



MIDDLE EAST TECHNICAL UNIVERSITY
NORTHERN CYPRUS CAMPUS

Computer Engineering Program

CNG 492
GRADUATION PROJECT

Hand Gesture Segmentation from Uniform and
Non-uniform Images

Final Report

Sezin Köktaş

Erencan Duman

Vuralcan Asal

2079382

1862630

1862531

Table of Content

List of Figures	5
List of Tables	7
1 Problem Definition	8
2 Literature Review	8
3 Identification of Constraints	21
3.1 Economic Constraints	21
3.2 Environmental Constraints	21
3.3 Social Constraints	21
3.4 Sustainability Constraints	22
4 Problem Area, Rationale & Objectives	22
5 Requirements	24
5.1 User Requirements	24
5.2 List of System Specifications	24
5.3 System Process Model	25
5.4 System High Level Sequence Diagram	25
5.5 Test Plan	26
5.5.1 Testing Strategy	26
5.5.1.1 Strategy for Image Processing methods	26
5.5.1.2 Strategy for YOLO	26
5.5.1.3 Strategy for Refinenet	27
5.5.2 Test Items	27
5.5.2.1 Unit testing	27
5.5.2.2 Integration testing	29
5.5.2.3 Smoke testing	29
5.5.2.4 Performance testing	29

5.5.3	Summary of features to be tested	29
5.5.4	Summary of features not to be tested	30
5.5.5	Item pass/fail criteria.....	30
5.5.6	Unit testing criteria for Image Processing Algorithms.....	30
5.5.7	Unit testing criteria for YOLO.....	30
5.5.8	Unit testing criteria for RefineNet.....	31
5.5.9	Test deliverables	31
5.5.10	Test schedule.....	31
6	Research Method, Design Details	32
6.1	Research Method	32
6.2	Design Details	33
6.2.1	List of Functions and Definitions.....	33
6.2.2	List of Special Algorithms	34
6.2.3	Process Diagram	35
7	Implementation details	36
7.1	Project Management	36
7.1.1	Estimation	36
7.1.1.1	COCOMO	37
7.1.1.2	Jones's First-Order Effort	39
7.1.2	Tasks	39
7.1.3	Milestones	40
7.1.4	Activity Diagram.....	41
7.1.5	Gantt Chart.....	42
7.1.6	Task Allocation	43
7.2	Testing	44
7.2.1	Unit Test.....	44
7.2.2	Integration Testing	60

7.2.3	Smoke Testing	60
7.2.4	Performance Testing	64
7.2.5	Test Requirements.....	67
8	Conclusion.....	68
9	Installation Manual.....	69
10	User Manual	77
	References.....	81

List of Figures

Figure 1: Static Gesture Set (Ghotkar, Khatal, Khupase, Asati, and Hadap, 2012).....	10
Figure 2: The Output with Intermediate Steps (Ghotkar, Khatal, Khupase, Asati, and Hadap, 2012)	11
Figure 3: The Outputs of the Steps for Plain Background (Randive, Mali, and Lokhande, 2012)	13
Figure 4: The Outputs of the Steps for Complex Background (Randive, Mali, and Lokhande, 2012)	13
Figure 5: Person Parts example (Lin, Milan, Shen, and Reid, 2017).....	18
Figure 6: System Process Model.....	25
Figure 7: System High Level Sequence Diagram	25
Figure 8: Process Diagram.....	35
Figure 9: Input_1 & Output_1 for Illumination	45
Figure 10: Input_2 & Output_2 for Illumination	45
Figure 11: Input_3 & Output_3 for Illumination	46
Figure 12: For Skin Detection, Successful Examples with Inputs & Outputs	47
Figure 13: For Skin Detection, A Fail Example with Input & Output.....	48
Figure 14: For Filtering, Successful Examples with Inputs & Outputs	49
Figure 15: For Filtering, a Fail Example with Input & Output.....	49
Figure 16: For YOLO Training, Successful Examples with Inputs & Outputs	50
Figure 17: For YOLO Training, A Fail Example with Input & Output.....	51
Figure 18: For YOLO Testing, A Successful Example with Input & Output.....	52
Figure 19: For YOLO Testing, Fail Examples with Inputs & Outputs.....	52
Figure 20: For K-means Clustering, Two Examples with Inputs & Outputs.....	53
Figure 21: For EGOHAND Model, A Successful Example with Input & Output.....	54
Figure 22: For EGOHAND Model, Fail_1 with Input & Output	55
Figure 23: For EGOHAND Model, Fail_2 with Input & Output	55
Figure 24: For EYTH Model, A Successful Example with Input & Output.....	56
Figure 25: For EYTH Model, Failed Examples with Inputs & Outputs	56
Figure 26: For GTEA Model, A Successful Example with Input & Output.....	57
Figure 27: For GTEA Model, Failed Examples with Inputs & Outputs	57
Figure 28: For HOF Model, A Successful Example with Input & Output	58
Figure 29: For HOF Model, Fail Examples with Inputs & Outputs	58
Figure 30: For PASCAL Model, Successful Examples with Input & Output	59
Figure 31: For PASCAL Model, Fail Examples with Inputs & Outputs	59
Figure 32: In Smoke Testing, An example For Scenario_1 with Input & Output	61

Figure 33: In Smoke Testing, An example For Scenario_2 with Input & Output	62
Figure 34: In Smoke Testing, An example For Scenario_3 with Input & Output	63
Figure 35: In Smoke Testing, An example For Scenario_4 with Input & Output	64
Figure 36: The most successful segmented image in fail results example.....	66
Figure 37: The worst segmented image in fail results example.....	66
Figure 38: The most successful segmented image in successful results example	67
Figure 39: The worst segmented image in successful results example.....	67
Figure 40: Download MATLAB Step1	70
Figure 41: Download MATLAB Step2	71
Figure 42: Download MATLAB Step3	71
Figure 43: Download MATLAB Step4	72
Figure 44: Download MATLAB Step5	72
Figure 45: Download MATLAB Step6	73
Figure 46: Download MATLAB Step7	73
Figure 47: Download MATLAB Step8	74
Figure 48: Download MATLAB Step9	74
Figure 49: Download Python Step1	75
Figure 50: Figure 15: Download Python Step2	75
Figure 51: Update PIP.....	76
Figure 52: Go to the Folder by Using cd	76
Figure 53: PIP Installation	77
Figure 54: How to Work the Software, Step1.....	77
Figure 55: How to Work the Software, Step2.....	78
Figure 56: How to Work the Software, Step3.....	78
Figure 57: How to Work the Software, Completed Message	79
Figure 58: Output Folder	80

List of Tables

Table 1: Test order for hand gesture segmentation.....	28
Table 2: Test schedule	31
Table 3: Function Points	36
Table 4: General System Characteristic	37
Table 5: Function points estimation.....	37
Table 6: List of the project tasks and their durations	40
Table 7: Activity Diagram	41
Table 8: Activity Timeline.....	42
Table 9: Gantt chart	42
Table 10: Task allocation.....	43
Table 11: Staff allocation.....	43

1 Problem Definition

In our society, there are many people that live with disabilities. Hearing disability is one of the most common one. According to the World Health Organization (WHO), around 466 million people which means over %5 of the world's population has disabling hearing loss (WHO, 2020).

People who has the hearing disability face with many problems. The main problem is the communication. Sign language is developed to overcome this problem. However, the number of people who know the sign language is quite low. Therefore, deaf people still suffering from communication problem in daily life. So, we think that some kind of sign language translator would make deaf people more active in life. Essential parts to design such automated translator are segmentation and recognition of hand gesture.

In the literature, number of people that try to solve the segmentation problem is limited. Therefore, this project aims to find a solution to this real life problem by offering a successfully working segmentation algorithm. Image processing, computer vision and deep learning techniques will be used in development phase. The segmentation algorithm is designed to work with static data (images).

2 Literature Review

Many applications had been used for hand gesture segmentation. In this section, various techniques that are published in different articles will be described in order to give a path to find a way to handle the hand segmentation. Each article will be explained in detail, specifically by defining their weak and/or strong parts.

According to Ghotkar, Khatal, Khupase, Asati, and Hadaps' article in 2012, hand gestures are spontaneous and powerful communication modality for Human Computer Interaction (HCI). The

objective of their work is to overcome the vision-based challenges, such as dynamic background removal, skin color detection for HCI and variable lighting condition.

Ghotkar, Khatal, Khupase, Asati, and Hadap consider hand gesture recognition system “S” like “S = {I, G, M, F, O}” where “I” represents a set of input hand gestures, “G” represents a set of single-handed anticipated static gestures, “M” represents mouse operation like left or right click, “F” represents feature vector for “G”, and “O” represents output with application interface “A”.

The authors’ aim was to overcome the challenge of skin color detection for natural interface between user and machine. Hence, to detect the skin color under dynamic background, three color spaces has been chosen that are commonly used in computer vision applications, RGB (red “R”, green “G”, blue “B”), HSV (Hue, Saturation, Value), and CIE-Lab (Ghotkar, Khatal, Khupase, Asati, and Hadap, 2012).

The main advantage of RGB is simplicity. However, it is not uniform since it does not separate chrominance, which is the color information in a signal, and luminance, which is apparent brightness or in other words how bright an object appears to the human eye, and R, G, and B components are highly correlated (Ghotkar, Khatal, Khupase, Asati, and Hadap, 2012).

HSV express Hue with dominant color (red, green, purple, and yellow) of an area. Saturation measures the colorfulness of an area in proportion to its brightness. Value is related to the color luminance. This model discriminates luminance from chrominance. This is very useful model for computer vision since the “Value” or “Intensity” is independent of the color information. However, this model gives poor result where the brightness is very low (Ghotkar, Khatal, Khupase, Asati, and Hadap, 2012).

CIE-Lab separates a luminance variable L from two perceptually uniform chromaticity variable (a , b). “ L ” represents lightness, “ a ” represents red/green value, and “ b ” denotes the blue/yellow color value (Ghotkar, Khatal, Khupase, Asati, and Hadap, 2012).

Ghotkar, Khatal, Khupase, Asati, and Hadap use this hand gesture recognition system and these color models since their aim was to design efficient gesture recognition system (2012).

The first technique that they use for hand segmentation is HSV color space and sampled Storage approach. As you can see in Figure 1, authors created a dataset that has static hand gesture images. These images are processed, and then, the R , G , B values of these images are taken. The green color sample of the images is passed to the algorithm and from H - S histogram where the range of H and S are specified by themselves. Algorithm could able to subtract dynamic background. However, it is sensitive to little variation in color brightness (Ghotkar, Khatal, Khupase, Asati, and Hadap, 2012).



Figure 1: Static Gesture Set (Ghotkar, Khatal, Khupase, Asati, and Hadap, 2012)

HSL or Lab Color Space is the other methods which have been used for hand segmentation. Initially, the input acquired RGB image was converted to lab color space CIE-Lab. a^* axis and $+a$ direction shift towards the red while along the b^* axis $+b$ movement shift toward yellow. Once the image gets converted into a^* and b^* planes, thresholding is done. After that, convolution operation is applied on binary images for the segmentation. As a last step, morphological processing is done to get the superior hand shape. All these steps are shown in Figure 2. Although, this algorithm

works for skin color detection, it is not useful for complex backgrounds since it is sensitive for complex background (Ghotkar, Khatal, Khupase, Asati, and Hadap, 2012). In addition to that, the purpose of using convolution and wherewith convolution is applied are not known, so this is another weak part of this method.



Figure 2: The Output with Intermediate Steps (Ghotkar, Khatal, Khupase, Asati, and Hadap, 2012)

Hand Tracking and Segmentation (HTS) Algorithm is useful for skin color detection and removal of complex background. The limitation of previous two techniques is overcome with this proposed hand tracking algorithm. Step by step design and implementation of this algorithm can be found in the article. The algorithm is found to be working for dynamic color tracking under complex background and able to segment required hand image by using morphological operations and obtaining the contour (Ghotkar, Khatal, Khupase, Asati, and Hadap, 2012).

In summary, in Ghotkar, Khatal, Khupase, Asati, and Hadaps' article, HTS algorithm is found useful for hand segmentation. However, their results are not strong since all these techniques are tested only based static gesture set that is collected by authors. Therefore, these results may not be acceptable for a large gesture sets. Moreover, this hand gesture recognition system may not work for different database since they created that for their database.

In Randive, Mali, and Lokhandes' article in 2012, there are two types of constraints applied on an image, which are background constraint and foreground constraint. Background constraints refer

to the constraints on the environment in which the hand will be tracked. Foreground constraints refer to the hand itself. These constraints make such systems less flexible and robust to the applied environment.

Authors proposed different methods for successful gesture recognition. Firstly, experiments have performed on images which has a plain, uniform background. Once it has been shown that the proposed algorithm was successful on uniform images, further analysis has been performed based images with a complex background (Archana S. Ghotkar, 2012).

For complex background, they first tried for subtraction of two consecutive frames by considering the first frame as background and the second one as current frame. But in this case, if the person or signer holds the hand in front of camera continuously for few seconds, the present image (i.e. hand image) also goes into background (Randive, Mali, and Lokhande, 2012).

Then they tried for skin color detection algorithm that detects skin color accurately. But in this case, results are mostly depend on lighting conditions and if the background is also of skin color then it became difficult to detect hand from face and skin colored background. You can see their hand segmentation algorithm for both plain, uniform background, and complex background in their article (Dawod, Abdullah, and Alam, 2010) .You can see the outputs of steps for both plain and complex background in Figure 3, and Figure 4.



Figure 3: The Outputs of the Steps for Plain Background (Randive, Mali, and Lokhande, 2012)

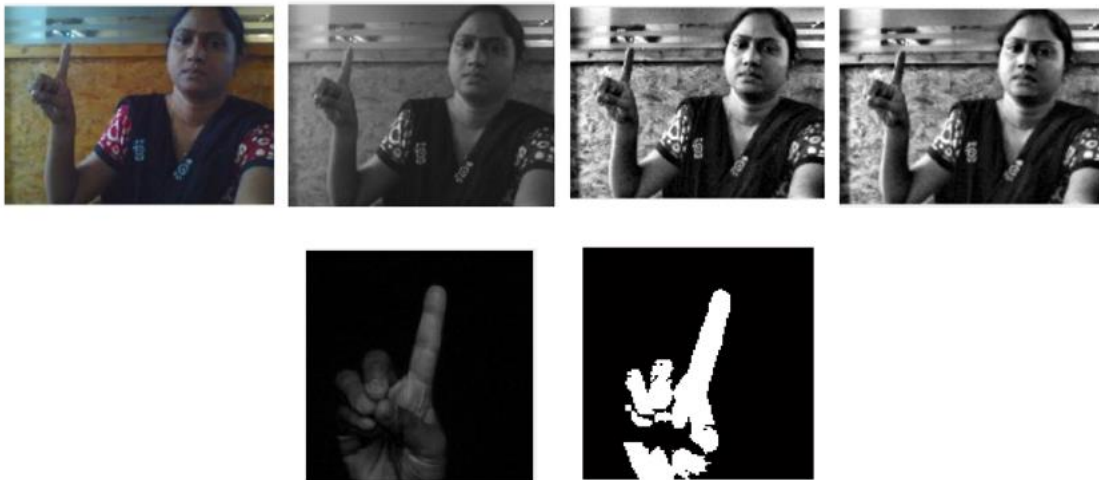


Figure 4: The Outputs of the Steps for Complex Background (Randive, Mali, and Lokhande, 2012)

In summary, Randive, Mali, and Lokhandes' work gives better results on plain and complex background with stable light conditions. However, it recognizes only static gestures. Their algorithm works well for getting good accuracies for plain (%82) and for complex (%85) background. However, their algorithm can be improved to fill the holes in threshold image or to use in dynamic gestures. Moreover, the dataset that they use is not mentioned or clear. Seems like they created their own dataset or they use random images that they found in the internet. Therefore, these results can be change for different data sets (2012).

The thing that we talked about but did not narrate well is thresholding. According to Bradley, and Roths' research in 2007, image thresholding segments a digital image based on a certain characteristic of the pixels (for example, intensity value). The goal is to create a binary representation of the image, classifying each pixel into one of two categories, such as "dark" or "light". This is a common task in many image processing applications and some computer graphics applications.

Furthermore, there is another research that may help us is about finger detection. Ravikiran, Mahesh, Mahishi, Dheeraj, Sudheender, and Pujari show that finger detection from the image can be done in four parts. First is getting the image then detecting edges then clipping the image and finally tracing the boundaries (2009).

