GRADUATE CERTIFICATE INTELLIGENT SENSING SYSTEMS PRACTICE MODULE REPORT: PROJECT GAMEFACE

Jun Ming LIM, Tadhg KENNEDY, Wee Ping YEONG

Institute of Systems Science, National University of Singapore, Singapore 119615

ABSTRACT

The motivation of our project is to explore using different human physical attributes, like head pose and face actions to implement a vision system that can help people with certain physical disabilities to be able to interact with a computer. Our team conducted research on and experimented with various detection algorithms with the aim of finding a suitable model that can help us achieve this goal. We have also explored various approaches to determine the different head pose and face actions in order to convert these responses into control commands that can be sent via a computer linked to a webcam. After all the experimentation of the various methods, the final system was designed utilizing the landmark tracking based approach - FaceMesh, which is one of the suite of Machine Learning tools from MediaPipe developed by Google. This light weight model was consistently able to help us achieve real-time detection in a fast and accurate way. Finally our team was able to successfully develop a vision system using FaceMesh as its main detection and tracking engine, and also demonstrated the ability for users to have fun with online games simply by moving their head and doing actions with their eyes and mouth.

Index Terms— *vision system, head pose detection, face action recognition, facial landmark tracking, HCI, FaceMesh*

1. INTRODUCTION

Currently, people living with different physical disabilities need to buy expensive third party peripherals in order to supply inputs to their computer systems and enjoy playing video games [1]. We aim to develop a vision system which will allow Human-Computer Interaction (HCI) in a fun and affordable way, using human gesture and pose recognition.

The input to our proposed system are video images which can be from a simple webcam linked to a computer. We then use a head pose and face action tracking model to identify various positions of the head and actions of certain facial features in order to provide real time differentiated responses to send commands to the computer. This allows people who may have difficulties moving their limbs to interact with online games using different head postures and face actions simply through a low cost webcam.

2. LITERATURE REVIEW

Over the course of the project we investigated many different forms of head pose estimation techniques. The different approaches can generally be categorized into two methods, two stage implementations and single stage implementations, we have investigated both over the course of the project. The following section will detail the papers and resource we referred to for the different approaches.

2.1. Two stage landmark models

For a two stage approach there are two major components, face detection and landmarks, and angle inference. The first stage is commonly performed using the dlib library, using the built in face detector, the face in the image is detected and cropped. This image of just the face region is then used for the landmark detection using the pre-trained 68 landmark detection model from dlib. From these landmarks we then need to infer the pose angles, this involves estimating the 3D point locations based off their 2D positions in the image plane. This is commonly referenced to as a Perspective-n-Points problem, which involves using mathematical transforms to estimate the pose such as the Direct Linear Transform (DLT) algorithm followed by Levenberg-Marquardt Optimization (LMO). Luckily for us OpenCV has a built-in function we can use instead of writing out the maths ourselves manually. We can use the solvePnP and solvePnPRansac functions, solvePnP by default uses an algorithm similar to DLT followed by LMO. The solvePnPRansac function however is more robust because it uses Random Sample Consensus (RANSAC). By using RANSAC the algorithm is able to counteract noisy points by selecting the minimum number of points required and identifying inliers, finding the solution with the maximum number of inliers. Once we've solved the PnP problem, we can use the rotation matrix and translation matrix to estimate the euler angles of the head.

This method has the advantage of being a well established method with existing functions in place to conduct the necessary calculations. It also doesn't require any training of models to work, as the existing pre-trained models are so well optimized.

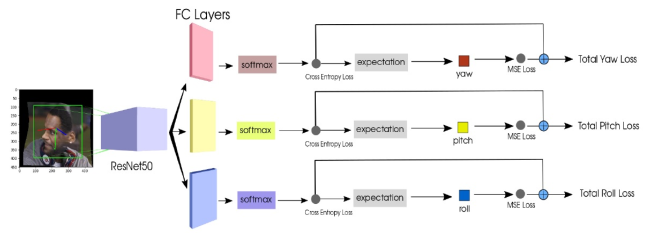
The model can struggle however with the stability of it's landmarks, this in turn makes the estimated angles unstable. Facial feature such as glasses and beards also impact the overall accuracy of the landmarks. You can see how the landmarks struggle to stay in place in the videos on the Towards Data Science article [2].

2.2. One stage head pose model

Single stage models seek to perform the angle inference directly using a single deep neural network (DNN) without using any facial landmarks. The model will still require a cropped face image using the face detector as input. By training a DNN with annotated face images at different angles, the system can learn to extract the features from the data and estimate the Euler angles in a single shot. In some use case, the head pose may just be all that is required to achieve its objective and the facial landmarks may not be useful in this case. Hence, the single stage models seek to reduce the training and inference time by skipping the landmark detection and 3D point location estimation step using a DNN based model architecture. At the same time, this approach will eliminate the dependency of the head pose estimation on the accuracy of facial landmark detection which often plagues the landmark-to-pose methods mentioned above. This is aligned with our objective where the exact location of all facial landmarks are not necessary for performing some of the simple HCI such as tilting or turning the head to one direction.

For the context of our project, we studied two state-of-the-art models that use the one stage head pose estimation approach, Hopenet [3] and WHENet [4]. The Hopenet uses a multi-output and multi-loss approach in estimating the head pose using Euler Angles[5]. The model uses a backbone network, ResNet50 as the feature extractor and followed by 3 separate fully-connected layers that form 3 outputs from the network, one for each direction of the Euler Angles: pitch, yaw and roll. Each output is in the form of 66 bins where each bin cover 3 degree angles for a total range of -99 to 99 degree. Each direction of Euler angles (pitch, yaw, roll) is then estimated as the softmax expectation of the 66 bins. The custom loss function used for each of the network output consists of 2 components, the bin classification loss and regression loss. The bin classification loss utilizes a softmax layer and categorical cross-entropy to calculate the loss or error bin classification. The regression loss is calculated as the Mean Squared Error (MSE) of the estimated Euler angles against the ground-truth angles. The final total loss is then defined as the sum of the bin classification loss and a weighted regression loss. The weight of the regression loss is controlled by multiplying it with a coefficient α, which is shown to affect the model performance in the original paper. Figure 1 below shows the summary of the model architecture of Hopenet as well as the custom loss function used following the original paper.

