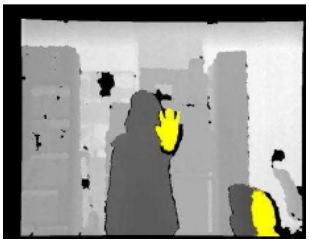
F. Dadgostar & A. Sarrafzadeh in their paper(TOOD:cite paper4), presents very good example of skin probability function and its uses, and how it can track wrong regions in a scene. They mention about Hue color space where hue is independent of intensity and can be ideally suitable for skin detection. The problem arises when along with user there are object which has skin color tone. For example a table may have certain wood like color which can be misidentified as skin color tone with Hue thresholding. Now if we are using skin probability function, where it represents the lighter pixels as skin

and darker as background. As shown in a Figure 2.1, the face region has darker pixels compare to the table in a background and if we solely rely on probability function where, higher probability will indicate the table region in background and that means a weak skin detector! That's why they introduce the motion detectors which can identify moving region of interest as compared to static background.

M. Park, Md M. hasan etal (TODO:cite:paper5) calculates depth histogram and thresholds depth image based on that. They assume that the hands are in front of the camera all the time and the rest of human body is considered as part of background. They calculate distance of hands and other body parts from accumulated histogram by analyzing rapid increase in the histogram slops. They assume that the hands are in front of camera and if slope is greater than certain value then this region is selected as candidate ROI. They use color information to segment other objects at the same depth level, thus they turn to skin color detection. This method is highly restrictive for hand movement, since it requires hands to be in front of camera all the times.



(a) Histogram calculated from depth image



(b) Depth Image with candidate regions

Figure 2.2: Depth histogram based thresholding:TOOD:Ref5

S. Joo, S. Weon etal ()cite:paper6) deals solely with depth images and finds hand and its movement. They propose classifier that combines a boosting and cascade structure. The features for training are depth differences at the stage of detection and learning. They also implements depth region grow and Depth Adaptive Mean Shift. Although they claim successful results the tracking algorithm can be quite slow because of region growing method.

2.2 Approach for Tracking

Figure 2.3 shows the organizational block diagram for rest this chapter. The full functional diagram is shown at the end of this chapter Figure 2.26 with final explanation. This approach is highly efficient in terms of processing time, and has requirement of accessing frames at 30 fps. Lower computational time helps us to track hands in real time, careful operations has been conducted for carrying out list of tasks with optimized performance. It uses the existing functionality of camshift from opencv and combination of kalman filter for successful tracking. In addition, utilizes the depth information for distance tracking which will help us to limit search window in z direction. At the end, it will provide 3D

hand position coordinates for further processing; training and testing certain gestures made by hand movement or provided to Pi4 robot for direct control of robot's arm.

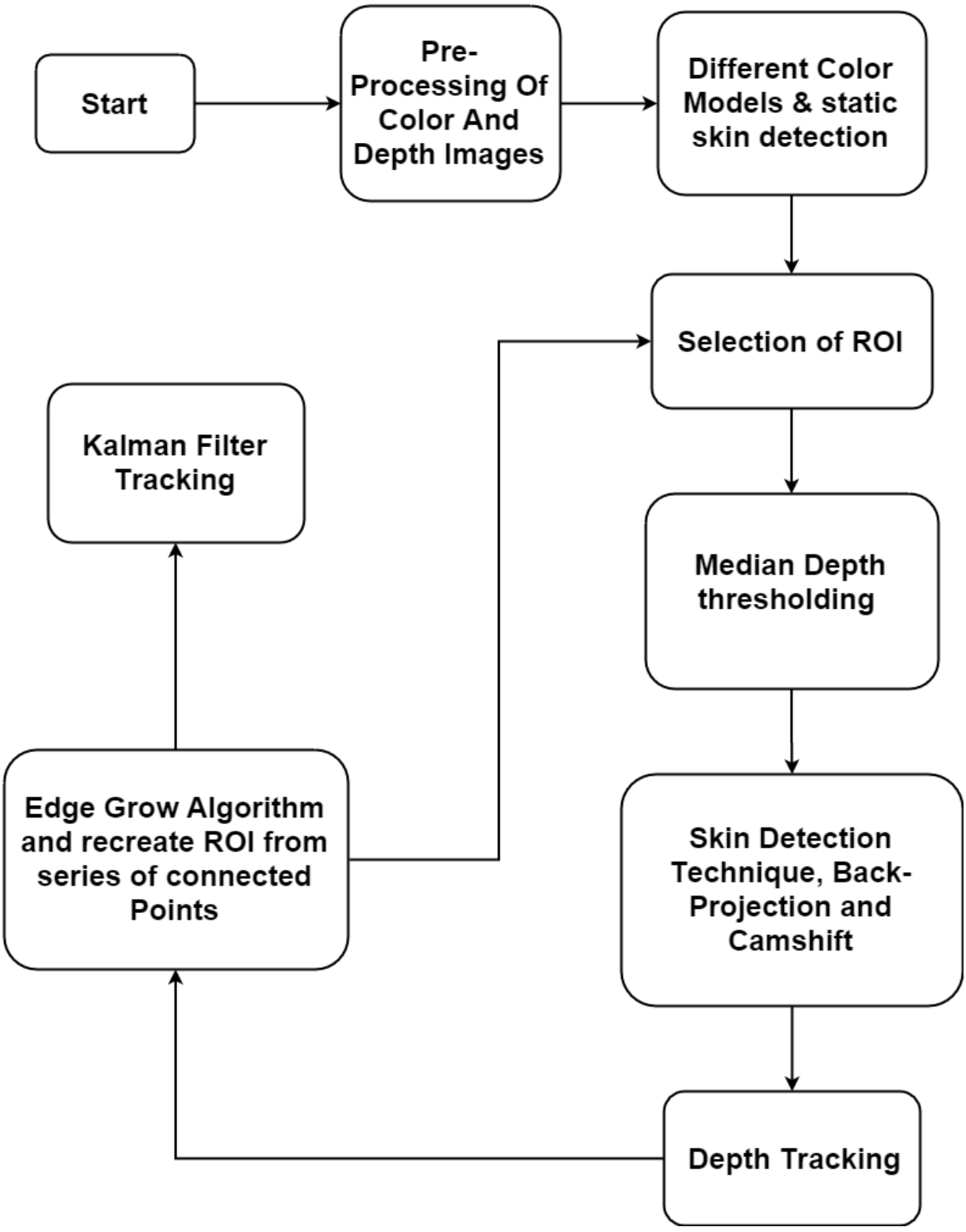


Figure 2.3: An organizational block diagram of Hand Tracking Algorithm

2.3 Pre-Processing of Color And Depth Images

Pre-processing is vital in any image processing application, such as scaling down for faster implementation, removing noise and creating masks for future usage. There are several embedded functions in opencv library some are very useful but others needs some time to perform and also quality needs to be improved. Hence some embedded functions needs to be manually implemented. One of the example is "PyrDown", it scale downs the image by specified size, but it is not suitable for depth images since it is required that the scaled down depth image is not altered in any way possible. Hence we need to manually scale down depth image for better results. Preprocessing steps are listed as below:

- Depth Image manual Scale down.
- Depth Image 2 meter distance background removal method and create depth mask. Multiply scaled down color image with depth mask, thus foreground segmentation based on depth. Also implement depth image flickering noise reduction technique.
- Depth Image edge Detection.

2.3.1 Scale Down of Images

For manual scale down of depth image, we loop through entire original depth image with size 640x480 and choose every fourth pixel in subsequent column and row to form scaled down image with size 160x120. If the pixel value is less than 500 which is kinect lower limit for correct depth detection, then we do median of 8 neighborhood around that pixel. It has certain advantages over conventional opencv method. Because opencv method not only scale-downs the size but it also smooths the image for better results as we can see from Figure 2.4. For hand detection task from scaled down image, smoothed or blurred regions will not be a clear way to distinguish certain features i.e. the finger details will not be as clear as in original image. It serves additional purpose with edge creation method to be more correct, this will be explained in depth-edge-image creation section 2.3.3. Although this method is not enough for kinect 2 depth image flickering noise, which clearly needs some more operations for improved performance.

- Loop through original depth image and check every fourth pixel in both x & y direction.
- If that pixel has value less than 500, then do median of 8 neighborhood pixels. Choose that value for current pixel.
- If current pixel value is greater than 500 then just select it.

(a) Scaled Down Depth Image: Manual

(b) Scaled Down Depth Image: Opencv

Figure 2.4: Depth Images Manual vs Opencv scale down

2.3.2 Background removal based on depth

In this step, we loop through scaled down depth image with size 160x120 and check for pixel values which are greater than 2000(mm) or 2 meter, we set them as null(0) and at the same time we create a map image which acts as mask for color image to remove unwanted background from the scene. This operation is quite helpful for foreground segmentation or background removal. The main limitation of this operation is that, depth image with kinect_1 depends upon infrared projected patterns and they are sensitive to direct sunlight. Hence if there is sunlight in our scene, then it will be difficult to operate on color image even though color image does not distort from sunlight. We allow that limitation in our case because in later operations, we need depth image without any distortion for calculating final ROI or track-window. Hence if depth image is not perfect it is the correct way to detect that there is some thing wrong with it. Also we can consider this operation as forming a point cloud with images where depth and color informations are fused together to represents a 2.5D point cloud. In that case however, if depth image is distorted, user can visualize by viewing cloud instead of checking depth image separately. Also this operation unveils that whether depth and color images are synchronized or not? This task serves one more advantage with eliminating shadow regions from depth image. As we can see from the Figure 2.5, the original color and depth image 2.5a,2.5c with back-ground and scaled-down color image without back-ground 2.5b,2.5d and these images will be utilized for further processing in our task. The steps are listed below:

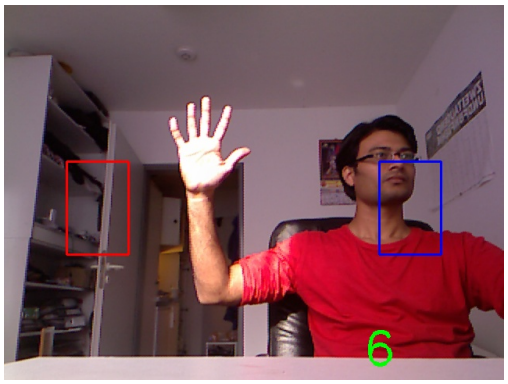
- Consider scaled down depth image with size 160x120.
- Loop through entire image and check if individual pixel value is greater than 2000?
- if so, mark it as 0, and update mask image.
- Utilize this mask with color scaled down image of same size. Do And operation between both images.
- Acquire color image with depth based background subtraction.

(a) Original Color Image

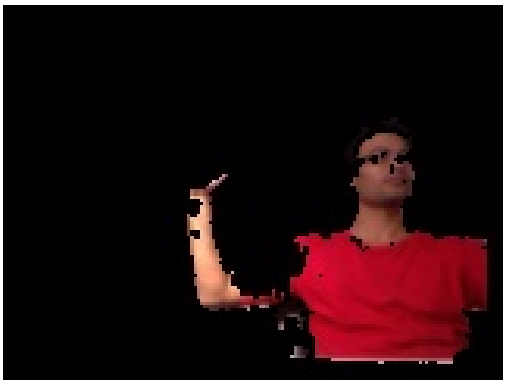
(b) Color Image with Back-Ground Subtraction

(c) Depth Image Without Back-Ground Subtraction

(d) Depth Image With Back-Ground Subtraction



(e) Color Image



(f) Color Image With depth illumination problem

(g) Depth Image With illumination problem

Figure 2.5: Color and Depth Images with/without Back-Ground Subtraction

Depth filtering is one of the operation which needs to be implemented smartly. With kinect_1 depth image does not have that much filtering requirement but with kinect_2 time of flight technology, it needs more attention. For this task we performed some morphological operations in the beginning such as opening and closing, but since depth should not altered in big way they are not so much effective. Hence we have tried other methods to reduce flickering noise like; the median and bilateral filters:TODO:ref:paper25. Problem with those filters is that they are quite slow. Thus, this task needs more attention in future implementations.

2.3.3 Depth Edge Image detection

Depth edge detection is one of the most important operation for our task. The result will be utilized for edge growing algorithm where we will rescale our tracking-window found from camshift. The idea is really quite simple, it checks each depth pixel and its 8 neighborhood pixels, if there is difference in value bigger than certain thresholds in our case 90 then the current pixel is defined as edge and it will be added in to map as gray image and depth edge image where original pixel value of edge will be preserved. Steps are as followed:

- Loop through scaled down depth image and check for each pixel, it's 8 neighborhood pixels values.
- If current pixel value and any of neighborhood values differ more than 90, then assign current pixel as depth edge pixel.
- Update depth edge image.

There are certain algorithms for edge detection like canny and sobel. But they detect 2D edge not in depth level. Figure 2.6 shows two different types of edge images, with first two figures 2.6a,2.6b we used traditional opencv pyrDown depth image for edge creation and it shows a lines of shadow image parallel to edges of the object. Next two images 2.6c,2.6d we use manual scale down method and it shows better results than earlier. This edge-depth image will be utilized for edge growing algorithm in the end where it will be helpful to find the top point of the hand and create a new ROI or track-window from that series of edge pixels.

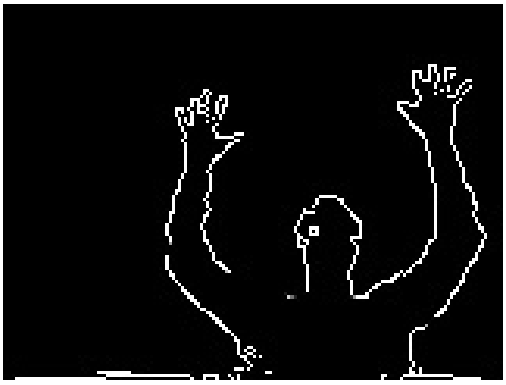
(a) Depth Image: opencv scaled down

(b) Edge Image: opencv scaled down

(c) Depth Image: manually scaled down

(d) Edge Image: manual scaled down

(e) Depth Image



(f) Canny Edge Image

(g) Edge Depth Image with Manual Method

Figure 2.6: Depth Edge Images with Back-Ground Subtraction

Canny and other edge detection algorithm can be useful, but the problem is, canny will return the gray image with edge pixels Figure 2.6f and we have to again create a depth map from those pixels to generate edge-depth image as shown in Figure 2.6g. The advantage of manual edge detection method is that, while finding and creating gray map for edge image, it can also create a edge map which has original depth value in edge formate, this will help us to calculate future edge grow algorithm.

2.4 Color Models And Static Skin Detection

Color is one of the most basic descriptor for object segmentation. There are different types of color models available for processing such as HSV, YCrCb, CIE-Lab, RGB and many more. The best choice for computer vision researchers for this task is HSV and YCrCb color spaces. We will take a closer look at different color models and see the usefulness and limitations.

RGB

RGB is Red, Green, and Blue color space which is one of the most simple color model. Different combination of all three main primary colors will make secondary additive chroma such as red and green will make yellow and white color can be created by full intensity of all three colors. Opencv has 8-bit,3-channel RGB color image representation, where 0 pixel value is represented as dark and 255 will be as maximum color intensity. These combinations can be quite useful for object detection and understanding. Although the main problem with RGB color space comes with the dependency over illumination. it is because of all three primary chroma components are highly correlated and in addition they do not have separation between luminance and chrominance (TODO Ref: paper3) . This does not mean that the RGB color model is not useful for computer vision applications, it is used in many different analysis and conclusions. In the end the final color information to humans must be provided in this color space!

2.4.1 HSV

HSV is one of the most popular cylindrical coordinate color space for skin segmentation and it is rightly so. It is also called HSB with BrightnessRef:wikipeadia. It can distinguish between chrominance and luminance Ref-paper3. **Hue** stands for dominant color representation, like red, green and blue. **Saturation** represents colorfulness with proportion to its brightness. **Value** or sometimes called **Intensity**, **Lightness** and/or **Bightness** is related to color luminance.

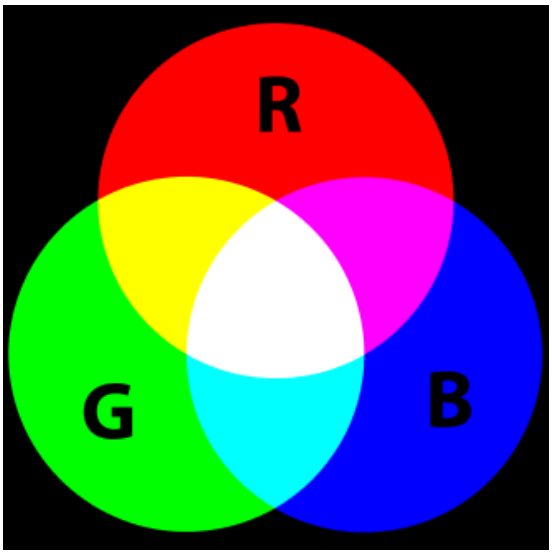
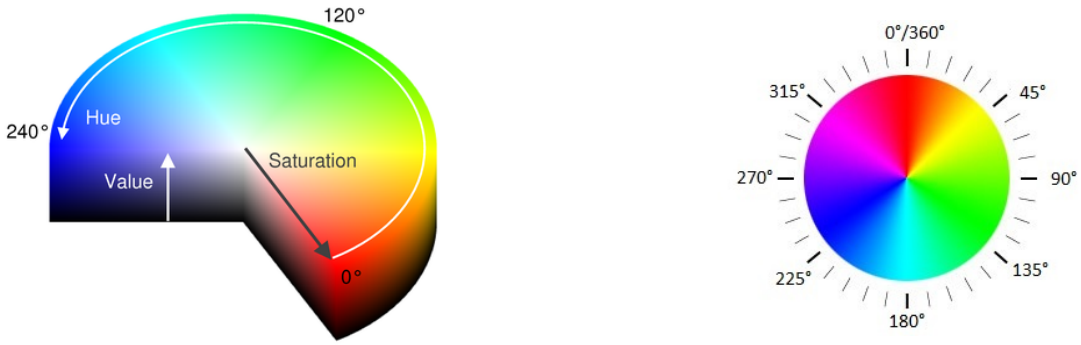


Figure 2.7: RGB Additive color representation



(a) HSV color space

(b) Hue color space

Figure 2.8: HSV color representation

Ref: wikipedia: In cylindrical representation the angle around central axis represents **Hue**, **Saturation** is the distance from the central axis and length along axis relates to **Value**. Red color starts as 0° Hue, Green at 120° and Blue at 240° and closing back to Red at 360°. Opencv Color representation is different than in theory, since opencv HSV color space only has 180° maximum limit for Hue, it is 0° for Red, 60° for Green and 120° for Blue and again 180° for Red. The reason behind is, since opencv represents HSV color space to 3 channel 8-bit image, Hue value has to be divided by 2 to fit into **uchar** unsigned character formate. Saturation and Value ranges from 0 to 255. The vertical axis

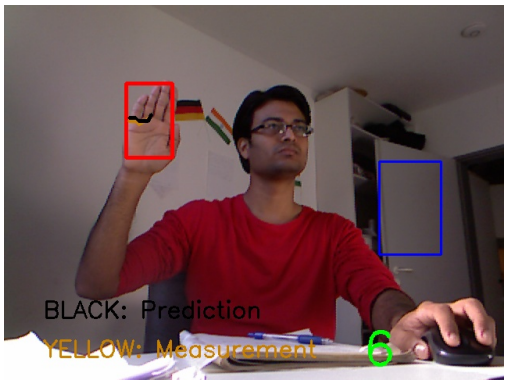
indicates gray levels black(0) at bottom to the top value 1 or 255(opencv).

$$
\begin{aligned}
V &= max\{R, G, B\} \\
S &= \begin{cases} \frac{V - min\{R,G,B\}}{V}, & \text{if } V \neq 0 \\ 0, & \text{otherwise} \end{cases} \\
H &= \begin{cases} \frac{60\{G-B\}}{V - min\{R,G,B\}}, & \text{if V = R} \\ 120 + \frac{60\{B-R\}}{V - min\{R,G,B\}}, & \text{if V = G} \\ 240 + \frac{60\{R-G\}}{V - min\{R,G,B\}}, & \text{if V = B} \end{cases}
\end{aligned}
\tag{2.1}
$$

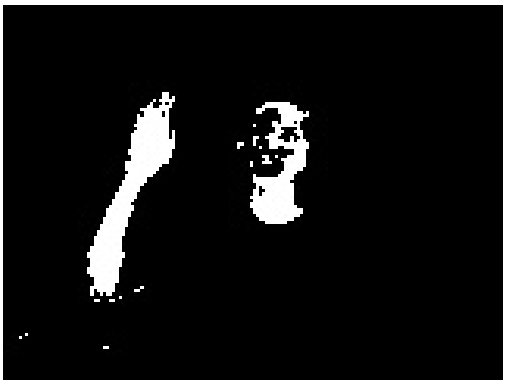
As per Equation 2.1, RGB to HSV conversion is done with opencv method with optimized way. This calculation will help us to understand the background conversion with opencv method. In literature, skin color model detection is trained by multiple color images with different background and illumination conditions. Skin probability density function can be obtained by those trained models, which would be helpful to calculate static as well as dynamic thresholds for upcoming frames. In several models the intensity is a key feature. If we cover bigger range for intensity the false detection will increase and would lead to less accuracy. However if we select specific values for intensity then it will reduce error but will also detect smaller number of skin pixels and that means a weaker classifier.

After many an experiments, it is understood that there can be two primary ranges for HSV model. Since skin color tone is similar to red hue color, the ranges include 0 to 22.5 and 157,5 to 180. We call it lower and upper range for skin detection. However, the main problem occurs when these ranges include false skin detection, for example the upper range is sensitive to different shades of red color, hence if there is pure red color in the scene then the classifier will produce wrong results. It can be also clear from the Figure 2.8b that if we start lower range from 0 then it will include pure red, hence we need to start from 2 to 22 and for upper range 157 to 175.