There is a method which is called canny edge detection. The canny algorithm uses an optimal edge detector based on a set of criteria which include finding the most edges by minimizing the error rate, marking edges as closely as possible to the actual edges to maximize localization, and marking edges only once when a single edge exists for a minimal response (Ravikiran, Mahesh, Mahishi, Dheeraj, Sudheender, and Pujari, 2009). According to canny, the optimal filter that meets all three criteria above can be efficiently approximated using the first derivative of a Gaussian function.

Clipping is used to cut a section through the image currently being rendered. The image contents that pertain to the area of interest are retained after clipping. The edge detected image contains portions that are unnecessary for further analysis. Hence we eliminate them by adopting two techniques discussed below (Ravikiran, Mahesh, Mahishi, Dheeraj, Sudheender, and Pujari, 2009). The first technique examines pixels from the bottommost y level and at each level checks if there are three or more consecutive white pixels. If the above condition is satisfied we mark this y-level

as “y1”. The second technique exploits the fact that most of the edge detected images of hand gestures have the wrist portion which has a constant difference between either ends on the same y-level. When it approaches the palm and region of the hand above it, this difference increases drastically. We make use of this fact and find the y-level where this event occurs and mark this y-level as “y2” (Ravikiran, Mahesh, Mahishi, Dheeraj, Sudheender, and Pujari, 2009).

Boundary Tracing is the last part after clipping. This phase of the algorithm is the heart of processing. The edge detected image which is clipped serves as the input to this phase. The output is the traced image where the trace-points are highlighted in blue and the points where the fingertip is detected are highlighted in red. This part also has other steps inside (Ravikiran, Mahesh, Mahishi, Dheeraj, Sudheender, and Pujari, 2009).

- Identifying the Optimal y-level
- Identifying the initial trace direction
- Tracing with appropriate switch of direction
- Rejoining the trace on encountering breaks
- Finger Tip Detection

In summary, the article that is written by Ravikiran, Mahesh, Mahishi, Dheeraj, Sudheender, and Pujari in 2009 has a weakness that we also might have in our process. The problem is happening when the background color and the hand color is the same. When they are making tests, they always used a uniform image that has a black or white background color. We do not think that this method is not suitable for non-uniform images.

Clustering is to group objects of a database into multiple classes or clusters so that objects in the same group have a large similarity and objects in the different groups have a large dissimilarity. There are many clustering algorithms like K-Means algorithm, the Cure levels-based algorithm etc. (Wang, and Su, 2011).

The basic idea of the K-Means clustering algorithm is that we firstly select k objects as initial cluster centers, then assign the distance, calculated by checking the distance between each object and each cluster center, to nearest cluster after, update the averages of all clusters, and repeat this process over and over again until we cannot find any different distance.

In our project, we use a ready function, provided in a toolbox of Matlab, called “kmeans()”. This function works with the same idea. It takes an n -by- p data matrix X and a k value stands for number of clusters or class that is needed. According to number of k , the initial cluster centers are assigned in different places randomly. Then it calculates the distances of all the points by using Euclidian distance to check which are close to these centers and reassign the centers every time. This process goes like that until we cannot find any different distance.

Until this point, we have made researches about image processing techniques that would help us. The main problems of the image processing techniques are complex background and illumination. Therefore, using only image processing techniques will not be useful in daily life. This situation shows the fact that the concepts of computer vision and deep learning are needed.

According to the Noh, Hong, and Hans’ research in 2015, convolutional neural networks (CNNs) are widely used in various visual recognition problems such as image classification, object detection, semantic segmentation, visual tracking, and action recognition. The representation power of CNNs leads to successful results.

Lin, Milan, Shen, and Reids' research states that very deep convolutional neural networks (CNNs) have shown outstanding performance in object recognition and have also been the first choice for dense classification problems such as semantic segmentation (2017). However, repeated subsampling operations like pooling or convolution striding in deep CNNs lead to a significant decrease in the initial image resolution. That is the main weakness of CNNs. To get rid of this weakness, they present RefineNet, a generic multi-path refinement network that explicitly exploits all the information available along the down-sampling process to enable high-resolution prediction using long-range residual connections. In this way, the deeper layers that capture high-level semantic features can be directly refined using fine-grained features from earlier convolutions. The individual components of RefineNet employ residual connections following the identity mapping mindset, which allows for effective end-to-end training. Further, they introduce chained residual pooling, which captures rich background context in an efficient manner. In particular, we achieve to get high results on the challenging PASCAL VOC 2012 dataset.

In Everingham, Van Gool, Williams, Winn, and Zissermans' article, there is more information about PASCAL VOC. It is visual objection classes which is used for recognition and detection an object. In PASCAL VOC 2012, there are different kind of image sets to be able to train different algorithms. Most of them are used in autonomous car drive. Car supposed to know its countenance while driving itself. Traffic lambs, people, kinds of animals and traffic sign boards are some of them (2010).

According to Lin, Milan, Shen, and Reids' research, they used PASCAL VOC 2012 datasets to have best segmentation models in RefineNet. They trained RefineNet with PASCAL VOC 2012-person dataset (2017). While they are using the person dataset, they improved the model. The

trained model can understand person. Additionally, it can understand person parts as well. So, it segments person parts with different colors. You can see the example below.



Figure 5: Person Parts example (Lin, Milan, Shen, and Reid, 2017)

The problem with this approach, hand is not segmented alone. It is same color from elbow to hand. Although we can use this trained model, we need to segment only hand in our project. This is still usable, but it cannot help alone. We still need to improve this kind of outputs.

According to Urooj, and Borjis' research in 2018, they also used RefineNet for segmentation of hand gesture. The main result of their research is the deployment of their own train models. They trained RefineNet with different hand gesture datasets. At the end, they obtained four different trained models. These models are HandOverFace, GTEA, Ego Hands, and EYTH. Each model is trained almost three hundred images within a week. They used Nvidia Titan X graphic card in training phase. In total, their training phase finished after a month even if they used one of the best GPU.

Thanks to Urooj, and Borjis' research in 2018, we can estimate how much time it would take if we try to have our own train model. Also, proposed train models have same the aim with our project. Therefore, we can try these models in our project too.

The weakness of Urooj, and Borjis' research in 2018 is that they used small dataset in training phase. Therefore, it may not be helpful if the input dataset are way different from the train dataset. It seems like they proposed these train models for specific purposes. From this point of view, we might not have a good success rate with these models because our input dataset is very large and different. The reason of this difference is that they used random hand gesture pictures in training phase for every model. However, our dataset has the hand gesture of the sign language images with varied aspects.

After analyzing the RefineNet and the train models, we decided to research more about object detection.

Yolo, which is referred "you only look once," which provides some abilities to machines. According to relevant research which was conducted by Redmon, Divvala, Girshick, and Farhadi in 2016 claims that YOLO has provided a human ability to machines, which helps the machines to detect objects in real life. Thanks to this feature, the device can provide a separate image in bounding boxes and associated class probabilities. For example, when an image which includes a dog is uploaded to YOLO, it can process it and find the dog in the image if the YOLO learns the dog before. They also mentioned that using YOLO image processing is simpler, quite straightforward, and extremely fast. By the way, the predictions of YOLO are international, so when you use YOLO, the predictions are global.

Moreover, object detection is not easy job to do, on the other hand, it is one of the challenging problems in the area of computer vision. Furthermore, although it is so challenging, YOLO is performed better in detect an object and localized it then label them in a bounding box (Shafiee, Chywl, and Wong, 2017).

Another research on YOLO explained that why YOLO is so essential. They said that the purpose of object detection is so important; that's why they think that it should be fast, accurate, and as well as the other features it also is able to recognize a wide variety of objects. In light of this information, they claimed that YOLO is one of the useful methods compared to its equivalent (Redmon, and Farhadi, 2017). Many other researchers prove that YOLO is superior software. According to a research, the YOLO is going to be much more useful and important in the future, because of the ability of detection of an object (such as people, animals, walls), the YOLO considered as the key software components in the near future for the autonomous car industry (Ćorović, Ilić, Durić, Marijan, and Pavković, 2018).

There are also other relevant researches which pointed out the importance and goal of the YOLO. For instance, in recent years, Huang, Pedoeem, and Chen state that YOLO has become a significant field in the computer vision area (2018). Moreover, thanks to YOLO, the machines are allowed to detect and classify an object, and this is so practical to use in face detection and face recognition which is so useful in many areas like security, observing nature, autonomous vehicles, etc. (Huang, Pedoeem, and Chen, 2018).

In this research project, one of the most challenging parts is to detect the hands on non-uniform images. Since the non-uniform images consist of sophisticated backgrounds or the same color with the objects, the objects are not easy to find. Based on the presented comprehensive literature review above, YOLO is the best option to detect the places of the objects.

Following testing methods will be applied to the end-product.

- Unit Testing
- Integration Testing

- Smoke Testing
- Performance Testing

3 Identification of Constraints

In this section, constraints of the segmentation of hand gesture function are evaluated.

3.1 Economic Constraints

The function has been implemented in MATLAB and Python. Users should not face problems with Python because it is open source. However, they should pay for MATLAB to use the function since it is only commercially available. Also, high GPU is essential since RefineNet needs at least Nvidia Cuda 9.0. The computer that we use to develop the function has Nvidia Quadro k620 graphic card which is not cheap at all.

3.2 Environmental Constraints

The function can only work with the computers. It does not need an internet connection. User that uses a laptop can face with problems when the laptop is in the battery mode. The reason is that function uses GPU and it might consume much power. Therefore, it is not recommended to run the function in battery mode for laptops. This might affect the mobility of users.

3.3 Social Constraints

The function's main feature is the hand gesture segmentation from images. So, users that are using the function are expected to work with sign language. However, users can use it to develop such games like rock-paper-scissor. As a result, the hand gesture segmentation function can be also used in different areas.

3.4 Sustainability Constraints

The function developed in Python 3.6.8 and MATLAB 2018a. Therefore, it might not work with other versions.

4 Problem Area, Rationale & Objectives

The aim of the project is to develop a hand gesture segmentation function that can create a new path on way of communication between the people using sign language and the people who do not know sign language. The function may help to create an application that can perform automated sign language translation. To do such application, segmentation and recognition stages are the essential parts. Also, a good segmentation function will decrease the number of challenges that developers may face when developing a recognition function. Therefore, we are going to start with basic steps. At first, we know that we need to analyze the images; that's why, MATLAB is the best option for us because MATLAB is very efficient when production code for numerical and also for scientific applications (Travinin Bliss, and Kepner, 2007), so we need to learn how to use MATLAB. Then, we need to find a proper dataset that has uniform and non-uniform images. Finding a useful dataset which is related with static data creates a real problem because of the ethics. After deciding which dataset we will work with, we tried to implement an algorithm by using common digital image processing techniques. Also, several static hand gestures segmentation algorithms were successfully implemented since it is very easy to segment hand shape from the images with only uniform background (i.e. with dark background). However, when the point comes to detecting hand shape forms on the non-uniform background (i.e., with complex background), it is such a challenging task to detect segment hand shape forms. We will start by analyzing the results obtained successfully by segmentation algorithms from the images with a uniform background. Besides, we will try to find the weak sides of the unsuccessful algorithms,

and then, find the way we should follow for clear results. Then, we will continue with thresholding. In order to obtain better segmentation results, we will use some filters which are most efficient and known, such as low-pass filter, high-pass filter, and Sobel filter, etc. After we successfully implement an algorithm that can segment hand gestures from uniform images, we are going to move to hand gesture segmentation the non-uniform images. According to the literature review, digital image processing techniques might not be enough to have success on non-uniform images. Therefore, we will use different techniques such as CNNs and object detection. Finally, end-product will give inspirations to develop an automated sign language translator by its success.

To sum up our expected problems listed below,

- Overcoming the noise problem; this problem is about having unexpected noises in binarized images (Khan, Ibraheem, and Meghanathan, 2012).
- Overcoming the background problem; this refers to the complex background where there is other objects in the scene with the hand objects and these objects might contain skin like color which would produce misclassification problem (Khan, Ibraheem, and Meghanathan, 2012).
- Overcoming the scale problem; this problem appears when the hand poses have different sizes in the gesture image (Khan, Ibraheem, and Meghanathan, 2012).
- Overcoming the variation of illumination conditions; where any change in the lighting condition affects badly on the extracted hand skin region (Khan, Ibraheem, and Meghanathan, 2012).
- Overcoming the object detection problem; YOLO has solved this problem by its advances abilities (Redmon, Divvala, Girshick, and Farhadi, 2016).

- Overcoming the segmentation problem; this problem occurs when RefineNet pre-trained models are used because they are prepared by different kind of datasets (Everingham, Gool, Williams, Winn, and Zisserman, 2010).

In the following parts, we will show how we can get rid of some of these problems.

5 Requirements

5.1 User Requirements

Functional:

- The users must use PNG formatted input images.
- The users must use letter based names for the input images.
- The users must use input images that do not include any space in the name.
- The users must use input image that are RGB.

Non-Functional:

- The users shall select their inputs by using a selection window.
- The user shall select one or multiple inputs.
- The user shall see all the outputs in the same output folder.

5.2 List of System Specifications

- GPU which can collaborate with Nvidia Cuda v9.0 or higher
- MATLAB 2018a
- Python 3.6.8

5.3 System Process Model

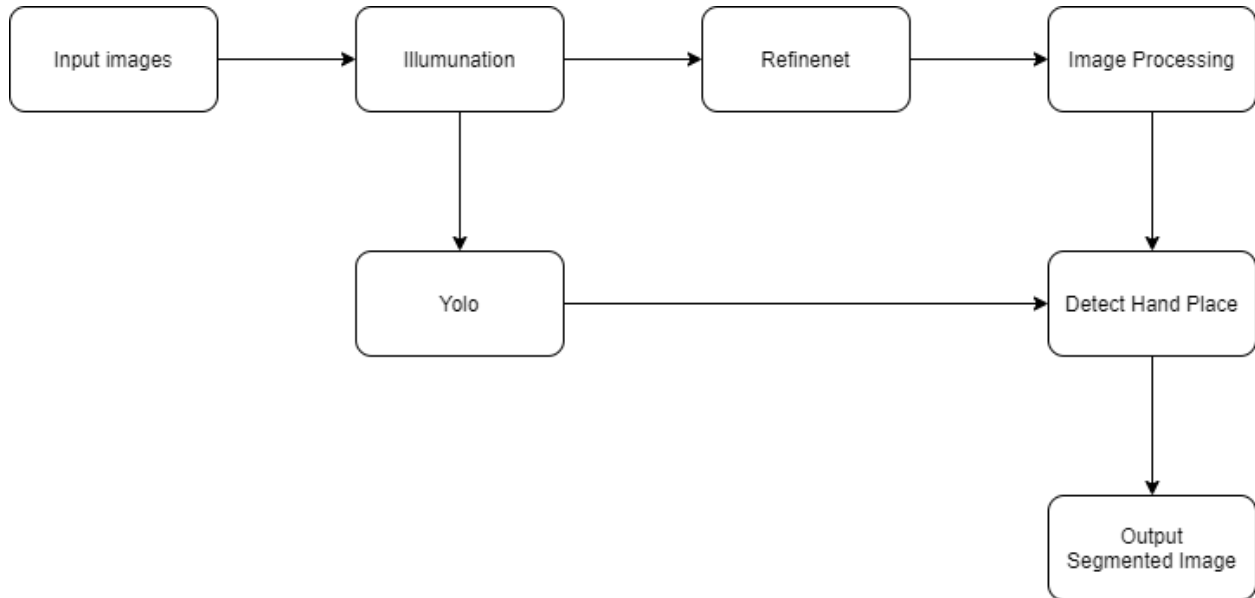


Figure 6: System Process Model

5.4 System High Level Sequence Diagram

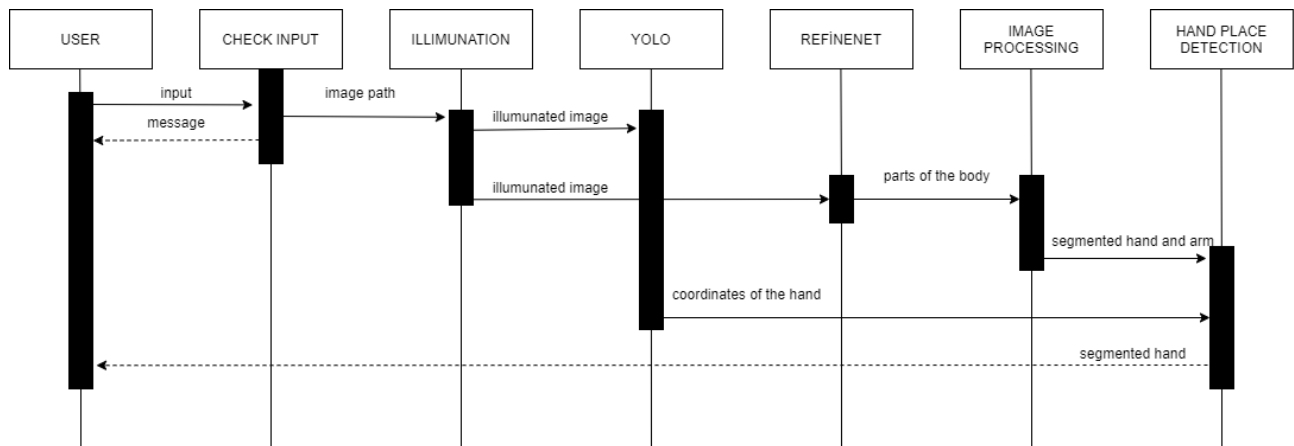


Figure 7: System High Level Sequence Diagram

5.5 Test Plan

5.5.1 Testing Strategy

As the project involves different image types to achieve its segmentation task, each major part will be tested separately with different methods based on its needs. The overall strategy is to test each simple component by unit testing, then integrating these. We will have different integration scenarios. That is why tests that are coming after this step will be done multiple times. After completing the integration test, we will move to smoke testing to test the project as a whole. Then, we will move to the performance testing which is a high order test. The success of the project is directly related to the outputs of this test.