WHENet [4] is one of the top ranking state-of-the-art models in head pose estimation[6] as of the writing of this report. It managed to achieve a very low average Mean Absolute Error (MAE) of the 3 Euler Angles at 3.48. The model adopted a similar architecture as in Hopenet but with some further customization in the loss functions used. Instead of using the full weight of bin classification, it added in a weight coefficient for the bin classification loss. Within the bin classification loss, it uses sigmoid activation instead of softmax activation and claimed that this helped to improve the accuracy marginally. Furthermore, the regression loss function used is a custom wrapped loss function that seeks to minimize erratic behaviour due subjects with large yaw angles. The WHENet also reduced the model complexity by using a lighter and more efficient backbone network as feature extractor, EfficientNetB0[7]. This has reduced the number of parameters in the network significantly and able to achieve real-time head pose inference.



$$\mathcal{L} = H(y, \hat{y}) + \alpha \cdot MSE(y, \hat{y})$$

Fig. 1: Hopenet Model Architecture and Loss Function

$$\mathcal{L} = \alpha \mathcal{L}_{reg} + \beta \mathcal{L}_{cls}$$

$$\mathcal{L}_{wrap}(\theta_{pred}, \theta_{true}) = \frac{1}{N_{batch}} \sum_{i}^{N_{batch}} \min[|\theta_{pred}^{(i)} - \theta_{true}^{(i)}|^2, (360 - |\theta_{pred}^{(i)} - \theta_{true}^{(i)}|)^2]$$

Fig. 2: WHENet Custom Loss Functions

As evident from the paper results, real-time head pose estimation without landmark detection is achievable with careful design of a very deep neural network and custom loss functions. Without the landmark detection as an intermediate stage, the accuracy of head pose estimation can be much higher than that of a landmark detection based model.

However, similar to any other DNN based models, they still come with some limitations that we need to take into consideration when designing the solution for our project. As these state-of-the-art models are only used to predict the Euler Angles, there are a limited number of recognized actions that can be derived from the 3 Euler Angles combined. Some examples of action recognition that are not achievable using a one stage head pose estimation approach are like eye or mouth open close, smile detection or even facial emotion recognition. In contrast, these actions can be easily recognized using the less effective two stage landmark models as discussed previously. Furthermore, we also notice that some of these state-of-the-art model are not fully open to public as well. In WHENet, we can see that the model training details and processes are omitted and trained model weights and parameters are not open to public access. In order adopt the same approach as WHENet, we would need to develop, train and implement the model ourselves. In addition, pertaining to our objective of allowing simple and effective HCI, there is no need for the system to estimate the Euler angles up to a fine-grained level of every single digit of degrees. We would only need trigger action command whenever there is an obvious change in head pose.

2.3. Landmark tracking model

Apart from the detection based approach, we also look at the tracking based approach, specifically facial landmark tracking approach. The idea of a landmark tracking approach is that instead of performing landmark detection task for each frame or image in a series of real-time video, we track the initially detected facial landmarks across the frames. This has the benefit of potentially much faster processing time as the facial landmark detection will only need to be performed once in the beginning. The remaining of the processes is just tracking the facial landmarks as the user moves around within the coverage area of the camera. Such an approach is especially suitable for our use case here where the sensor equipment used is a webcam or camera with a fixed position and the user is assumed to be moving within the video coverage area in order for the system to register any head pose changes or facial action recognition.

One of the more recent implementation of facial landmark tracking that we studied is the Face Mesh tracking function offered under MediaPipe[8][9]. MediaPipe is an open-source framework developed by Google for building different cross-platform and customizable ML pipeline and solutions. Within MediaPipe framework, there are a long list of ML solutions which includes face detection, facial landmark detection and tracking, hand, human posture tracking and many more. For the context of our project, we look at the Face Mesh solution which is developed based on highly efficient face detection and landmark tracking method. According to the original paper[10], the algorithm is an image processing pipeline where it first perform a face detection using a light-weight face detection model, BlazeFace[11]. The BlazeFace model architecture is designed with feature extractor similar to the MobileNetV1/V2 and a modified anchor scheme of Single Shot Multibox Detector(SSD). Once the a face detected, the pipeline continues with feeding the cropped bounding box of the detected face into a mesh prediction neural network[10]. The mesh prediction network follows a residual neural network architecture and produces the 3D facial landmark coordinates. Finally, with the facial landmarks detected, the algorithm tracks them across the frames in the real-time video camera. To further improve the accuracy of landmark predictions especially around the lips, eyes and irises, another model, attention mesh model[12] is used in place of the mesh prediction neural network. It is able to detect a total 478 facial landmarks, additional 10 more landmarks for the irises as compared to the mesh prediction model.