As shown in Figure 2.9b, with static thresholds for lower H: 2 to 12 and upper H: 168 to 175 with S: 65 to 255 and V: 20 to 255 HSV range limit, static skin mask will be generated. These ranges are selected as static thresholds and will be initialized at the beginning of the program.



(a) Original Color Image



(b) HSV static skin mask

(c) YCrCb static skin mask

Figure 2.9: Static skin thresholding with HSV and YCrCb models

2.4.2 YCbCr

YCbCr is the representation of RGB color space in to different form. Wikipedia describes it as a "encoded RGB information". This modeling technique is not only being used in vision based applications but in signal processing as well. Y represents luminance or brightness and Cb means blue color minus brightness and Cr represents red minus brightness. Green is "encoded" in the Y.

$$
\begin{aligned}
Y &= 0.299 * R + 0.587 * G + 0.114 * B \\
Cr &= (R - Y) * 0.713 + delta \\
Cb &= (B - Y) * 0.564 + delta \\
Inverse, \quad R &= Y + 1.403 * (Cr - delta) \\
G &= Y - 0.714 * (Cr - delta) - 0.344 * (Cb * delta) \\
B &= Y + 1.773 * (Cb - delta)
\end{aligned}
\tag{2.2}
$$

$$
Where, \quad delta = \begin{cases} 128, & \text{for 8-bit iamges} \\ 32768, & \text{for 16-bit iamges} \\ 0.5, & \text{for floating point images} \end{cases}
$$

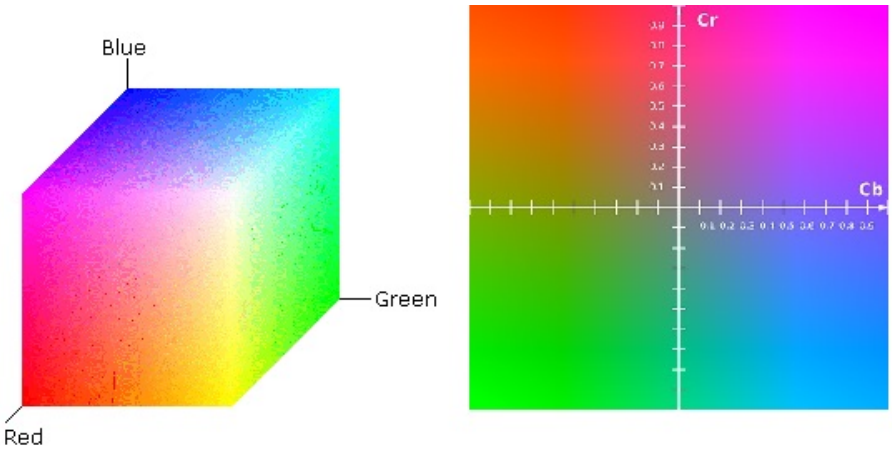


Figure 2.10: RGB and YcbCr

Opencv has very nice function for converting RGB color image to YCbCr and it can be observed from Equation 2.2 that the calculation is quite straight forward. The only limitation with this model is, it has similar sensitivity as upper hue thresholds limit. Different red color shades can be detected as false skin regions. As shown in Figure 2.9c, static skin mask is generated with red sensitivity. The benefit however is that this model is quite robust when it comes to illumination changes. It even acts normally during direct sunlight conditions and shadows without dynamic thresholds! It turns our static thresholding is more than enough for this model and it ranges for Y: 0 to 255, Cr: 145 to 170 and Cb: 75 to 255. Hence if we are open to introduce one limitation with not allowing complete red color to be present in the scene, this is the model to work with.

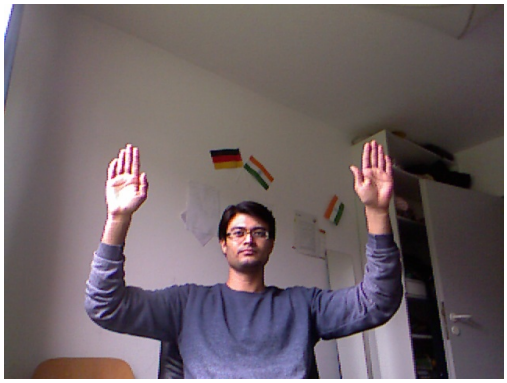
2.5 Selection Of ROI or Track-Window

This section is for initialization of region of interest in the image, this is the starting point of the hand tracking algorithm. It can be selected by two different ways. User can select it manually by mouse or use the intuitive initialization method to do so for both hands.

The automatic initialization works as mentioned below,

- Draw two Rectangles on screen for both hands.
- Use skin mask for this task.
- Loop thorough the mask and checks if the contour around the skin mask matches the size of initialized rectangles.
- If it does take it as the initial ROI or track-window.
- In the case of tracking loss because of bad illumination(white-window) or sudden jumps in depth(yellow-window), the actual tracking will be suspended. An automatic counter will start and if it reaches 50, this method will be called one more time.

- Hence after 50 frames user lost tracking, reinitialization will be offered to user once again.



(a) Original Color Image

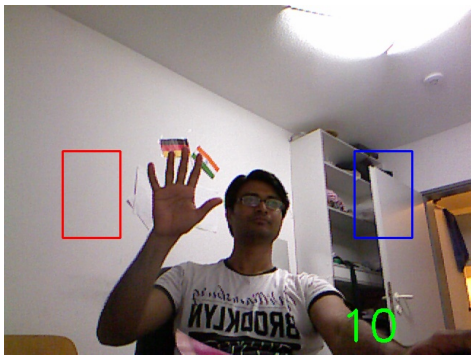
(b) Color Image-> Left Hand Initialized

(c) Color Image-> Right Hand Initialized

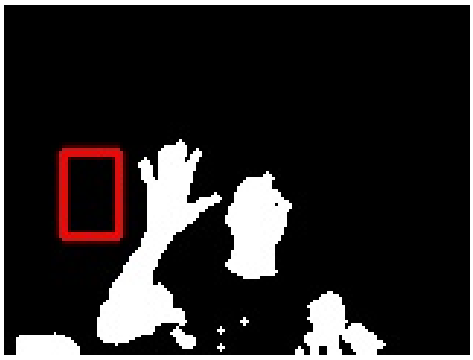
(d) Color Image-> Both Hands have been Initialized

Figure 2.11: Intuitive ROI Initialization

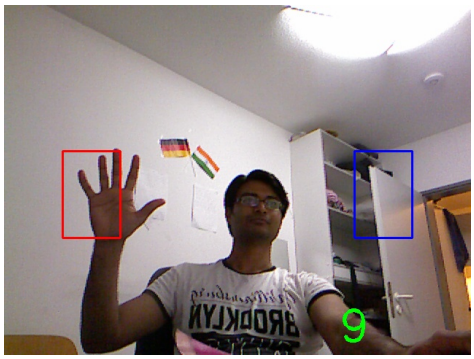
As we can see from the Figure 2.11, both hands needs to be placed close to the rectangles. Once the hands completely covers the individual rectangle, they are initialized. However sometimes because of bad illumination conditions, the skin mask would not be ideal for this task. During those situations, user needs to initialize hand position manually by mouse. The set of figures 2.12, explain in detail how it supposed to work with skin mask. As we can observe from those figures that once hand is near enough and large enough to cover initialized rectangle, it will be considered as likely candidate for intuitive hand initialization. Thus only partial occlusion with rectangles is not enough for tracking to start.



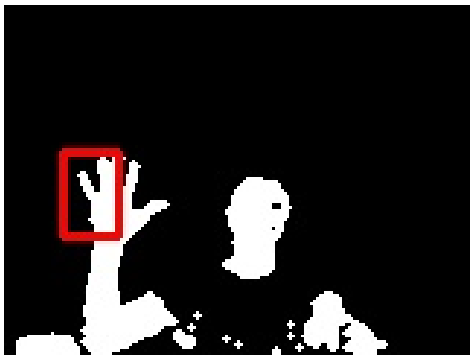
(a) Color Image-> Left Hand Initialized



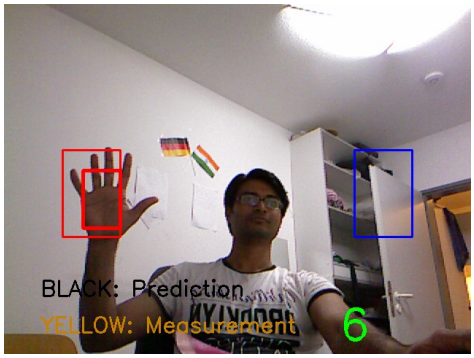
(b) Skin mask-> Left Hand Initialized



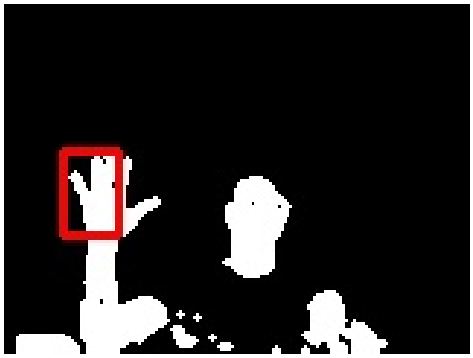
(c) Color Image-> Left Hand Initialized



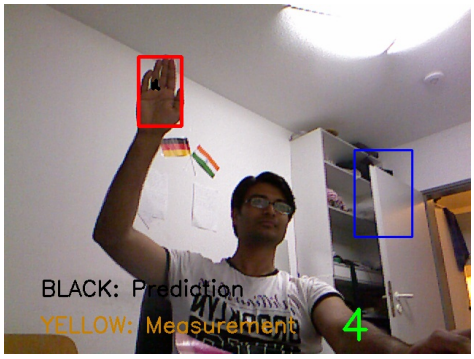
(d) Skin mask-> Left Hand Initialized



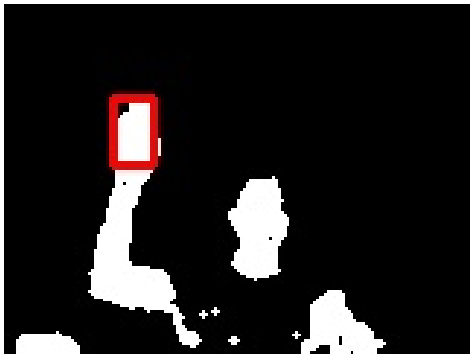
(e) Color Image-> Left Hand Initialized



(f) Skin mask-> Left Hand Initialized



(g) Color Image-> Left Hand Initialized



(h) Skin mask-> Left Hand Initialized

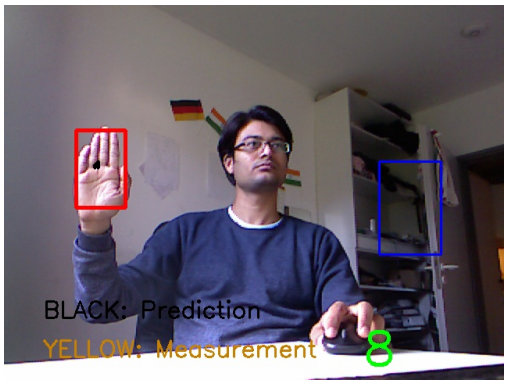
Figure 2.12: Intuitive ROI Initialization

2.6 Histogram based Back-Projection For Dynamic Skin Detection

Back-projection is the way to show the probabilities of a color in an image to appear in every pixel with the help of histogram.

Method works like this:

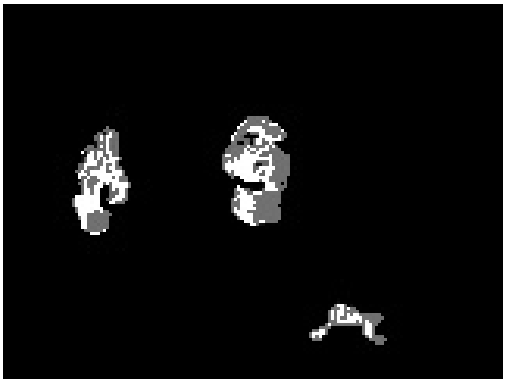
- Convert color to HSV image and separate hue channel from all 3 channels.
- Use primary static thresholds for hue skin thresholding, in our case lower-limit: 2 to 12 and upper-limit: 168 to 175, initialize dynamic thresholds with same values.
- Use this primary skin mask to calculate hue image histogram and normalize it.
- Initialize back-projection opencv method with normalized-histogram and input Hue image.
- Calculate secondary dynamic thresholds for next frame based on two biggest peaks of the histogram.



(a) Original Color Image

(b) Hue Image

(c) Histogram



(d) Back-Projection Image

Figure 2.13: Back Projection Images

As shown in Figure 2.13, we have color, Hue and Histogram images. Based on static skin mask and hue image, we calculate histogram of a model and see if this fits to given pixels of an image. Skin in hue image is analogous to different combinations of red color as explained earlier and can be seen from Figure 2.13c, hence if we have histogram of hand skin model then we would be able to find the exact combination for current illumination conditions.

Ref: 13 : Lets consider this case with more depth, we have calculated normalized histogram of skin model. Now we are given a test image and we need to find the skin distribution in that image with our previous knowledge based on normalized histogram. So we will check each pixel of test image p(i,j) and try to find correspondent bean value from our previous histogram for that pixel $\{h_{ij},s_{ij}\}$. Now assign that bean value to new pixel of a test image as a back-projection of it, do it for all pixels and soon we have entire back-projection as gray image. The value stored in image represents the probability of that pixel to be related to skin color model. And from Figure 2.13d it is clear that the lighter pixels are close to skin and darker ones represents background.

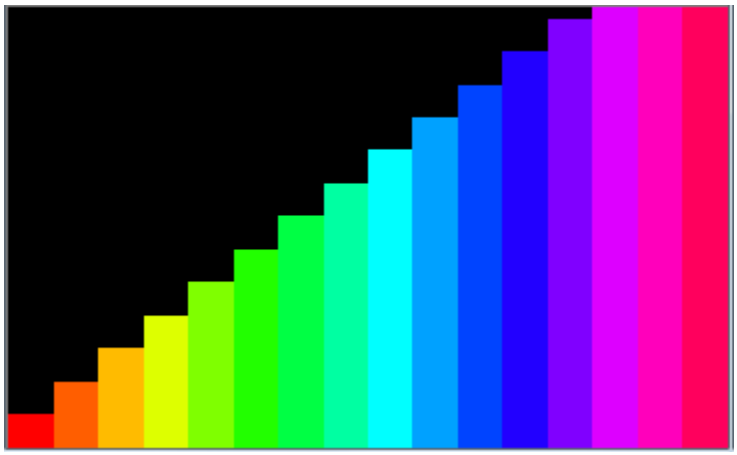


Figure 2.14: Histogram with All colors

Histogram with 16 beans will segment entire hue color span in equal 11.5 distance apart as we can see from Figure 2.14 with hue color space represented as histogram. Now from histogram, we can calculate two of the biggest peaks inside selected region. These two peaks will tell us about lower and upper thresholds limits of Hue for next frame. This procedure will be repeated each time from selected(track) window or ROI. Hence we will have dynamic hue values at each frames for hand skin color information. These threshold along with static thresholds will help us to find better skin mask from given colored image.

2.7 Depth Based Median Thresholding & Tracking

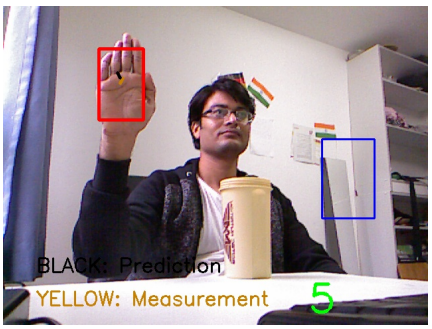
In this section, we use the tracking-window or ROI for calculating median depth of a region. This depth value provides vital information that how far our hands are from

camera. It will also be utilized for thresholding depth image and can create depth mask which should contain hands and other objects at that depth level; Figure 2.15c. Now in order to ignore other objects at same level as hands are, we again use track window to crop ROI from that mask as shown in Figure 2.15d. This mask will be multiplied with skin mask from last section 2.6 and then combination of depth and skin mask will be utilized to initialize camshift algorithm in section 2.8. It is expected that to avoid hand face occlusion problem, combination of this depth mask and back-projection image with camshift should have positive results. Since back-projection is 2D image, it does not differentiate between hand or face. Now if we carefully segment hand with depth level and multiply with this back-projection image, we have added depth information to this 2D mask.

Now, in addition of x and y directional tracking, if we can track with z direction then it would be the ideal 3D tracking. Sometimes during hand movement, depth suddenly jumps from one level to other which simply should not be allowed. It is the result of wrong median calculation because of dominance of background depth pixels in ROI. This can occur when our track-window is too large to cover hand and it will include background pixels from depth image as well. Now, since we are calculating median from the cropped ROI area, those pixels will be counted as well. It happens more often when hand moves toward direction of face and occlusion occurs as shown in Figures 2.16. Additionally user will see yellow text on screen top side indicating "Left-Dpth-median". The right solution is to count median depth from pixels of hand contour inside our track-window. Since hand-contour will include only hand pixels and not from background, depth tracking will be much more smooth. The steps are simple and explained as,

- Initialize track-window or ROI. Use this ROI with depth image.
- Take median of all corresponding depth pixels which are part of ROI and are related to skin mask, independent for both hands.
- Compare current median depth and store it in global variable to be utilized later.
- Do thresholding of depth image based on calculated median depth, independent for both hands.
- Crop thresholded mask with ROI and multiply with back-projection image, independent for both hands.
- Use this final mask as camshift input.
- Tracking: For the next frame, calculate difference between current and previous median depths.
- If it is greater than certain threshold, in our case 100 mm or 0.1 meter then raise flag.

- Once flag has been raised, stop camshift tracking and draw last position of track-window with "yellow" color to indicate tracking loss, write text on screen as "Left-Dpth-median"; Figures 2.16.



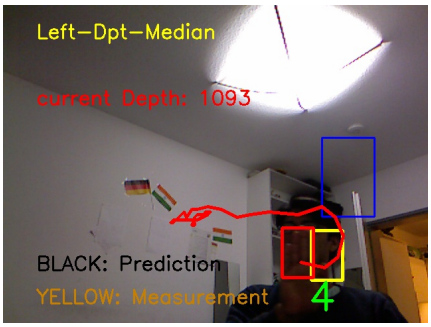
(a) Original Color Image

(b) Original Depth Image

(c) Depth Median Mask

(d) Depth Median Mask->Cropped

Figure 2.15: Median Depth Thresholding



(a) Color Image->Tracking lost

(b) Back-Projection Image->Tracking lost

(c) Depth Mask->Tracking lost

(d) Skin mask for camshift input->Tracking lost

Figure 2.16: Tracking loss temporarily because of occlusion and sudden jumps in depth

2.8 Continuously Adaptive Mean Shift Tracking (CAMShift)

The CAMShft algorithm or continuously adaptive mean-shift, which is modification of meanshift is used for finding the bounding region around maximum density of probability distribution in the histogram based back-projection image. It uses the track window or ROI for restricting the search and the mask we calculated above which is combination of back-projection and depth thresholded mask. In any tracking application, estimate of the position of an object is vital task. And in many cases for doing so, we do not need entire image to process. Often the close vicinity of the previous position can help us to relocate the new position of an object.

Meanshift calculates the centroid of entire gray image which is back-projection image. This centroid is used for weighting purpose such that, it gives higher weight to the lighter pixels, lower weight to relatively darker pixels and 0 to black pixels in a back-projection image. Probability density function or confidence map will be created for current frame based on hue color image histogram calculated from the previous frame. Meanshift will be used for finding peak of a map near object's previous position.

There is very good example illustrated in opencv tutorial (TODO:cite:Ref 11) for meanshift functioning. Consider we have a set of points, now the task is to find the highest pixel density in those set of points. We are given an area lets say a circle to represent the maximum pixels density, as shown in figure 2.17.

Now the initial window was shown as blue circle named as "C1" and center as small

blue rectangle marked as "C1_o". When Meanshift iterate over this circle, it will find the centroid or maximum density point as "C1_r" shown as small blue circle inside big circle. Hence it will shift the position of current circle or window to new centroid as its center point. Now it again repeats the process and probably shifts the window to final position marked as small green rectangle. It allows some degree of error in that position but mostly is very accurate in locating the centroid position.