5.5.1.1 Strategy for Image Processing methods

The testing plan for Image Processing methods involves checking if they are working correctly or not. Unit testing will be applied to all components. Results will be checked one by one manually since it will take too much time to create an autonomous system to test all the outputs.

Approximate Time: 3 days

5.5.1.2 Strategy for YOLO

The testing plan for YOLO part involves checking if detection of hand is done correctly or not. Unit testing will be applied both training and testing parts. Results will be checked one by one manually because it will take too much time to create an autonomous system to test all the outputs.

Approximate Time: 2 days

5.5.1.3 Strategy for Refinenet

The testing plan for Refinenet part involves checking if the segmentation is done successfully or not. Results will be checked one by one manually since it will take too much time to create an autonomous system to test all the outputs.

Approximate Time: 2 days

5.5.2 Test Items

5.5.2.1 Unit testing

Algorithm ID	Algorithm	Participant	Methodology
1	Illumination	Erencan Duman	Change in the RGB values and pixels will be checked. Bright ones should be darker, and, darker ones should be brighter.
2	Skin Detection	Erencan Duman	Pixels which has RGB values between thresholding intervals will be analyzed.

3	Filtering	Sezin Köktaş	Number of black pixels will be counted. It should decrease since some noises will supposed to be corrected.
4	Edge Detection	Sezin Köktaş	Number of edges will be counted.
5	YOLO train model	Vuralcan Asal	The algorithm will be tested with its' train dataset. Check the success rate is %100.
6	YOLO	Vuralcan Asal	The actual test result will be analyzed manually.
7	Refinenet	Erencan Duman	Pre trained models will be analyzed with the non-uniform dataset. Results will be analyzed manually.
8	K-means clustering	Sezin Köktaş	Results will be checked manually.

Table 1: Test order for hand gesture segmentation

5.5.2.2 Integration testing

The nature of research-based projects leads us to have different scenarios. Passed items from the unit testing will help us to provide multiple scenarios to get the best performance result. That is why we will integrate different algorithms. In this test phase, we will provide the communication of multiple integrated algorithms.

5.5.2.3 Smoke testing

This testing method will be applied to integrated algorithms that are prepared according to our scenarios. We will decide that the build is stable or not by controlling the outputs.

5.5.2.4 Performance testing

In this part, scenarios that passed smoke testing will be tested. They are all will be checked with the same test data which has a thousand images. Then outputs will be controlled manually since it is subjective. In the end, the success rate will be calculated for all scenarios.

5.5.3 Summary of features to be tested

The image processing methods, the multi-path refinement network and the YOLO are the three main components that are to be tested in this system. After unit testing of the components is completed the testing progress will move on to integration testing. This phase will test the components amongst themselves. Then, smoke testing will be applied. If it has successful results we will move to the performance testing which is high order.

The performance test is critical in deciding if the integrated algorithms are answers to the problems that the projects have. It will be concluded that which scenario needs to be optimized more or enough.

5.5.4 Summary of features not to be tested

We will not use some low priority testing methods in our project. Firstly, alpha/beta testing is not included because we do not have any customers. Also, stress testing is not necessary because we do not have any web based system. Moreover, security testing will not be conducted since our project does not need any protection. Finally, recovery testing will not be in our test plan because there is no such system that would fail the algorithms.

5.5.5 Item pass/fail criteria

Unit testing criteria for the algorithms are below. Scenarios will be considered according to passed items from the unit testing. These items will be integrated in line with scenarios. Then smoke and performance testing will be applied. If a scenario will have problems in the smoke testing, it will be eliminated. We are planning to have success rate for all the different scenarios after performance testing. Higher ones will be the used for the project.

5.5.6 Unit testing criteria for Image Processing Algorithms

We will run the algorithms function with number of images. We will check outputs by eyes since success is subjective. If the success rate is below %85, this unit will not be included in the scenarios.

5.5.7 Unit testing criteria for YOLO

We have two different data for the YOLO. They are called train and test data. Trained model will be tested with the train data. Success rate is supposed to be %100 with the train data. This will also give the information that algorithm is working correctly.

5.5.8 Unit testing criteria for RefineNet

Some samples will be selected from the non-uniform dataset. Each pre-trained models will be analyzed if they are working as they are expected or not.

5.5.9 Test deliverables

1. Test plan document
2. Test cases
3. Test design specifications

5.5.10 Test schedule

Milestone	Required Time (days)
Unit testing	5-7
Integration testing	3-5
Smoke testing	2-3
Performance testing	7-8

Table 2: Test schedule

After the start point, testing phase is expected to take two weeks. It will not take more than two weeks because all tests can be done simultaneously.

In the first week, unit and integration testing will be finished. The following week will be spent on smoke and performance testing.

6 Research Method, Design Details

6.1 Research Method

In the light of the presented literature review, it is clear that image processing, computer vision and deep learning techniques are essential in order to implement a successfully working segmentation algorithms. Therefore, our project focuses on these areas mostly. As mentioned before, the most common problems are illumination, object detection and segmentation algorithms.

To get rid of illumination problems, image processing techniques are used. Input data converted to the RGB images. In addition, mean values of the R, G and B channels are calculated. Moreover, mean of the R and B channels are changed according to the mean of G channel. This approach might not solve all the illumination problems because if the input image is much shiny, there is no way to understand the real image. In this kind of situations, every color gets closer to the white. The other image processing technique is the morphological operations. It is used to get smoother output images.

Moreover, only computer vision techniques can be enough to have good segmentation results. CNNs are widely used in the semantic segmentation. As we mentioned before, RefineNet is one of the best CNN in the literature. The main problem of the RefineNet is the training part. It needs very high-level computer systems. In our current circumstances, it is not possible to create our own model with RefineNet. However, there are some pre-trained models for different purposes in the literature. In this project, these models will be used. Segmented hand gesture figures are expected as a result.

Object detection is another method that is widely used in semantic segmentation. Finding the coordinates of the object that needs to be segmented can be rescuer in some cases. In the literature, YOLO is one of the most common object detection algorithm. Because of the nature of deep learning, training is also needed for YOLO. Unlike RefineNet, it needs lower computer systems to train your own data. Therefore, it is possible to train our own dataset with YOLO. At the end, finding the coordinates of the hand is expected.

6.2 Design Details

6.2.1 List of Functions and Definitions

Input Function:

We design input function as it shows a file selection window to user to pick several data as input. Then all the other implementations will be executed according to the data.

Output Function:

Output function saves output data to a static folder to prevent folder mess.

Illumination Function:

Taken data is initially in RGB color format as default. In this function, we separate channels of RGB model. Then we remove the illumination from the image channels.

YOLO:

The coordinates of hand gesture of the input images are detected and saved into an excel file by using our own train model.

RefineNet:

The pre-trained models are used to segment hand from the input images. Additional segmentation methods are also used. In the end, segmented hand gesture is expected as output.

Image Processing Function:

The input of this function is supposed to be segmented. Morphological operations are done in this function. Output of this function is expected to be smoother and fully filled.

Detect Hand Place Function:

In this function, the excel file contained from YOLO is used to detect the place of the hand in the segmented images. It is expected to collect cropped images that has only segmented hand gesture figures.

6.2.2 List of Special Algorithms

- Illumination Algorithm
- YOLO
- RefineNet
- Image Processing Algorithm
- Detect Hand Place

6.2.3 Process Diagram

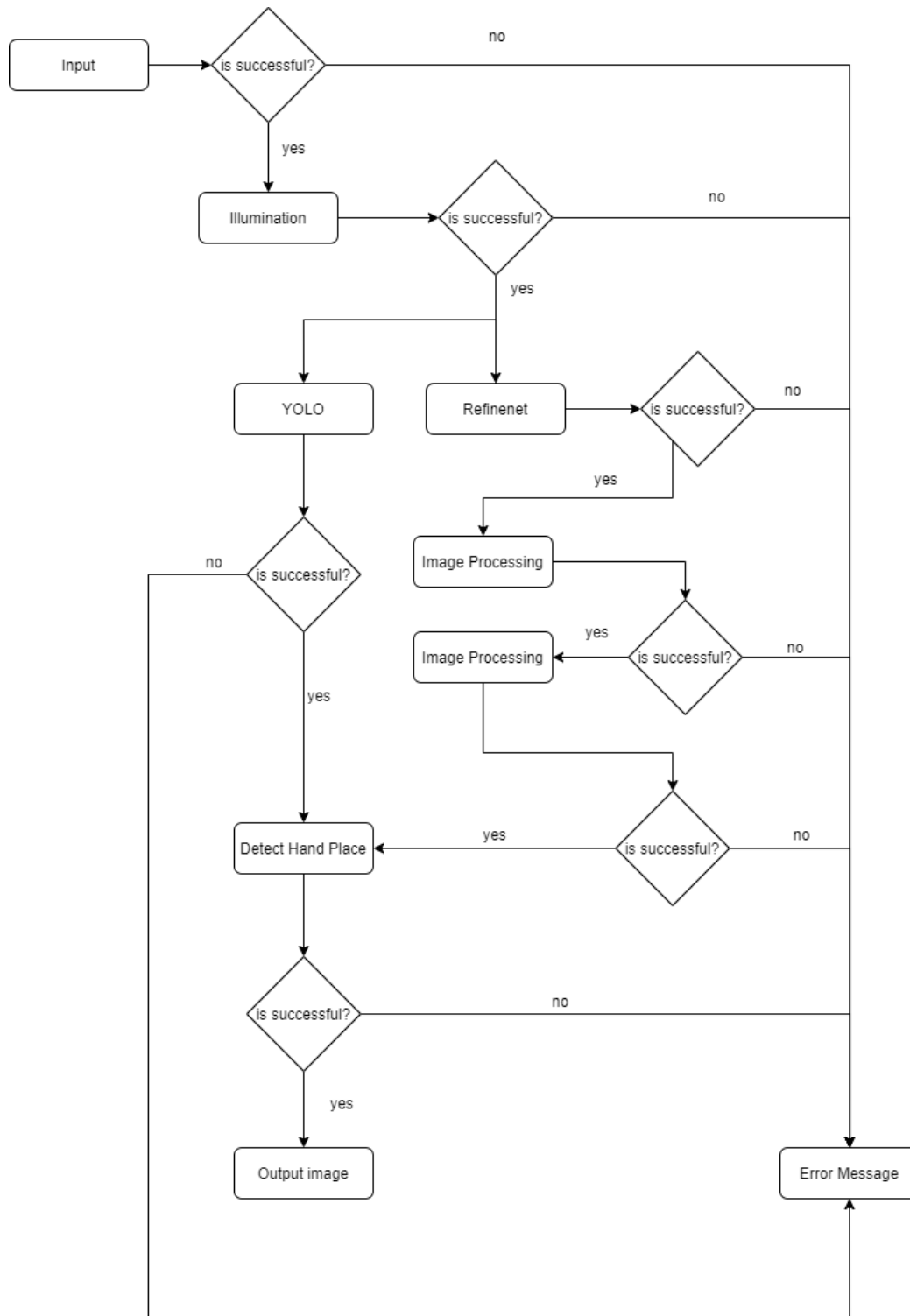


Figure 8: Process Diagram

7 Implementation details

7.1 Project Management

7.1.1 Estimation

Function Points

Program Characteristic	Low Complexity	Medium Complexity	High Complexity
Number of Inputs	1x3	1x4	2x6
Number of Outputs	2x4	0x5	0x7
Inquiries	2x3	2x4	2x6
Logic internal files	3x7	1x10	0x15
External interface files	0x5	0x7	1x10

Table 3: Function Points

General System Characteristic (GSC)

General System Characteristic	Degree of influence (DI) 0 (least) – 5 (most)
Data communications	4
Distributed data processing	4
Performance	3
Heavily used configuration	2
Transaction rate	3
On-Line data entry	0
End-user efficiency	2
On-Line update	1
Complex processing	1
Reusability	5
Installation ease	1
Operational ease	0

Multiple sites	0
Facilitate change	1
TDI=29	

Table 4: General System Characteristic

Influence multiplier or VAF= $27*0.01+0.65 = 0.92$

Function Point Estimation

Unadjusted total of function points	92
Influence multiplier	0.92
Adjusted total of function point	86.48

Table 5: Function points estimation

$ATFP = UTFP * IM = 92*0.94 = 86.48$

7.1.1.1 COCOMO

We used MATLAB. Its language unit size is 40, 55 and 80.

With minimum unit size, $LOC = LOC=ATFP * Language\ unit\ size = 86.48*40 =3459$

With mode unit size, $LOC=ATFP * Language\ unit\ size = 86.48*55 =4756$

With maximum unit size, $LOC = LOC=ATFP * Language\ unit\ size = 86.48*80 =6918$

$KDSII = LOC/1000 = 3459/1000=3.459$

$$KDSI2 = LOC/1000 = 4756/1000=4.756$$

$$KDSI3 = LOC/1000 = 6918/1000=6.918$$

We choose COCOMO semi-detached. In semi-detached, $a=3.0$, $b=1.12$, $c=0.35$.

With minimum unit size;

$$MM1=a*(KDSI1^b) = 12.04$$

$$TDEV1=2.5*(MM1^c) = 5.97 \text{ months}$$

With mode unit size;

$$MM2=a*(KDSI2^b) = 17.18$$

$$TDEV2=2.5*(MM2^c) = 6.76 \text{ months}$$

With maximum unit size;

$$MM3=a*(KDSI3^b) = 26.18$$

$$TDEV3=2.5*(MM3^c) = 7.84 \text{ months}$$

If we chose COCOMO organic; $a=2.4$, $b=1.05$, $c=0.38$.

