A Robust Head-Driven Camera Mouse using Facial Landmark Extraction and Optical Flow Tracking

Antonis Moussoulides - 2004482

A Thesis submitted for the Degree of Master of Science in Advanced Computer Science

Supervisor:

Dr John Woods

School of Computer Science and Electronic Engineering

University of Essex

August 2020

Abstract

As of today, the mouse and keyboard is still the conventional method to interact with a computer. However, individuals with physical impairments can find this method of Computer Interaction obstructive, since it requires the full engagement of the eyes, head, arms, and fingers. Opportunely, the motivating factor is that typical consumer-grade webcams are very common and inexpensive nowadays, and Computer Vision techniques have technologically advanced. Therefore, this paper presents a non-intrusive Human-Computer Interaction system, in the form of a user-friendly application, that uses Computer vision techniques and lets the user control their mouse with only their head movement alone, using a consumer-grade webcam. Various algorithms used for this problem are justified and evaluated. Face detection is used to detect the user's face and define a face bounding box. Facial Landmark Extraction is used to identify a single point between the user's eyes within the face bounding box. Lucas Kanade Sparse optical flow tracker is used to track the defined point robustly between consecutive frames, thus ultimately allowing the user to control the mouse cursor with their head. Three different modes for triggering mouse events are also presented, to give the freedom of choice to the user. The robustness, speed and precision of the system are also evaluated and measured. The evaluation results indicated that this system can be a viable hands-free alternative mouse for individuals with physical impairments.

Table of Contents

1. Introduction & Purpose

1.1 Problem & Motivation

The rapid growth of Personal Computers (PC) in the early 1980's introduced the mouse and keyboard as the most conventional and straightforward interface between a human and a computer. Up to the present moment, the mouse and keyboard are nonetheless considered the most common Human-Computer Interface (HCI) devices. This is due to the fact that they have an intuitive and ergonomic design for the wrists and hands and offer a variety of practical shortcuts and functions which makes the interaction and experience simple between a user and a computer. While the mouse and keyboard might seem the most natural method to interact with a computer, individuals with physical and motor impairments will find them hindering to use since they require the full use of eyes, head, arms, and combination of fingers. In particular, individuals suffering from brain and/or neurological disorders such as Amyotrophic lateral sclerosis (ALS), which can target and paralyze the upper limbs, can find the use of the conventional mouse virtually impossible to use, while individuals with other physical impairments that might have retained their upper limb movement, can find the use of the mouse difficult [1]. As a result, individuals with these physical impairments cannot effectively use a computer mouse properly and intuitively which can consequently have an effect on their personal life and employability opportunities in the future, as they cannot access and use a computer efficiently [2].

Nowadays, several assistive applications and interfaces have been presented in the industry. Their main focus is to make a PC accessible for individuals with physical impairments or individuals unable to use a mouse and keyboard. These interfaces serve as a bridge of communication between the user and the computer or other devices, which improve the efficacy of the accessibility between a user with physical impairments and a computer. Some widely used technologies today include the utilization of wearable peripherals, such as electrodes attached to functional muscles, that can carry electrical signals and thus able to trigger an event, for example, a mouse click. Other prominent solutions, that can be also combined with electrodes, are the use of headbands, goggles or wristbands with infrared sensors embedded in them that can effectively track the intended motion of the user and as a result, simulate a mouse [3] [4]. While these interfaces discussed might provide an alternative and accurate solution to an individual with physical impairments, the key problem is that these interfaces might seem extremely intrusive to some users as they can take a significant amount of time to set up and use each time the individual wants to access their PC. They can additionally cause physical discomfort after wearing them for an extended period since they introduce plenty of cables. Another significant disadvantage worth mentioning is that these wearables can be relatively expensive to purchase and maintain.

However, over these past two decades, the field of Computer Vision (CV) and Artificial Intelligence (AI) has introduced a significant boom of interest on scientists and researchers, both in the industry and academia. Recent advancements in these fields bring forward the opportunity to enhance current HCI systems for individuals with physical impairments without the need for expensive assistive peripherals. Additionally, the significant expansion in the software ecosystem of CV and AI libraries that are offered today make a task that was considered to have had a very complex nature in past years be now implemented relatively easily. In particular, the motivating aspect behind this dissertation is that these advancements make it technologically feasible to develop an HCI system that can track the rotation of the user’s head from their facial landmarks in real-time using various CV techniques, with only the use of a low-cost consumer-grade web camera, which nowadays is very commonly present in almost every PC. By tracking in real-time the position of the user’s head using only a single web camera, we can estimate where the user is likely looking on their screen and thus ultimately simulating and controlling a non-intrusive, hands-free mouse, with only the user’s head movement [5].

Having stated the motivating factors for this paper, the core purpose of this dissertation is to develop, evaluate and present a non-intrusive and robust HCI system as an alternative hands-free solution to individuals unable to use a mouse. In conjunction with tracking the user’s head in real-time, various useful mouse functions, such as left, right and middle-clicking and scrolling can be implemented by using the user’s voice, facial gestures or keeping still in a point of interest to initiate them, thus ultimately letting the user control their mouse effectively without using their hands.