The main difference between meanshift and camshift is, that the size of the tracking window stays constant in case of meanshift(TODO:cite:Ref:11). The problem comes when object is too close or too far away from camera. It makes sense to change window size simultaneously as well. Here camshift (Continuously Adaptive Meanshift) comes with the solution. It was first published by Gary Bradsky in his paper "Computer Vision Face Tracking for Use in a Perceptual User Interface" in 1988. It uses meanshift at its core and once meanshift finds its results, it updates the size of a tracking window as per Equation 2.3. Camshift also calculates the orientation with best fitting ellipse and returns rotated rectangle with opencv function.

$$s = 2 * \sqrt{\frac{M_{oo}}{256}} \tag{2.3}$$

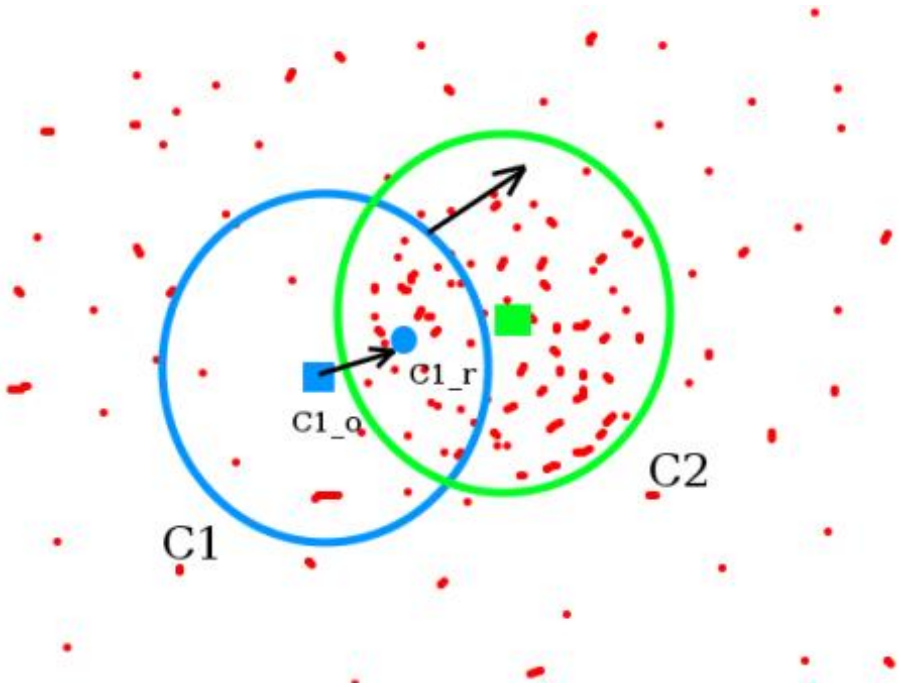


Figure 2.17: A Set of Points TODO:cite:ref11

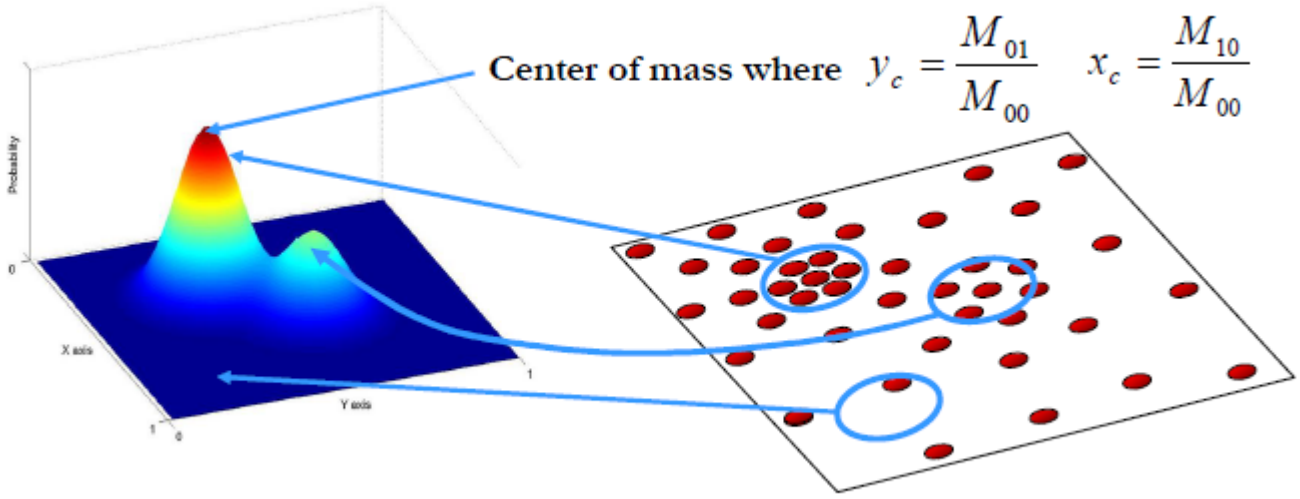


Figure 2.18: Camshift Probability Distribution [**?**]

In literature Ref7, wikipeadia, Moments computed from selected region by camshift are denoted as M_{00},M_{10},M_{01}. Ratio of these moments as shown in Figure 2.18 will be helpful for computation of centroid. This ratio will also be utilized for calculating gradient vector which will be used for next iteration. Once this vector is zero, it is assumed that the tracker is converged to center of an object. "Results are improved with camshift by allowing size of tracking-window to fluctuate and grants object to move away or towards the camera and returns size, aspect, and orientation of the window for further processing."

Hence for discrete 2D image probability distribution, the computation of mean center location (x_c,y_c) can be detected using the moments which we recorded earlier. "

$$x_c = \frac{\sum_x \sum_y xP(x,y)}{\sum_x \sum_y P(x,y)}; y_c = \frac{\sum_x \sum_y yP(x,y)}{\sum_x \sum_y P(x,y)} \tag{2.4}$$

Here, in Equation 2.4, P(x,y) = hI(x,y) is backprojection probability distribution at point x,y within search window I(x,y) which is computed from histogram h of I. Equation 2.4 is simplified to,

$$x_c = \frac{M_{10}}{M_{00}} y_c = \frac{M_{01}}{M_{00}} \tag{2.5}$$

Where;

$M_{00} = \sum_x \sum_y P(x,y) - 0^{\text{th}}$ *moment*

$M_{10} = \sum_x \sum_y xP(x,y) - 1^{\text{st}}$ *order moment*

$M_{01} = \sum_x \sum_y yP(x,y) - 1^{\text{st}}$ *order moment*

These moments represents the weighted average of the image pixel intensities. Once algorithm converge, two Eigenvalues in terms of major length and width of the probability distribution can be found as follows:

$$Let \quad a = \frac{M_{20}}{M_{00}} - x_c^2, \quad b = 2\left(\frac{M_{11}}{M_\infty} - x_c y_c\right) and \quad c = \frac{M_{02}}{M_\infty} - y_c^2 \tag{2.6}$$

The second order moments are defined as:

$$M_{20} = \sum_x \sum_y x^2 P(x,y); M_{02} = \sum_x \sum_y y^2 P(x,y) \tag{2.7}$$

A covariance matrix of

$$conv(P(x,y)) = \begin{bmatrix} a & \vec{b} \\ b & c. \end{bmatrix} \tag{2.8}$$

Finally the length and width can be represented as eigenvalues of matrix:

$$\begin{aligned} \lambda_i &= \frac{a+c}{2} \pm \frac{\sqrt{4b^2 + (a-c)^2}}{2} \\ Or; \quad l(length) &= \sqrt{\frac{(a+c) + \sqrt{b^2 + (a-c)^2}}{2}} \\ And \quad w(width) &= \sqrt{\frac{(a+c) - \sqrt{b^2 + (a-c)^2}}{2}} \end{aligned} \tag{2.9}$$

The orientation of major axis:

$$\theta = \frac{arctan\left(\frac{b}{a-c}\right)}{2} \tag{2.10}$$

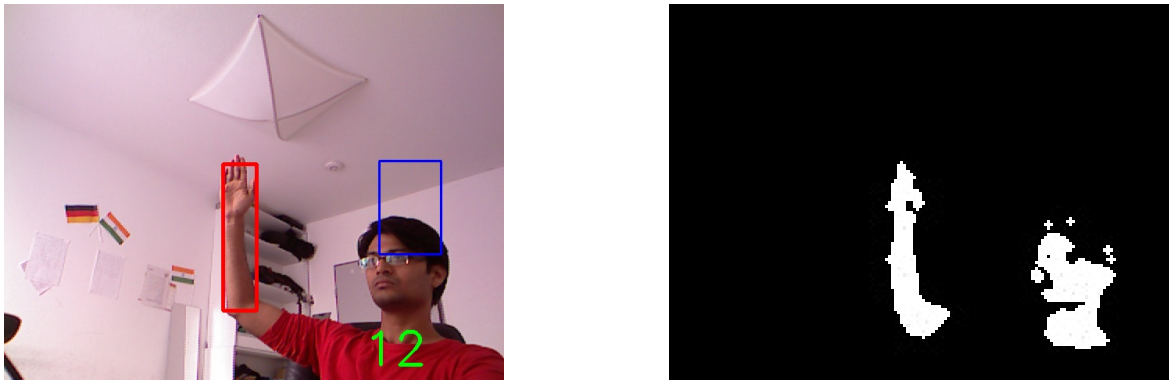
Camshift has efficiency in performance, ability to track object even if it is partly occluded, hence no dependency over shape and size of an object.

2.8.1 Limitation of CamShift

Camshift only tracks object in x and y coordinate system and is unable to track depth of an object, thus we have created our own depth tracker, which keeps the track of depth values for both individual hands. For hand tracking based on skin color, it is quite challenging for hand face occlusion situations. Since face has bigger area than individual hand has, the camshift will always tries to move towards face region.

The second limitation, is not camshift limitation per say, but it is real problem when we want to track hands. As shown in Figure 2.19, camshift will consider the entire arm as ROI or the target to track based on skin mask. Therefore tracking window would be

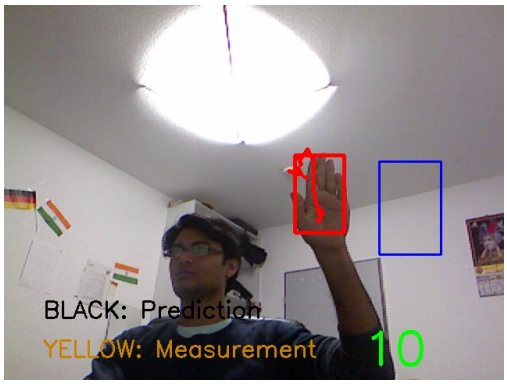
undesirably large, which should not be the case when we want to track palm instead of entire hands. Ideally, tracking-window should cover if not complete palm then as much palm as possible while being as small as possible. Thus, the need for additional algorithm to modify the large tracking window based on some close loop scenario case. Finding top point of hand or palm should be ideal for relocating such a window. The large camshift resulted window not only covers hands but back-ground, face and other unwanted regions too. By including background pixels will cause median depth algorithm work poorly and simultaneously presenting challenge for depth tracking 2.7. Thus we need to rescale our track-window found by camshift for better tracking results, this task would be explained in next section 2.10 of edge growing.



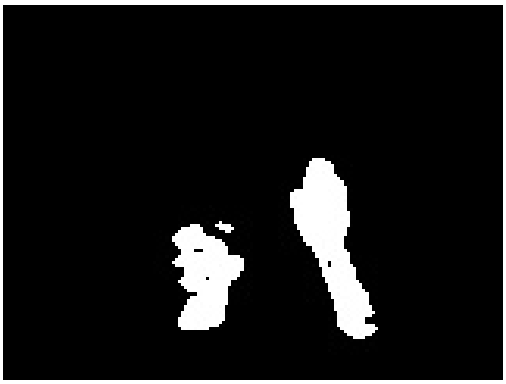
(a) Camshift result for entire arm (b) Skin mask-> Camshift

Figure 2.19: Camshift result for entire arm

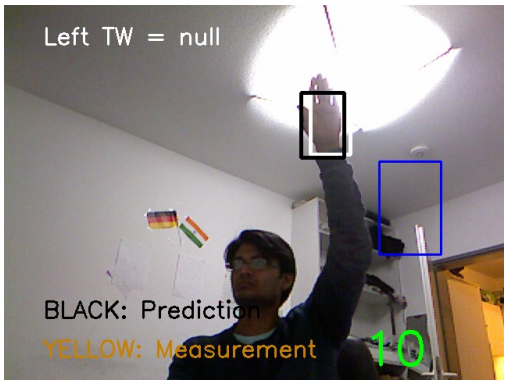
There will be time when skin mask will not be ideal, because of bad illumination conditions. The situation will make skin mask to null and even though depth is there, the combined median depth plus skin mask will be zero. Whenever this situation occurs, the camshift will output zero as well, hence there will not be any ROI to track down. During this situation, we output last position of track-window as "white" rectangle, it will be indication to the user that, in some part of the scene, the illumination conditions are not ideal to work as shown in Figure 2.20.



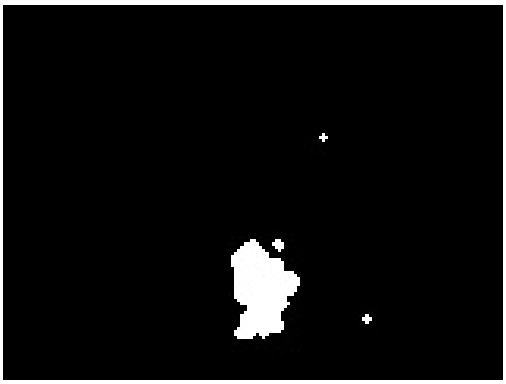
(a) Color Image->Normal



(b) Skin Mask->Normal



(c) Color Image->Tracking lost



(d) Skin Mask->Tracking lost

Figure 2.20: Tracking loss because of bad illumination condition

2.9 Tracking loss

There are two main types of tracking loss in this algorithm. Figures 2.20 are result of tracking loss because of bad illumination conditions. As we can see from the Figure 2.20d that the skin mask has no valuable data at the point where hand supposed to be, this results in to camshift tracking failure and camshift will output null. As it happens flag will be raised indicating whether left or right hand is null and user can see white tracking window which will be followed by Kalman prediction which is black tracking window in this case. If user gets out from that region that causes skin mask to be corrupted then tracking window will reinitialize itself from previous position where it lost its tracking. If not then we need to reinitialize manually by mouse or wait for 50 frames to pass and automatic initialization window will reappear.

There is second kind of tracking loss and was discussed in depth based median thresholding section 2.7 briefly, it is highlighted in Figures 2.16. This happens when hand comes close enough to face and occlusion situation occurs. In this case we have taken help from depth tracking and even use thresholded depth mask to get rid of face problem. But still as we can see from figure 2.16a, depth jumps from it's previous frame where it was 745 and in current 1093 where tracking was lost only for single frame. A flag will be raised named

Left/Right-dpt-Median, which is not discrete message but it does it's job! Although it is safe to say that if this happens for short period of time then it does not matter to us that much and at the same time we can see our original tracking-window as red in figure 2.16a. But if it happens for long then user has to reinitialize or go to that depth where it was lost and it will be initialize itself automatically.

2.10 Edge Growing

This algorithm comes to the picture once camshift publishes it's result. It uses the result for the initialization and then works with depth edge image for edge growing task. As we have discussed one of the limitation of camshift for hand palm tracking, that it covers entire arm if user chooses to wear t-shirt or other half sleeve shirt or top. This task will be explained in the detail here and will be justified it's purpose.

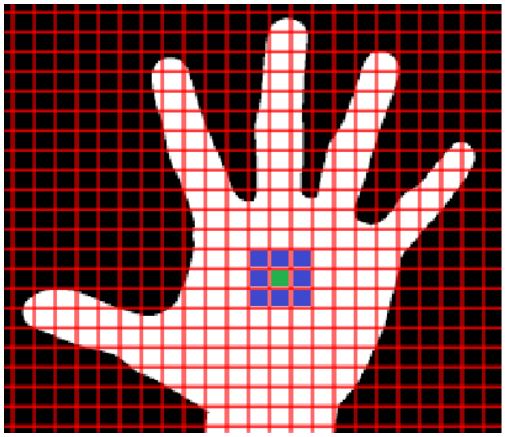
2.10.1 Region Growing

Initially, we were working with depth image to create hand geometry with depth *region growing* algorithm. The algorithm works as follow:

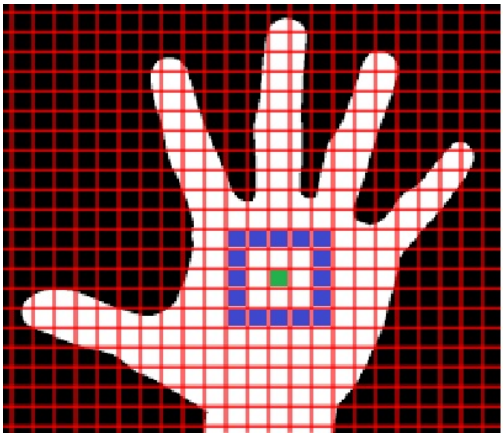
- Initialize seed point for Region-Grow Algorithm from camshift track-window center point.
- Now, try to find depth value to that center pixel.
- Look into 8 neighborhood pixels and check, if they have the similar depth or value in certain range.
- Create a map image and initialize the points which satisfies above condition and make them as new seed points for next iteration.
- Check the size of region width and height at each iteration.
- If the size of grown region reaches certain level according to depth level, stop recursive region growing. That means if depth is closer to camera, the region must be large for palm to cover and if depth is far away from camera then palm can fit into relatively smaller size of region.

It works fine but the problem with that task is that, it is quite slow for real-time hand tracking, although it will indicate hand palm nicely such as open or closed palm and number of fingers and so on. We also tried to make it fast by skipping certain neighborhood pixels, such as instead of considering immediate next 8 neighborhood pixels Figure 2.21a, we can jump to every second level of neighborhood Figure 2.21b. Here in both Figures, green is the seed pixel, and blue are the selected neighborhood pixels. Nevertheless, it can be utilized when user needs to know the shape of the hand. This algorithm can be

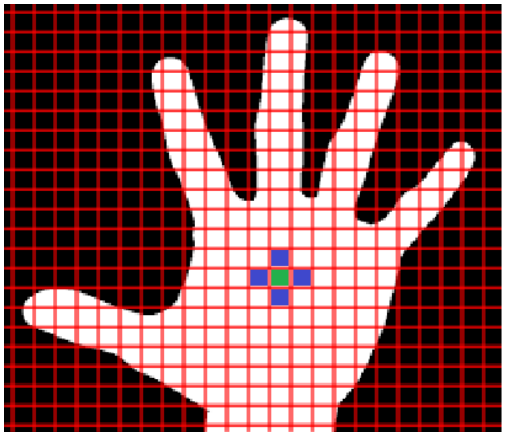
utilized certainly for much smaller edge growing region. It would not have the difficulties of being too slow for the real time operation.



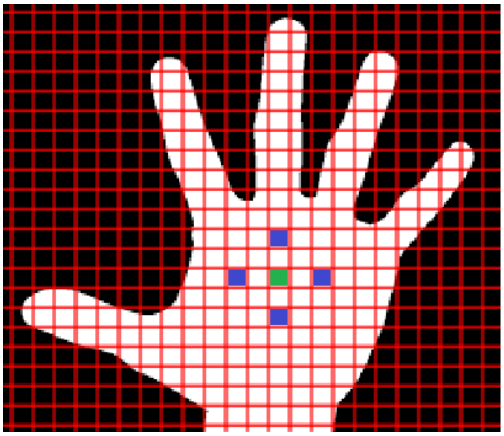
(a) 8 neighborhood pixels selection



(b) 16 neighborhood pixels selection



(c) 4 diagonal neighborhood pixels selection



(d) 4 sparse diagonal neighborhood pixels selection

Figure 2.21: Region Growing Seed Point and Neighborhood pixels selection

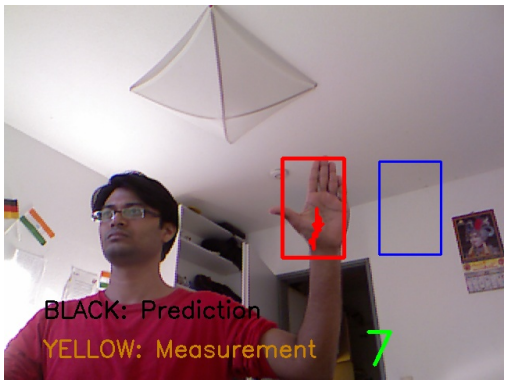
2.10.2 Approach

As we have discussed above region growing with depth image has certain limitation, however it has quite a nice application, which will be very useful for hand shape understanding. The other brief way to check out hand shape is to extract the contour around it and then try to figure out the shape from it. Although in our task we are not interested in the geometry of hand it can be useful for future applications. Now the idea is to consider depth edge image discussed in the section 2.3.3, and based on that image, we can grow edge just like we grow our depth region earlier.

The algorithm works as follow:

- Utilize camshift track-window center point and size. Based on depth level, try to find two closest points in horizontal or vertical directions.

- If camshift track-window has height greater than width then choose horizontal ie two different points on same image y-axis XL and XH TODO: ADD fig. YL and YH for track-window width greater than height. TODO: ADD fig
- Choose XL or YL as starting point of edge grow algorithm and XH or YH as stopping point.
- From start until stop point do edge/region growing and collect all set of points in both direction from start point as shown in TODO: Fig??.
- Now, we have the contour around our hand, but it still includes some unwanted points in the other direction do start point.
- In order to remove those points, make sure the end of these collection of points is stop point.
- Do closest neighborhood search from stop point, or ideally the last point of collected set of points. In this search, make sure each point from set is arranged from stop point until start point. This way we can ignore the other unwanted points.
- Now, we have a real set of connected points which include start and stop points.
- Once we have that set of points, try to find extreme points in both x and y directions. We will be able to find extream top-point from the set as shown in TODO: Figure?? and then two width points used for creating new track-window which should cover most of the palm area if not all.
- This new track-window found from these three points will be the final track-window of our hand tracking algorithm which can be utilized as kalman measurement data and for next frame ROI initialization.