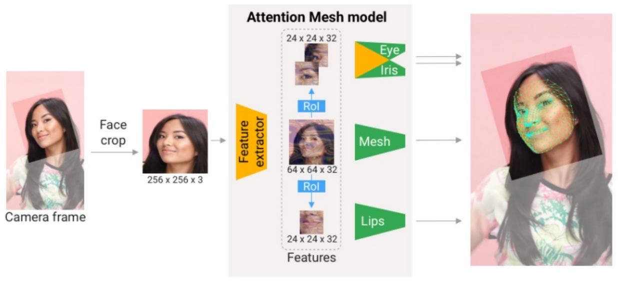


Fig. 3: MediaPipe Attention Mesh Model

As mentioned previously, the landmark tracking based model has the benefit of much faster inference time as facial landmark detection is only performed once in the beginning. However, this also means that the approach is prone to some of the common issues that generally plague object tracking algorithms in computer vision. Some of the issues are like object occlusions and accuracy of tracking can be affected when there are multiple similar objects within the image or frames. Comparing to the second approach on one stage head pose estimation approach, the landmark tracking method is not only faster than the prior but also able to perform many different kind of real-time facial action recognition such as eye or mouth open close etc. Unlike the methods we have discussed previously, the MediaPipe is a complete bundle of ML solutions that comes in the form of ready-to-use applications. Hence, there is no detailed model architecture or model training information available. For Python based application, MediaPipe is a Python library package that we can install and call the functions such as the Face Mesh function and apply directly on images or frames for detection or tracking inference.

3. DATASET

3.1. Closed Eyes in the Wild

The Closed Eyes in the Wild (CEW) dataset was used during our experimentation to train a CNN for detecting if a person's eyes are open or closed. The dataset consists of 2423 subjects with 1192 subjects with both eyes closed and 1231 subjects with eyes open are selected from the Labeled Face in the Wild. We specifically used the dataset of patches which contained the eye and were split into right eye and left eye, each 24x24 pixels. When we used the data in the model training we used the DataGenerator function in TensorFlow to perform data augmentation such as translation, rotation and zoom to bolster the data set and help the model to be more robust to varying situations. The lighting can vary greatly within the dataset, as can be seen in Figure 4.

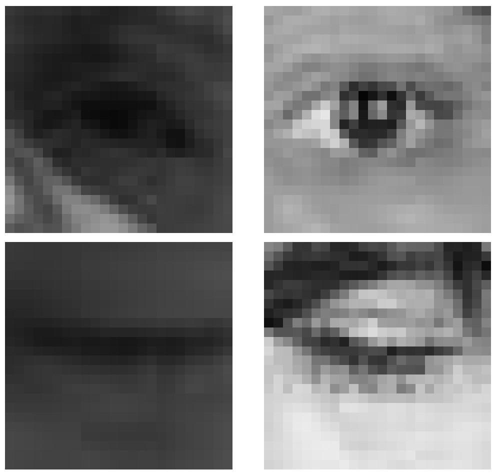


Fig. 4: Samples of CEW dataset

3.2. 300W-LP

Another experimentation that we have conducted in the project is to train a head pose estimation neural network. The 300W-LP[13][14] is used for this purpose. The 300W-LP dataset is a collection of synthetically expanded images of the dataset 300 Faces In-The-Wild (300W)[15][16][17]. A face model is used to morph the original face image from 300W dataset into various angles of yaw. This has generated a total of 61,225 synthetic face images. In total, there are 122,450 images of face including both the original images from 300W dataset and the synthetically generated images. Each image only have 1 face and the annotations for each image includes various attributes of a face such as the position of the 68 facial landmarks, Euler Angles (pitch, yaw, roll) and more. For the context of training a head pose estimation model, we only use the Euler Angles annotations from the dataset. By training our model on a large dataset, we can ensure that the model can adapt to the wide variation of faces and angles. Figure 5 shows some examples of the original image and synthetically generated images.

4. PROPOSED SYSTEM

Our overall system architecture (Fig.6) starts with getting real-time video images as the inputs where all frames will be utilised. The reason why we want to do this is because the team had taken into consideration that different users may have webcam of different FPS (frames per second) rate.

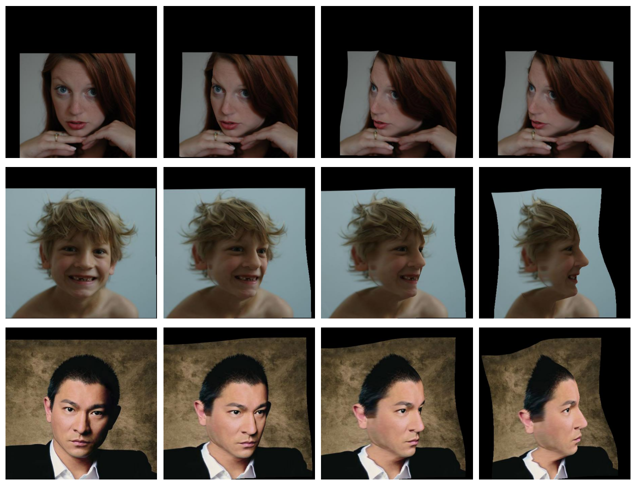


Fig. 5: Samples of 300W-LP Dataset

On top of that, we want our system to be able to capture any changes in head posture in order to be able to provide a zero-latency, speedy response to the computer interface.

Another design consideration is on the ease of usage and smooth gaming experience that can suit the individual preferences of each user. Hence we have included a configuration file that the user can simply edit and load into the system before the actual start of game play. The file content is split into various sections for different groups of variables with clear description on what to change and for which purpose.

The head pose and face action detection model will be the main engine behind our entire system. Here we aim to deploy a fast light-weight model so that every single frame from the video images can be used for real-time inference to detect changes in face direction, and even opening of mouth or closing of eyes. After all the experimentation on the various approaches mentioned in the literature review section, the final system is designed utilizing the landmark tracking based approach. Even though the selected model is already pre-trained, we evaluated that FaceMesh by MediaPipe was able to satisfy this requirement through its face detection algorithm and facial landmark tracking which proved to be highly accurate.

With the ability to instantaneously track face landmarks for every frame, we then employ heuristic methods to estimate the head pose together with any movements on the eyes and mouth based on the selected landmark coordinates of interest. Such methods can be distance based, where we evaluate the ratio between certain points to determine the face direction, or use arctan function to obtain the estimated angle of tilt.