1.2 Aims & Objectives

This kind of application will deem most beneficial towards individuals that are unable to use a conventional mouse due to physical and motor impairments but have retained the ability to rotate their head. The major aim of this dissertation is to give these individuals a robust and hands-free alternative to a conventional mouse by using only an inexpensive web camera and the motion of their heads. Hence, this dissertation seeks to examine, implement, and evaluate various CV techniques that track the rotation of the head effectively, based on facial features extracted from the user’s head, to simulate the motion of the mouse. Furthermore, various methods used to trigger mouse functions such as facial gestures, voice or hovering over a point of interest for a certain period will be presented, implemented, and evaluated.

The objectives that this dissertation aims to tackle are the following:

- Investigate, implement, and evaluate various CV algorithms used to track the position of the human head in real-time

- Research, implement and evaluate various methods used as means to trigger useful mouse functions

- Implement an easy-to-use and accessible application that:
    1. Allows physically impaired users to control their mouse using only their head movement
    2. Uses various techniques to trigger a mouse function, such as using the user’s voice or facial gestures
    3. Has an easy-to-use GUI for using the system

2. Background

2.1 Face Detection

To be able to track the rotation and movement of the human head, a vital prior phase is to first detect and classify a human head from a single frame. Face detection is a widely discussed subject matter in the field of Computer Vision over these past two decades and has proven to be a challenging task due to the nature of the human head. In particular, the challenging aspect of face detection is the varying shapes and poses those human faces can appear in images and additionally the task of distinguishing a human face from different backgrounds and illuminations [6]. Opportunely, different implementations and extensions of face classifiers have been presented and evaluated in recent years. The underlying goal of a face classifier is to correctly detect and learn a human face in an unseen image, where the face classifier has been trained on hundreds, possibly thousands of given images containing human and non-human faces. In this sub-section, the two most fundamental algorithms used for face detection will be discussed and evaluated.

The first algorithm that is going to be discussed is by the work of Viola and James [7]. The authors in this novel paper presented an object detection machine learning approach, which achieves rapid face detection using boosted cascades of more complex classifiers. Instead of classifying the raw pixels from an image as features, the authors in this paper proposed a novel method where features are represented as rectangular regions of the image. These features represented as rectangles hold pixel values within them, thus estimating if a particular window in the image contains a face. The reason behind this is that these Haar-like features are calculated considerably faster rather than calculating the pixels directly. In more detail, as described in [7], these rectangular regions are rapidly calculated using an integral image, which is an intermediate representation of the image. An integral image allows the sum of the values that these rectangles represent to be calculated using only four values, which considerably saves computational resources and time.

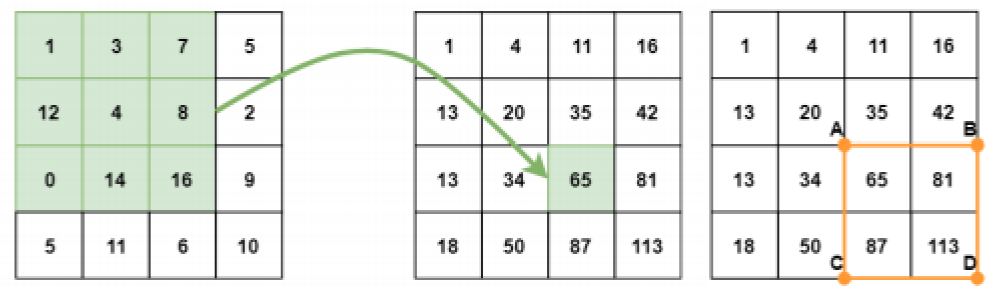


Figure 1. An integral image where the sum of a rectangle value are the top left rectangles (Source: [8]*)*

Secondly, Adaptive boosting (Adaboost), which is a machine learning algorithm, is trained to eliminate any redundant features from the image and keep only relevant features. Since an image sub-window can have over 180,000 rectangular features, the goal of Adaboost is to select only a small amount of very important and relevant features of the face. These best-selected features are then combined to form a different classifier of its own, where every classifier is responsible to detect a region of the face, such as the eyes, nose, or mouth. When combined, these "weak" classifiers are then able to classify a human face correctly and accurately.

Lastly, the authors in [7] also address the problem of distinguishing a region of the image where it contains a human face. It is assumed that only a very small window of the image will contain a face and the major portion of the image will not contain a face. Hence, they proposed a method called "Cascade Classifiers", a simple method used to check and evaluate whether the selected region of the image contains a face by discarding regions that are known to not contain one. Every selected region is passed through various stages and classifiers and discarded if one of these stages fail. If one region of the image passes all of the stages successfully, then it is assumed that the aforementioned region contains a face. Thus, using these techniques discussed above, the authors achieved rapid face detection from an image.