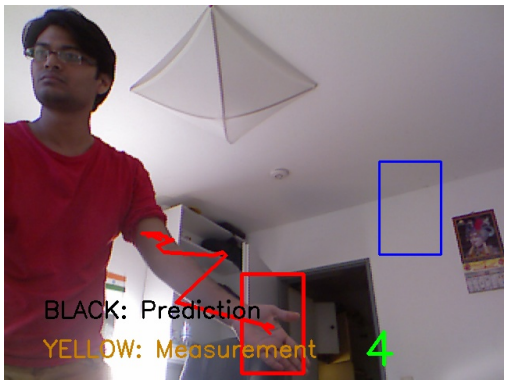


(a) Color Image

(b) Depth Image with start & stop points

(c) Edge Grown

(d) Final set of Points



(e) Color Image

(f) Depth Image with start & stop points

(g) Edge Grown

(h) Final set of Points

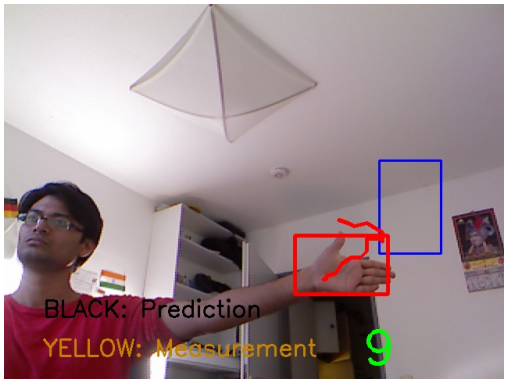
Figure 2.22: Depth Edge Grow Cases->TOP & BOTTOM

(a) Color Image

(b) Depth Image with start & stop points

(c) Edge Grown

(d) Final set of Points



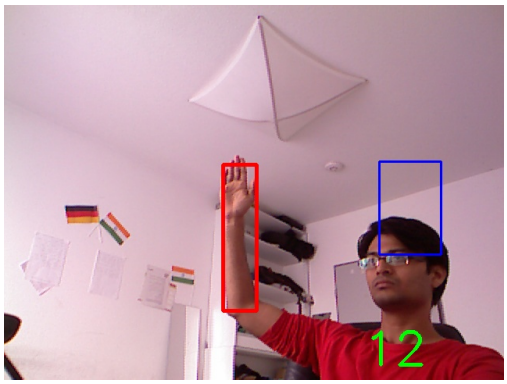
(e) Color Image

(f) Depth Image with start & stop points

(g) Edge Grown

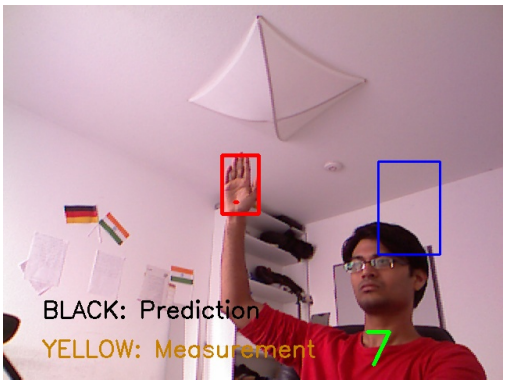
(h) Final set of Points

Figure 2.23: Depth Edge Grow Cases->LEFT & RIGHT



(a) Original Camshift result

(b) Skin mask-> Camshift



(c) Rearranged track-Window with edge grow algorithm

Figure 2.24: Camshift result vs Rearranged track-window from Edge Grow Method

2.11 Kalman Filtering and tracking

The Linear Quadratic Estimation(LQE) or in other words Kalman Filter is one of the most popular filtering algorithm. Series of measurements are being used along with measurement error for prediction of unknown variables. It has interesting application in control and robotic motion planning and estimation. It is fast robust and simple to implement. It comprises of two steps, first is measurement stage where the measurement data is being fed to predictor along with estimation errors. It will be used for prediction in next step where all those errors has been taken into consideration. Once prediction is done, for the next frame, the estimation and real errors will be utilized to make prediction more accurate.

Measurements in real system or in model are prone to some degree of noise either it is sensor noise or unknown environmental noise. It is a great deal for system to predict and estimate its measurement data in advance for more reliable operation. Kalman with weighted average of new measurement data and averaging of system states produces more reliable results than other prediction algorithms. Its dependency over only the last measurement rather than entire history of data is also one of the reasons for its fast response.

The gain of the filter is dependent of accuracy of measurement and estimate of state vector which can be tuned for desired performance.

2.11.1 State Space Representation Ref: paper7,15,16,17,18,19

In control theory, state space representation of system model is one of the most efficient way to describe the system. It can be seen from Equation 2.11

$$x_k = Ax_{k-1} + Bu_{k-1} + e_k \tag{2.11}$$

Here, for discrete-time linear system; k is discrete time index, x_k is state vector, A is transition matrix and e_k is the noise in the system. It is also called process noise and represented as covariance matrix Q. u_{k-1} represents external input, since in our case the hand positions are moving without any external force, this term will be zero.

For simplicity of the implementation, we use position x,y and width,height of an object as state variables, we do consider velocity in account but do not measure it in measurement matrix H. Now the above state space equation 2.11 is represented in matrix form,

$$\begin{Bmatrix} x_k \\ y_k \\ x'_k \\ y'_k \\ w_k \\ h_k \end{Bmatrix} = \begin{bmatrix} 1 & 0 & dT & 0 & 0 & 0 \\ 0 & 1 & 0 & dT & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} * \begin{Bmatrix} x_{k-1} \\ y_{k-1} \\ x'_{k-1} \\ y'_{k-1} \\ w_{k-1} \\ h_{k-1} \end{Bmatrix} + e_k \tag{2.12}$$

Equation 2.12 represents the state prediction, and the measurement prediction is given by Equation 2.13,

$$\begin{Bmatrix} Zx_k \\ Zy_k \\ Zw_k \\ Zh_k \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{Bmatrix} x_{k-1} \\ y_{k-1} \\ w_{k-1} \\ h_{k-1} \end{Bmatrix} + v_k \tag{2.13}$$

or;

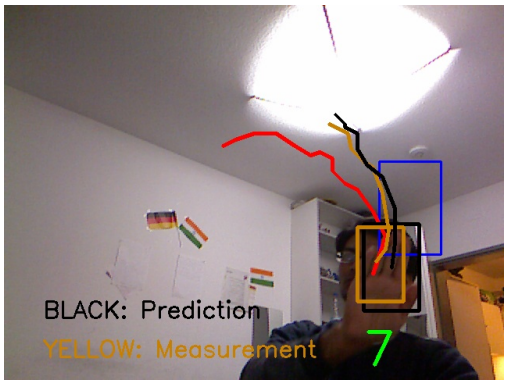
$Z_k = H * x_k + v_k$

Hence, the observation or measurement Z_k will have measurement-matrix H which ignores the velocity components in x and y directions, and x_k is the state estimate and v_k is measurement noise assumed to be zero mean Gaussian white noise with covariance Rk.

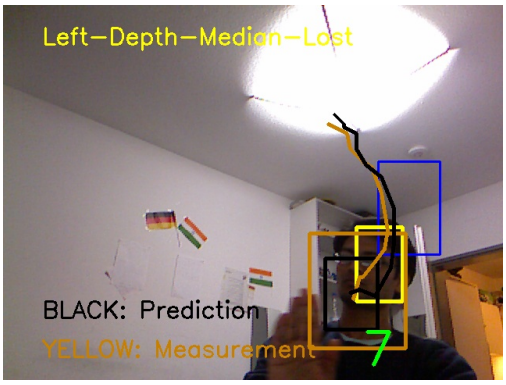
Being a recursive algorithm, it depends upon estimate of previous time stamp and measurement of current time stamps needed to compute estimate of current state. Hence no history of observations are needed.

In our case, kalman prediction has been utilized whenever there is tracking loss. It does not matter whether it is due to illumination conditions or may be sudden depth

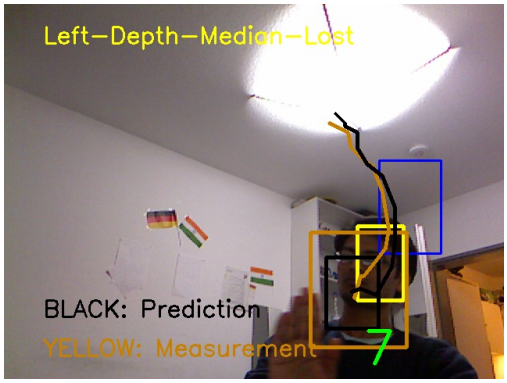
jumps from one level to other due to false skin detection. Kalman will predict for some frames the size track-window and position of palm center for utilization. We consider the prediction for only few frames and if during that time hand comes to right position from where it lost its tracking then this prediction results will be considered as part of final trajectory hand has made. In future, we need to implement kalman for z direction or depth for both hands as well, so that it can track not only x and y but complete 3D hand trajectory.



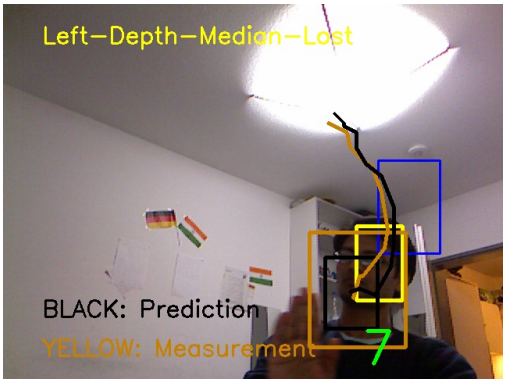
(a) Color Image->Normal



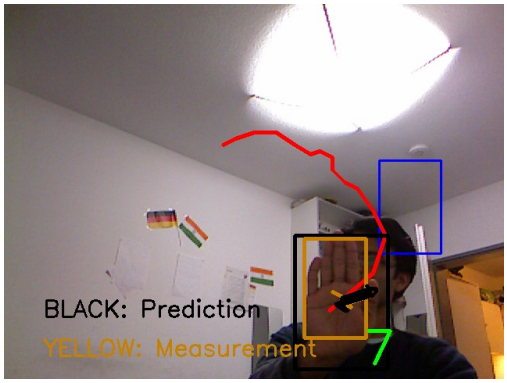
(b) Color Image->Tracking lost_1



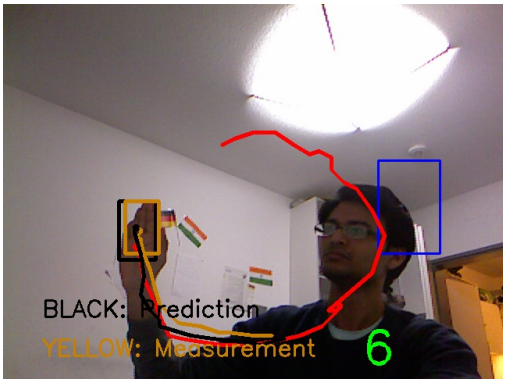
(c) Color Image->Tracking lost_2



(d) Color Image->Tracking lost_3



(e) Color Image->Normal Again_1



(f) Color Image->Normal Again_2

Figure 2.25: Kalman tracking with normal trajectory

2.12 Full Hand Tracking Block Diagram

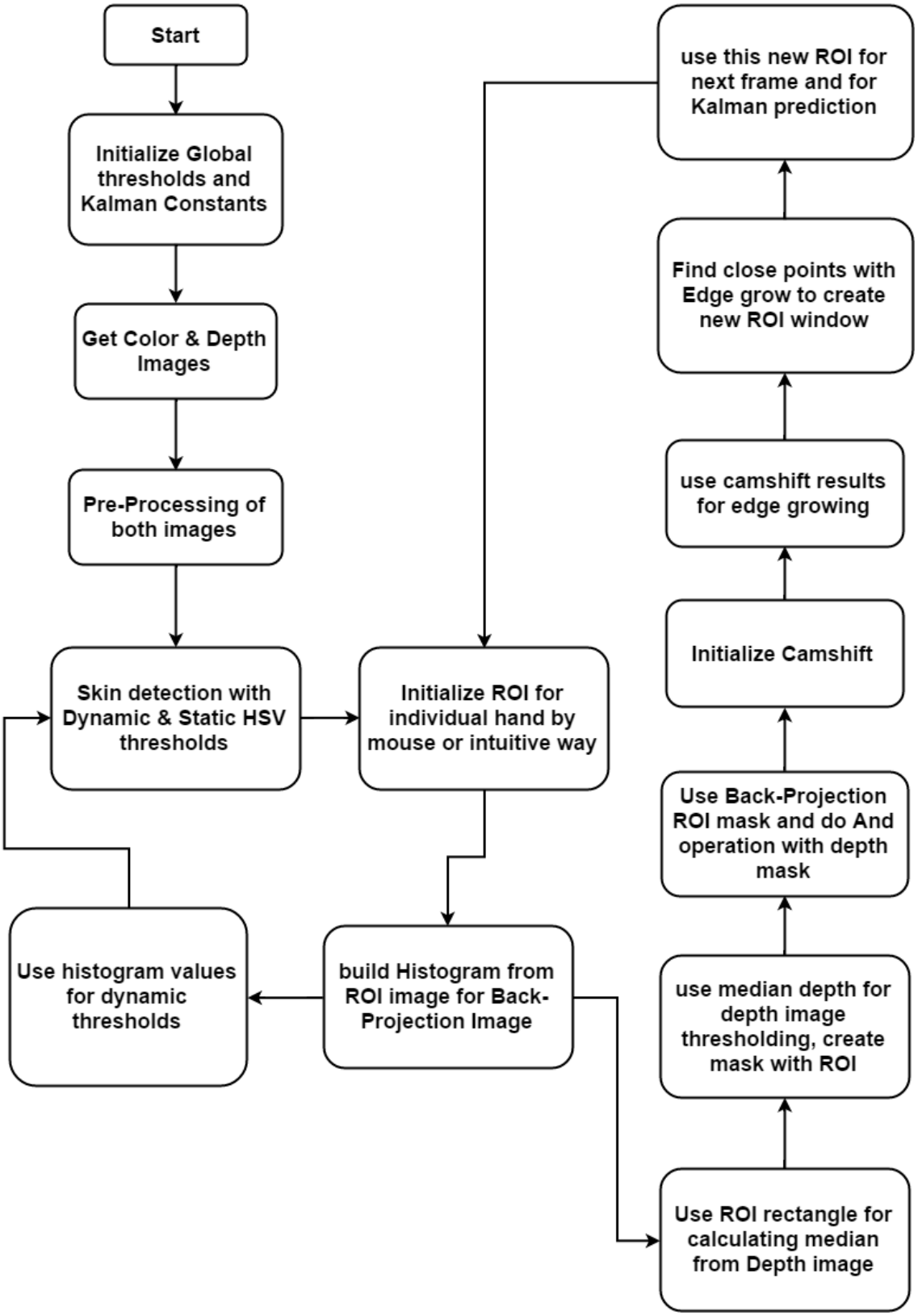


Figure 2.26: A Complete block diagram of Hand Tracking Algorithm

2.12.1 Summery

Finally at the end of this chapter, we need to summarize all the steps above for making it more understandable for common readers. One can refer Figure 2.26 for the same purpose.

- Initialize global variables such as Hue and YCrCb color spaces static thresholds and kalman coefficients and others.
- Retrieve color and depth images and do preprocessing where; scale down, depth background segmentation, depth image filtering and creation of depth edge image are done.
- Use static thresholds for both color models and find out primary skin mask. Use dynamic thresholds once hand is initialized to segment more accurately.
- Initialize ROI or tack-window for both hands.
- Find histogram from ROI of an image and use it for calculating back-projection image as well as dynamic thresholds for Hue range.
- Use ROI with depth image and find out median of depth points in that region which are related to skin pixels. Threshold depth image based on newly found median depth, create depth-mask by ignoring other objects at the same depth with help of ROI.
- Do And operation with back-projection and depth masks, for reducing hand face occlusion problem.
- Apply this final mask to camshift algorithm for calculating new track-window.
- Since camshift will compute entire arm if provided in the mask, we need to rescale this track-window manually.
- Use the center of this ROI and find two adjacent points on x or y axis and assign them name XL,XH or YL,YH.
- These set of points will be start and stop point of our edge grow algorithm.
- From depth edge image, search for neighborhood edge pixels and grow edge until it reaches to stop point.
- Collect all those points and remove unwanted points.
- From close set of points, find Top-point "TP", low-point "LP" and high-point "HP" in respective direction.
- Based on these final three points calculate new and improved track-window which will be used for kalman measurement and in next frame as ROI.
- Use kalman results when camshift looses it's tracking for some frames.

2.12.2 Limitation of this approach

As always the case with approaches, there will be certain limitation for this approach as well. We tried our best ot hide or improve camshift limitations but sometimes in real world one can not anticipate the conditions and situations which results in to tracking loss. Such as..,

- This approach heavily depends upon skin mask for canshift prediction, thus if the illumination conditions are not good, it won't work smoothly.
- Hand movement has to be limited at certain speed, otherwise it will result the ROI of next frame to be lost.
- We need to get rid of YCrCb color space for introducing red sensitivity in skin segmentation. Thus, again ambiguity between better vs accurate skin mask.
- Input frame rate has to be 30fps or more.
- With kinect 2, the depth image flickering problem is far more worse than it is with kinect 1. Hence there is scope of improvement in kinect 2 depth image filtering.
- Since final track-window size depends upon depth edge image, results with kinect 2 are not so good as compare to kinect 1.

2.12.3 Scope Of Improvement

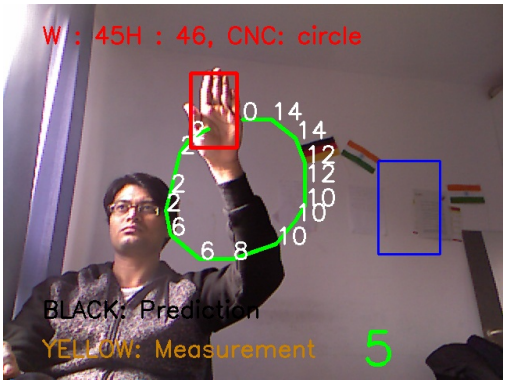
- Don't consider ROI but contour inside ROI for each image.
- After sometime do depth tracking only. and if you feel deoth is lost based on size of your contour reinitialize or reintroduce skin pixels again!
- Kinect 2 depth image filtering. With time of flight camera limitation.
- Assigning labels to hands more smartly one camshift limitation is, if both hands are together it is hard to seperate both trackers...
- Kalman 3D tracking instead of 2D.
- There can be the third camshift for face tracking wihch will indicate face size and depth level, thus we can be pre-aleart when user proceds towards face.
- Along with this algorithm, there can be one more feature classifier can be developed for hand contours understanding, which will classify hand from other false skin pixels as well as face contour.

Chapter 3

Gesture/Trajectory Understanding

Understanding gesture is an important task with HMI(human machine interface) and HCI(human computer interface). We humans and animals use gesture as way of expressing our feeling and in some case communication. Humans use hands as tools to interact with natural world. As we know human hand is quite smaller compare to entire body and with most degrees of freedom(27 DOF)[TODO:ref27]. Therefore modeling of human hand will be quite challenging task, although most real world applications does not require that much degrees of freedom, for example in our case Pi4 robot has 6 DOF arm hence complex hand modeling is not necessary in that regard. The gesture ideally can be of three types; static, dynamic or both. Static gestures will not require movement of hand but rather complex combination of certain poses/signs indicating different codes or meanings as shown in Figures 3.1a. Whereas, others include movements of arms up-to some level to form a trajectory 3.1b. In our case, human gesture must be understood by Pi4 robot in MeRoSy project, who is supposed to work side-by-side with humans on assembly lines. Therefore, in this section we will show primary movements of hand trajectory to decode predefined simple gestures.

(a) Static Hand Gestures



(b) Dynamic Hand Gesture/Trajectory

Figure 3.1: Types Of Gestures

3.1 Background And Related Works

Several state of the art papers have described interesting theories for understanding hand gestures both static and dynamic, in this section we will take a close look at some of them.

In machine learning theory, classification problem is divided in to three types of learning modalities; supervised, unsupervised and reward based. It is said that for supervised learning, a set of training data needs to be provided along with labeling of individual data represents different class. Here, it is expected to find correlation between data to class based on set of training examples provided by user. The resulted output might be analytical function of a model of the system. In this method the accuracy depends heavily upon the training data and successful labeling of those data.

The second type is unsupervised learning, where classifier is provided with bulk of data without any priory knowledge and it is expected to cluster the data to some group. According to wikipedia, "it is the method for finding hidden structure from unlabeled data". For example, in the Figure 3.2, we can see a basket of fruits where each fruit has it's own color, shape and size. Now an algorithm needs to gather as much information as possible for each type of object form provided image or set of images. After that it needs to gather data in to undefined classes based on similarity it found between them.

Third type of learning is a new one and is based on rewards, such as reinforcement learning. It is also called learning through demonstration, as it can be clear that there will be a training period and during that time positive and negative rewards will be assigned to certain actions which will be later used to relate modeling of an environment. The benefit of this kind of learning technique is that learning can be performed during on-line, that means during state transition a set of rules will decide positive and negative behaviors and can be updated almost immediately. This will be helpful in exploration of new dynamic environment where conditions can change dynamically and algorithm needs to adapt simultaneously. This approach has been studied in control, gaming, information and artificial intelligence theories. Figure 3.3, helps to understand basic difference between all three main methods.

Figure 3.2: A basket of fruits

Hierarchical matching pursuit along with sparse coding is one of most interesting research mentioned by Paper:ref: 30. They use language models along with gesture

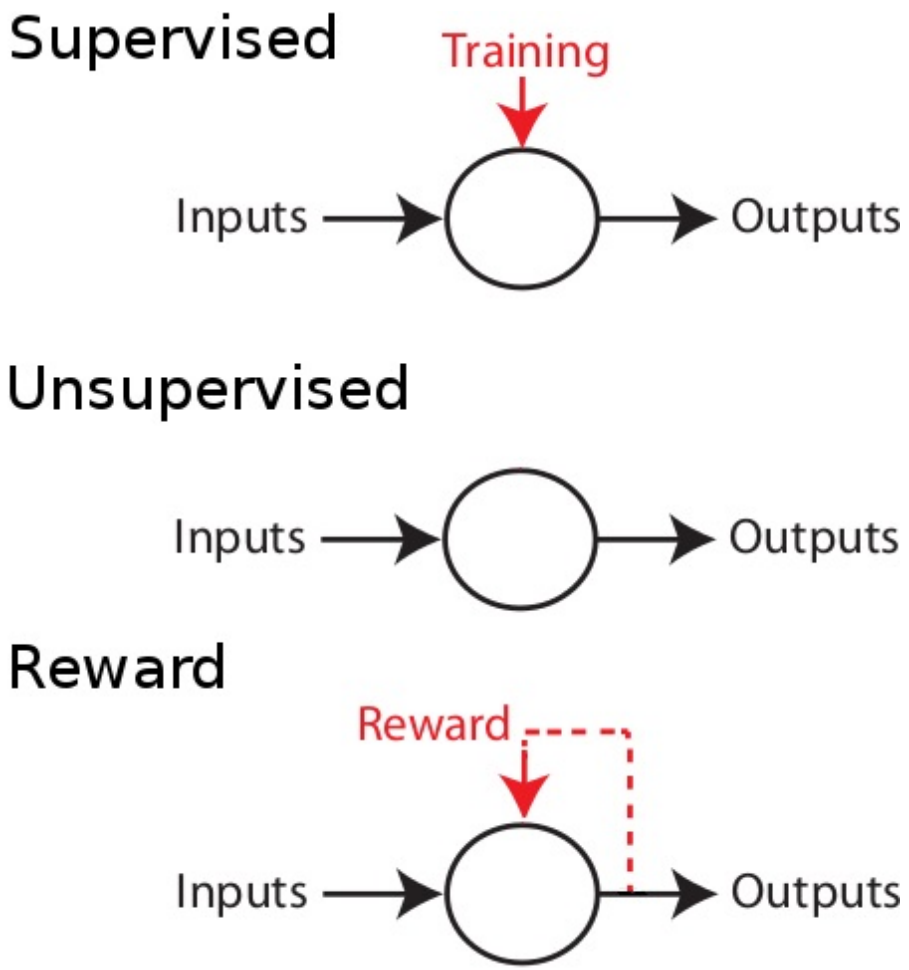


Figure 3.3: Learning

models to describe an object. Robot learns through demonstration that the object with specific color and size is pointed with finger by user to grasp as shown in Figure 3.4. Sparse coding helps to find an input signal from a rich set of data without labeling, thus higher level features needs to be discovered from a set of data having no need for modeling the feature set manually. A dictionary learning algorithm K-SVD helps to model data which is a sparse linear combination of codewords. Codewords from codebook: which is a set of feature vectors needs to be learned in this task.