With minimum unit size;

$$MM1=a*(KDSI1^b) = 8.83$$

$$TDEV1=2.5*(MM1^c) = 5.72 \text{ months}$$

With mode unit size;

$$MM2=a*(KDSI2^b) = 12.34$$

$$TDEV2=2.5*(MM2^c) = 6.49 \text{ months}$$

With maximum unit size;

$$MM3=a*(KDSI3^b) = 18.29$$

$$TDEV3=2.5*(MM3^c) = 7.54 \text{ months}$$

7.1.1.2 Jones's First-Order Effort

$$\text{Estimated Effort (man-month)} = [((ATFP) ^ (3*\text{Class Exponent}))/27]$$

An average system requires 0.45 as class exponent.

$$((86.48) ^ (3*0.45))/27 = 15.25711449 \text{ man-month}$$

$$\text{Schedule in months} = [3*(\text{man-months} ^ (1/3))]$$

$$3*(15.25711449 ^ (1/3)) = 7.440670219 \text{ months.}$$

7.1.2 Tasks

Project Tasks	Start dates - Durations
(T1) Research about computer vision methods	01.02.2020 –2 weeks
(T2) Research about deep learning methods	01.02.2020 – 2 weeks
(T3) Research about K-means clustering in segmentation area	01.02.2020 –2 weeks

(T4) Research about CNNs	15.02.2020 – 3 weeks
(T5) Research about YOLO	15.02.2020 – 3 weeks
(T6) Implementation of K-means clustering	15.02.2020 – 4 weeks
(T7) Implementation of YOLO	07.03.2020 – 4 weeks
(T8) Implementation of RefineNet	07.03.2020 – 4 weeks
(T9) Testing	07.04.2020 – 6 weeks
(T10) Improvement	21.05.2010 – 2 weeks

Table 6: List of the project tasks and their durations

7.1.3 Milestones

- Implementation RefineNet
- Implementation YOLO
- Implementation of K-means clustering
- Integration
- Improvement

7.1.4 Activity Diagram

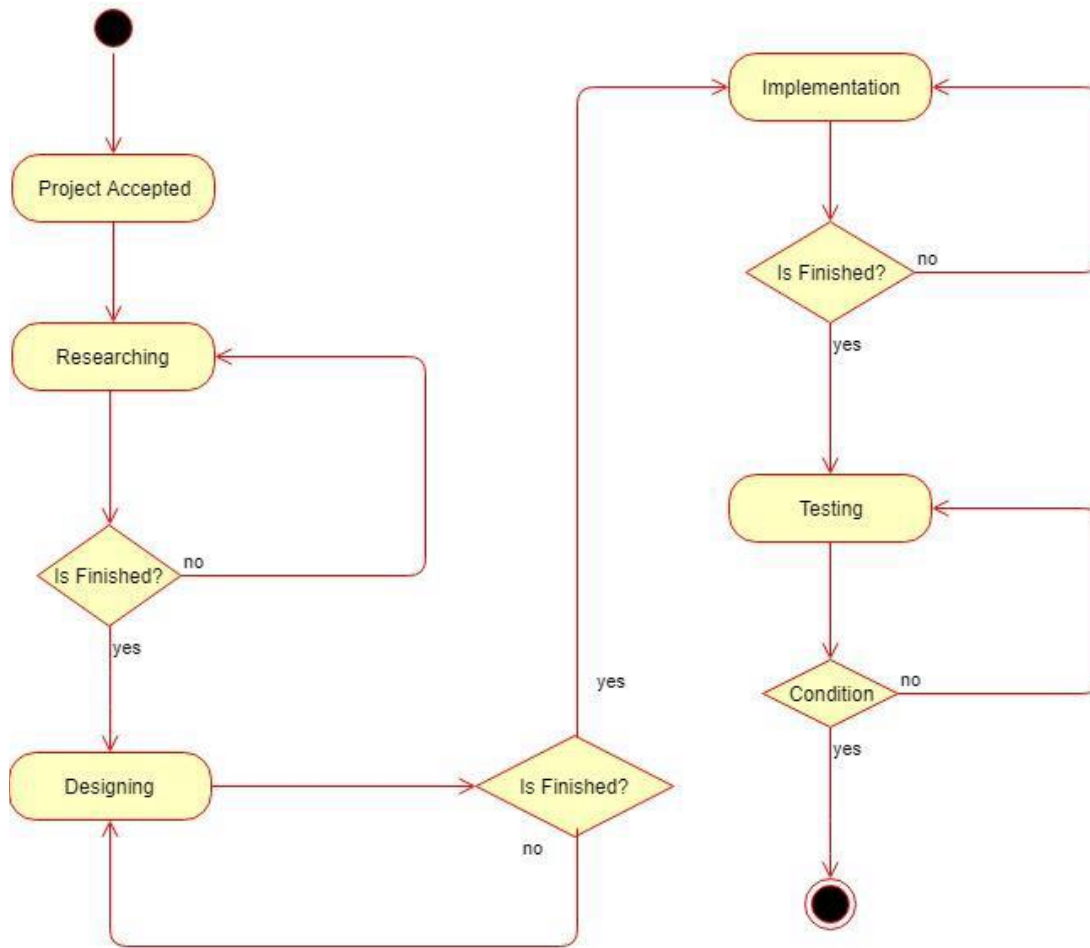


Table 7: Activity Diagram

The above diagram is showing our main activity algorithm and plans. Our plan is mainly if a step didn't fully finish, then do not pass the next step.

The above diagram is showing our main activity algorithm and plans. Our plan is mainly if a step didn't fully finish, then do not pass the next step.

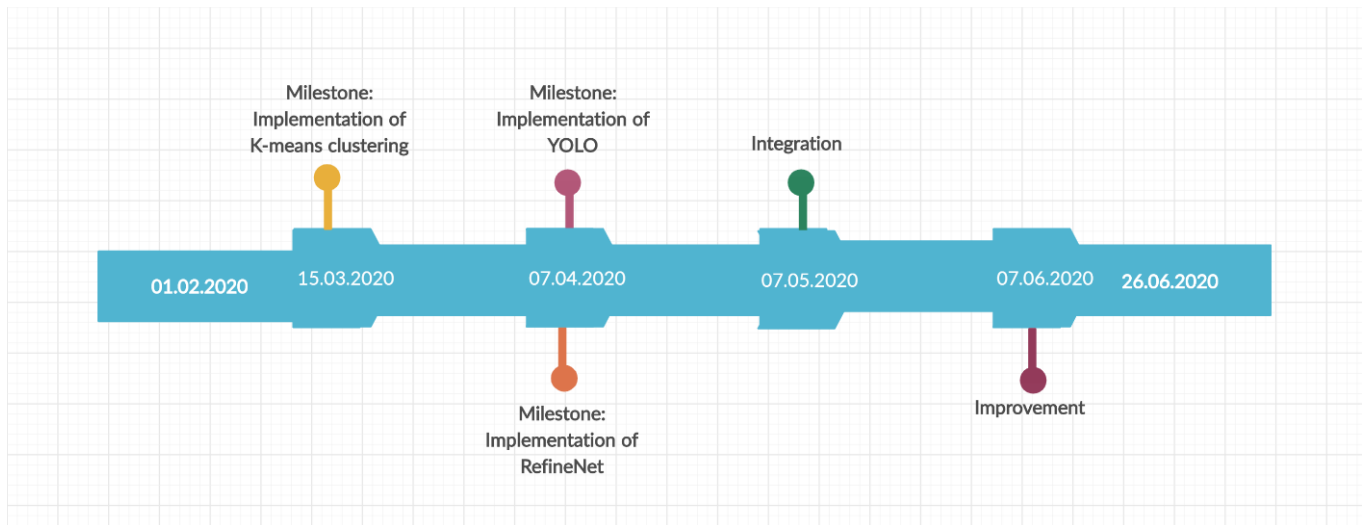


Table 8: Activity Timeline

7.1.5 Gantt Chart

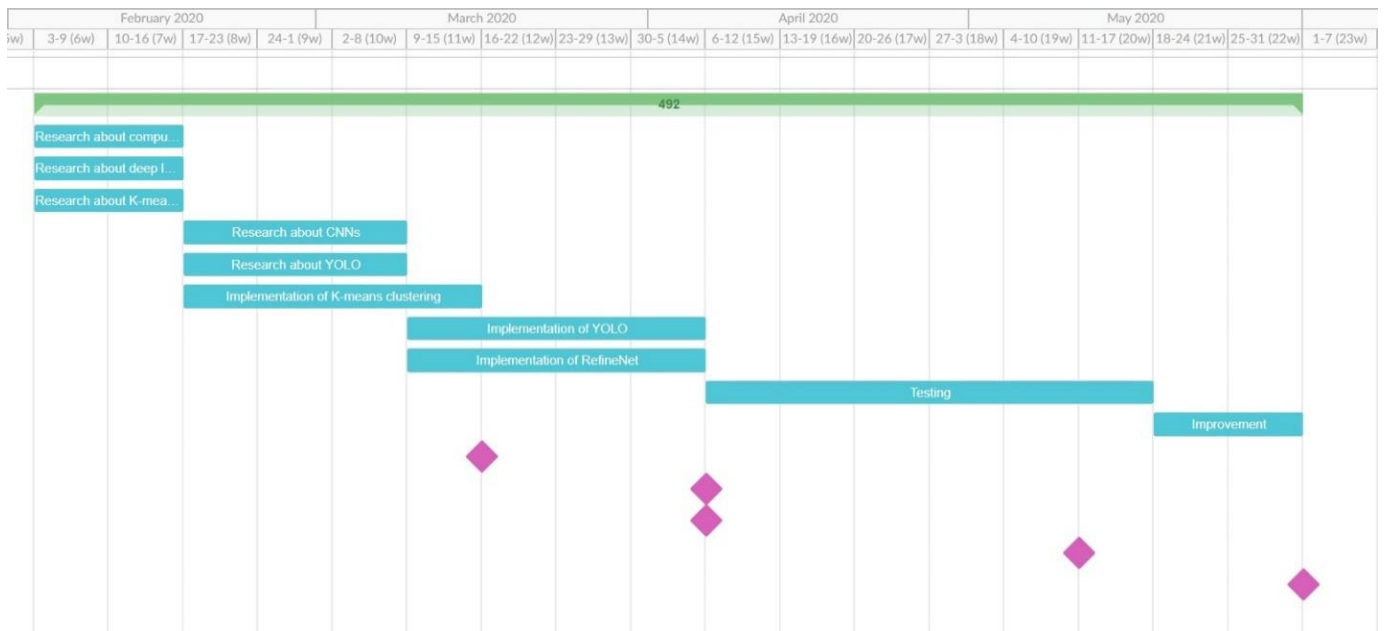


Table 9: Gantt chart

7.1.6 Task Allocation

Hand Gesture Segmentation					
Task name	Assigned	Start date	Duration (week)		
Literature Research		30.09.2019 09:00	9w		
492		03.02.2020 09:00	17w		
Research about computer vision m...	Eren Can Duman	03.02.2020 09:00	2w		
Research about deep learning met...	Vuralcan Asal	03.02.2020 09:00	2w		
Research about K-means clusterin...	Sezin Köktaş	03.02.2020 09:00	2w		
Research about CNNs	Eren Can Duman	17.02.2020 09:00	3w		
Research about YOLO	Vuralcan Asal	17.02.2020 09:00	3w		
Implementation of K-means clustering	Sezin Köktaş	17.02.2020 09:00	4w		
Implementation of YOLO	Vuralcan Asal	09.03.2020 09:00	4w		
Implementation of RefineNet	Eren Can Duman	09.03.2020 09:00	4w		
Testing	Whole Team	06.04.2020 09:00	6w		
Improvement	Whole Team	18.05.2020 09:00	2w		
Implementation of K-means clustering	unassigned	16.03.2020 09:00			
Implementation RefineNet	unassigned	06.04.2020 09:00			
Implementation YOLO	unassigned	06.04.2020 09:00			
Integration	unassigned	11.05.2020 09:00			
Improvement	unassigned	01.06.2020 09:00			

Table 10: Task allocation

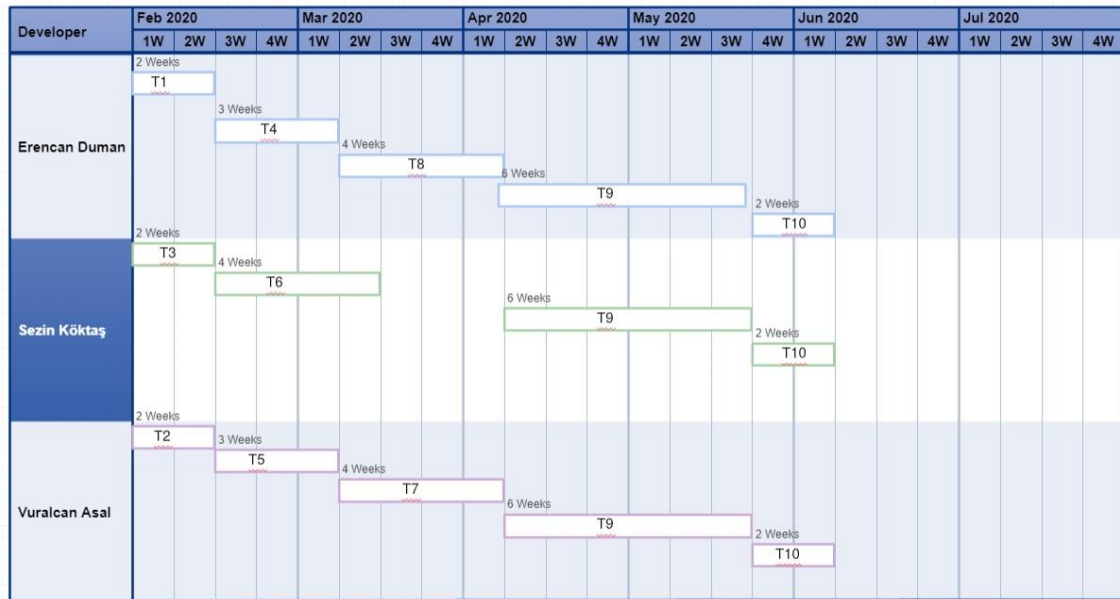


Table 11: Staff allocation

7.2 Testing

In the test part of the project, unit testing, integration testing, smoke testing, and performance testing models were used. All functions and methods were tested one by one in the unit testing. After that, a platform is created to integrate the functions and methods. For the integration part, all functions and methods are tried to integrate, and it worked. After seeing that the integration platform works correctly, different scenarios were created. In the smoke testing, these different scenarios were tried to run, and take notes their results. For the performance testing, the results which were taken from smoke testing were evaluated. For the project, the most success rate of the output result is satisfied. To decide the correct outcome, a small code created to compare inputs and outputs, so deciding that the output is correct or not is more easy and safe thanks to the small code.

7.2.1 Unit Test

The Unit Name: Illumination function

Participants: Erencaan Duman

Pass/Fail criteria: In this function, it is expected that mean values of R&B channels of the image should be changed according to the mean of Green channel (Arandjelović, and Cipolla, 2009). We can compare the R&B channel's mean value of input and output image.

Results: We tested this function with all test images. We calculated all the R&B mean values of the input and output images. Then, we compared the new values with the G channel's mean.



Figure 9: Input_1 & Output_1 for Illumination

Input_1 mean of R channel: 119.9793

output_1 mean of R channel: 106.9743

Input_1 mean of G channel: 106.9908

output_1 mean of G channel: 106.9908

Input_1 mean of B channel: 107.0750

output_1 mean of B channel: 107.0750



Figure 10: Input_2 & Output_2 for Illumination

Input_2 mean of R channel: 103.3273

output_2 mean of R channel: 97.2155

Input_2 mean of G channel: 97.2031

output_2 mean of G channel: 97.2031

Input_2 mean of B channel: 97.7615

output_2 mean of B channel: 97.1584



Figure 11: Input_3 & Output_3 for Illumination

Input_3 mean of R channel: 135.0292

output_2 mean of R channel: 119.9161

Input_3 mean of G channel: 119.9343

output_2 mean of G channel: 119.9343

Input_3 mean of B channel: 125.6647

output_2 mean of b channel: 119.9289

As you can see from the values above, output images' R&B channels are closer to the G channel's mean. Thus, we can adjust all the channels of an image according to the Green channel. A code that we created gives the values. Firstly, we need to run the illumination function than store its inputs and outputs. Secondly, we can run the code with all inputs and outputs to collect means. This testing is important because we will decide either using this function in our integration scenarios or not. As a result, it gives expected outputs. Therefore, we can use this function to get rid of illumination problems.

The Unit Name: Skin Detection function

Participants: Erencan Duman

Pass/Fail criteria: In this function, it is expected that the function detects the pixels which is related with the human skin. The function works step by step. First of all, it started working by

converting RGB image to YCbCr image, and then, it takes the YCbCr image and create a kernel. While creating the kernel, it uses two columns by using the size of the YCbCr image and initializes the kernel with zero '0'. After that, it initializes the kernel with one '1' according to Cb and Cr channels. The condition of initialization with one is $(Cb > 77 \ \& \ Cb < 127 \ \& \ Cr > 133 \ \& \ Cr < 173)$ (Sakkari, Zaied, and Amar, 2012). At the end, the kernel is the output which detects the pixels related with the human skin. If the rate of successful images bigger than fails, it is acceptable.

Results: We tested the skin detection function with test dataset. We compare the outputs with the inputs by using another code. The code takes output image which is in binary format, than copies the detected pixels on to the input image. Copied pixels are red. The successful images are decided by using human eyes one by one. After the unit test of the skin detection part, it is seen that the function can detect the skin color, which means the function is acceptable. Also, you can see two successful examples in figure 12 and one fail example in figure 13 to understand clearly.



Figure 12: For Skin Detection, Successful Examples with Inputs & Outputs



Figure 13: For Skin Detection, A Fail Example with Input & Output

The Unit Name: Filtering function

Participants: Sezin Köktaş

Pass/Fail criteria: In this function, it is expected that the function removes the parts which are out of the hand. The function used after skin detection. The function takes the images which is decided the skin parts as an input. The principle of the function is that the main object must be the hand, so remove the other parts. Also, fill the last part if there is space. Therefore, the function calculates the biggest part in the image, and removes the other small part to clear the image. If the rate of successful images bigger than fails, it is acceptable.

Results: We tested the function with all outputs. We compare the outputs with the inputs by using same code with skin detection. The code works same; that means it takes the original images and the output as the inputs, and compare them. After the unit test for filtering, it shows that the function can remove irrelevant parts and fill the necessary holes, so the function is acceptable. Also, , you can see two successful examples in figure 14 and one fail example in figure 15 to understand better how we decide successful or not.