Once our system is able to accurately detect head pose and face activity changes, we then gain the capability to map these actions to different control commands which will then be automatically input to the computer system in real-time. With all these in place, the user can then utilise our vision system and enjoy playing online games simply by moving their head and doing actions on their eyes and mouth.

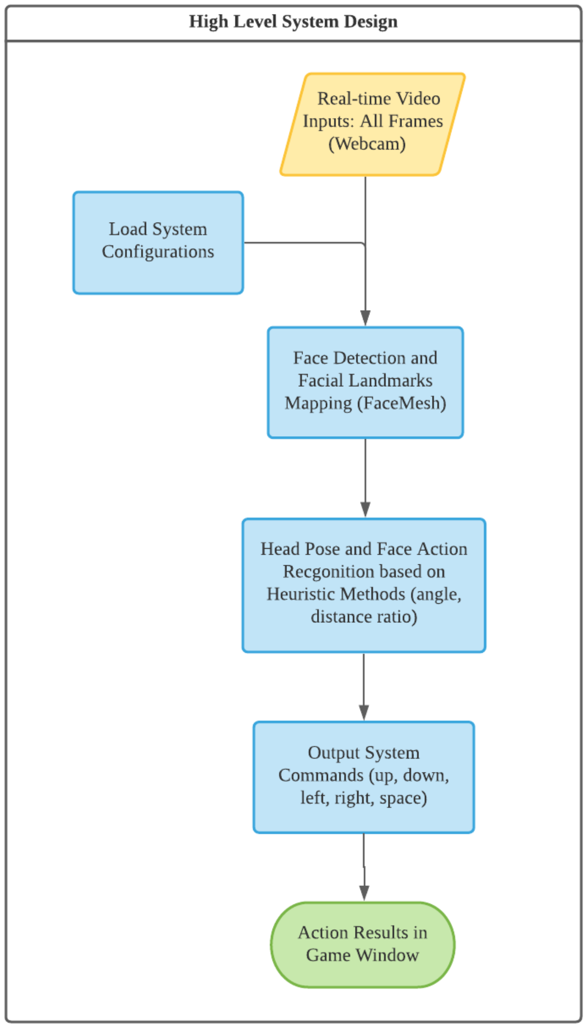

High Level System Design
Real-time Video Inputs: All Frames (Webcam)
Load System Configurations
Face Detection and Facial Landmarks Mapping (FaceMesh)
Head Pose and Face Action Recgonition based on Heuristic Methods (angle, distance ratio)
Output System Commands (up, down, left, right, space)
Action Results in Game Window

Fig. 6: GameFace System Design Flowchart

5. EXPERIMENTAL RESULTS

5.1. Two stage model

We investigated the use of dlib for landmark detection as part of a potential two stage model, this was very straight forward as there are numerous resources available for simple dlib face detection and landmark detection functions [2] [18]. As we were also investigating the use of mouth and eye detection, we first attempted to use the landmarks from the dlib detector to detect if the mouth and eyes were open or close. Using a guide online [19] we were able to implement a rudimentary mouth detector. By calculating the average Euclidean distance between points on the top and bottom edge of each lip, then calculating the distance between the lips we can check how far the mouth is open relative to the size of the person's lip. This allows the calculation to handle different sized mouths better than a fixed threshold distance. We developed similar logic based on the distance between the corners of a person's eye, and their top and bottom eyelid. Both of these calculations include a scaling factor which we were able to fine tune to adjust the sensitivity of the functions.

However, when we tried using the dlib facial landmarks for live video, we found that the inference time was very slow, reducing the frame rate of the output video to only 1-3 frames per second. Since the frame rate is effectively the number of frames being processed, we found that the time between actions and the feedback time for the user was too slow for it to be usable in controlling the game. Due to these drawbacks we shifted focus to the single stage models.

5.2. CNN based eye detection

After we decided that dlib landmark detection wasn't suitable for our use case, we investigated using a convolutional neural network for the eye and mouth open and closed detection. Using the CEW dataset we trained a simple CNN model to recognize if a person's eyes were open or closed, using this guide [20] we were able to implement the model to detect if a person's eyes were open of closed. This guide still used the dlib landmarks for detecting the eye position but we hoped to be able to use a more lightweight landmark system in practice. However, we ran into two major issues before proceeding. While we found a dataset for the open and closed eyes, we were unable to find an easily accessible and suitable dataset for the mouth detection. Additionally the initial results weren't promising from the CNN, the detection wasn't picking up the eye being open or closed well. In light of this, and the work that would be involved in curating our own dataset for mouth detection we made the decision to pursue a different approach.

5.3. Head Pose Estimation Model

While we saw that the landmark detection based approach dlib is not useful for developing a responsive vision system for head pose estimation due to slow inference time, we decided to experiment on the effectiveness of a one stage head pose estimation model. We adopted a similar neural network architecture to Hopenet and WHENet discussed above and trained a head pose neutral network that takes in a face image as input and produces 3 outputs, 1 for each of the Euler angles. The aim here is not to reproduce the exact replicate head pose estimation model as we saw in Hopenet and WHENet, but to test out some fine tuning to the model architecture to cater for the scope our project.

Similar to WHENet, we use the EfficientNetB0[7] as the feature extractor of our head pose net. There are 3 network output, each of them is a 66 bins class output where each bin cover 3 degree angles for a total range of -99 to 99 angle. This is the same output structure as we saw in Hopenet. We do not intent to cover a wider range of angle output as what WHENet did. This is because for the practical reasons, we do not expect the users to turn their head to extreme angles when performing HCI using our GameFace application. As the main objective of our system is to allow HCI through a real-time camera or webcam, we would assume that the user face and attention would be facing largely around the computer screen. Hence, extreme angles such as yawing to left or right for more than 90 degrees does not make sense for our use case.