The second fundamental algorithm that is widely used today to detect human faces, is by the work of Dalal and Triggs [9]. The authors in this paper outperformed the work in [7] by presenting Histograms of oriented gradients (HOG) descriptors that robustly and rapidly capture facial features from a human face. The overview of this algorithm is as follows. Firstly, the input image is divided into equal regions or "cells". Moreover, the input image's colour and gamma are normalized, which happens to characterize the distribution of edges of the image rather well and removes any redundant regions of the image, such as the background. Afterwards, each cell holds a value that represents the orientation of the edges for that particular cell's pixels, which corresponds to a 1-D histogram. All histograms from every cell are then combined to compute a feature vector. Finally, the feature vector is then fed to a Support Vector Machine (SVM) with a linear kernel, which can correctly classify a face from an input image with significant accuracy.

Having discussed the importance of the Viola-James Algorithm and the HOG algorithm for detecting faces, the speed and accuracy of these two algorithms need to be discussed as well. Rahmad et al presented experimental results which compare and evaluate these two factors of both algorithms [8]. Specifically, the authors in this paper evaluated the two algorithms by training them with a dataset consisting of 300 images of different backgrounds and illuminations. The results that they presented demonstrate that although the V-J algorithm recognized frontal faces very well, it was prone to false positives, especially with faces occluded with glasses or hats. On the other hand, the HOG algorithm was proven to be 5% more accurate on very small faces and detected them relatively faster. For this dissertation's application, this minor increase of accuracy is slightly significant, since we can assume that the user will be situated near to their computer, thus expecting a large amount of the image to contain a face. However, the significance of HOG for this dissertation comes from its improved speed since the application will make use of face detection in real-time, meaning multiple images per second. Hence, the concluding remark for this sub-section is that this dissertation will use the HOG algorithm for face detection due to its improved speed.

2.2 Facial Landmark Extraction

Another important concept for this dissertation that is deemed essential to discuss is the extraction of facial landmarks. Thus, it is required to establish a solid review for this topic. In Computer vision, facial landmark extraction is a fundamental task used to map key facial features in a face using points as coordinates [10]. Several applications make use of facial landmark extraction, such as facial animations, 3D face reconstruction and facial expression analysis. Briefly summarizing, facial landmark extraction aims to detect the overall structure of a face and then subsequently to map each point of the structure with coordinates, inside a rectangle where a face has been detected. This is achieved by training a machine learning algorithm or a Convolutional Neural Network (CNN) with a dataset comprising of multiple faces with already labelled mappings. Ultimately, the CNN will be able to learn and correctly classify the structure of the face and key facial characteristics in new unseen faces in varying poses, such as the corners of the eyes, nose, chin, and mouth. Facial landmark extraction allows us to also detect eye blinking and winking of each eye, as well as detecting the opening and closing of the mouth, using a certain threshold. The points that represent the corners of the eyes are detected using facial landmark extraction and then subsequently, the Euclidean distance between those points is found, thus effectively detecting an eye blink. Detecting eye blinks can seem beneficial for applications such as systems that assess the drowsiness level of a driver, but for this scenario, it can also be beneficial for triggering certain events, such as mouse clicks. Specifically, this can be done with the Eye Aspect Ratio (EAR) formula, which was introduced by Cech et al [11] :

$$EAR = \frac{\|p2 - p6\| + \|p3 - p5\|}{2\|p1 - p4\|}$$

Where *p1,…,p6* are the corner coordinates of the eye, as shown in Figure 2

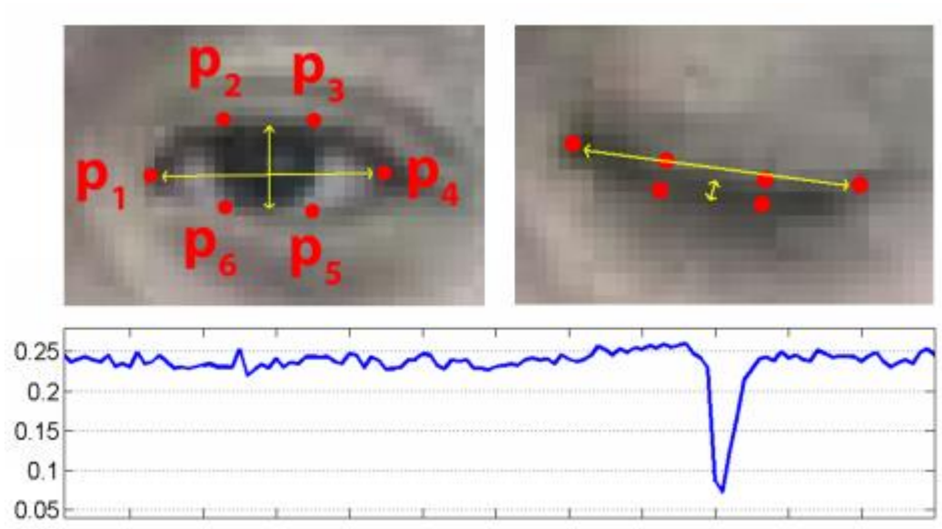


Figure 2. The 6 points of the eye, which are detected using facial landmark extraction. A blink is detected when the EAR value between those 6 points falls below 0.1 (Source: [11]*)*