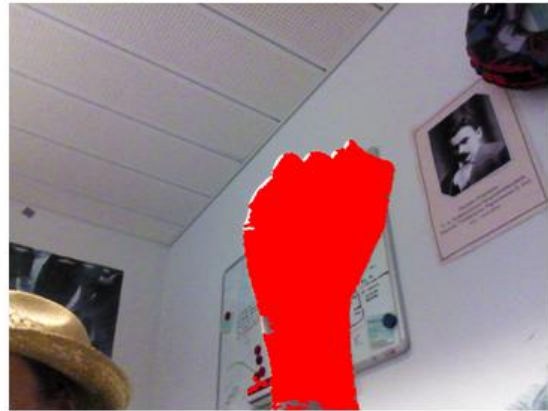


Figure 14: For Filtering, Successful Examples with Inputs & Outputs



Figure 15: For Filtering, a Fail Example with Input & Output

The Unit Name: YOLO train model

Participants: Vuralcan Asal

Pass/Fail criteria: In the unit test of YOLO training model, it is expected that the function detects the hand correctly. The function takes the inputs where the hands places are determined manually, and then learn how the hand looks like and its position. The test way for the YOLO training model is giving the input which is original training images, and detects the hand place. It is expected that almost all of the outputs have the correct place of the hand. If it is, the training model is acceptable.

Results: We tested the model with all training images. After the unit test, the function cannot find the correct place of the hand for very few images. That means, the result is almost 100% correct, so the training model is acceptable. Also, figure 16 includes two successful examples, and figure 17 shows one fail example. You can check these two figures to understand better how we decide successful or not.

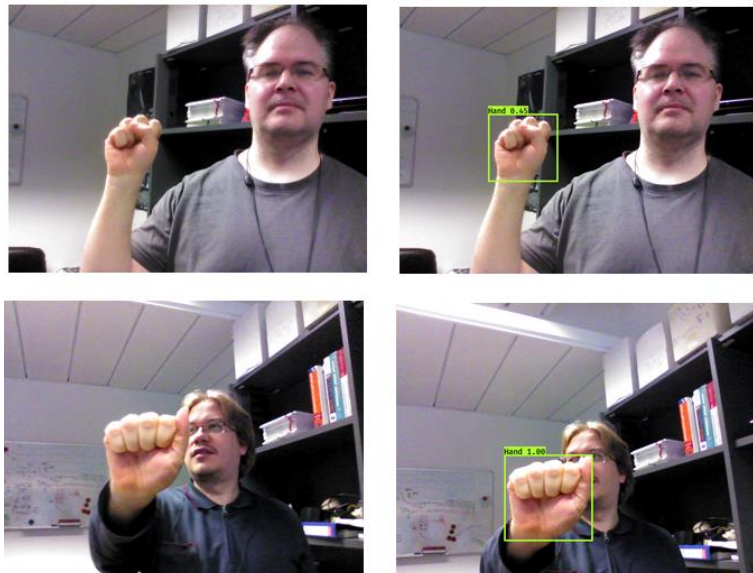


Figure 16: For YOLO Training, Successful Examples with Inputs & Outputs

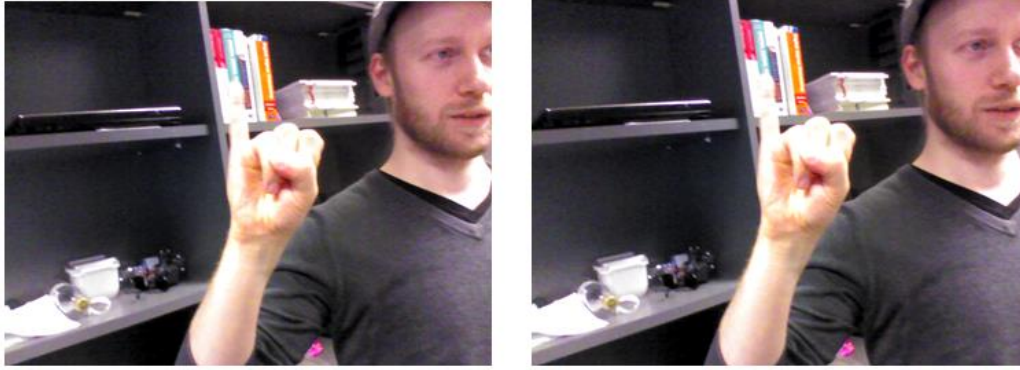


Figure 17: For YOLO Training, A Fail Example with Input & Output

The Unit Name: YOLO

Participants: Vuralcan Asal

Pass/Fail criteria: In the unit test of YOLO, it is expected that the function finds the hand correctly. The function takes the input images which is not seen before by the function. After that, it tries to find the hand's place correctly, and take notes the coordinates of the hand (Du, 2018). At the same time, it draws a square the place of the hand on the image. If the rate of the correct place has a huge part, YOLO is acceptable.

Results: In the unit test of YOLO, it is expected that the function detects the hand correctly. The function takes the input images which are not seen before by the function. After that, it tries to find the hand's place correctly, and stores the coordinates of the hand (Du, 2018). At the same time, it draws a square at the place of the hand on the image. If the rate of the correct place has a huge part, YOLO is acceptable. To see the YOLO result example, you can check figure 18 which includes one successful example, and figure 19 which includes two fail examples.



Figure 18: For YOLO Testing, A Successful Example with Input & Output

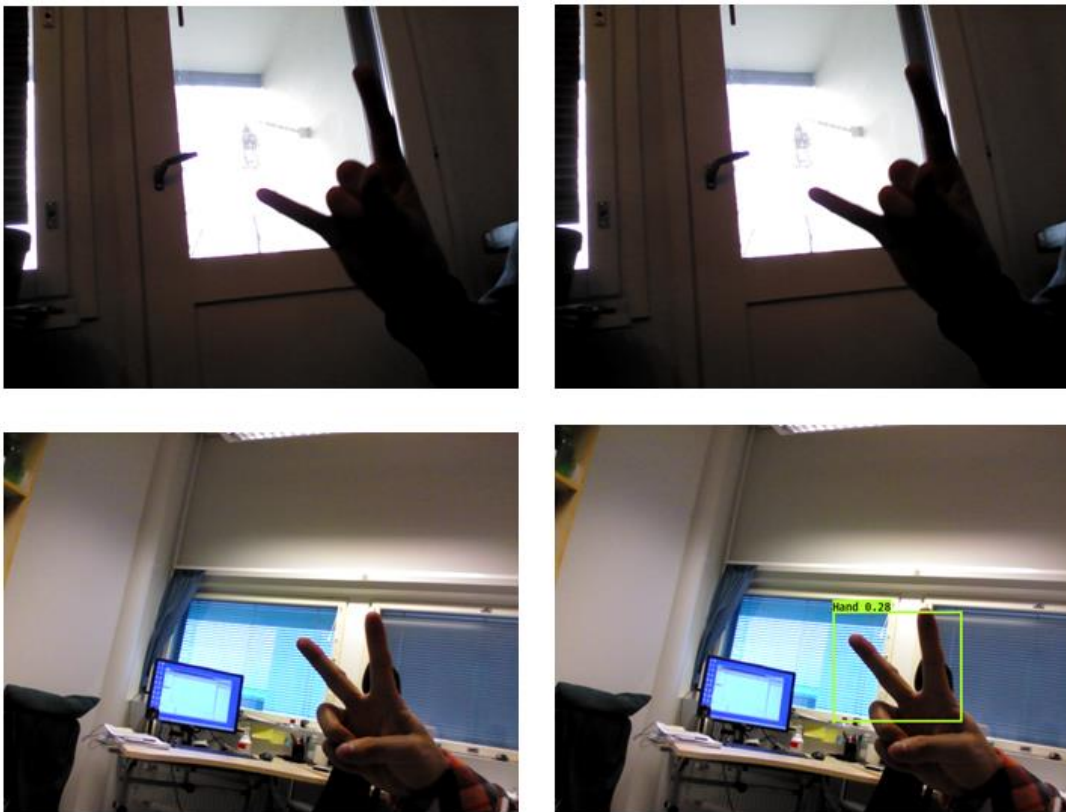


Figure 19: For YOLO Testing, Fail Examples with Inputs & Outputs

The Unit Name: K-means Clustering function

Participants: Sezin Köktaş

Pass/Fail criteria: In the unit test of K-means, it is expected that the function divides the original image into k groups or regions. The function takes the input images and the number that shows the number of clustering regions, k. After that, it segments the image into k clusters by performing k-means clustering and returns the segmented image (Kodinariya, and Makwana, 2013). If it gives an image clustered in two regions, K-means is acceptable.

Results: We tested K-means with all images. After the unit test, the function groups the original image into two regions. Also, you can see two examples in figure 20 to understand better how we decide the function is acceptable.



Figure 20: For K-means Clustering, Two Examples with Inputs & Outputs

As you see in the example. The function can divide the images into two regions.

The Unit Name: Refinenet and Pre-trained Models

Participants: Erençan Duman

Pass/Fail criteria: In the unit test of Refinenet, we have four different pre-trained models that are taken from Urooj and Borji's research in 2018. We could not prepare our train model due to lack of high GPU. Also, Everingham, Van Gool, Williams, Winn, and Zissermans' research in 2010 provides a model to separate the parts of a human body. We will run all of these models one by one with our test dataset. According to the Urooj and Borjis' research, all the models are trained to segment hand from static images (2018). If the model can segment the hand clearly, it will pass. In Everingham, Van Gool, Williams, Winn, and Zissermans' research in 2010, the body parts can be seen clearly to pass if the Pascal model is used. We will analyze all the outputs by eyes. Then, success rate of each model will be cared.

Result of Model 1: EGOHAND

To check figure 21 for successful example and figure 22 and 23 for fail example.



Figure 21: For EGOHAND Model, A Successful Example with Input & Output



Figure 22: For EGOHAND Model, Fail_1 with Input & Output

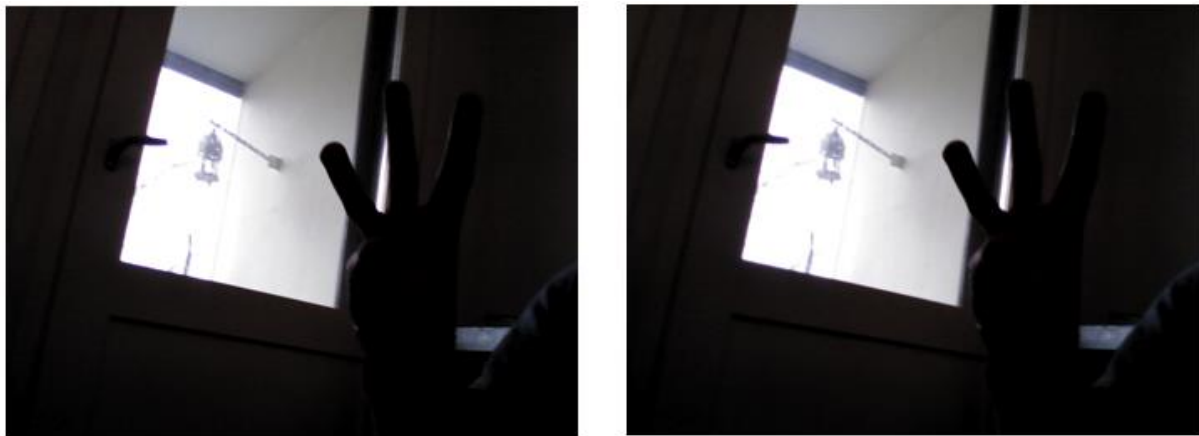


Figure 23: For EGOHAND Model, Fail_2 with Input & Output

According to the evaluation for this model, some of the images can be segmented successfully, so the model can be acceptable.

Result of Model 2: EYTH

You can see a successful example in figure 24, and two fail example in figure 25.



Figure 24: For EYTH Model, A Successful Example with Input & Output



Figure 25: For EYTH Model, Failed Examples with Inputs & Outputs

According to the evaluation for this model, some of the images can be segmented successfully, so the model can be acceptable.

Result of Model 3: GTEA

Figure 26 includes one successful example and figure 27 shows two fail examples. You can see these figures above.



Figure 26: For GTEA Model, A Successful Example with Input & Output

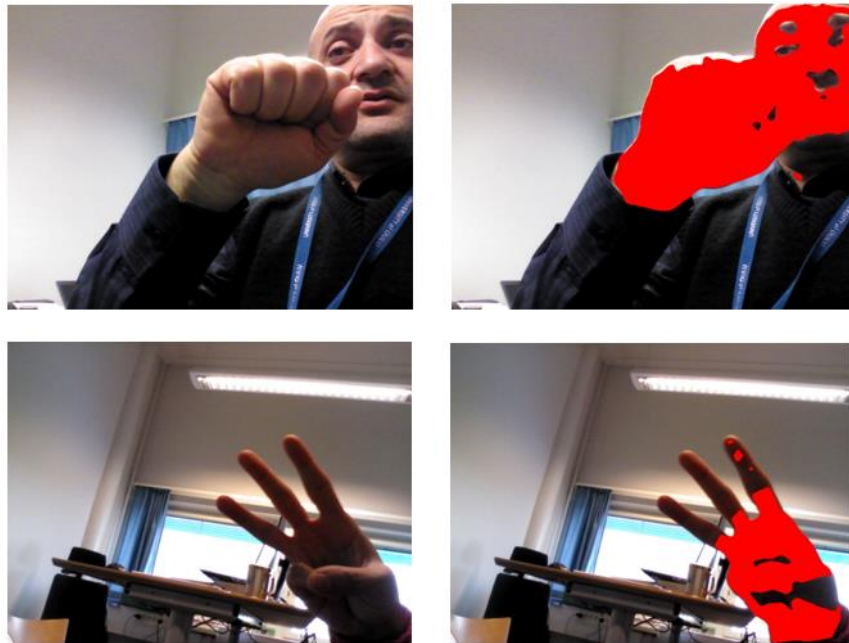


Figure 27: For GTEA Model, Failed Examples with Inputs & Outputs

According to the evaluation for this model, some of the images can be segmented successfully, so the model can be acceptable.

Result of Model 4: HOF

Figure 28 is for one successful example and figure 29 is two successful images.



Figure 28: For HOF Model, A Successful Example with Input & Output

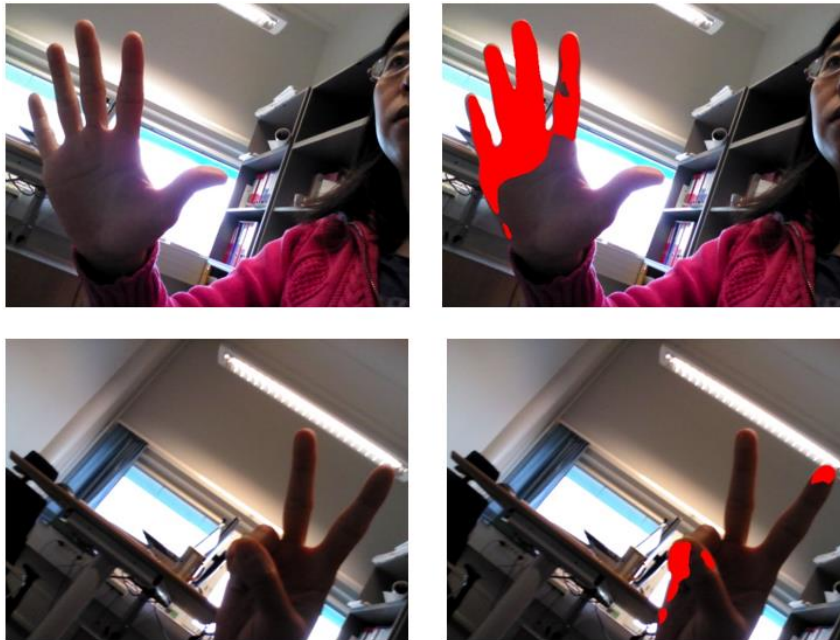


Figure 29: For HOF Model, Fail Examples with Inputs & Outputs

According to the evaluation for this model, some of the images can be segmented successfully, so the model can be acceptable.

Result of Model 5: PASCAL

For Pascal model, you can analyze figure 30 for a successful example, and figure 31 for a fail example.