The loss function we adopted is similar to Hopenet which is the sum of softmax categorical cross-entropy loss and weighted MSE loss. The weight or coefficient used for training our head pose model is $\alpha = 1$. Given the scope of the project, we did not further explore all possible values of the coefficient α as we saw from the Hopenet studies, there was only a marginal improvement for testing with various values of α. Figure 7 shows an illustration of the model architecture used for training the head pose net which resemble the architecture of Hopenet.

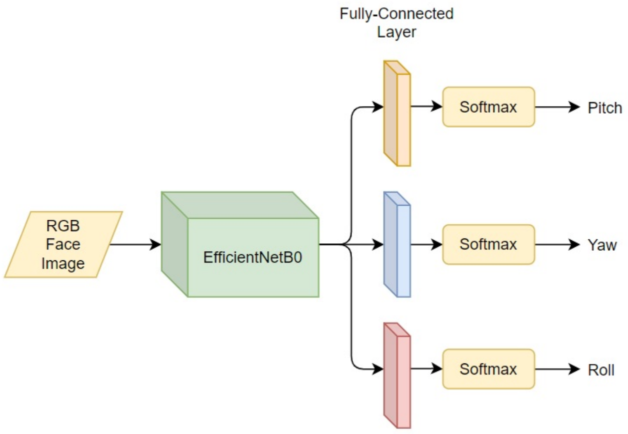


Fig. 7: Head Pose Net

As the model only takes in face image as input, we would need to further apply a face detector to the raw frames to obtain the bounding box of the face. The face detector we used here is an ultra-light face detector[21] that is very small in size and has a very fast inference speed. As mentioned in the discussion of the 2 state-of-the-art models Hopenet and WHENet, some of the information about the training process of the models are not available. Hence, the head pose net is developed in tensorflow. Model subclassing method is used for building the network architecture in tensorflow instead of functional API as model subclassing would allow the flexibility of using custom loss functions. A 70/30 train test split was used for the training. The metric used for evaluation is the Mean Absolute Error (MAE). As the dimension of the last CNN block of the EfficientNetB0 is 1,280, using a Dense layer of size 66 bins would create 84,546 trainable parameters for each of the Euler Angle output.

Table 1 shows the evaluation result of the head pose net trained on the 300W-LP dataset. We can see that purely using the EfficientNetB0 as feature extractor with all layers frozen, the yaw angle error at 36.1 MAE is much higher as compared to the pitch and roll angle. This is likely due to about half of the 300W-LP dataset are synthetically created images. The yaw angles annotation itself of these synthetically created images might not be very accurate as a result. In an attempt to improve the model performance, we turn on the last group of the CNN block in the EfficientNetB0 which consists of 16 layers to be trainable and we can see that pitch and roll angle improved with MAE at around 2.5, while the yaw angle MAE increased up to 49.6. Besides, the model loss chart shows that the model is slightly overfitted especially after epoch 25. Finally, we decided to test out a head pose model that only has 2 output angle, pitch and roll since the model is not performing on yaw angle using the 300W-LP dataset. With last group of the CNN block in EfficientNetB0 turned on to be trainable, we further added in batch-normalization layer and dropout layer before the dense layer to help reduce the model overfitting. The final MAE of the pitch and roll are at 3.0 and 2.6 respectively.

Model Architecture	Number of Parameters Trainable	Pitch MAE	Yaw MAE	Roll MAE	Avg. MAE
No fine-tuning	253,638	4.07	36.11	7.24	15.81
16 trainable layers	1,375,222	2.51	49.63	2.25	18.13
16 trainable layers with BN and Dropout Layers (Pitch and Yaw)	1,293,236	3.02		2.59	2.80

Table 1: Head Pose Net Training

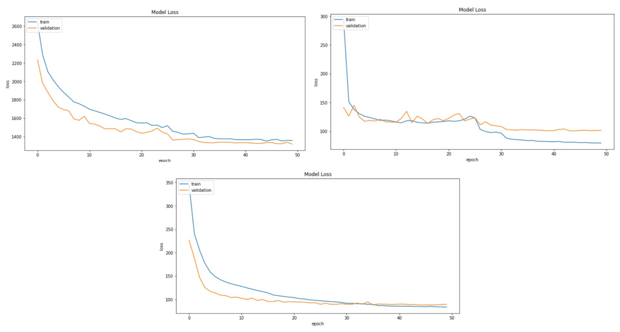


Fig. 8: Head Pose Net Training Loss

We used the trained model and tested it with a webcam for real-time performance. The inference speed performance is faster than the two stage landmark detection based approach at an average 20 frames per second. To trigger any action, we set a threshold such that whenever the inferred pitch and roll angles exceed the threshold, an command control will be triggered. Setting up a quick control system with just the 4 directions (pitch up, pitch down, roll left and roll right), we were able to use the system to play a simple game such as the Tetris game. We used the roll left and right with an angle threshold of 15-20 degree to execute the move left and right button in the game. We then used the pitch up and down with a similar threshold of 15-20 degrees to execute some other commands such as rotating the Tetris block or pausing or resuming the game. The system seems to be capable of handling simple real-time command inputs using the pitch and roll angles inferred from the trained head pose model taking the real-time video frames as input. However, after a few rounds of testing the system, we noticed that the user experience is not ideal and smooth as what we expected. In some occasions, actions are triggered unexpectedly as we saw that the real-time pitch and roll angles inferred tend to fluctuate at a range of around -5 to +5 degrees. Besides, there also some other unexpected behavior observed such as when the user closes their eyes, the model inferred it as a 5-10 degrees roll to the right which can then trigger actions unexpectedly.