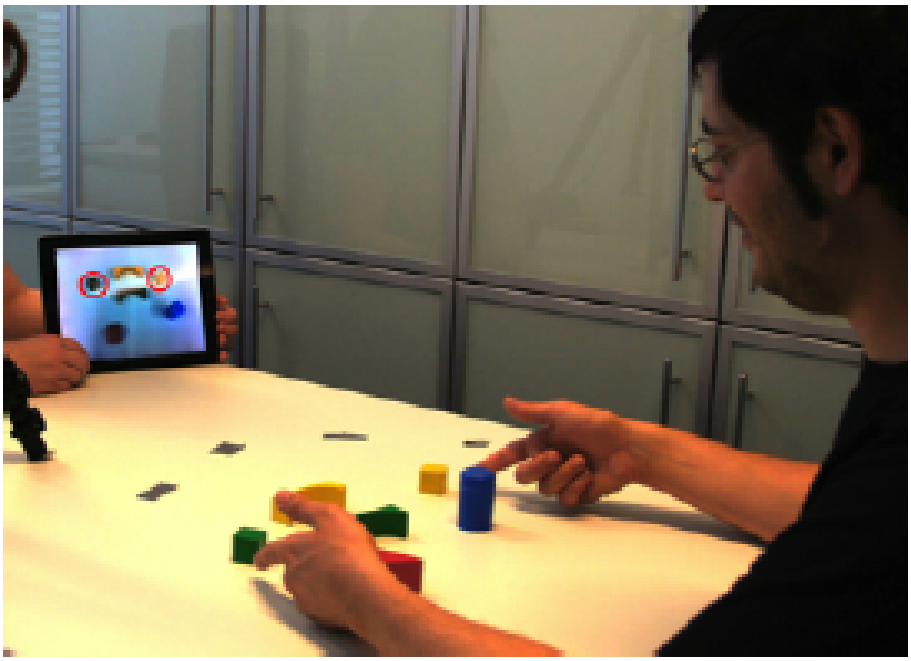


Figure 3.4: Object is being indicated by user for robot to grasp

"To handle noisy input data from kinect", Z.Ren, J. Yuan suggests a distance metric; Finger-Earth Mover's Distance(FRMD):Ref:31 to measure dissimilarity between different hand shapes. This algorithm matches only the fingers not entire hand thus can better distinguish hand gestures with slight fingers movement differences.

Dynamic as well as static gestures are recognized by D. Michel,K.Paputsakis and A. Argyros:Ref32 by creating dataset of 5 different gestures. They use intuitive set of rules predefined for each individual gestures to be classified from a clustered environment.

A substantial amount of work has been done by M. Elmezain,A. Al-Hamadi etal in ref:33,34,35 and by X. wang,M. Xia etal in ref: 36 with Hidden Markov Model based dynamic trajectory understanding. It is quite interesting for trajectory understanding and movement analysis. We got our primary idea for trajectory data processing from all those papers. It is easy to represent a set of points to a sequence of codes which will be primary operations done on different trajectories.

3.2 Approach for Gesture Understanding

In this section, we will explore our idea of trajectory understanding from a set of points. This trajectories are simple and does not require lots of data to process. It is the attempt on primary work for exploring possibility to decode and understand trajectories. From hand tracking section, we get 3D points in world coordinate system, which is collected in this task and the entire hand movement will be recorded for further processing.

The motivation behind this approach was initially drawn from paperTOOD:??, where writers use HMM (*Hidden Markov Model*) to predict shape of the trajectory. Although HMM works well, it has some prime limitation such as; Viterbi algorithm used for decoding

gestures is quite expensive both in terms of computation time and memory. For sequence of data n, best path for model with s states and e edges requires sn amount of memory and en amount of time[TODO:Ref39]. Therefore we take help from SVM classifier to identify our trajectory. SVM was selected because of it's simplicity, predictability and fast speed, additionally opencv has great interface for implementing classifier.

Once a complete set of 3D data recorded as single trajectory, it will be analyzed in different ways. Data collection and Processing sections explain how individual points are scanned and utilized, then appropriate angles between subsequent points are found. Later those angles are converted to codes for better representation. Thus from a set of points, we will have sequence of codes which will be the feature vector to our SVM for both training as well as testing. The entire process is explained in detail in summery section 3.7 with a complete block diagram.

Trajectory can have 3 different attributes, the first is to consider position of individual points in all directions. However the simple positions can not be considered for understanding task because of the variability of size, shape and positions itself in a 3D world. Second, trajectory will have velocity by which the gesture is performed, it will have different set of points at each different velocity. The third and most reliable solution for decoding trajectory is to find an angle between subsequent points. An angle will be independent of scale, size and velocity of the gesture and will be perfect contender to represent the set of points as sequence of angles.

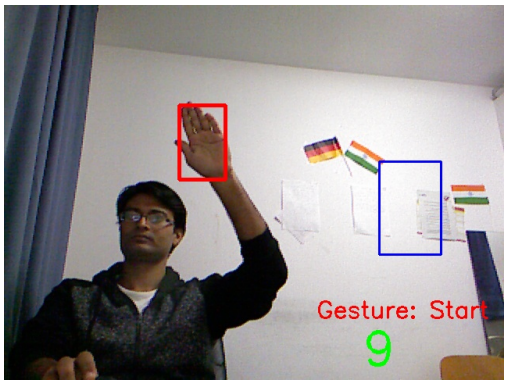
3.3 Data collection

As in every important applications, data collection is one of the main tasks needs to be done neatly and efficiently. Here, data will be the collection of 3D points for hand motion trajectory. However, we will only consider 2D(X,Y) points for simplicity.

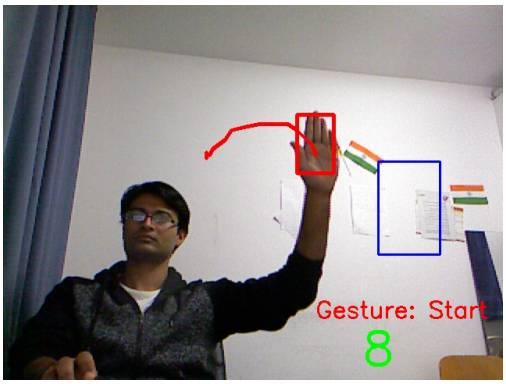
Here, at first we need to figure it out points from where data collection needs to be started & stopped. It would be great if we can somehow indicate to our data collection method to start collecting from certain time/position and to stop at different time/position. In the next subsection for intuitive start and stop, we discuss exactly that problem.

3.3.1 Intuitive Start and Stop

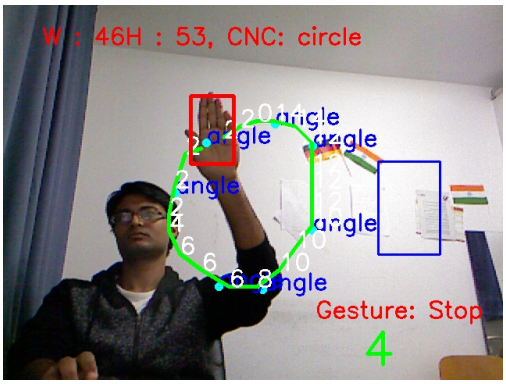
It is important for user to have intuitive interface when it comes to interacting with live environment. It would be much easier to inform user in advance to behave certain way so that the algorithm can adjust itself and thus user do not need to set parameters manually. This is one of those interface where user needs to know that from when the data collection or trajectory tracking is started and when and where it is stopped.



(a) Gesture->Start



(b) Gesture->Intermediate Points are being recorded



(c) Gesture->Stop

Figure 3.5: Start & Stop of Gesture

The interface is really simple to understand and implement. For initialization or start, user needs to keep it's hand steady at fixed position for sometime. In the background, every point is analyzed regarding it's position. Hence if we get some points which are found at the same place for sometime, it would be start condition for gesture to begin. It is shown in the Figure3.5, where hand is steady for some frames and then the gesture tracking starts. The process is explained as below:

- Once hand has been initialized, hand tracking algorithm will return 3D points.
- Store every 30 points and find out if the start and stop point are still in the range along with others, if so then raise the flag for gesture to start.
- Once flag has been raised for start process, record all consecutive points and inform user with text on screen and draw trajectory with different(red) color;Figures 3.5a & 3.5b.
- Again store every 30 points while trajectory is being recorded, and check continuously if the hand has stopped moving.
- Once that condition realizes raise other flag for gesture-stop and reset flag for gesture-start making way for new gesture.

- In the same frame, process data first and then go to gesture recognizer method and check if recently stored trajectory data matches any predefined gesture or not?
- If it does, print out the information on screen for user to visualize; Figure3.5c.

3.3.2 Data Preprocessing, flickering noise elimination

In this section, we will discuss how individual points are selected for creating a sequence of codes for SVM to train or test. Once, trajectory recording is finished, we will have entire set of points. The problem with this points is, they are affected sometimes with flickering noise of depth image. Since we are rescaling our tracking-window size based on depth edge image as referred in section of Edge Grow 2.10, the tracking window will have flickering effect as well. Thus if we consider points very close to each other, they will have misleading information about the orientation.

Therefore, we choose every pixels that are at the safe distance from each other in either X or Y direction. To do so, we simply check from start point till end of collection and select those points that are at least at 3 mm distance of each other in scaled down image(160x120). This will help us to achieve much reliable trajectory and get rid of false orientation.

3.3.3 Scaled Down Trajectory

This section describes one of interesting operation on set of points collected from previous operations. Here, we do scaling of individual points for better results. The argument for this operation is that, if we can select a standard size for all trajectories and then can successfully scale down original size of trajectory to that specific size, it would reduce the amount of training data needed for specific gesture to train. Such as, if we consider the original trajectory size then we will have n number of shapes ranging from smaller size to the bigger one and we need to train our classifier for all those sizes. Thus if we perform, some scale-down operation and rearrange those set of points to smaller set of points, it would give us standard size for individual gesture and hence the sequence of codes must be limited to certain level.

We have trained our classifier with and without this operation. In one case where we have considered original shape without any scaling-down operation and then calculating angles between points and creating a sequence of codes and use that for training as well as testing phases. From many experiments, we concluded that in traditional case without scaling-down, we need more training data to train SVM classifier. Although with sufficient data, the ambiguity for SVM to separate two different gestures will be reduced.

Whereas, scaling down of trajectory presents advantages like, the training data will be reduced. Additionally, if during faster movement of hands some points are missed by hand tracking algorithm, and missing data will not be glorified and gesture shape will have minor distortion as compared to original size. However this advantage becomes disadvantage when it comes to distinguish between two gestures, such as Digit-5 vs letter-S and Digit-4 vs Digit-9, these limitations will be explained at the end of this chapter

3.8.

As we can see from Figure 3.6, original trajectory is represented as green points and standard scaled down trajectory is shown in red points. We can confirm from figure that the shape is pretty much preserved for simple gesture. Here the large amount of original data will be forced to fit into smaller section of scaled-down trajectory and based on our 3 mm distance criteria, most of them will be removed and only the one which has significance difference in position will be accounted for further angle detection between them.

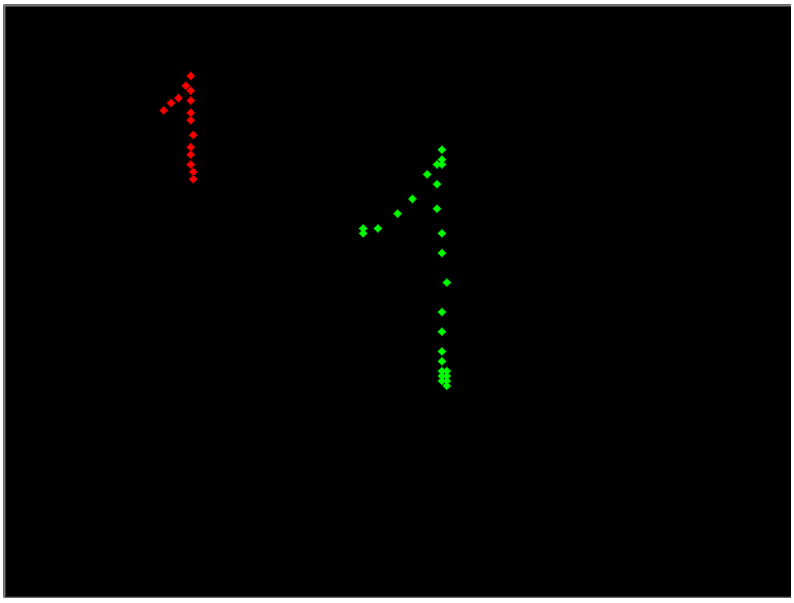


Figure 3.6: Scaled Down Trajectory

3.3.4 Finding Angles and Creating Sequence of Codes

As we can see from the Figure 3.7a, orientation between two points will be calculated with respect to +ve x axis. An angle will be useful information as explained earlier by being independent of shape, size and velocity of trajectory. There can be four cases where point $P_2(x_{t+1}, y_{t+1})$ with respect to $P_1(x_t, y_t)$ is placed in four different quarters. As shown in Figure 3.7c, we need simple geometry to solve this puzzle. Orientation can be found from

Equation 3.1 below;

$$\theta_t = \begin{cases} \theta_c, & 1^{\text{st}} \text{ Quater} \\ 180 - \theta_c, & 2^{\text{nd}} \text{ Quater} \\ 180 + \theta_c, & 3^{\text{rd}} \text{ Quater} \\ 360 - \theta_c, & 4^{\text{th}} \text{ Quater} \end{cases} \tag{3.1}$$

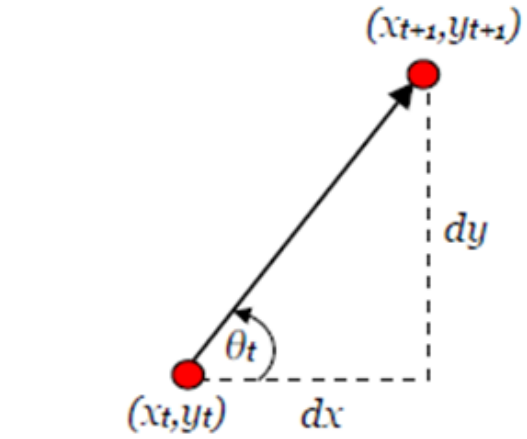
$$where;$$

$$\theta_c = \frac{\arccos(\frac{dx}{P_1P_2}) * 180}{\pi};$$

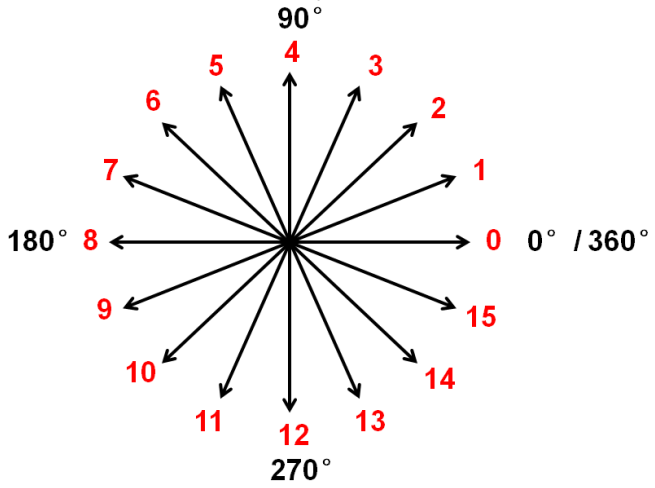
$$P_1P_2 = \sqrt{(P_1.x - P_2.x)^2 + (P_1.y - P_2.y)^2}$$

Here, P_1P_2 is the distance form point P_1 to P_2 and dx is difference in x direction. With this equation, we can compute angle from 0° to 360°.

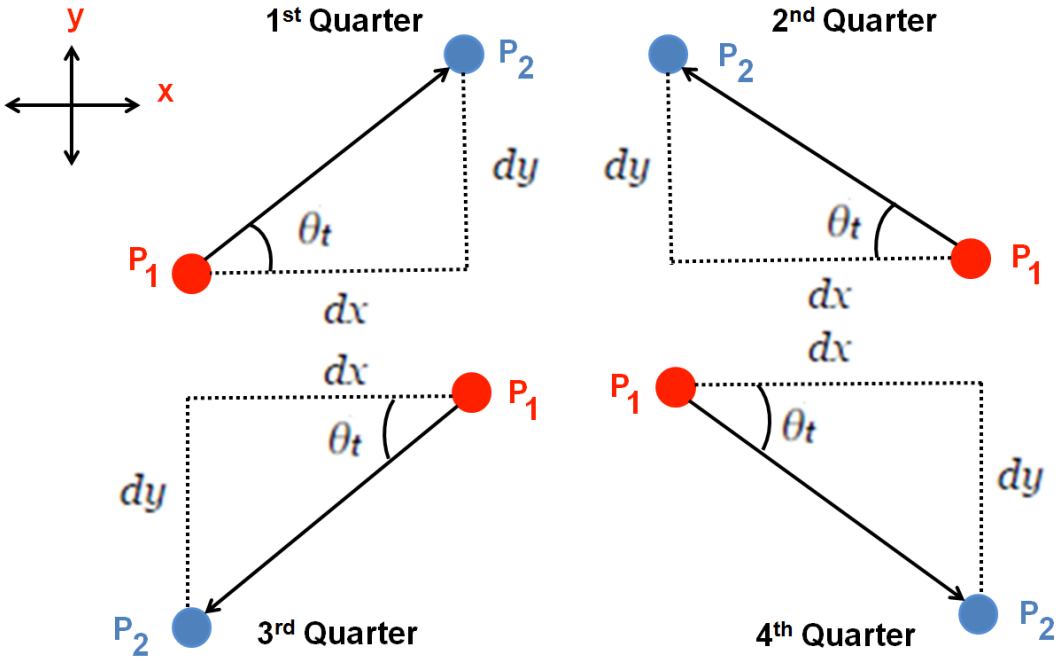
Now for simplicity of further calculation, ww divide rotational space into 16 different sectors and assign them label from 0 to 15 as shown in Figure 3.7b. They are called *codes*, and by this way we create a *sequence of codes* for our trajectory. Now this *sequence of codes* is ready to be utilized as training or testing data for our classifier.



(a) Trajectory Orientation between two points



(b) Codes



(c) Different Orientation Cases

Figure 3.7: Orientation & Codes

3.4 Support Vector Machine Theory

In machine learning field, support vector machine is one of the popular theory for classification and regression problems. This technique was originally invented by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963.TODO:Ref:Wikipedia. It can be utilized as linear and non-linear classifier model with different kernel-tricks. Linear kernel was used

for simplicity in this case. Different set of training data based on requirements can be provided for classification purposes.

A classification problem usually requires training set of data, where each individual data is labeled as representation of class. The duty for SVM classifier is to calculate optimal distance between each individual class. If there is binary classifier, then only two main classes to predict, SVM will calculate an optimal line separating two classes. And a plane for 3D data, for higher dimensional data, SVM will form a hyperplane.

Functional margin which is the largest distance from nearest training data point needs to be calculated optimally, since it will be final factor for SVM classification error. It can be understood that the bigger the margin, the lesser the classification error.

Lets consider simple example of binary classification with 2D data. Point (X_i, Y_i) is the training data point where i=1...n. A set of all these points is D, and it is stated that: Ref:wikipedia

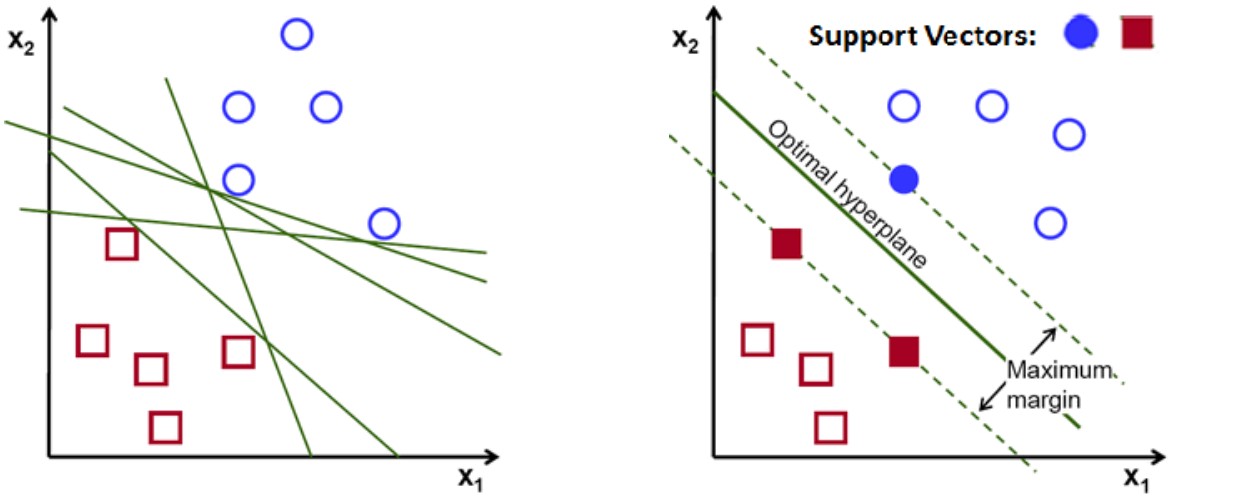
$$D = \left\{ (X_i, Y_i) | X_i \in \mathbb{R}^P; Y_i \in \{-1, 1\} \right\}_{i=1}^{n}$$

Here; Y_i is the class label for each training data X_i, which is either -1 or 1. X_i is real vector with p dimensions in this case p = 2. Now the goal is to find a optimal margin bwetween two classes represented as a line which separates points of class 1 from class -1.