Figure 30: For PASCAL Model, Successful Examples with Input & Output

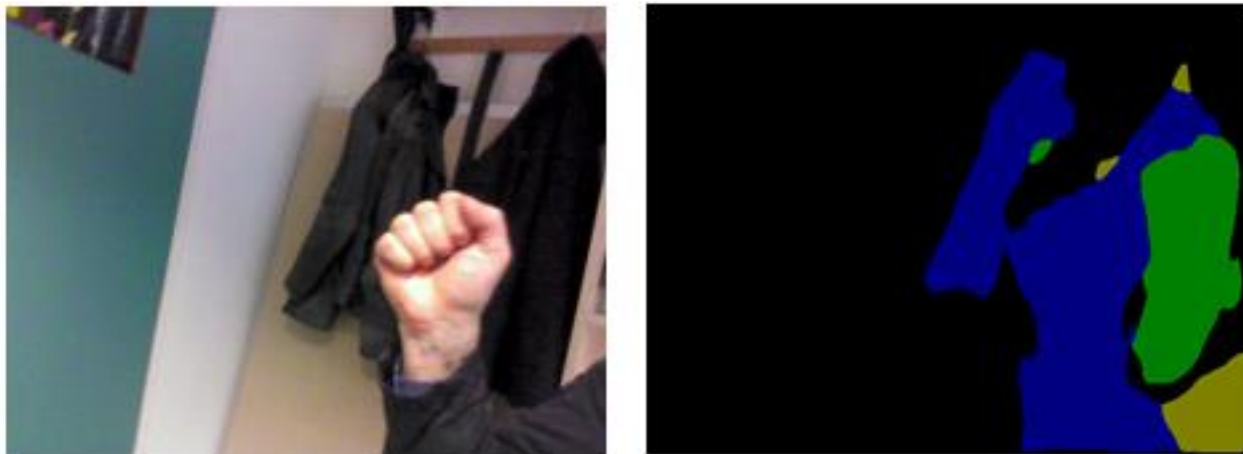


Figure 31: For PASCAL Model, Fail Examples with Inputs & Outputs

According to the evaluation for this model, some of the human bodies in images can be separated their parts successfully, so the model can be acceptable.

7.2.2 Integration Testing

Due to the nature of research-based projects, there are several different scenarios to consider. As many as possible different scenarios can be tried to get the best performance. These scenarios can be created by connecting different algorithms. Also, it should be flexible in case of applying different scenarios. At that moment, a good quality code needs a good connection and structure. In the project, two different languages are used, which are Python and MATLAB. The integration code is developed by using MATLAB because majority of functions are written in MATLAB. Thus, the program is started in MATLAB, and it manages functions without concern whether it is developed in MATLAB or Python. For the MATLAB-Python integration, MATLAB's own tools are used. "System ('command')" which is the function in MATLAB is very helpful to run Python systems. The integration between different functions that are written in different programming languages are handled successfully and clearly in our implementation. For instance, which function called, when a function works, which platform (Python or MATLAB) is used, or when the platform is changed, these types of decisions are taken in the integration code which is prepared for the project. In the integration testing part, all functions, methods and platforms are integrated themselves successfully.

7.2.3 Smoke Testing

For smoke testing, as many as possible scenarios were created. These scenarios were applied by using the integration system. The scenarios are explained step by step below.

Scenario 1:

1. Take the input from the user
2. Detect the hand by using YOLO

3. Segment the hand by using Refinenet models one by one

For the scenario, the program runs four times since there are four different models of Refinenet to test. Therefore, in that part, it can be said that four different scenarios are tested.

According to the results, the first scenario has successful connection. The input can pass all the processes in the scenario_1 even if refinement models change. There is an example in figure 32.



Figure 32: In Smoke Testing, An example For Scenario_1 with Input & Output

Scenario 2:

1. Take the input from the user
2. Detect the hand by using YOLO
3. Use the Illumination function
4. Use the Skin Detection function

According to the result, the second scenario has successful connection. It is seen that the input object can pass all the processes in the scenario_2. You can see an example in figure 33.



Figure 33: In Smoke Testing, An example For Scenario_2 with Input & Output

Scenario 3:

1. Take the input from the user
2. Detect the hand by using YOLO
3. Using the Illumination function
4. Use the Skin Detection function
5. Use the Filtering function

In figure 34, you can see an example for scenario 3.



Figure 34: In Smoke Testing, An example For Scenario_3 with Input & Output

According to the result, the second scenario has successful connection. It is seemed that the input object can pass all the processes in the scenario_3.

Scenario 4:

1. Take the input from the user
2. Using the Illumination function
3. Detect the hand by using YOLO and store the coordinates of the hand place
4. Segment and separate the human body by using Refinenet with Pascal model
4. Use the Filtering function
5. Find the hand place by using the coordinate of the hand

You can see the success of the scenario 4 in figure 35.



Figure 35: In Smoke Testing, An example For Scenario_4 with Input & Output

According to the result, the fourth scenario has successful connection. It is seen that the input object can pass all the processes in the scenario_4.

7.2.4 Performance Testing

As we mentioned in the test plan, the success rate for each scenario will be calculated by following formula.

The Success Rate Formula: Number of successfully segmented images are divided by total number of the input images.

Scenario 1 with the EGOHANDS model:

Almost %8 of the input images are segmented correctly.

Scenario 1 with the GTEA model:

Almost %25 of the input images are segmented correctly.

Scenario 1 with the EYTH model:

Almost %5 of the input images are segmented correctly.

Scenario 1 with the HOF model:

Almost %15 of the input images are segmented correctly.

Scenario 2:

This scenario has almost like %24 success rate.

Scenario 3:

This scenario has almost like %33 success rate.

Scenario 4:

This scenario has almost like %71 success rate.

The number of input images: 1000

The number of successfully segmented images: 712

The success rate = $712 / 1000 = 0.712$

In the performance testing, the best success rate that can be seen is %71. Also, the worst one has %5 success rate. As a result, at the first steps of the project, the success rate was under 5%. For now, the system could reach 71% success rate, so scenario four will be used for the project at this time.

In this part, we want to show some successful and unsuccessful outputs of scenario 4. As a result, how categorization of outputs is done, can be more understandable.

We will start with unsuccessful outputs than continue with successful ones.

As you can see in figure 36, the output is very close to being successful. This is one of the unsuccessful outputs with lower pixel loss.



Figure 36: The most successful segmented image in fail results example

Figure 37 shows one of the unsuccessful outputs with higher pixel loss.

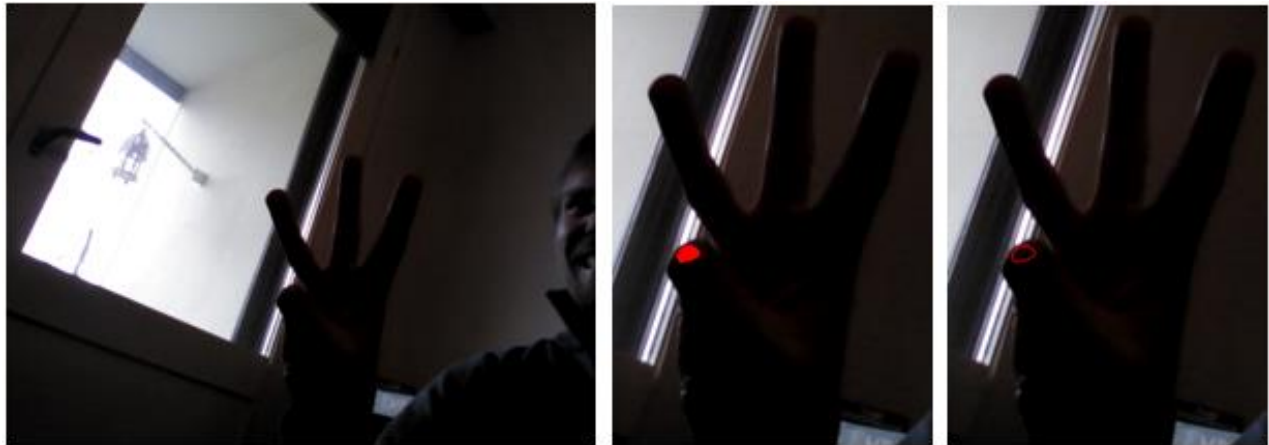


Figure 37: The worst segmented image in fail results example

Figure 38 shows one of the successful outputs with lower pixel loss.



Figure 38: The most successful segmented image in successful results example

Figure 39 shows one of the successful outputs with higher pixel loss. The reason for being counted as successful is that although pixel loss is a bit much, it is still recognizable.



Figure 39: The worst segmented image in successful results example

7.2.5 Test Requirements

Functional:

- All functions and methods must work properly
- All functions and methods must be connected as required
- The segmented outputs must be the segmentation of the hand in the image
- All outputs must be stored in a folder

- All input images must be in PNG format
- All input images must be RGB
- All input images' name must be one word

Non-Functional:

- The users shall select their input images
- The users shall choose one or multiple input images
- The users shall see the segmented images with the names which are same with the inputs
- The user shall use a selection window to select their inputs
- The segmented image shall include only the hand in the image

8 Conclusion

It has been eight months since we started the project. From the beginning until the end, we tried to see different techniques by researching the different articles to have a better segmented image as much as possible. We tried to implement the techniques that we found by searching the articles, into our algorithm, and tried to get better segmentation algorithm. In our project, researching many articles is very important in terms of the development of the algorithm and understanding the basics of the image processing, computer vision and deep learning techniques. Hence, we also took Fundamentals of Image Processing and Computer Vision courses to understand the basics of the techniques and implementation ways of them.

The first milestone is the implementation of K-means clustering. As you can see, we did not use it in the end-product because implementing this algorithm has showed that the image processing methods will not be enough to achieve our goal. There are many other algorithms that we implemented but not used. It is the nature of research-based projects. This situation made us to

change our aspect to the computer vision and object detection techniques. The second milestone is implementing YOLO. This part is a huge step for our project because detection of the hand coordinates showed that we can zoom into the hand by cropping the images according to the hand coordinates. The third milestone is the RefineNet. It showed that how CNNs are strong in the segmentation area. The hardest part of the RefineNet is that if you do not have a computer that has high-level components, it is not possible to get your own train model. Under these circumstances, we tried to find pre-trained models that would fit our project. We found 5 different pre-trained models that can be used for hand gesture segmentation. The other milestone is the integration of these algorithms. Until this point, every algorithm was working well alone. However, when we integrate them, the results were not as expected. This situation lead us to understand that we need to make some improvements which is the last milestone. The improvement was not about the implementing different algorithms from scratch. It was about changing the order of functions that we already had. The only thing that we add to the algorithm is a small piece of segmentation algorithm. This improvement has affected our success rate positively. The best success rate was %33 before the improvement. In the end, our end-product has %71 success rate.

9 Installation Manual

If you want to download **HandSegmentation** folder to your computer, you can reach it from this link; “<https://drive.google.com/drive/folders/1miXRzf4ZJA4uy-hl7jqB9ouLKw4rXrCC>”

In order to, use this project, there are some necessary softwares:

MATLAB 2018a: The best option for this project was MATLAB 2018a because the older version as not as useful like 2018a version.

Python 3.6.8: This project was prepared with Python 3.6.8. The other version were not useful for our project to get reliable and valid results.

Cuda 9.0: In this project, NVIDIA Quadro K620 was used as graphic card. Also, the project needs to use Cuda 9.0 or above because the older versions of Cuda not able to handle our project.

*If you want to reach the Python 3.6.8 and Cuda 9.0 setup files, you can find them in **user_requirements** file.

Download MATLAB 2018a:

You need a MATLAB account to enter the MATLAB world, but MATLAB is a paid software. After you create your account, you can download the setup file and the product key. 2014 and later versions are installed in the same way. If you have all the required files to install and you have a file installation key, you can follow the steps.

Choose “Use a File Installation Key” and click Next button.

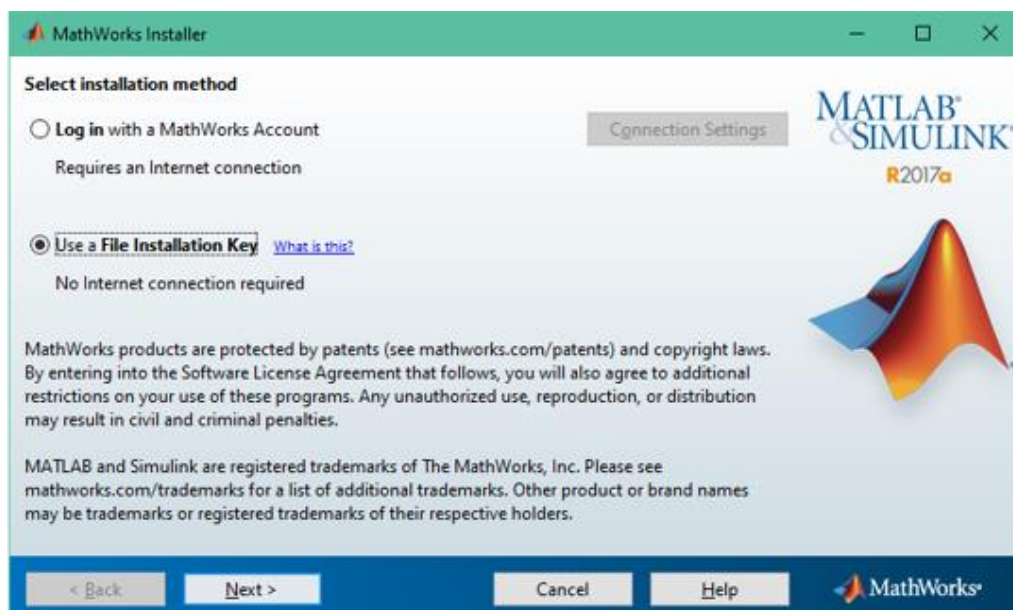


Figure 40: Download MATLAB Step1

Accept license agreement and click **Next** button.

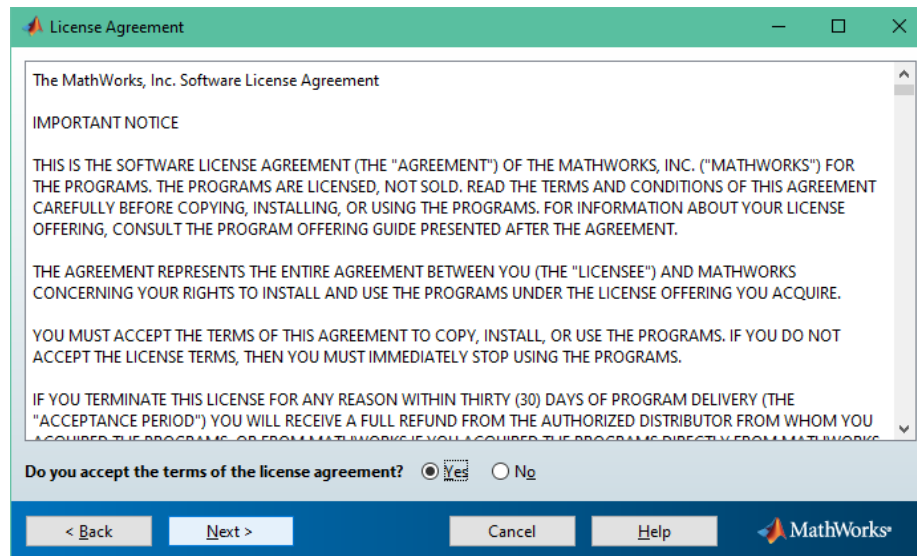


Figure 41: Download MATLAB Step2

Choose “**I have the File Installation Key for my license**”, enter **your installation key** and click **Next** button.

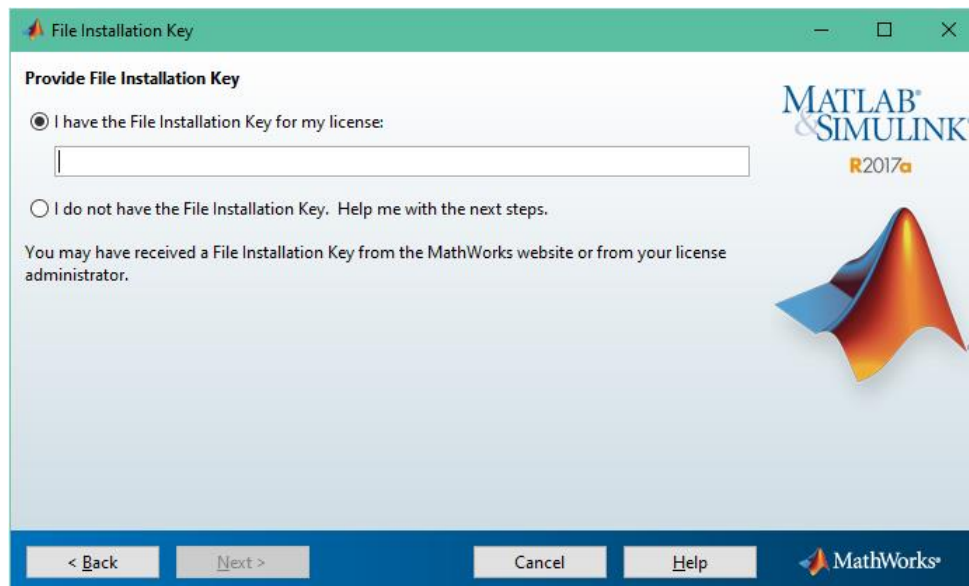


Figure 42: Download MATLAB Step3

Choose your installation folder and click **Next** button.