On top of the limitations observed with the trained head pose model, it is also difficult to expand the usability of the system to include more commands. Hence, we did not proceed further to tune or improve head pose model. Together with the previous issue on eye detection using CNN model, we decided to explore and experiment with the third approach for both head pose estimation and facial action recognition using the landmark tracking based approach. It was at this time that we became aware of MediaPipe and this was our next and final approach.

5.4. MediaPipe

As discussed previously, MediaPipe is a framework that consists of various ML solutions that can be used to develop different kinds of application across multiple platforms. Within the Python ecosystem, the MediaPipe is a Python library package which we can install and directly call its functions on images or frames to perform various tasks such as face detection, facial landmark detection and tracking etc. For the context of our project, we tested the Face Mesh function which can detect and tracks up to 478 facial landmarks in real time.

According to the original MediaPipe source, the Face Mesh solution is wrapped within a built-in function call FaceMesh. A FaceMesh object would first need to be initialized and then it can be directly applied on any images or frames. The function will produce an object output which its attributes are the normalized x and y coordinates of the facial landmarks. To convert the normalized x and y coordinates back to the scale of the original input image or frame, we simply just multiply the x value by the width and multiply y value by the height of the original input image or frame. To identify which location that each of the 478 landmarks refers to on a face, we can make use of the visualization diagram, Figure 9 as taken from the original MediaPipe source.

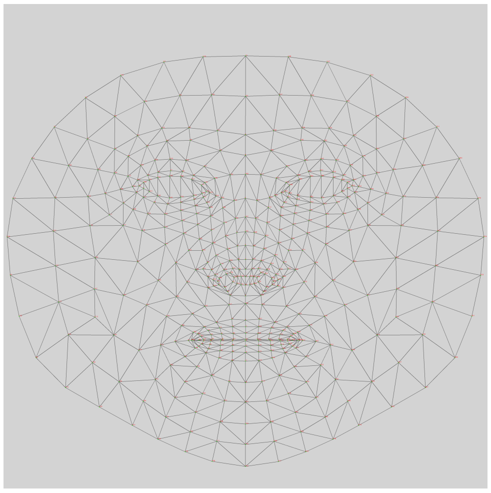


Fig. 9: FaceMesh Canonical Face Model Landmark Indices

We tested the performance of the FaceMesh function in real-time webcam video and it seems to perform the best among all approaches that we had experimented. The inference speed of the facial landmark detection and subsequent landmark tracking is very impressive. The frame per second count is close to 30 on average given the algorithm is applied on every frame without any skipping and is tested on a webcam that records at 30hz frequency. The landmark tracking is also performing well where it is able to track the facial landmark accurately as the user moves around in front of the webcam. As the MediaPipe is developed by Google with portability in mind, it is capable of running with desktop CPU. We tested the FaceMesh performance running on CPU and the resulting frame per second is still close to 30 on average. However, as discussed previously, a landmark tracking approach for face pose estimation and facial action recognition do comes with limitations such as object occlusions. We tested and found that the landmark detection and tracking is less accurate when the user face is partially occluded. This is more apparent when the user is wearing a mask or a sunglasses.

Nonetheless, the real-time landmark detection and tracking is one of the best performing approach that we had experimented in this project. As such, the team had decided to adopt the landmark detection and tracking based approach by using the FaceMesh for the GameFace system implementation.

5.5. Heuristic Head Pose and Facial Action Recognition

To develop a useful system that is able to recognize changes in head posture or facial actions such as eyes or mouth open close, we adopted a heuristic based approach that utilizes the facial landmarks for inference. We have defined and developed a total of 8 actions to recognize in the system using the facial landmarks that are being tracked by the FaceMesh function. They are the pitch up, pitch down, yaw right, yaw left, roll right, roll left, eyes close and mouth open.

To estimate the roll angles, we can draw a horizontal line across the face using just the widest landmark point on the left and right cheek of the face. The estimated roll angle is then defined as the arctan of the height difference of the 2 points over the width difference of the 2 points. To check if the user is rolling to left or right, we just need to check if the left cheek landmark point is lower or higher than the right cheek landmark point in the image or frame. We can then set a threshold where if the angle difference of rolling left or right to the neutral or origin position is higher than the threshold, then an action command will be triggered.

For the yaw angles, we are not able to estimate the exact angles in degree using the 2D facial landmarks. Instead, we utilized the facial landmarks to compare the distance from the left eye to the nose bridge versus the distance from the right eye to the nose bridge. With the camera placed in front of a person, when the person is yawing to the left, the distance from the right eye to the nose bridge will seemingly increase in length while the distance from the left eye to the nose bridge will decrease and vice versa when the person is yawing to the right. Hence, by comparing the ratio of the 2 distances, we can identify if the user is yawing to the left or right. To trigger action command using the yawing, we just need to again set a threshold for the ratio of the distances such that an action will be triggered whenever the user yawing to 1 direction.

Similarly, estimating the pitch angles in degree is not possible using the 2D facial landmarks. To recognize a pitching action, we can draw 4 lines with 2 on the upper and lower level of the forehead landmarks and remaining 2 on the upper and lower level of the chin landmarks. We then compare the distance from the upper to lower level of forehead lines versus the distance from upper to lower level of chin lines. When a person pitches up, the distance from the upper to lower level of chin lines will seemingly increase while the distance from the upper to lower level of forehead lines will decrease and vice versa when the person pitches down. Similar to triggering action commands with yawing, we just need to set a threshold for the ratio of the 2 distances calculated using the 4 lines formed from the facial landmarks to trigger pitch commands.

One additional benefit of using landmark based model here is that we can use the logic we developed as part of the dlib landmark based on mouth and eye detection with the FaceMesh landmarks. We can easily check if the eyes or mouth are open or close using the landmarks around the eyes and mouth. By comparing the ratio of the distances calculated using the eyes and mouth landmarks, we can determine if there is a change in facial action which will then allow action commands to be triggered.