As seen in figure 2, the value of the Euclidean distance when the eye is open is almost a constant value (0.25). However, when the eye is closed, the said value drops below 0.1 or a certain defined threshold to detect a blink. Hence, by using the EAR formula, we can identify whether the left or right eye is open or closed. The inverse function of the EAR can be also done on the mouth (which is called Mouth Aspect Ratio), detecting whether the Euclidean distance of the points of the mouth represents an open or closed mouth. With this having been said, different useful functions can be implemented to trigger an event, such as right or left-clicking, using the appropriate eye blinks, or opening of the mouth.

However, there is a key disadvantage when using facial features to generate mouse events. The main problem, as Nabati et al described in their work, is that facial features in an input image (or multiple images) exist within only a very small area of the image [12]. Hence, it is therefore assumed that the image must be of very high resolution, which typical consumer-grade web cameras typically do not possess yet. Other problems arise also, such as the occlusion of facial features when the head is rotating in odd poses. For blink recognition to work as intended, a clear frontal image of the face is required. Specifically, since in this application we assume that the user will be turning their head, where, for example, the left or right eye can be occluded (hidden) in an input image. Consequently, the system will not be able to recognize an eye blink if the user is rotating their head in a pose where one of the eyes is occluded. Lastly, different individuals have different Eye Aspect Ratios. Thus, it is implied that it would be very difficult to find a common threshold value to recognize an open eye from a closed one, that is suited for every user.

2.3 Real-Time Head Pose Estimation

To better understand how real-time head pose estimation can be applied to this system, a literature review about this topic needs to be established. Head pose estimation has been widely utilized in a vast majority of applications in recent years, such as gaze modelling, driver attention systems, behaviour analysis and other AI applications, yet it has proven to be a challenging task. Describing it on a high level, head pose estimation is the task of predicting the pose of a human head by identifying the degrees of its position in 3D space [13]. More in detail, the goal of head pose estimation is predicting the three dimensions of the rotation of the head in degrees, which are the yaw (y-axis), pitch(x-axis) and roll(z-axis), as shown in figure 3. This can be achieved by training a CNN with multiple input images of different human heads in different poses. Firstly, for head pose estimation to work, it is assumed that face detection has been implemented. Subsequently, the 2D facial landmarks and the overall structure of the human face are identified as described above, in section 2.2. By identifying these 2D points, the 3D points of a model head are defined and subsequently mapped to the corresponding 2D projection points of the image, to provide to us the rotation of the head (yaw, pitch and roll) [14]. A 3x3 camera matrix containing the approximation of the radial distortion, focal length and the optical centre of the web camera is defined. For this case, we assume that there is no lens distortion. By having the 2D and 3D points of the head, as well as the camera matrix, we can therefore calculate the rotation and translation vector. Since, we now know the world coordinates of the image, which are the 3D feature points, we can convert them to camera coordinates, which can be found using the rotation and translation vector. Firstly, we convert the rotation vector into a 3x3 rotation matrix **R** using decomposition, and then calculate the location of the three dimensions (X,Y,Z) in the camera coordinate system using the following equation as described in Nabati et al [12] :

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = T_{CH} \begin{bmatrix} X_H \\ Y_H \\ Z_H \\ 1 \end{bmatrix}$$

$$T_{CH} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

*Where R11, R33 are the parameters of the 3x3 Rotation Matrix **R** and Tx, Ty, Tz represent the three parameters of the translation vector **T***

The final results are the XYZ coordinates of the head converted into the camera coordinate system, expressed in radians. A simple conversion to degrees will provide you with the three Euler's angles of the human head of an input image, or in real-time, as a video. The conclusion drawn from this is that these coordinates can be directly mapped as 2D mouse coordinates (ultimately discarding the Z-axis), which consequently can let the user move the mouse with only using their head.