In general, a hyperplane can be defined as a function of X as;

$$f(X) = \beta_0 + \beta^T X \tag{3.2}$$

In Equation 3.2, β represents weight vector which is also normal to the hyperplane, β_0 as bias



(a) Set of Points (b) Line as optimal hyperplane found between points

Figure 3.8: SVM set of points

There can be an infinite possible combinations for hyperplanes defined for this case as shown in Figure 3.8a by changing values of β_0 and β. Equation 3.3 represents an optimal

value for hyperplane;

$$1 = |\beta_0 + \beta^T X| \tag{3.3}$$

Here, X represents the closest training data point to the hyperplane, it is also called support vector. Distance between X and optimal hyperplane will be represented by,

$$distance = \frac{|\beta_0 + \beta^T X|}{||\beta||} = \frac{1}{||\beta||} \tag{3.4}$$

From Equation 3.4, we can derive that the maximum margin between both classes support vectors would be;

$$\text{Max Margin}, M = \frac{2}{||\beta||} \tag{3.5}$$

With these equations 3.4 & 3.5, we will get to know an optimized separation between two classes of data. When new test data needs to be classified as either side of the line, it will be easy and fast to predict where it belongs.

This relatively simple example can help us understand complex cases where data can be quite large. In our example as we have discussed in the section for data collection 3.3, we will have a sequence of codes which will be our data for particular gesture. We will have mainly two classes either the given data is the specified gesture class or it's not. Hence it is safe to say that, we are working with multidimensional binary classifier. For each individual gesture, a separate classifier needs to train and test. This is called multiple binary classifications, there exists multi class SVM classifiers but they are not consider in this case for simplicity.

In the next sections, we will consider training and testing phases for individual gestures. Here, we will compare mainly two different ways of data modeling: with and without scaling-down of trajectory.

3.5 Training Phase

As mentioned earlier, there are two ways to collect training data, first is to keep original trajectory size and find *sequence of codes* and the second is the scale down original trajectory and do the same. Here in this section, we are going to discuss both approaches and number of training data set required for each gesture.

	Original		Scaled Down	
Gesture	+ve data	-ve data	+ve data	-ve data
Digit_0/Circle	153	45	29	66
Digit_1	80	69	14	69
Digit_2	66	54	10	51
Digit_3	74	51	20	51
Digit_4	69	74	10	74
Digit_5	66	52	9	52
Digit_6	52	93	35	32
Digit_7	66	52	66	52
Digit_8	41	45	9	20
Digit_9	47	74	13	10
Letter_S	100	21	16	21

Table 3.1: All gestures training data set with original and scaled down approach

As we can observe form table 3.1, the positive data set reduced when it comes to rescaling(scaled-down) approach compared to original trajectory size approach. This can be possible because of standard size specifications, when we rescale our original trajectory, it will get reduced to certain points. If we try to compute sequence of codes from those reduced list of points, it will make smaller set of codes compare to original size of trajectory where number of codes could reach up-to 30 to 35, which can be limited between 15 to 20. It is needless to say that, we add an extra computing when we do rescaling and it can be helpful for some cases. Training data is saved in an array of integers and this array has maximum size equals to 50, that means no matter which method we adopt, the total number of codes in one sequence would be limited to 50. However it is completely independent for individual gesture to have any number of codes inside 50.

3.6 Testing Phase

In this section, we will discuss the results of testing data and preconditions placed for better results. The need for preconditions before SVM prediction is due to the limitation of SVM where some times, it can not distinguish between complete gesture or half of the gesture. For example if we draw Digit_3, it would contain upper part of the digit and even if the lower part is missing but test data matches up-to more than 50% of calculated SVM hyperplane limits, it would be announced as Digit_3, where as in reality it was only the part of gesture. Here, we can not increase negative data for upper half part of digit because it will reduce the prediction accuracy.

3.6.1 Gesture: Circle, Digit_0 or Ellipse

As we can see from the Figure 3.9, there is very fine difference between circle and digit_0. Actually SVM only decides whether it is round gesture or not? Based on gesture width and height ratio, we can decide whether it is circle or ellipse/Digit_0? In addition to this, we also have one more condition called closeness criteria, it is calculated according to Equation 3.6, and represents if the start point and stop point are close enough to finish the round trajectory? This will be utilized for all shapes like triangle, Square/Rectangle, Pentagon, Hexagon and Digit_8.

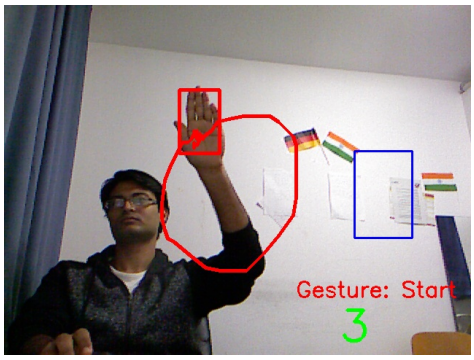
Distance between start & stop points,

$$D = \sqrt{(startPoint.x - stopPoint.x)^2 + (startPoint.y - stopPoint.y)^2} \tag{3.6}$$

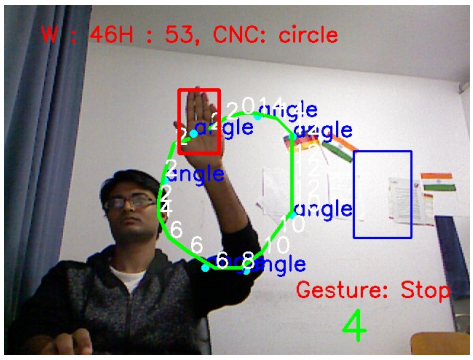
Finally, steps needs to be performed for Circle, Digit_0 gesture detection:

- Find start and stop points of the trajectory.
- Calculate distance between them as shown in Equation 3.6.
- If distance is close enough, ask SVM whether it is round shpae or not?
- Once SVM outputs positive result, calculate ratio of trajectory height and width.
- If ratio is 1.4, assign gesture name as Digit_0 or Ellipse, other wise Circle.

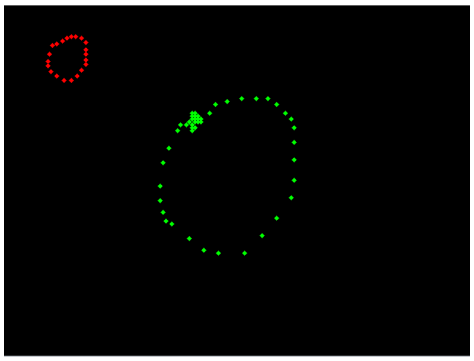
As we can see from the Figure 3.9b & 3.9e, the sequence of codes is represented as white numbers on image. Ideally for circle the sequence must contain all the 16 codes. No matter from where circle starts, it must end close to where it has started and this should cover entire 360° rotational space, however in real cases it is not often the case and some codes can not be covered but the training data must include all those cases and it should predict correctly. The main limitation with this sequence of codes along with SVM appears when user tries to draw a square or other close shape, since any close shape will include all the codes which are similar to circle, close shape detection with SVM is not a good idea. We realized this limitation and found other solution to this problem. As we can see from Figures 3.9b & 3.9e, blue text called *angle* is also drawn on images. This is the result of finding convex hull around the close shape, which is explained in detail in section 3.9.



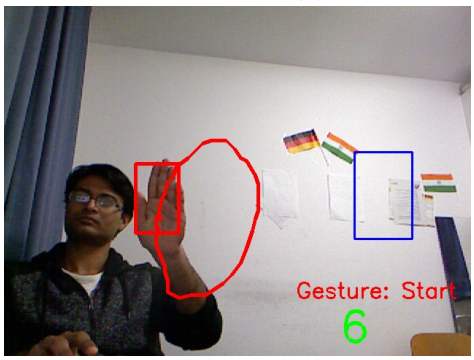
(a) Circle Start



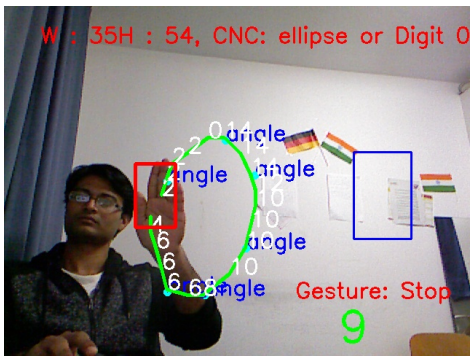
(b) Circle Stop



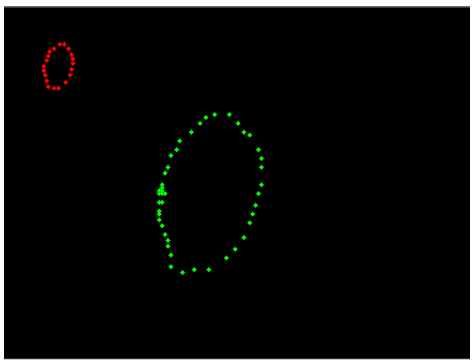
(c) Circle Scaled Down Trajectory



(d) Digit_0 Start



(e) Digit_0 Stop

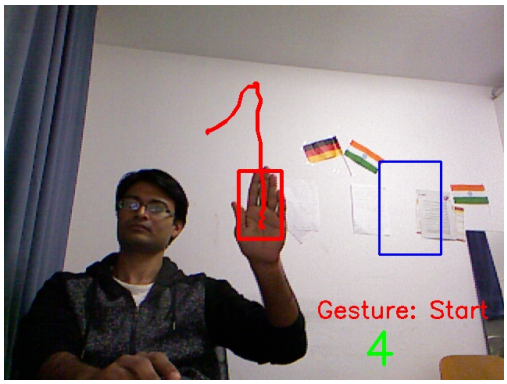


(f) Digit_0 Scaled Down Trajectory

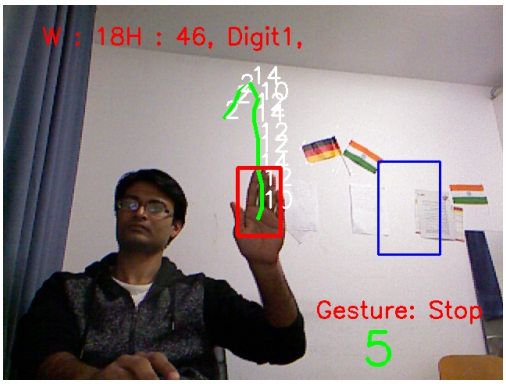
Figure 3.9: Gestures-> Circle, Digit_0

3.6.2 Gesture: Digit_1

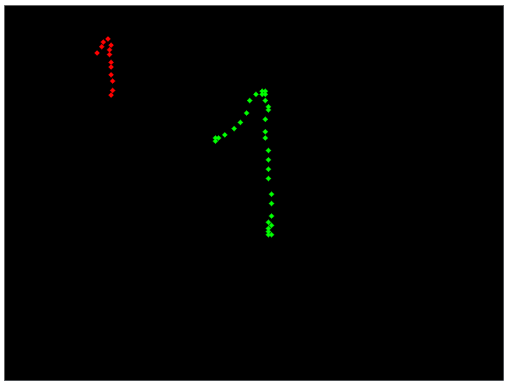
Digit_1 is one of the simple gesture, it includes mostly straight line and does not require more training data when it comes to scaled down case. The only problem is that it is quite similar to Digit_7 for SVM classification, an SVM is not a good indicator between straight line and slightly tilted line which is with Digit_7. In order to reduce false prediction, we have included some additional preconditions to differentiate both shapes, as it we can see from Figure 3.10d, normal codes are shown in red color, the start codes are shown in purple and stop codes are shown in green color. This means that the sequence of codes needs to start from either of 1,2,3 codes and must stop at 11,12,13 codes. If this precondition is satisfied then only we get to ask SVM it's prediction. This precondition will not be dependent on size or position of the point in 2D space and thus are considered as part of classification process.



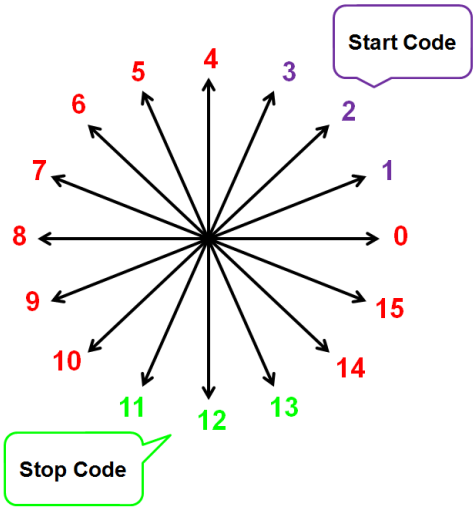
(a) Digit_1 Start



(b) Digit_1 Stop



(c) Digit_1 Scaled Down Trajectory

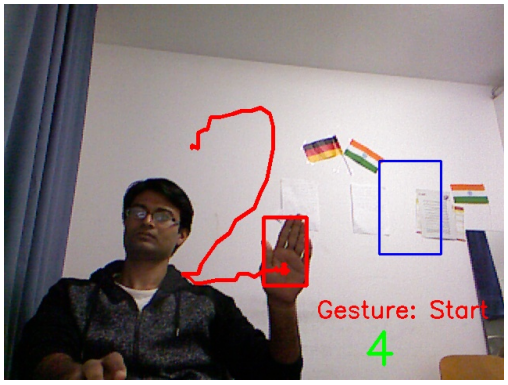


(d) Digit_1 Precondition

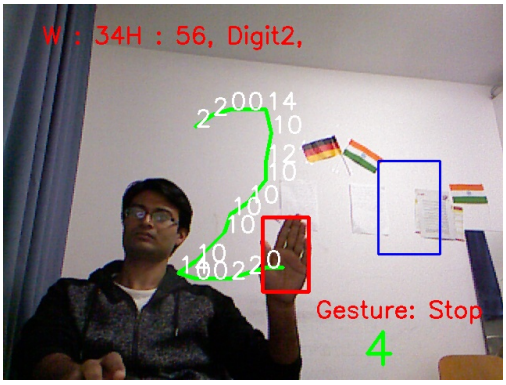
Figure 3.10: Gestures-> Digit_0 & Digit_1

3.6.3 Gesture: Digit_2

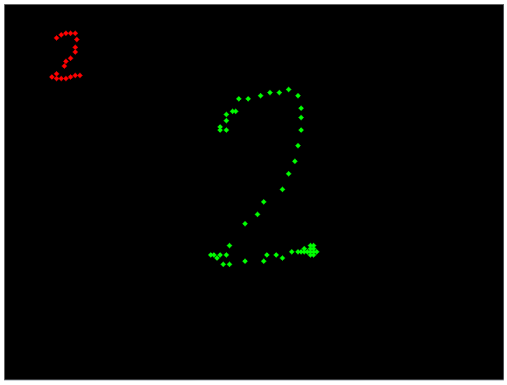
Second gesture in this catagory is Digit_2, this is also simple trajectory with most of the part is some what similar to Digit_3's upper part. This trajectory also needs to have start and stop conditions, and as shown in Figure 3.11d, start codes must be either of 0,1,2 and stop codes are also 0,1,2 and additional 15(green). If start and stop codes are same the they are indicated as blue digit in mentioned figure. As we can also see from the table 3.1, we need only 10 position data for scaled down model however with original size trajectory approach, it would be 66! The negative set of data are almost the same.



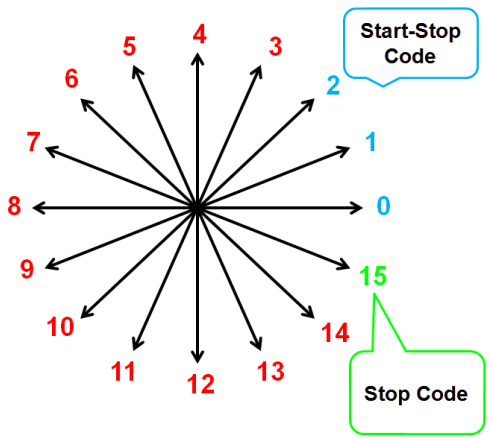
(a) Digit_2 Start



(b) Digit_2 Stop



(c) Digit_2 Scaled Down Trajectory

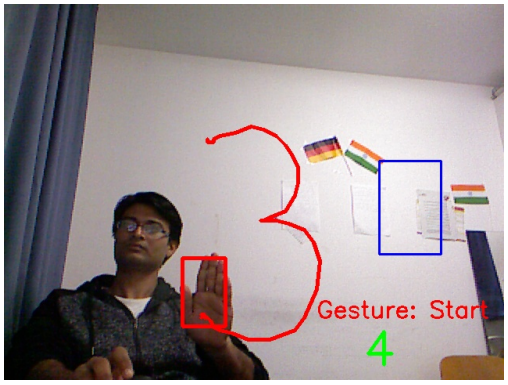


(d) Digit_2 Precondition

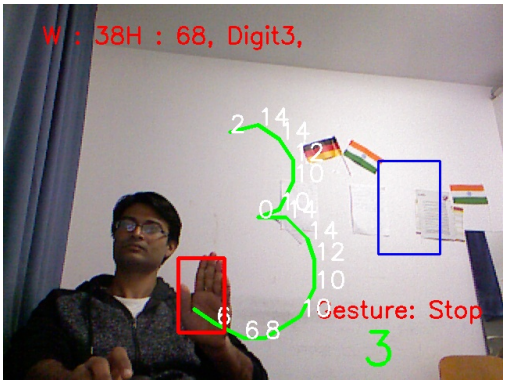
Figure 3.11: Gestures-> Digit_2

3.6.4 Gesture: Digit_3

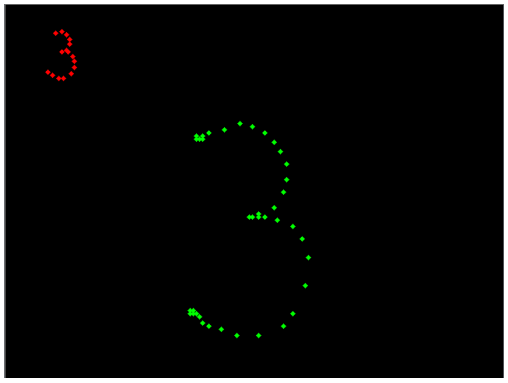
Digit_3 has most number of start and stop codes as shown in Figure 3.12d and it is rightly so. Since number 3 has very wide range of motion as uncertainty around it's stopping point, we need to consider almost entire left half of that circle as stopping point criteria. However, this digit needs a lots of data for original trajectory size to train, since it is quite dynamic when user wants to draw a trajectory shaped as Digit_3. The start codes are mentioned in purple: 0,1,2 & 15 and stop codes are in green: 5, 6, 7, 8, 9 & 10. It requires 20 positive data for scaled down and 74 for original size trajectory.



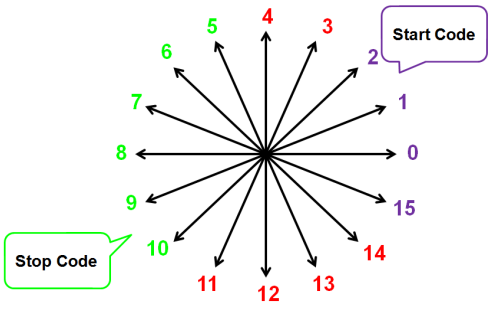
(a) Digit_3 Start



(b) Digit_3 Stop



(c) Digit_3 Scaled Down Trajectory

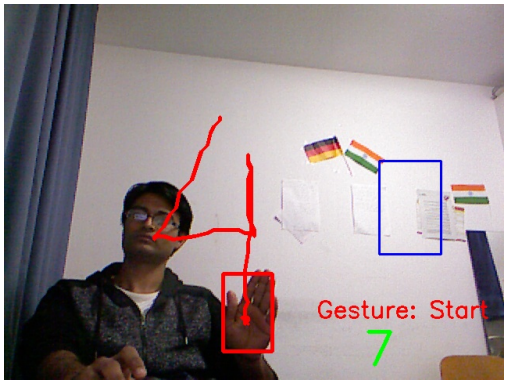


(d) Digit_3 Precondition

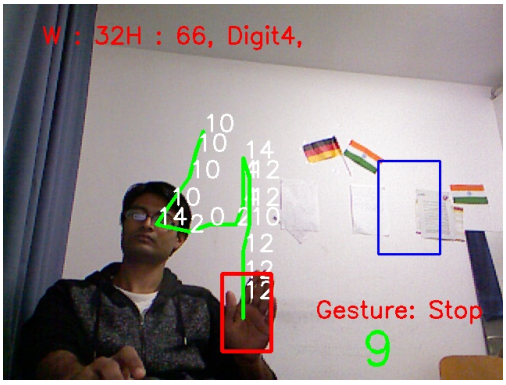
Figure 3.12: Gestures-> Digit_3

3.6.5 Gesture: Digit_4

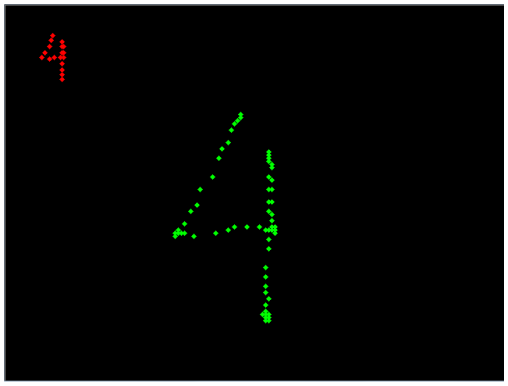
This digit is simple in looking but it is not easy when it is scaled down. The problem often comes because of sequence of codes matches Digit_9 almost and thus creating lots of preconditions to separate both digits. Figure 3.13d only shows the start and stop conditions where start codes are: 10, 11 & 12 and stop codes: 11, 12 & 13. Additionally one more condition is, the start point must be at specific distance from stop point in x direction. This condition will separate Digit_4 from 9 because 9 must have start and stop point almost on straight line.