The figure 10 shows the lines and landmarks used for identifying the 8 actions using heuristic based approach. The green color line is used for roll angles, red color lines for yaw directions, blue color lines for pitch directions, orange color lines for mouth open close and yellow lines for eyes open close.

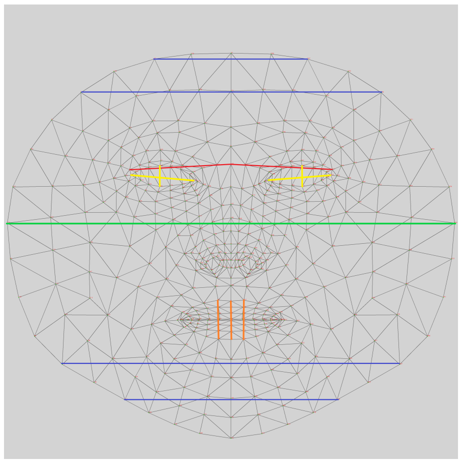


Fig. 10: Heuristic Based Head Pose and Action Inference

To further automate and introduce flexibility into the system, we integrated all the action command inferences under a rule-based system. All thresholds and action commands mapping used for the system are saved and recorded in a configuration file. The user can easily change the response time or sensitivity of the action inference easily by just adjusting the threshold and limits within the configuration file. There is a mapping of head pose or facial action recognition that would allow the user to fully customize the keyboard buttons to execute for each of the 8 actions inference. The execution of keyboard buttons using the action inference is done using the keyboard interaction function, pyautogui in Python. To facilitate a smooth user experience, we designed an operation process flow for the system where the system operates on 2 modes: standby mode and play mode. Upon launching the GameFace app, the system will initialize in the standby mode after loading the settings from the configuration file. During the standby mode, no actions will be registered besides the start action(close eyes) to go into the play mode. This is a time frame set to allow the user to get ready to go into the play mode. Whenever play mode is triggered, the system will calibrate the neutral position of the 3 directions (roll angles, yaw distance ratio, pitch distance ratio) and the neutral position will be set for the entire session of the play mode.

To re-calibrate the neutral position, the user would need to go back into the standby mode and re-trigger into play mode again. Within the play mode, all head pose and facial actions inferred using the logic explained above will be triggered in real-time. Finally, one of the action(by default the close eyes action) is reserved to switch the user back into the standby mode. The switching of standby and play mode is designed to prevent unintended actions being triggered when using the GameFace app. The standby mode serves as a quick pause as and when the user needs to attend to other matters. The figure 11 shows a summary of the operation of the rule-based systems for the standby mode and play mode.

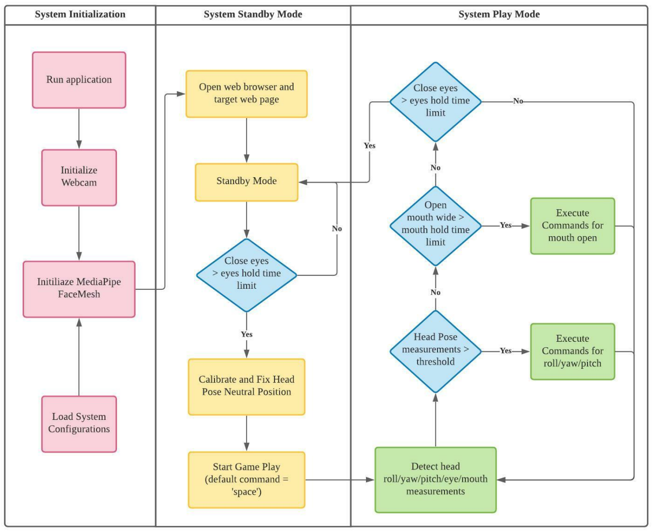

System Initialization
System Standby Mode
System Play Mode
Run application
Initialize Webcam
Initiliaze MediaPipe FaceMesh
Load System Configurations
Open web browser and target web page
Standby Mode
Close eyes > eyes hold time limit
Yes
No
Calibrate and Fix Head Pose Neutral Position
Start Game Play (default command = 'space')
Close eyes > eyes hold time limit
No
Yes
No
Open mouth wide > mouth hold time limit
Yes
Execute Commands for mouth open
No
Head Pose measurements > threshold
Yes
Execute Commands for roll/yaw/pitch
Detect head roll/yaw/pitch/eye/mouth measurements

Fig. 11: System Operation Process Flow Chart

We have tuned the system configuration based on testing with some simple games such as the Tetris game and Pac-man doodle game. With the FaceMesh utility, we were able perform accurate and responsive action commands easily in real time through a webcam. Finally, the system is deployed as a python file where the user would just need to run the python file to start the GameFace app.

6. CONCLUSIONS

Given the limited time frame for us to work on this project, our team has successfully developed a vision system that is able to detect the various head postures as well as face actions in real-time and provide differentiated control commands to play online games via the use of a simple webcam linked to a computer. During the course of the project, we experimented with different methodologies like detection-based models which can be 2-stage or 1-stage approach, as well as tracking-based models to infer the head angles and even face actions. We had also explored various ways to estimate the different head poses, like directly estimating the angle of tilt via the model, or using heuristic method of calculating euclidean distances from facial landmarks and evaluating the ratio to determine the outcome.

Initially, we started working with the detection-based Head Pose model that directly estimates the face angle. After a series of training and tuning the model, we ultimately managed to obtain very good results but we noticed an obvious reduction in the frames per second since it is doing inference frame by frame. The team then went on to search for an alternative algorithm and managed to find MediaPipe, which is developed by Google, that offers a pipeline of cross platform Machine Learning solutions. It offers a suite of tools that include FaceMesh, which has face detection, landmark detection and tracking . We realised that this can actually perform much faster real-time head pose estimation than the previously mentioned Head Pose model due to speedier inference time. Therefore we selected FaceMesh by MediaPipe as the main engine behind our head pose and face action detection system so that we can deliver a smoother interface and this method of inferring the outcome is also much less complex.