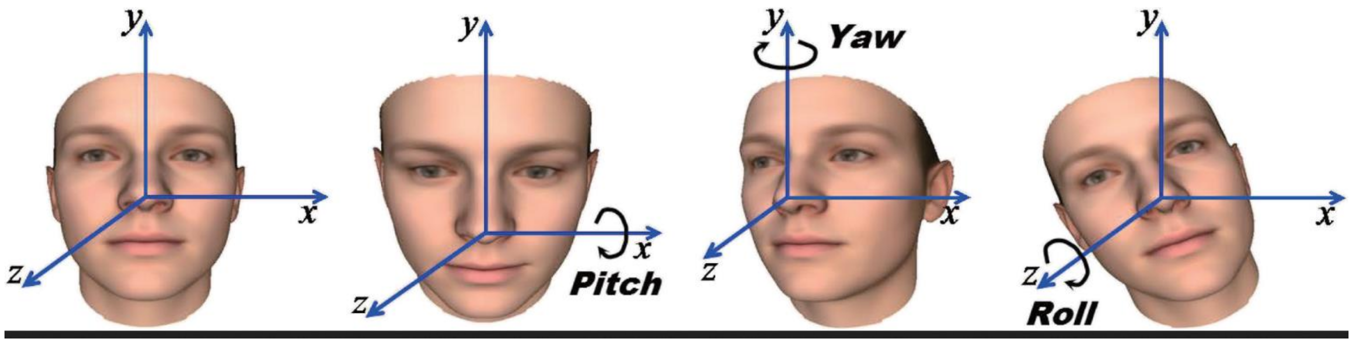


Figure 3. The Pitch (X), Yaw(Y) and Roll(Z) rotation of a human head (Source: [15]*)*

Is it however worth mentioning the drawbacks of head pose estimation, which Ruiz et al discussed in their work [13]. The authors in this paper discuss the limitations that come with estimating the pose of the head with approximate values. They discuss that the performance of the head pose estimation algorithm can be solely reliant on the performance of the identified facial landmarks. Thus, by approximating the 3D position of the captured facial landmarks, and then projecting them into a 2D image plane, it can produce unreliable and non-robust results. Additionally, in real-time, the rotation of the head using head pose estimation is rarely the true value, but rather revolves around the true value, which in consequence, can cause jitter and compromise precision. In this system, the precision of the head-driven mouse is of vital importance, since we have to ensure that the user can zero in on small buttons and text and have an overall smooth experience.

2.4 Lucas Kanade Optical Flow Tracker

Quintessentially, a vital topic for the implementation of this head-driven camera mouse system is the Lucas Kanade optical flow tracker algorithm and especially its sparse variant. This novel algorithm was developed and presented by Bruce D. Lucas and Takeo Kanade in the early 1980s, as a method to visualize and track the motion of an object in consecutive frames (video) [16]. Recent breakthroughs in research enhanced and expanded this algorithm and are now widely used in a vast plethora of applications, such as object detection, segmentation, video stabilization, and tracking. In Particular, the goal of the LK algorithm is to capture the motion or change of an object from one frame to another, thus effectively tracking its movement robustly. An object, or in this scenario, a face is detected, then subsequently the LK algorithm can predict the displacement of the face in the next frame, relative to the camera [17]. There are two variants of this algorithm, which are the sparse and dense optical flow trackers. The Sparse variant relies on a sparse set of pixels (or edge features) of an object which is then tracked frame by frame, to find the velocity of the motion of the tracked object. The Dense variant of the algorithm tracks the entire flow vector, meaning all of the pixels within the object, which gives more accuracy, albeit it being more computationally expensive rather than the sparse variant.

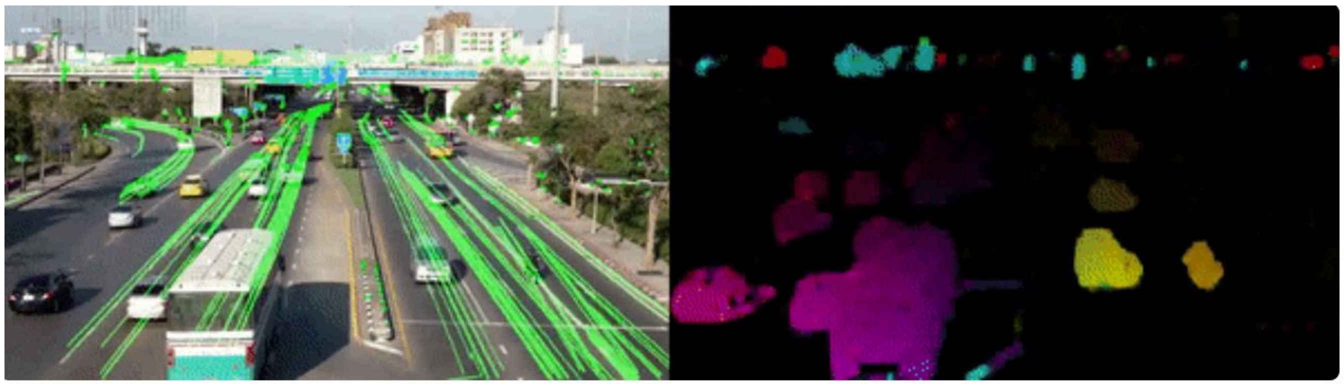
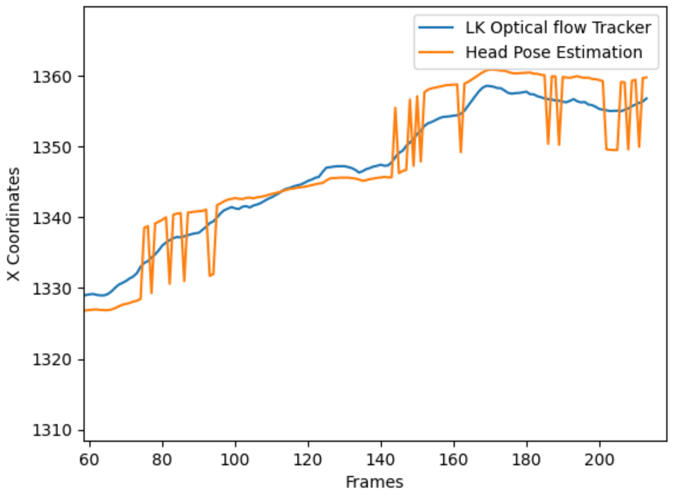