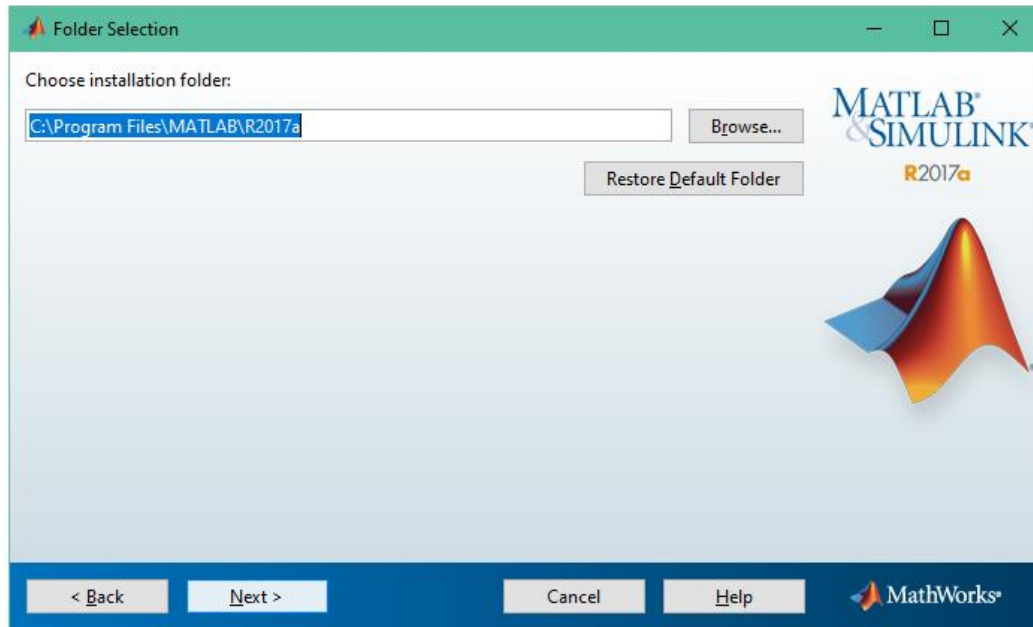


Figure 43: Download MATLAB Step4

Click **Next** button without doing any changes.

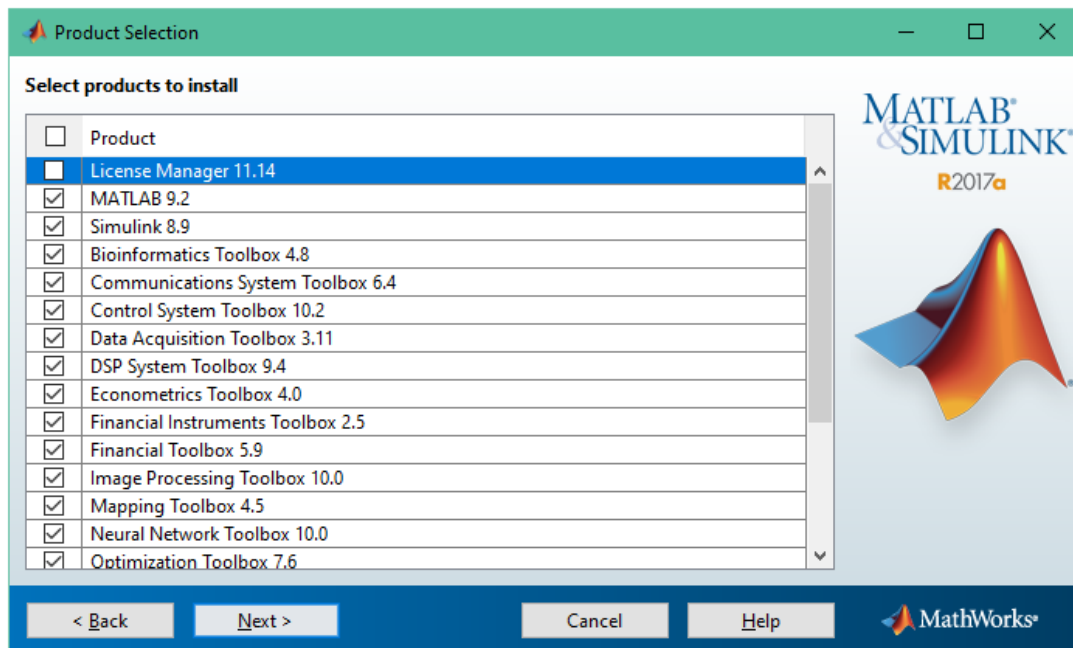


Figure 44: Download MATLAB Step5

Enter full path the “**license.dat**” file and click **Next** button.

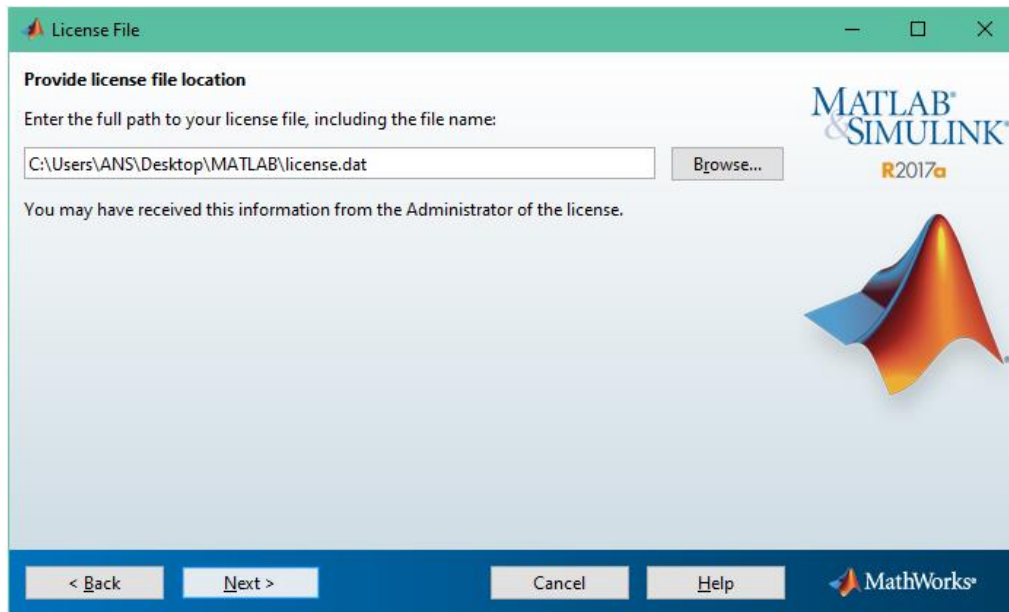


Figure 45: Download MATLAB Step6

Click **Next** button.

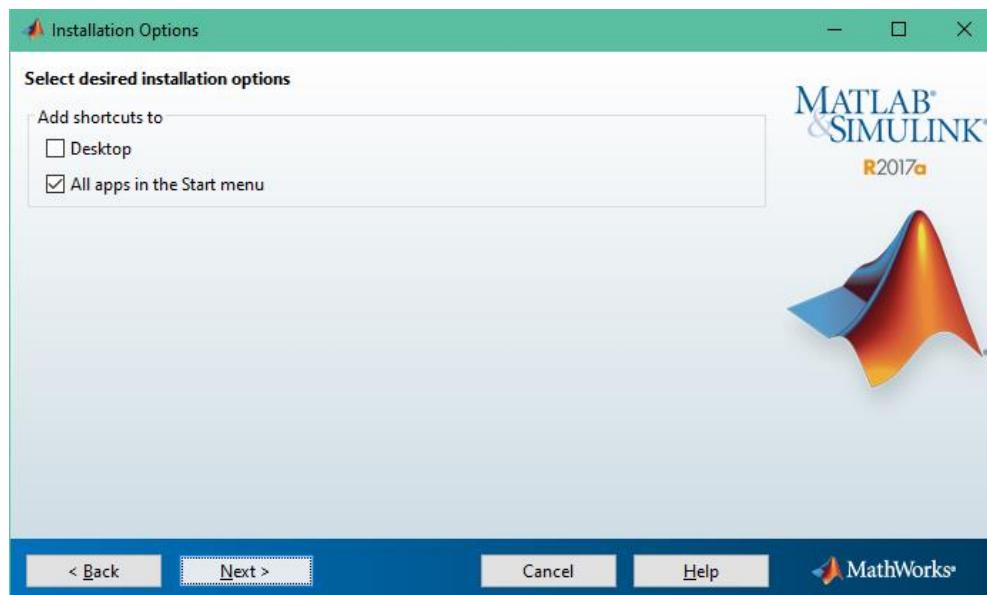


Figure 46: Download MATLAB Step7

Click **Install** button to start installation.

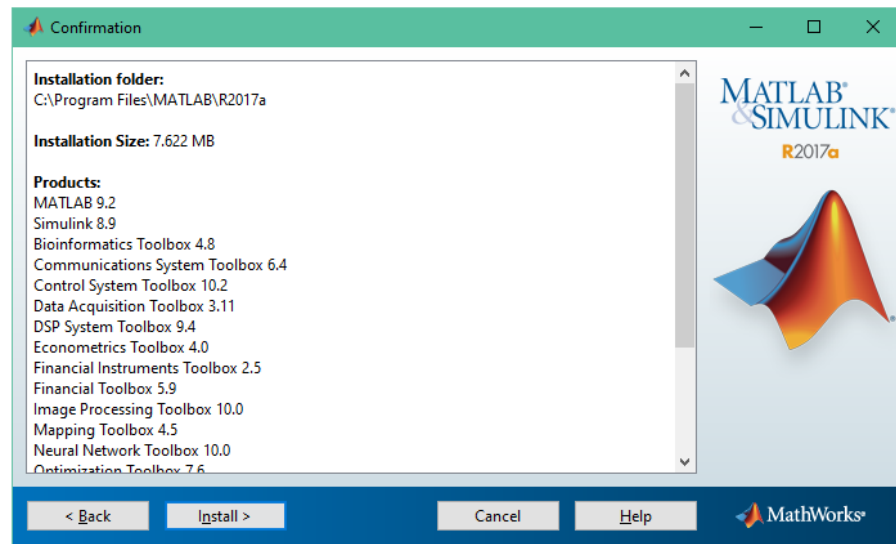


Figure 47: Download MATLAB Step8

Wait until the installation is completed. After the installation, click the **finish** button.

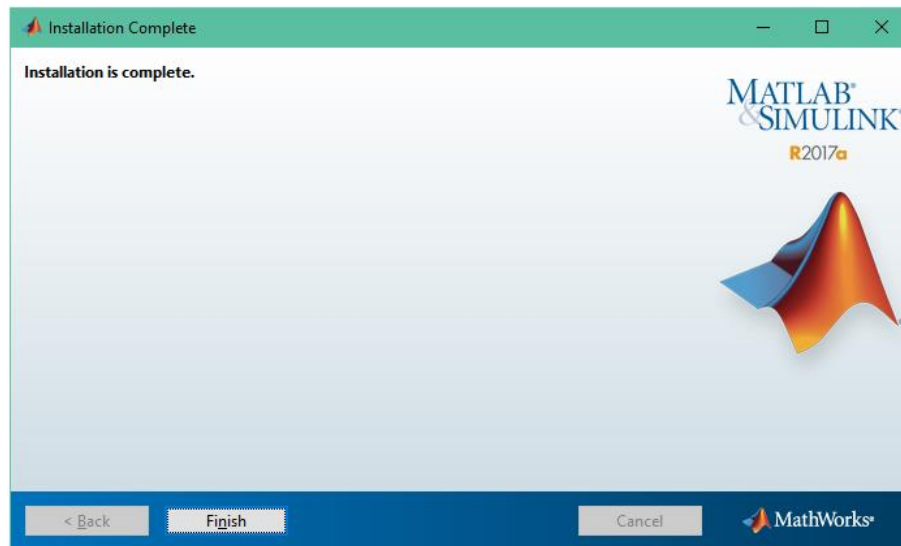


Figure 48: Download MATLAB Step9

Other option: If you do not have file installation key, you can follow the step which is in the link:

<https://www.mathworks.com/videos/how-to-install-matlab-1525083586145.html>

Download Python 3.6.8:

Python is a free software. You can download it from its own web page;

“<https://www.python.org/downloads/>“.

You need to find python 3.6.8. After you have the setup file you can follow the steps

Run the setup file, select “Add Python 3.6 to PATH” and click install now.

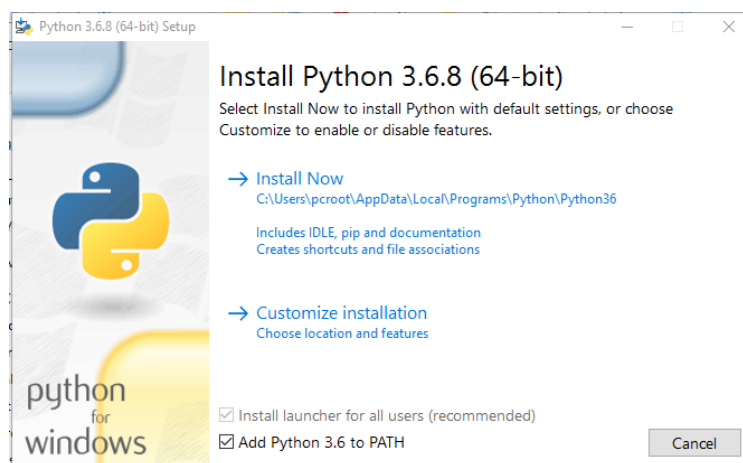


Figure 49: Download Python Step1

Click the **close** button.

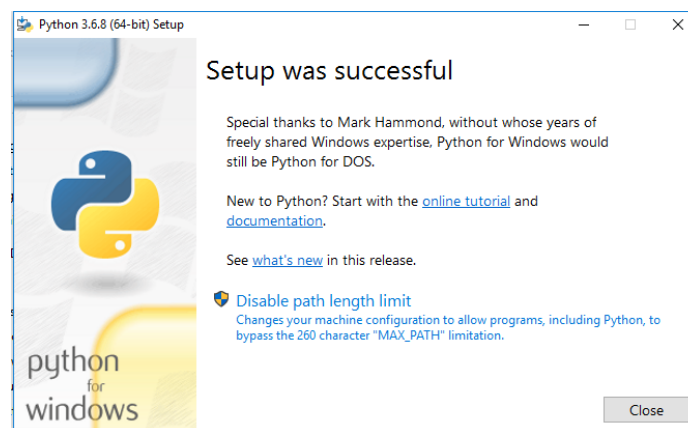


Figure 50: Figure 15: Download Python Step2

Download Cuda 9.0:

The important part is that you need to find a compatible Cuda version for your graphic card. You can see the details about Cuda from the web site:

“<https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html>”

You can download the setup file of the Cuda toolkit from:

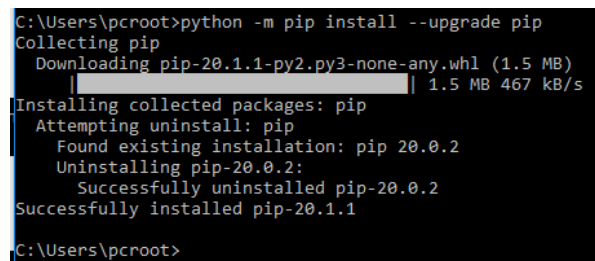
“<https://developer.nvidia.com/cuda-toolkit-archive>”

Download your Cuda version and follow its instructions.

Prepare the project to use:

Open Command Prompt (CMD)

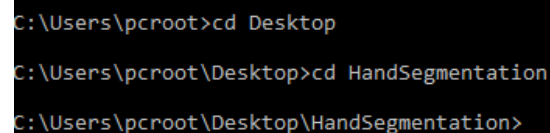
Upgrade the pip version by typing “**python -m pip install --upgrade pip**”.



```
C:\Users\pcroot>python -m pip install --upgrade pip
Collecting pip
  Downloading pip-20.1.1-py2.py3-none-any.whl (1.5 MB)
    |#####| 1.5 MB 467 kB/s
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 20.0.2
    Uninstalling pip-20.0.2:
      Successfully uninstalled pip-20.0.2
  Successfully installed pip-20.1.1
C:\Users\pcroot>
```

Figure 51: Update PIP

Go to the inside of the project file.



```
C:\Users\pcroot>cd Desktop
C:\Users\pcroot\Desktop>cd HandSegmentation
C:\Users\pcroot\Desktop\HandSegmentation>
```

Figure 52: Go to the Folder by Using cd

Install the requirement files by typing: “**pip install -r requirements.txt**”.

```
C:\Users\pcroot\Desktop\HandSegmentation>pip install -r requirements.txt
```

Figure 53: PIP Installation

Wait until all requirements are installed and close the CMD window.