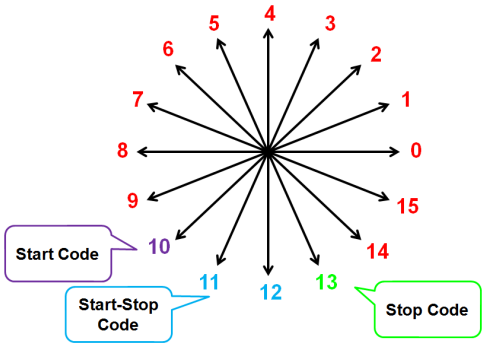
(a) Digit_4 Start



(b) Digit_4 Stop



(c) Digit_4 Scaled Down Trajectory

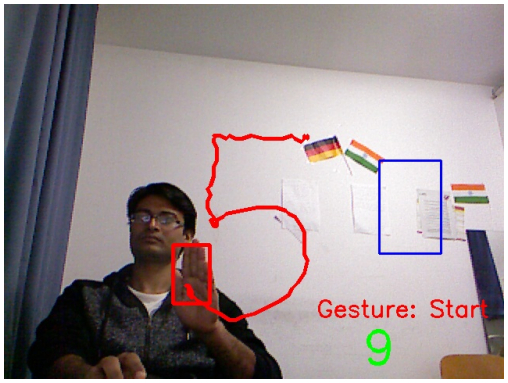


(d) Digit_4 Precondition

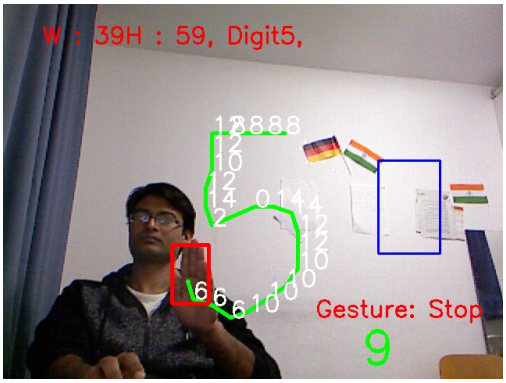
Figure 3.13: Gestures-> Digit_4

3.6.6 Gesture: Digit_5

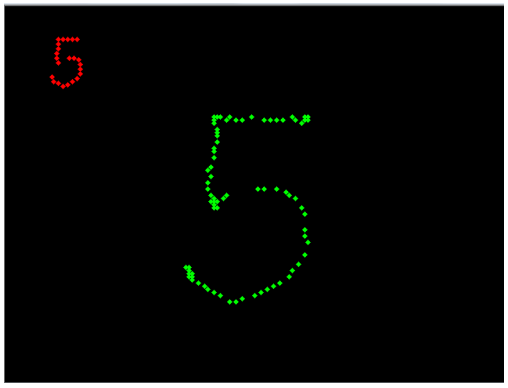
Just like Digit_4, Digit_5 also has it's so called twin and it is letter_S. There has to be some way to differentiate both trajectories from each others and here comes additional conditional again. For Digit_5 these conditions are, the trajectory must include first straight line going towards left and then second going down. Start codes are 7, 8 & 9 and stop codes: 7, 8, 9 & 10. For scaled down case it requires only 9 positive data to identify the gesture.



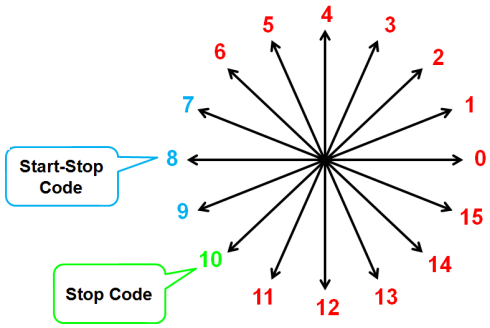
(a) Digit_5 Start



(b) Digit_5 Stop



(c) Digit_5 Scaled Down Trajectory

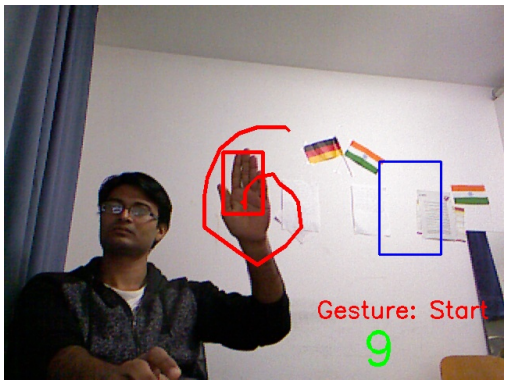


(d) Digit_5 Precondition

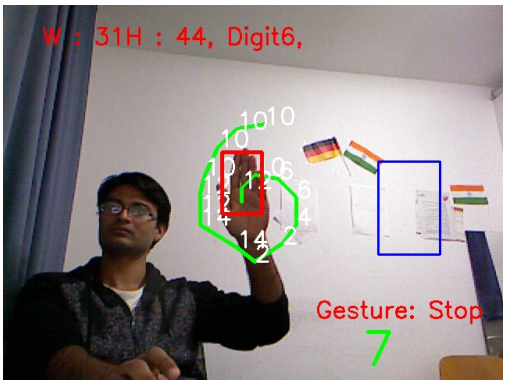
Figure 3.14: Gestures-> Digit_5

3.6.7 Gesture: Digit_6

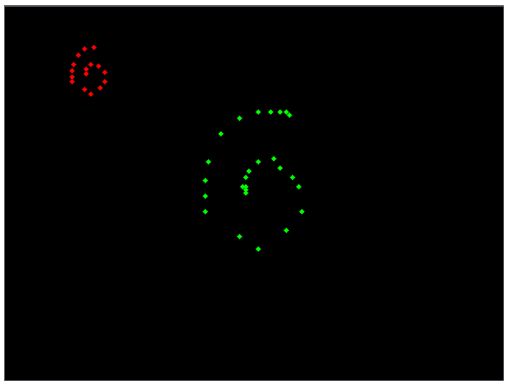
This gesture will have almost every codes in rotational space since it also looks like a round shape if looks carefully but the luckily it does not satisfy closeness criteria 3.6 we found earlier in Circle's case. The start codes are: 7, 8, 9 & 10 where as stop codes are: 8, 9, 10, 11 & 12.



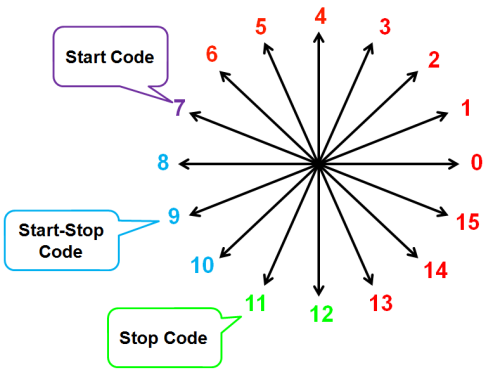
(a) Digit_6 Start



(b) Digit_6 Stop



(c) Digit_6 Scaled Down Trajectory

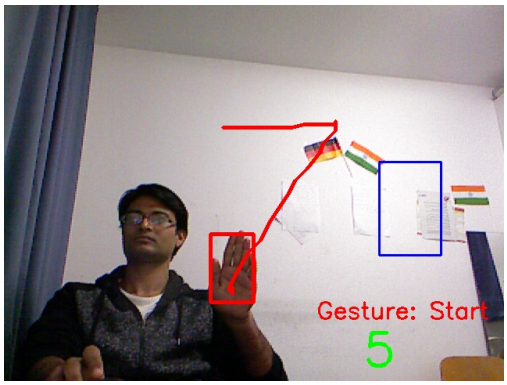


(d) Digit_6 Precondition

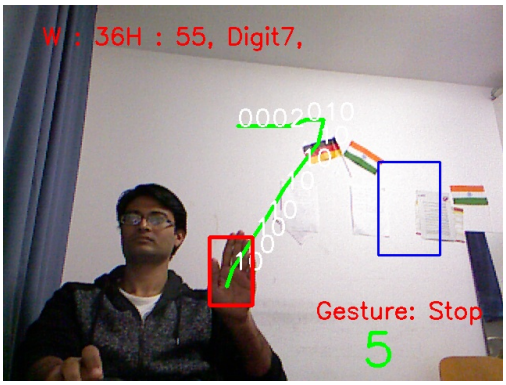
Figure 3.15: Gestures-> Digit_6

3.6.8 Gesture: Digit_7

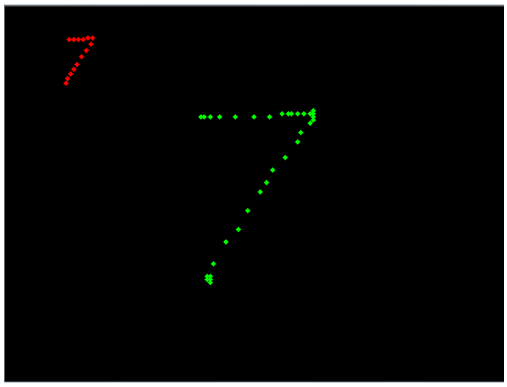
As explained earlier in Digit_1 section, Digit_7 has more similarity to Digit_1 then normally we used to think. Unless we include angle feature here to find out the tilt of line in Digit_1 & 7, we need to rely on start and stop codes which are: 0, 1 & 15 and 9, 10 & 11 respectively. We also added additional criteria just like Digit_5 that the start of the gesture must include a straight line pointing towards right.



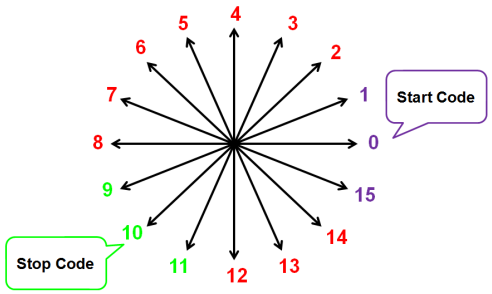
(a) Digit_7 Start



(b) Digit_7 Stop



(c) Digit_7 Scaled Down Trajectory

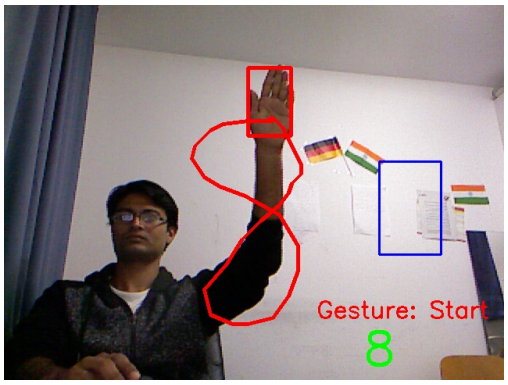


(d) Digit_7 Precondition

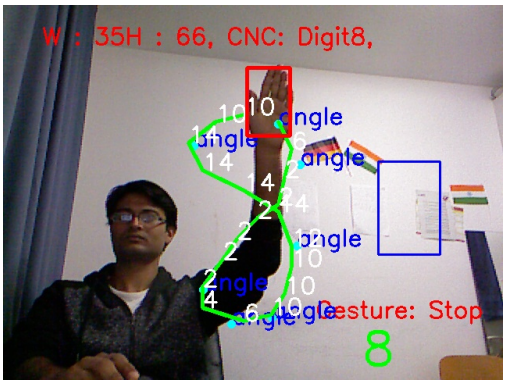
Figure 3.16: Gestures-> Digit_7

3.6.9 Gesture: Digit_8

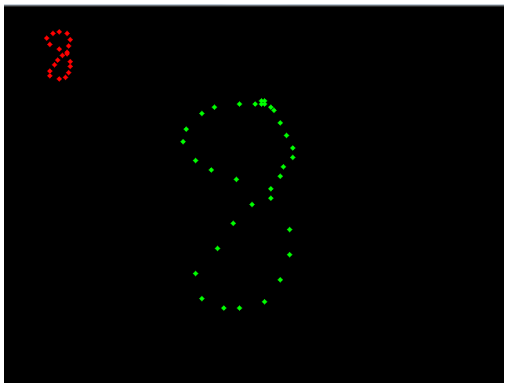
This gesture is the only one in digit category which satisfies the closeness criteria 3.6 discussed in Circle section. For this gesture to idetify, start codes are: 7, 8, 9, 10 & 11 and stop codes are: 4, 5, 6 & 7. There is again very less positive data for this gesture in scaled down case which is only 9, for original case it is 41.



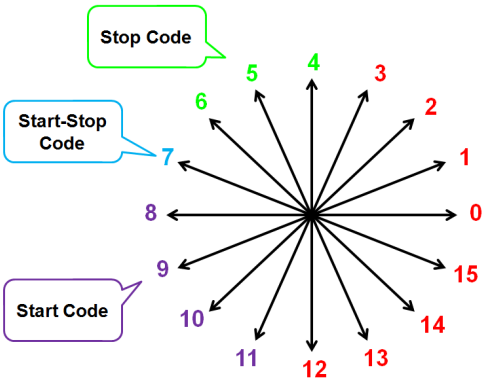
(a) Digit_8 Start



(b) Digit_8 Stop



(c) Digit_8 Scaled Down Trajectory

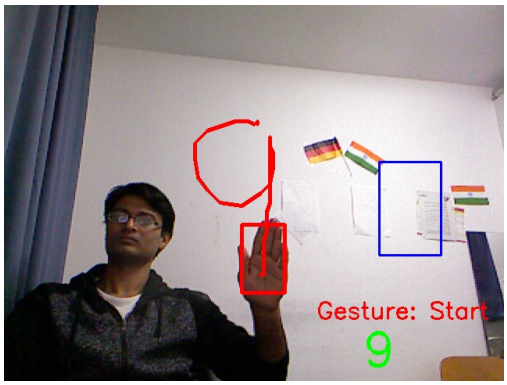


(d) Digit_8 Precondition

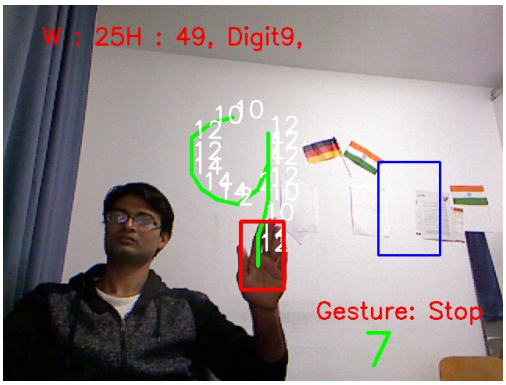
Figure 3.17: Gestures-> Digit_8

3.6.10 Gesture: Digit_9

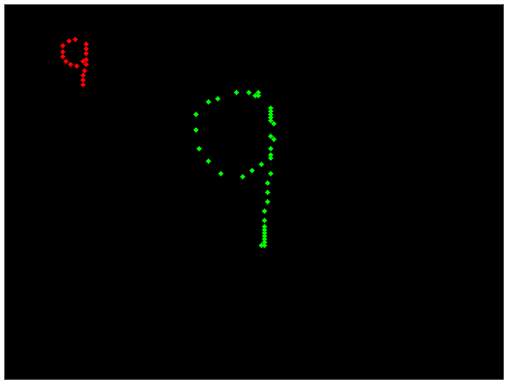
Digit_9 is the last gesture for digit category, and is very closer to Digit_4 as explained earlier. However additional conditions makes it separable from Digit_4 which is: the start and stop points are almost on straight lines. Additionally, start codes are: 7, 8, 9 & 10 and stop codes are: 11, 12 & 13. There are 13 positive training data for scaled down and 47 for original size of trajectories needed.



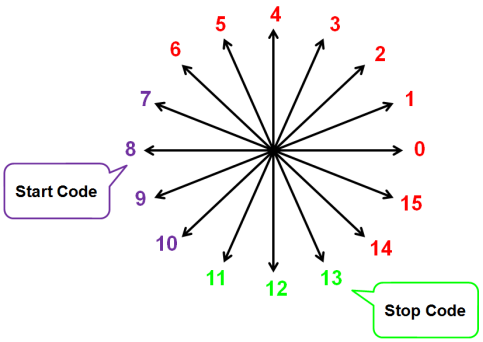
(a) Digit_9 Start



(b) Digit_9 Stop



(c) Digit_9 Scaled Down Trajectory



(d) Digit_9 Precondition

Figure 3.18: Gestures-> Digit_9

3.6.11 Gesture: Letter_S

3.7 Summery

The entire process it explained in the block diagram shown in Figure 3.19. Here the steps are listed as below:

- First of all, wait for intuitive gesture start. To do so, keep hand still for few frames.
- Once gesture recording is started, it will be displayed on screen.
- Next perform certain movement of hand as Digit form or simple shape form.
- Keep hand still again for stopping gesture data recording.
- In the same frame, go to gesture recognizer method and scale down original trajectory to specific size. For flickering noise reduction, choose 3 mm distance points from each other in both X or Y directions.
- Now, calculate an angle between two subsequent points and divide it by 22.5, and thus create codes ranging from 0 to 15.
- Once we have sequence of codes, it will be utilized for both training phase and testing phase.
- For **Training Phase**, collect all possible sequence of codes for all size of shapes.
- For **Testing Phase**, check precondition based on each individual gesture.
- If precondition satisfies then ask SVM prediction with sequence of codes as input vector.

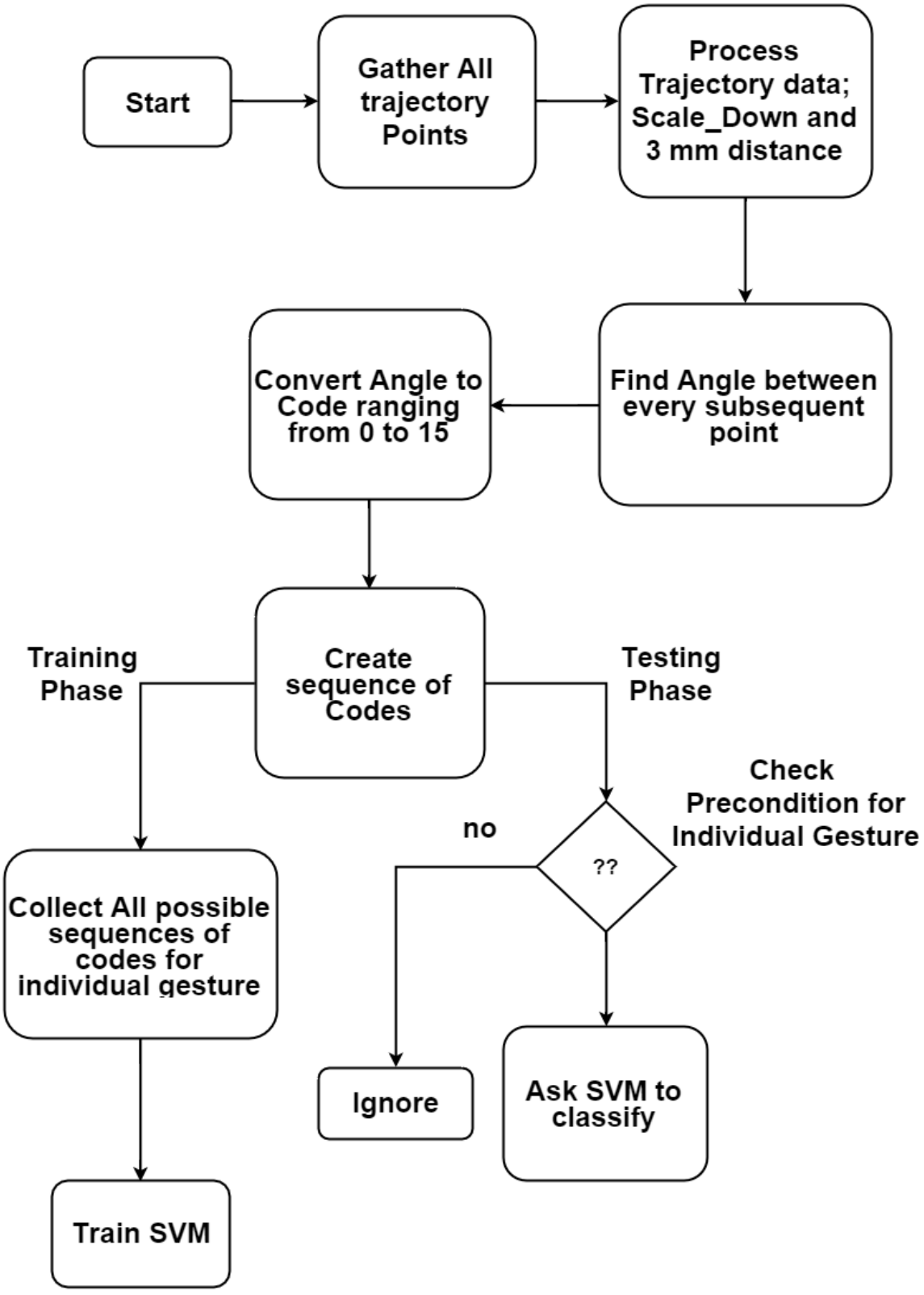


Figure 3.19: A Complete block diagram of Gesture Understanding Algorithm

3.8 Limitation of this approach with SVM

- Although sequence of codes is smart way to represent trajectory points, it is difficult for SVM to distinguish between some shapes like, Square and Circle.
- As it was explained in the SVM section, that classifier calculates an optimal hyperplane to distinguish between classes. If the sequence are so much similar as with Digit_1 & 7 or with Digit_4 & 9, the prediction needs to include some more preconditions before asking SVM for classification.
- Binary classifier will be based on probability of -1 or 1, where 1 represents the desired class and -1 is other. Hence every case where probability is more than 50%, it will be classified as a gesture class.
- Ideally there should be one additional criteria where classifier should point out how much percentage the given test data meets the desired class.
- For negative data set, if we provide data for other Digit for example of Digit_1 vs Digit_7 where most of the time the sequence of codes will matches each other, it will reduce the classification accuracy.
- Scaling down of trajectory will have restrictions when it comes to differentiate fine details about trajectory.
- With original trajectory size, above problems are somehow minimized with lots of training data provided for each single probable size, thus making training time longer.