Figure 4. Difference between the Sparse (left) and Dense (right) optical flow tracker (Source: [17]*)*

The problem of optical flow tracking can be expressed as the image intensity ***I*** which contains the ***x,y*** coordinates of the feature (point) and the time ***t*** between consecutive frames. The displacement, meaning the difference of the old position of the feature and the new position in space ***dx, dy*** is defined. Thus, if we have the current image's coordinates ***x,y*** and move their position to the new position of the feature (***x + dx, y + dy***), we ultimately have the new image's coordinates and the flow of motion of the object is identified [17]. It is however assumed that the time ***t*** between frames cannot be 0, as the objects are then displaced, meaning that it is effective only on relatively slow motion. Secondly, we also assume that the lighting remains constant between consecutive frames. By using a feature detector algorithm, such as the Shi-Tomasi edge corner detection, we can successfully identify useful features and subsequently track them, or extract them from a facial landmark descriptor [18]. Lastly, to circumvent the problem of only tracking small motions in consecutive frames, a pyramidal method of LK can be implemented. The pyramidal extension to LK scales down large motions and removes any small motions, for the algorithm to distinguish all motion, whether it is fast or slow.

To demonstrate how the LK sparse optical flow algorithm compares to tracking with head pose estimation, a preliminary experiment was conducted. The two algorithms were compared with the same test video of a person sitting completely still, to evaluate the precision of the tracking between the two algorithms. Specifically, the same set of facial landmarks has been used on both algorithms and were subsequently mapped as mouse coordinates. Figure 5 reveals the precision of the X and Y coordinates between the two algorithms on consecutive frames.



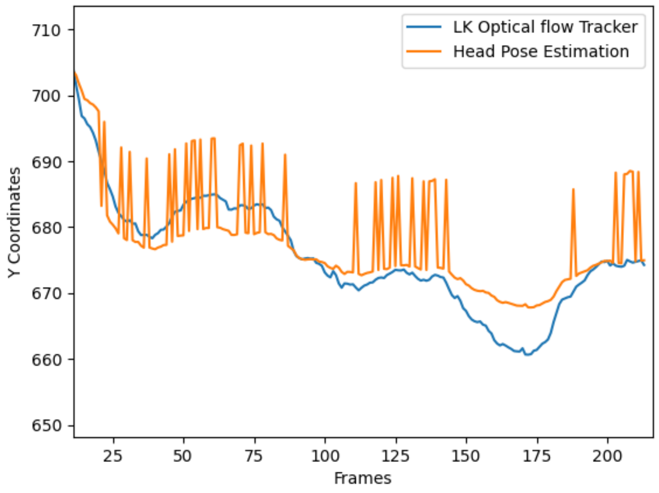


Figure 5 The precision of the X (left image) and Y (right image) coordinate of the mouse using the LK algorithm and head pose estimation for tracking the head

As Figure 5 shows, we can acknowledge that the LK Optical flow tracker gives rise to better results. As discussed in the previous sub-section, tracking with head pose estimation gives approximate results, resulting in severe jitter and noise. On the other hand, the LK algorithm outperforms head pose estimation in terms of tracking, which gives significantly better results in terms of precision and smoothness of the overall tracking of the facial features. We can also claim that head pose estimation is mostly used for applications that might require an approximate position of the head, such as the person looking left or right. While the LK algorithm might not give us the approximate rotation of the head in degrees, it can provide us with the exact position of a tracked facial feature in consecutive frames. Thus, the concluding remarks are that the LK algorithm will be used for tracking the user's facial features since it provides us with robust accuracy and precision in terms of tracking.

2.5 Mouse Interpolation

We have discussed how tracking can be implemented using the LK algorithm with a set of facial features and subsequently map the position of those facial features as mouse coordinates for the user to be able to control the mouse using their head. However, what is yet unknown for the system is the amount of movement the mouse can achieve with respect to the screen resolution and camera resolution. Specifically, we have to relate the movement of the tracked facial features from the camera to the screen. Hence, in this sub-section, the interpolation of the mouse coordinates relative to screen and camera space will be discussed. Let us assume that we have a set of points or coordinates and the range of those points are known. Linear Interpolation can estimate a new set of data points from a set of already known points, or coordinates and find an unknown value between two known values [19]. Putting it differently, using the camera's width and height dimension and also the screen's width and the height dimension, we can convert the range of X and Y coordinates of a tracked point to be able to cover the entirety of the screen. In this case, the interpolation function can be expressed as follows [20]:

$$y(x) = y1 + (x - x1)\frac{(y2 - y1)}{(x2 - x1)}$$

Where:

- y is the interpolated coordinate
- x is the known coordinate
- x1 and y1 are the coordinates below the known x coordinate
- x2 and y2 are the coordinates above the known x coordinate

For this scenario, the width of the camera dimension and the width of the screen dimension in pixels can be applied as y2 and x2 to find the interpolated X coordinate and 0 for x1 and y1, since we assume that they are below the known value. To find the interpolated Y coordinate, the same principle can be applied using the height of the camera dimension and the height of the screen dimension. Ultimately, this will allow the user to control the mouse with their head and the motion will be in respect to the entirety of the screen, thus allowing the mouse to reach the whole screen.

2.6 Related Work

This paper focuses on the implementation of an assistive HCI system that lets the user control the mouse only using their head movements. Over these past few years, different implementations have been presented for this problem. In this sub-section, related work with similar implementations will be discussed.

Tu et al, in their work, presented a camera mouse system [21]. The authors in this paper demonstrate the implementation of a real-time head tracking system, which uses mouth expressions derived from a facial landmark descriptor as functional mouse events. They have used a 3D model of a face which captured the facial features of a real face in real-time. Moreover, they have used a facial landmark descriptor to portray changes in the expressions of the mouth. These various mouth expressions then correspond for the use of mouse events, such as right and left clicking, dragging, and scrolling. In their experiment, they have evaluated the accuracy of their system by determining the localization error in pixel of a mouse click in a crosshair ten times. Their results concluded in a good overall accuracy of the system; however, the reliability of the head tracking module was lacking very precise results.

Fu et al presented another camera mouse implementation which expanded the work above [22]. Particularly, contrary to the method above, this implementation focused on the 2D points of the human head. The authors in this paper discuss the use of real-time head pose estimation and face tracking to implement an assistive HCI system for the physically impaired. Further, in detail, it calculates the pitch, yaw, and roll motion of the head, as discussed in section 2.3, and used 2D head tracking of the head to simulate mouse motion. Moreover, they have used head tilts as a way to initiate useful mouse events without using hands. Experimental results from this paper indicated robust face tracking and detection, which even succeeded in partial face occlusion, extreme movement and jumping and blurriness. However, the authors discussed some drawbacks in this implementation, such as the cursor not providing enough smoothness for users to press small buttons, due to the low amount of FPS that the web camera provided and also some lighting conditions that proved sub-optimal. According to user judgement, the precision of the head pose estimation was not significantly precise.

To conclude this sub-section, similar implementations and assistive systems have been presented in recent years. However, as discussed above, they use techniques to track the head such as head pose estimation, where it can produce unreliable results in terms of tracking. It is also worth mentioning that they also use facial gestures to initiate mouse events. The drawbacks that arose using this method were also discussed. Particularly, the occlusion of key facial features may compromise the robustness of the system. Opportunely, this system focuses on reliable head tracking using the LK algorithm, which gives robust results. For mouse events, since the use of facial gestures can provide unreliable results, other techniques can be used, such as using voice commands for mouse events, or keeping still for a certain amount of time to initiate a mouse event. These techniques and the general implementation of the system will be discussed more in detail in the next section.

3. Methodology

3.1 General Overview of the Method

In this sub-section, the general implementation of the assistive HCI system will be discussed and presented. Firstly, it is important to discuss the pre-processing pipeline of the system and how in particular the input image is processed throughout the system, for face detection, facial landmark extraction and LK flow tracking to take place. Figure 6 demonstrates the process of the system to achieve mouse motion and how an input image is manipulated from the web camera.