Now you complete installation part, and the project is ready to segment your hand.

10 User Manual

Using the software is very easy. Follow the steps:

Important: Your input images must be “png” format, and you do not should have any white space in input names.

Example: **example_input.png**

Go to the project folder and open the **HandSegmentation.h** by using MATLAB file.

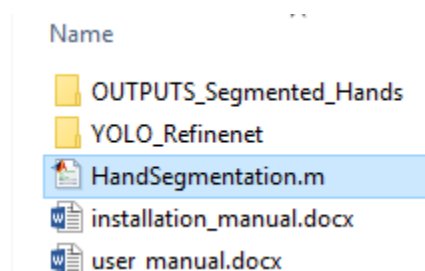


Figure 54: How to Work the Software, Step1

To run the project, click the **run** button.

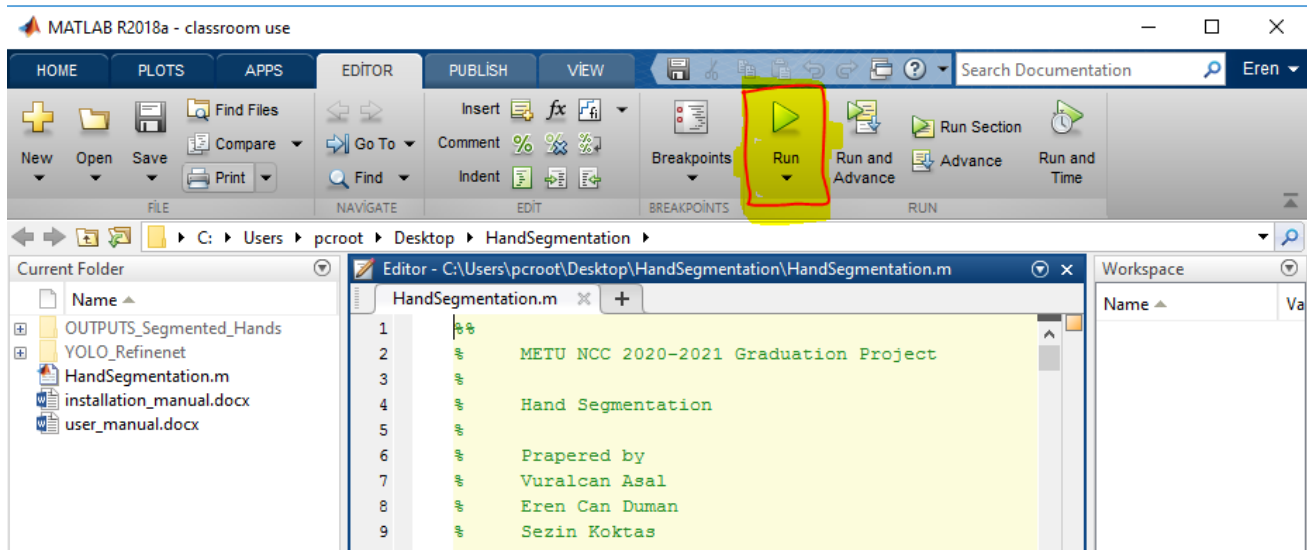


Figure 55: How to Work the Software, Step2

After opening the **select files window**, go to your images file and select the images you want to segment. You can select **one or multiple** images. Then, **click** the open button.

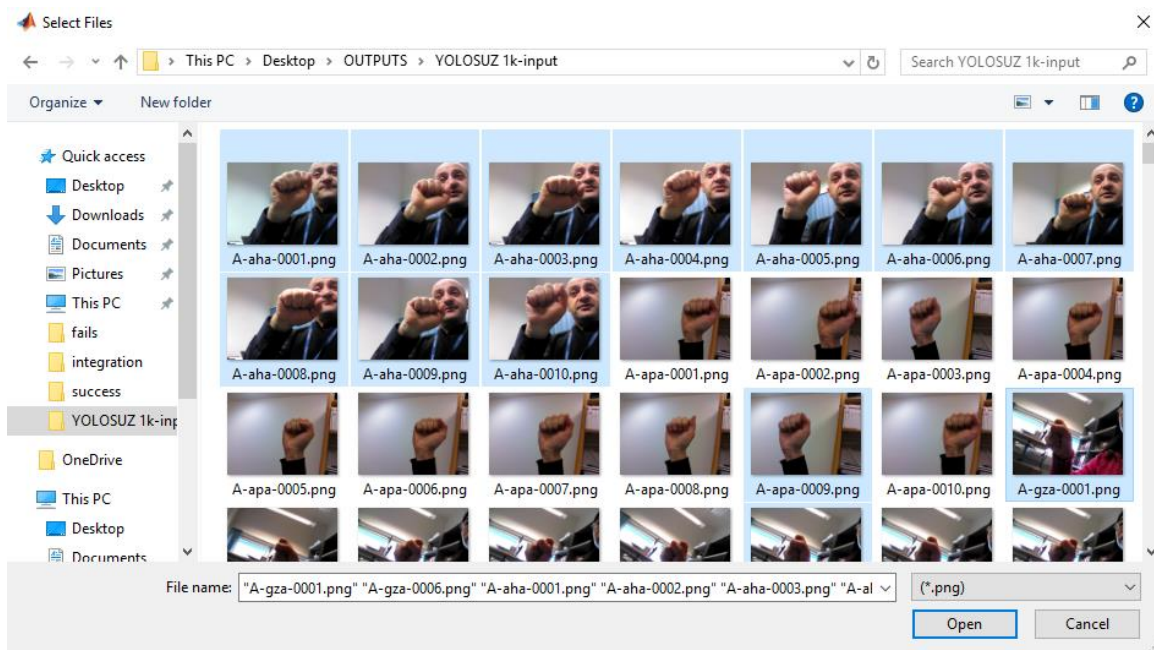
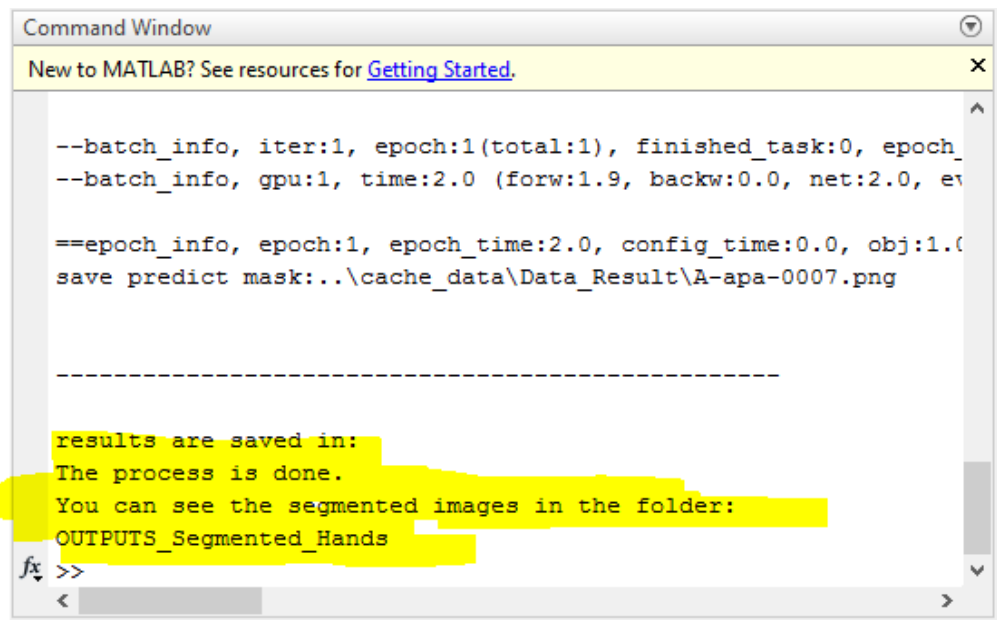


Figure 56: How to Work the Software, Step3

Wait until the process is finished. It takes a bit time, do not worry.

After seen “**results are saved in:**” on command window, you can close the HandSegmentation file.

A screenshot of a MATLAB Command Window. At the top, there is a yellow banner that says "New to MATLAB? See resources for [Getting Started.](#)". Below this, the command window displays several lines of text: "--batch_info, iter:1, epoch:1(total:1), finished_task:0, epoch_", "--batch_info, gpu:1, time:2.0 (forw:1.9, backw:0.0, net:2.0, ev", "==epoch_info, epoch:1, epoch_time:2.0, config_time:0.0, obj:1.0", and "save predict mask:..\cache_data\Data_Result\A-apa-0007.png". A dashed line separates this from the final message: "results are saved in:", "The process is done.", "You can see the segmented images in the folder:", and "OUTPUTS_Segmented_Hands". The prompt "fx >>" is visible at the bottom left of the window.

```
Command Window
New to MATLAB? See resources for Getting Started.
--batch_info, iter:1, epoch:1(total:1), finished_task:0, epoch_
--batch_info, gpu:1, time:2.0 (forw:1.9, backw:0.0, net:2.0, ev
==epoch_info, epoch:1, epoch_time:2.0, config_time:0.0, obj:1.0
save predict mask:..\cache_data\Data_Result\A-apa-0007.png

-----
results are saved in:
The process is done.
You can see the segmented images in the folder:
OUTPUTS_Segmented_Hands
fx >>
```

Figure 57: How to Work the Software, Completed Message

You can see the all segmented outputs in the folder which is **OUTPUS_Segmented_Hands** in the project folder as figure58.

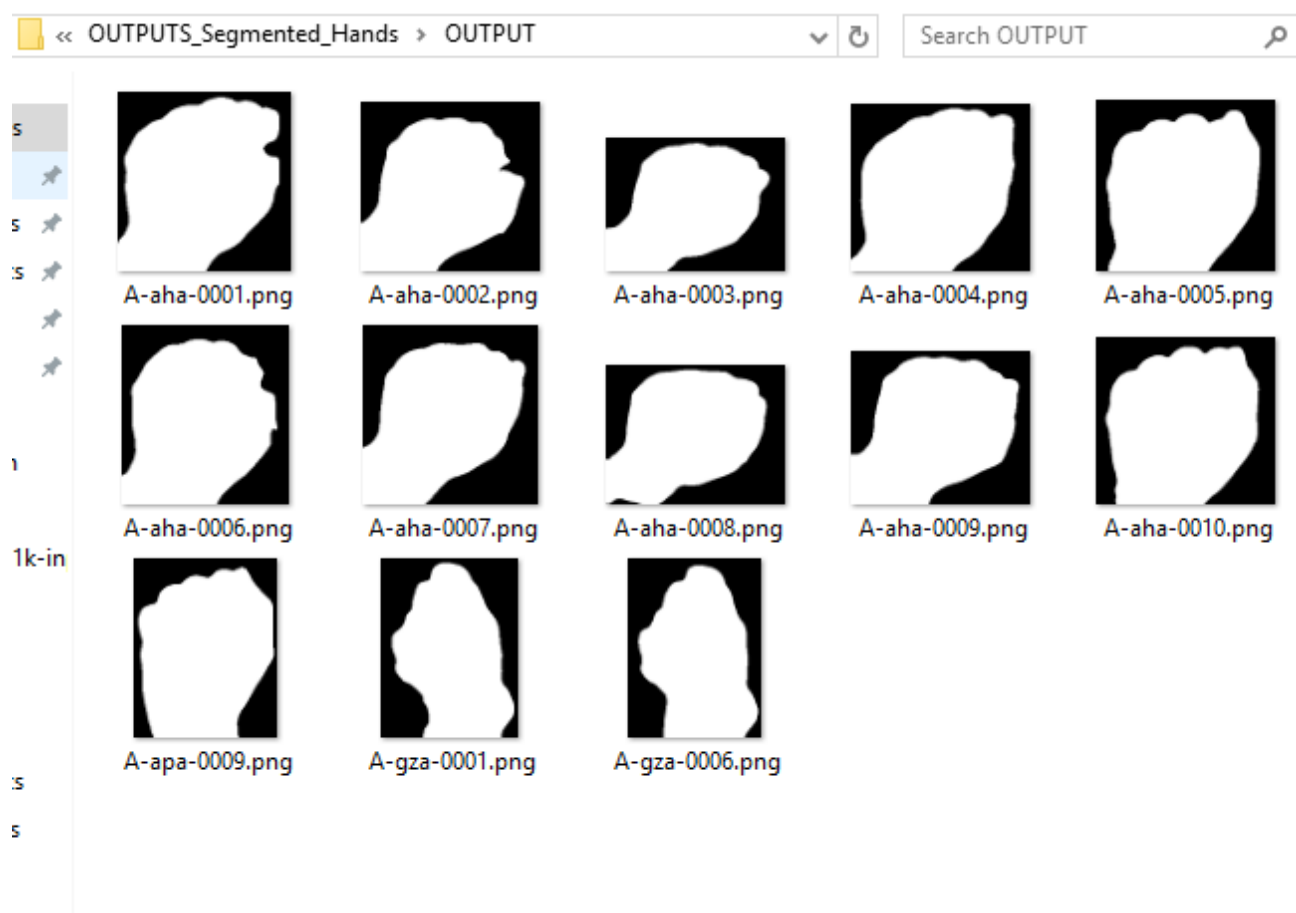


Figure 58: Output Folder

References

- Bradley, D., & Roth, G. (2007). Adaptive thresholding using the integral image. *Journal of graphics tools*, 12(2), 13-21.
- Ćorović, A., Ilić, V., Durić, S., Marijan, M., & Pavković, B. (2018). The real-time detection of traffic participants using yolo algorithm. In 2018 26th Telecommunications Forum (TELFOR) (pp. 1-4). IEEE.
- Dawod, A. Y., Abdullah, J., & Alam, M. J. (2010, December). Adaptive skin color model for hand segmentation. In 2010 International Conference on Computer Applications and Industrial Electronics (pp. 486-489). IEEE.
- Deafness and hearing loss. (2020, March 1). WHO | World Health Organization. <https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss>
- Everingham, M., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2), 303-338.
- Ghotkar, A. S., Khatal, R., Khupase, S., Asati, S., & Hadap, M. (2012, January). Hand gesture recognition for indian sign language. In 2012 International Conference on Computer Communication and Informatics (pp. 1-4). IEEE.
- Huang, R., Pedoeem, J., & Chen, C. (2018). YOLO-LITE: a real-time object detection algorithm optimized for non-GPU computers. In 2018 IEEE International Conference on Big Data (Big Data) (pp. 2503-2510). IEEE.

- Khan, R. Z., Ibraheem, N. A., & Meghanathan, N. (2012, January). Comparative study of hand gesture recognition system. In Proc. of International Conference of Advanced Computer Science & Information Technology in Computer Science & Information Technology (CS & IT) (Vol. 2, No. 3, pp. 203-213).
- Lin, G., Milan, A., Shen, C., & Reid, I. (2017). Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1925-1934).
- Noh, H., Hong, S., & Han, B. (2015). Learning deconvolution network for semantic segmentation. In Proceedings of the IEEE international conference on computer vision (pp. 1520-1528).
- Randive, A. A., Mali, H. B., & Lokhande, S. D. (2012). Hand gesture segmentation. International Journal of Computer Technology and Electronics Engineering, 2(3).
- Ravikiran, J., Mahesh, K., Mahishi, S., Dheeraj, R., Sudheender, S., & Pujari, N. V. (2009, March). Finger detection for sign language recognition. In Proceedings of the international MultiConference of Engineers and Computer Scientists (Vol. 1, No. 1, pp. 18-20).
- Redmon, J., & Farhadi, A. (2017). YOLO9000: better, faster, stronger. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7263-7271).
- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).
- Shafiee, M. J., Chywl, B., Li, F., & Wong, A. (2017). Fast YOLO: A fast you only look once system for real-time embedded object detection in video. arXiv preprint arXiv:1709.05943.

- Travinin Bliss, N., & Kepner, J. (2007). 'pMATLAB Parallel MATLAB Library'. The International Journal of High Performance Computing Applications, 21(3), 336-359.
- Urooj, A., & Borji, A. (2018). Analysis of hand segmentation in the wild. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4710-4719).
- Wang, J., & Su, X. (2011, May). An improved K-Means clustering algorithm. In 2011 IEEE 3rd International Conference on Communication Software and Networks (pp. 44-46). IEEE