In addition, in order for our system to be able to offer a better gaming experience, and even be versatile enough to play different games, the team has also designed it in such a way that users can adjust various settings according to their own preferences by editing a configuration file.

7. LIMITATIONS AND FUTURE WORK

The current limitation to our system is that there are only 8 unique control outputs available. For future enhancement there is potential to scale up in order to play more complicated games by having more combinations of detection with even more controls. For example, we can have detection for single eye wink, smile, gaze estimation or even different facial expressions for a richer gaming experience.

To cater to more people with other forms of disability, we can also further extent our detection system to other parts of the body, like arm or fingers movement so that controlling commands are not only limited to the face.

To optimize our model inference time, which is currently tracking 478 facial landmark points with FaceMesh, we can customize and train our own models with fewer landmark points that only covers the necessary facial features of interest. This can also potentially decrease the amount of time needed for model training.

8. CONTRIBUTIONS

This section describes the contributions of each team member to this project in order to make it a success, while having fun at the same time!

Table 2: Team member contributions:

Name	List of contributions
Jun Ming	Head Pose Model Training and Testing MediaPipe Face Mesh Landmarks Head Pose Estimation using Face Mesh Heuristic Based Command Trigger Rules Testing and tuning of system System Integration and Deployment Report writing and documentation
Tadhg	Lightweight face detection dlib landmark detection method Mouth and eye detection methods Angle visualisation functions Testing and tuning of system Report writing and documentation
Wee Ping	Originator of project idea Automated keyboard interaction (pyautogui) Input command buffer function Testing and tuning of system Report writing and documentation

9. REFERENCES

[1] Sintia Radu, “Who’s paying for assistive technology?,” https://www.usnews.com/news/best-countries/articles/2017-12-01/assistive-technology-keeps-growing-but-paying-for-it-is-next-challenge, 2019.

[2] Vardan Agarwal, “Real-time head pose estimation in python,” https://towardsdatascience.com/real-time-head-pose-estimation-in-python-e52db1bc606a, 2020.

[3] Nataniel Ruiz, Eunji Chong, and James M. Rehg, “Fine-grained head pose estimation without keypoints,” *CoRR*, vol. abs/1710.00925, 2017.

[4] Yijun Zhou and James Gregson, “Whenet: Real-time fine-grained estimation for wide range head pose,” *CoRR*, vol. abs/2005.10353, 2020.

[5] Wikipedia, “Euler angles,” https://en.wikipedia.org/wiki/Euler_angles.

[6] Paperswithcode, “Head pose estimation,” https://paperswithcode.com/task/head-pose-estimation.

[7] Mingxing Tan and Quoc V. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” 2020.

[8] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, Wan-Teh Chang, Wei Hua, Manfred Georg, and Matthias Grundmann, “Mediapipe: A framework for building perception pipelines,” 2019.

[9] Google, “Mediapipe,” https://google.github.io/mediapipe/.

[10] Yury Kartynnik, Artsiom Ablavatski, Ivan Grishchenko, and Matthias Grundmann, “Real-time facial surface geometry from monocular video on mobile gpus,” 2019.

[11] Valentin Bazarevsky, Yury Kartynnik, Andrey Vakunov, Karthik Raveendran, and Matthias Grundmann, “Blazeface: Sub-millisecond neural face detection on mobile gpus,” 2019.

[12] Ivan Grishchenko, Artsiom Ablavatski, Yury Kartynnik, Karthik Raveendran, and Matthias Grundmann, “Attention mesh: High-fidelity face mesh prediction in real-time,” 2020.

[13] Xiangyu Zhu, Zhen Lei, Xiaoming Liu, Hailin Shi, and Stan Z. Li, “Face alignment across large poses: A 3d solution,” *CoRR*, vol. abs/1511.07212, 2015.

[14] Xiangyu Zhu, Zhen Lei, Xiaoming Liu, Hailin Shi, and Stan Z. Li, “Face alignment across large poses: A 3d solution, 300w-lp,” http://www.cbsr.ia.ac.cn/users/xiangyuzhu/projects/3DDFA/main.htm.

[15] Christos Sagonas, Georgios Tzimiropoulos, Stefanos Zafeiriou, and Maja Pantic, “300 faces in-the-wild challenge: The first facial landmark localization challenge,” in *2013 IEEE International Conference on Computer Vision Workshops*, 2013, pp. 397–403.

[16] Christos Sagonas, Georgios Tzimiropoulos, Stefanos Zafeiriou, and Maja Pantic, “A semi-automatic methodology for facial landmark annotation,” in *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2013, pp. 896–903.

[17] Christos Sagonas, Epameinondas Antonakos, Georgios Tzimiropoulos, Stefanos Zafeiriou, and Maja Pantic, “300 faces in-the-wild challenge: database and results,” *Image and Vision Computing*, vol. 47, pp. 3–18, 2016, 300-W, the First Automatic Facial Landmark Detection in-the-Wild Challenge.

[18] Satya Mallick, “Head pose estimation using opencv and dlib,” *LearnOpenCV*, 2016.

[19] Peter Xie, “How to detect mouth open for face login,” https://towardsdatascience.com/how-to-detect-mouth-open-for-face-login-84ca834dff3b, 2019.

[20] Jordan Van Eetveldt, “Real-time face liveness detection with python, keras and opencv,” https://towardsdatascience.com/real-time-face-liveness-detection-with-python-keras-and-opencv-c35dc70dafd3, 2019.

[21] Linzaer, “Ultra-lightweight face detection model,” https://github.com/Linzaer/Ultra-Light-Fast-Generic-Face-Detector-1MB.