3.9 Shape Descriptors

There are methods in image processing to identify basic desired shapes from geometry. In this section, we are going to utilize that existing functionality from opencv tutorials:Ref:37. As we can see from Figures 3.22,3.23,3.24 and 3.25, there are basic shapes with distinct geometry which can be easily identified without any complex classifier. Here, we make use of opencv method *ApproxPolyDp* which is abbreviation of approximate polynomial double precision points. This method "calculates approximation of a polynomial curve with desired precision:Ref:opencv: 3.20", which is quite helpful for finding the convex hulls of the shape. As we will see all the figures now on with *angle* in blue color on each convex hull points.

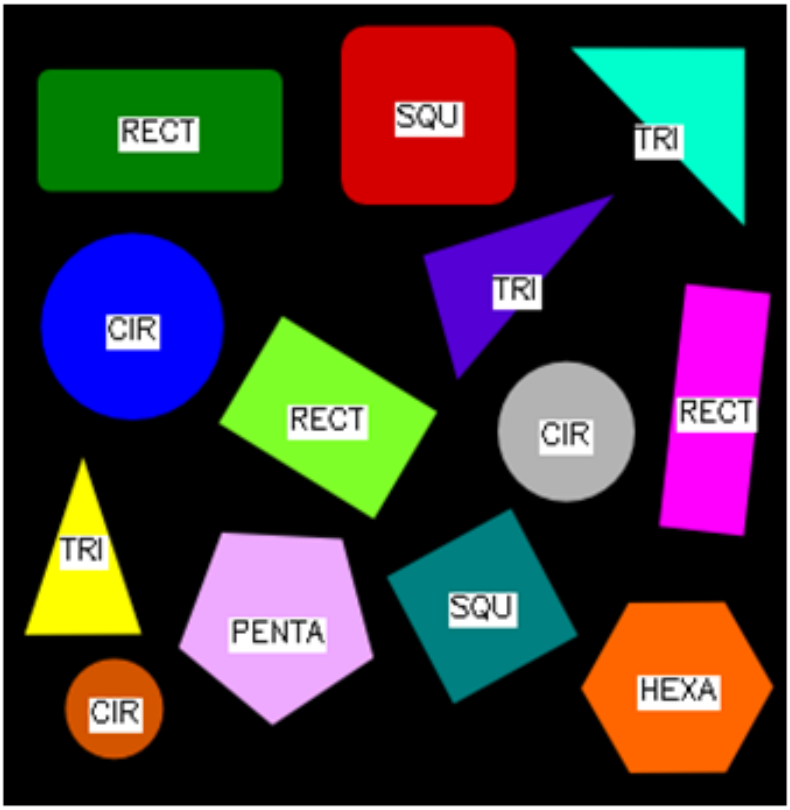
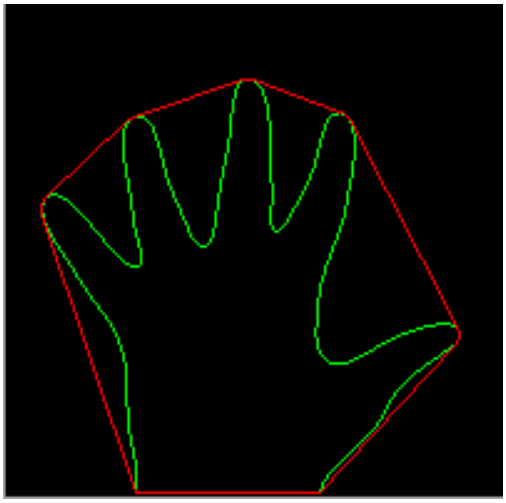
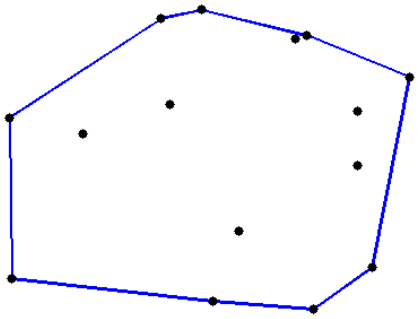


Figure 3.20: Basic Shapes:ref37



(a) Convex Points around Hand Contours



(b) Convex Hull around set of points

Figure 3.21: Convex Hull Examples

3.9.1 Approach

In theory, this techniques should be able to distinguish up-to pentagon and hexagon as shown in Figure 3.20. However user can not draw such as fine shapes in 3D world environment as shown in Figures 3.24 & 3.25, the trajectory will start looking circle in hexagon case. Thus, we restrict our selves up-to square and rectangles. The idea is very simple, find out convex hull of the trajectory and count no of points in the approximation curve vector.

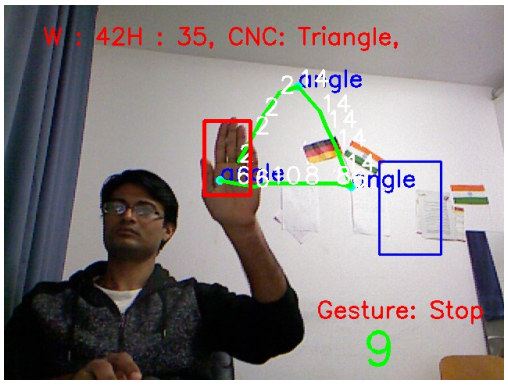
For **Triangle**, the conditions are:

- The convex hull points must be three.
- All 3 angles with neighborhood points from that vortexes must have summation close to 180°.

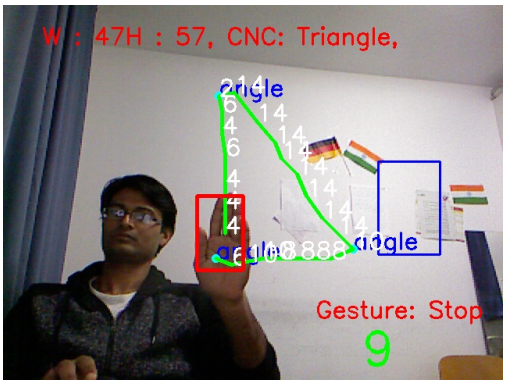
For **Rectangle/Square**, the conditions are:

- The convex hull points must be four.
- All 4 angles with neighborhood points from that vortexes must have summation near 360°.
- Individual angle can not be greater than 100°.
- Ratio between width and height must be greater than 0.9 and less than 1.1 in Square's case.
- If all above cases satisfies and it's not Square then it must be rectangle.

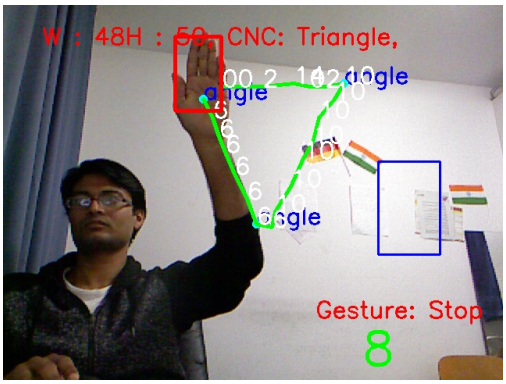
Requirement of this approach comes from the limitation of our previous approach with SVM classifier for basic shape detection. As it was mentioned earlier, SVM cannot distinguish between circle and Square trajectories represented as *sequence of code*, thus we needed a new approach to find out new ways to discover different shapes.



(a) Triangle_1

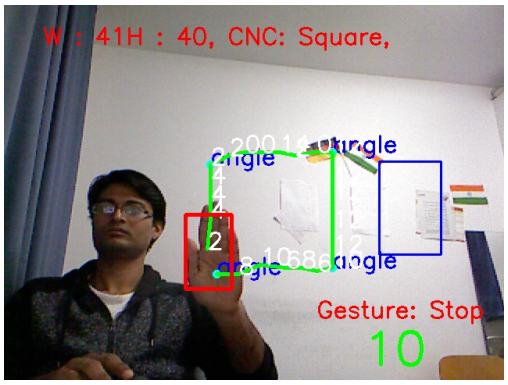


(b) Triangle_2

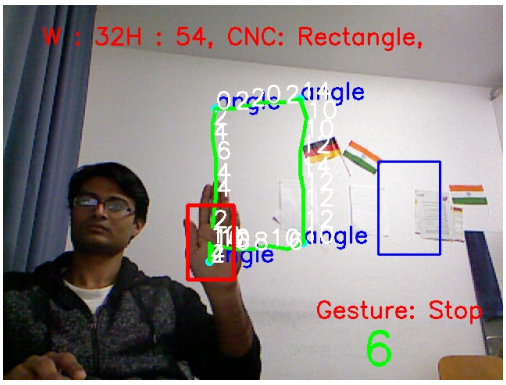


(c) Triangle_3

Figure 3.22: Gestures-> Triangle

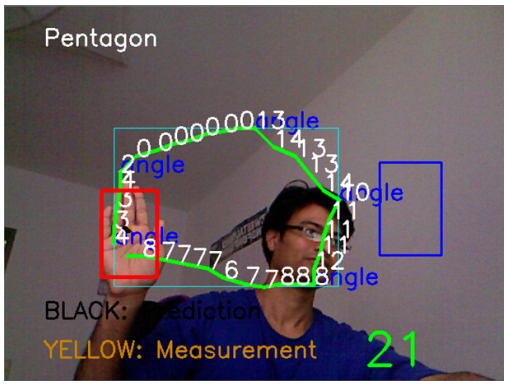


(a) Square

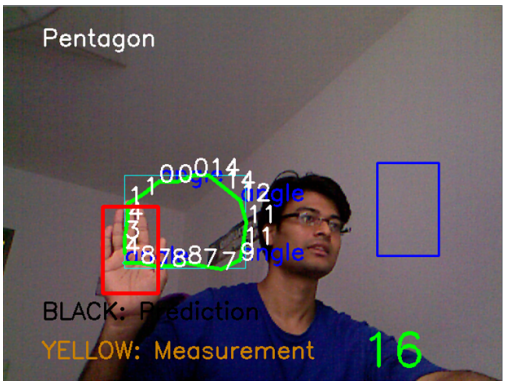


(b) Rectangle

Figure 3.23: Gestures-> Square & Rectangle

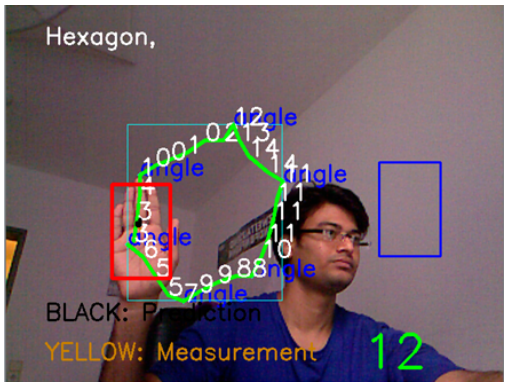


(a) Pentagon_1

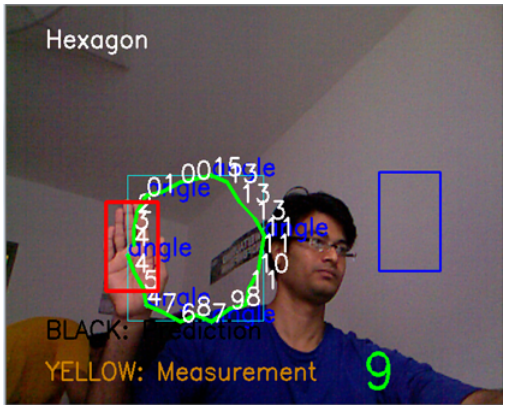


(b) Pentagon_2

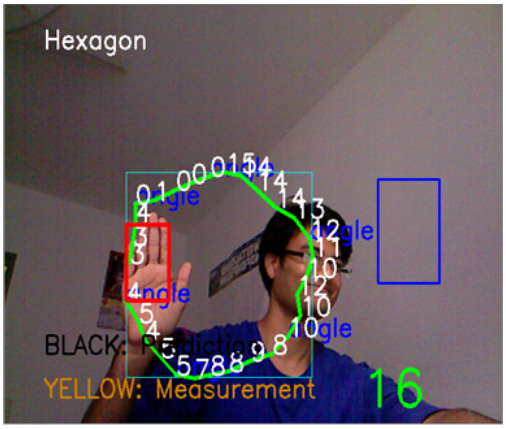
Figure 3.24: Pentagon Shape



(a) Hexagon_1



(b) Hexagon_2



(c) Hexagon_3

Figure 3.25: Hexagon Shape

3.10 Further Research And Scope Of Improvement

- Scaled down approach needs more training data for better robust detection.

- More intuitive way describing gesture not only sequence of codes but may be shape and size to be included in SVM classifier.
- Different classifier such as neural network and HMM can be explored for this approach.

Chapter 4

Platform

two para intro.

This task is specifically developed for MeRoSy project as stated earlier. Ii would be first step towards human machine interface in this big project. MeRoSy stands for *Mensch Roboter Synergie* which is basically used for human robot cooperation for installation and other assembly tasks. Figure 4.1 shows the task scenario with this project. This project is currently in progress at IAT department of Unniverstiy of Bremen.

Current task provides basic hand tracking application with built in methods with gesture/trajectory understanding. In previous chapters, we had extensive look at the methods which have implemented both tasks. Currently there are many interesting tasks going on with this project where object recognition, path planning and reinforcement learning fields are included. In future robot must be trained to be aware of it's surroundings.

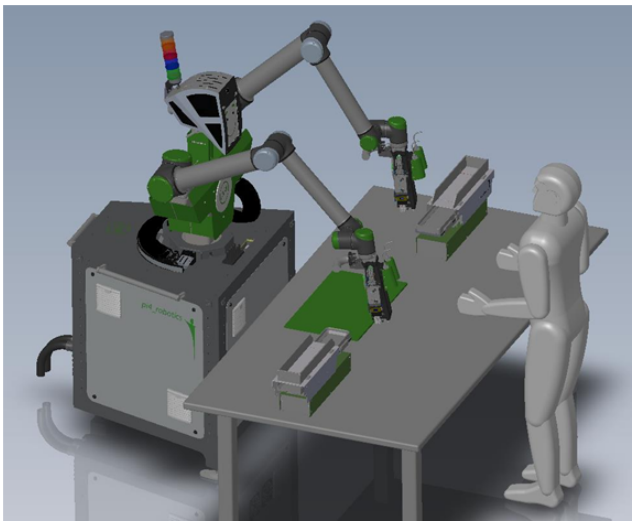


Figure 4.1: MeRoSy task scenario

4.1 Pi4 Robot

Pi4 is the type of robot manufactured by ?? institute. It is currently semi automated. Both arms of the robot has 6 degrees of freedom and can be controlled by remote. In future scenario, it is expected to track human hands by live kinect camera recording. The field of view is shown in Figure 4.1 where camera should be mounted on top of the robot and looking downwards.

Figure 4.2: Pi4 Robot

4.2 Kinect1 with Windows

opencv 2,4, openni 2,2,windows 7, kinect Xbox 360

Kinect 1 or in our case kinect Xbox 360 is the one of first camera Microsoft used for gaming console in it' famous Xbox series video game console which enables players to control their video games using their own body gestures and voice commands. The popularity of this camera had crossed not only gaming industry but had generated lots of interest in computer graphics field. Kinect's main appeal was depth data along with RGB color images in real time, and it was possible cheaply. Earlier scientists had to use more costly time of flight cameras for depth measurement.

The depth sensor consists of two parts, an infrared projector, which is located on the left side of the RGB camera, and receiver which is located on the right side of the camera. Figure 4.3 shows the components of the sensor. The most important components are the infrared transmitter and the receiver. The most common way to calculate the distance of an object is by projecting infrared light on it and then calculating the time it takes the light to reflect off the object and return to the receiver, but the camera sensor works in a different way; the infrared projector projects a fixed pattern of infrared spots onto the object, then the receiver captures a shifted grid of these spots, the processor then calculates the offset of each of the spots to generate a depth map. The camera sensor can measure the distance of an object 2 meters away within 1 cm accuracy and maximum range up-to 3.5 meters.Ref:wikipedia

Openni(Open Natural Interaction) is a open source project originally founded by PrimeSense, provides kinect interface with computer. It also provides low level support for various kinect features. In this project, we have used mostly Openni2, with OpenCv 2.4.9 on windows platform. Latter, this task was shifted to kinect 2 with ros indigo on Ubuntu platform.

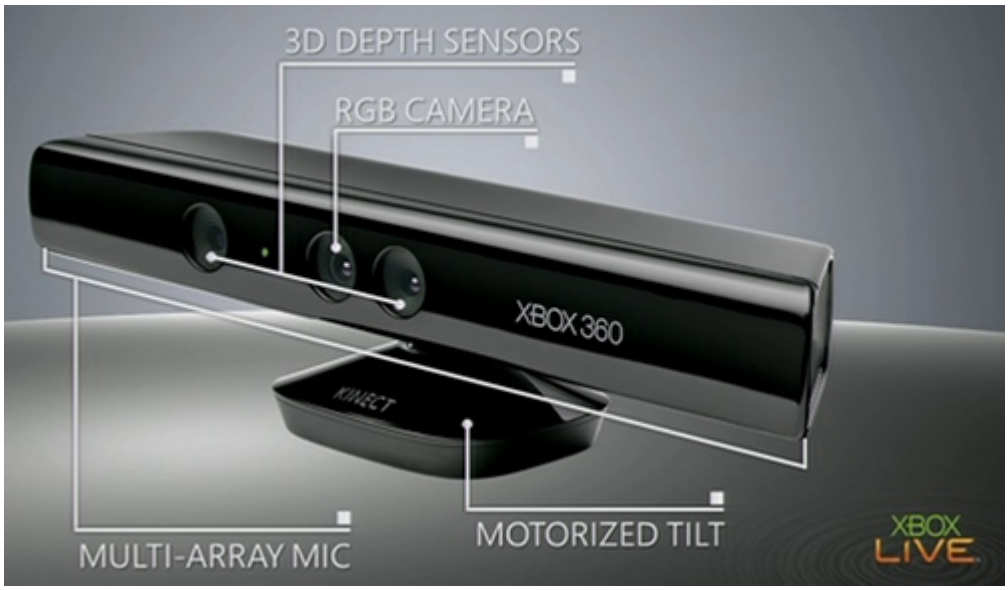


Figure 4.3: Kinect Xbox 360

As mentioned earlier, kinect_1 depends upon infrared projected patterns, this will cause problem when there is direct sunlight in the scene. Light will distort the pattern and hence depth image will be corrupted, and thus kinect_1 can only work indoor environment. These limitations is eliminated by time of flight technology adopted by kinect_2.

4.3 Kinect2 with ROS & Ubuntu

Kinect 2 is the latest developement in kinect series, and is equipped with different time of flight technology. Microsoft has made this expensive technology available to common users with low cost of kinect and with better results.

4.3.1 Kinect2 Limitation

mention depth image noise, specially include depth edge image with noisy backgnrd vs kinect1 .. mention something abt time of flight camera limitations.....

References

[1] Clark. Validity and reliability of the nintendo wii balance board for assessment of standing balance. pages Gait & Posture, 31(3), 307–310, 2012.

[2] Fraisse etal. Cotton, Vanoncini. Estimation of the centre of mass from motion capture and force plate recordings: a study on the elderly. *USA: CRC Press*, pages (pp.272–291), 1990.

[3] S. Cotton, A. Murray, and P. Fraisse. Statically equivalent serial chains for modeling the center of mass of humanoid robots. In *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*, pages 138–144, 2008.

[4] S. Cotton, A.P. Murray, and P. Fraisse. Estimation of the center of mass: From humanoid robots to human beings. *Mechatronics, IEEE/ASME Transactions on*, 14(6):707–712, 2009.

[5] R. Contini Drillis and Bluestein. M. body segment parameters: A survey of measurement techniques. 1964.

[6] Bernard Espiau and Ronan Boulic. On the computation and control of the mass center of articulated chains, research report, no. 3479. 1998.

[7] Machline Meir Ashdod Arieli Yoel Jerusalem. Freedman Barak Binyamina, Shpunt Alexander. Depth mapping using projected patterns. 2008. retrieved april, 2012. www.freepatentsonline.com.

[8] Fairhead Harry. All about kinect. 2012. retrieved april, 2012. *i-programmer : http://www.i-programmer.info/babbages-bag/2003-kinect-the-technology-.html*.

[9] P. Sardain and G. Bessonnet. Forces acting on a biped robot. center of pressure-zero moment point. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 34(5):630–637, 2004.

[10] VN & Chugunova LG (1990) Zatsiorsky, VM; Seluyanov. Methods of determining mass-inertial characteristics of human body segments. *USA: CRC Press*, pages (pp.272–291), 1990.