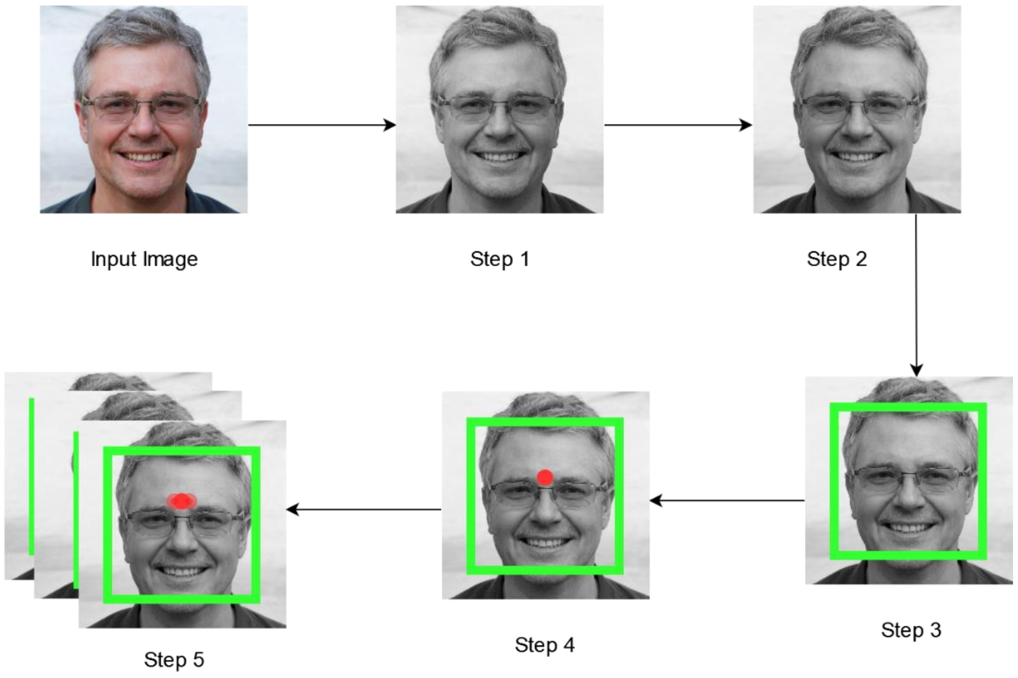


Figure 6. The processing pipeline of the system (Person generated with a GAN [23]*)*

Firstly, a raw input frame is received from the web camera using a separate thread from the main one, for improved performance.

Subsequently, step 1 converts the frame to grayscale, removing the colour of the image. This is since face detection and facial landmark extraction do not require colour to function, thus the colour of the image, in this case, is redundant. By removing the colour of the raw image, we can see an increase in performance in terms of computational speed since redundant information of the image is removed.

Step 2 consists of making the resolution of the raw frame significantly smaller and flipping the image vertically. Typical web-cameras output images in resolutions ranging from 1280x720, up to 1920x1080. These large resolutions substantially impact the speed of the system, due to the large image space that face detection and facial landmark extraction require to search, ultimately making each frame to process much slower. Hence, we reduce the image resolution significantly since face detection and facial landmark extraction can still function well, even on faces at 80x80 pixel resolution [24]. We have found that the resolution of 420x200 was the most ideal balance between facial accuracy and speed. Moreover, the image is then flipped vertically to simulate a "mirror-like" image. Web cameras tend to output images that do not represent the natural direction of movement and is counterintuitive. Since we want the user to control the mouse using natural movements, i.e. looking left to move the mouse to the left side and looking right to move the mouse to the right side, inverting the image is an essential step.

Step 3 implements the HOG face detection algorithm on the pre-processed image to detect the user's face. This will generate a face bounding rectangle where it contains the face of the user as shown in figure 6. Using the X and Y coordinates of the face bounding box in the image, as well as the width and height of the bounding box, we can identify exactly in which region of the image the face is located.

Step 4 contains the process of facial landmark extraction. Within the face-bounding box that we have identified in step 3, 68 facial features of the face are found using the already pre-trained model "shape_predictor_68_face_landmarks" [25]. This will give the general structure of the face and identify the corners of the eye, chin, nose, and mouth. However, we have only used one point of the face, which is located in the forehead, particularly between the eyes of the user. This is because the point between the eyes of the user represents an ideal middle position of the face, where it can robustly track the rotation of the user's head in all directions.

Finally, when all of the prior steps are completed, step 5 implements the LK sparse optical flow tracker in the defined point between the user's eyes for consecutive frames in real-time. Particularly, we have identified the single point in step 4 within the face bounding box. Subsequently, we give as parameters the location (of X and Y coordinates) of the point from the previous frame to the LK algorithm, and the current frame. The LK algorithm will then subsequently predict the position of the point in the current frame. The previous point and the inferred previous point of the frame are then compared to evaluate the error difference. If the error is less than 1 pixel, the points are valid, otherwise, the points are discarded. Finally, the final points are then interpolated as discussed in section 2.5, and then successively mapped as the X and Y coordinates of the mouse position respectively. This process is iterative and repeats for every input frame in a series of frames, to effectively track the position of the user's head and ultimately let the user control the mouse with their head, in real-time.

This concludes the processing pipeline of the proposed algorithm to capture the motion of the head robustly and subsequently map them as mouse coordinates, ultimately letting the user move the mouse cursor with their head. However, what is yet to be discussed is how the user can trigger useful mouse events, such as left and right-clicking, double-clicking, scrolling and middle mouse click. Since we do not want to constrain the user with only one method of triggering mouse events, we introduce three distinctive modes for every type of user, which are called "Hover Mode", "Voice Mode", and "Motion-only Mode". These modes will give the user the freedom of choice to trigger mouse events in different ways, according to their preference, and they will be discussed more in detail in the following sub-sections.

3.2 Hover Mode

In this sub-section, the first mode will be discussed, which is called “Hover Mode”. Specifically, this mode can let the user trigger certain mouse events by staying completely still on a point of interest. After a certain amount of time that the user stood still, a mouse event will get triggered. A small helper GUI located in the top left of the screen, as seen in figure 7, will let the user choose their preferred mouse function, such as left-click, right-click, double click, and middle mouse click, which can be also used for scrolling.

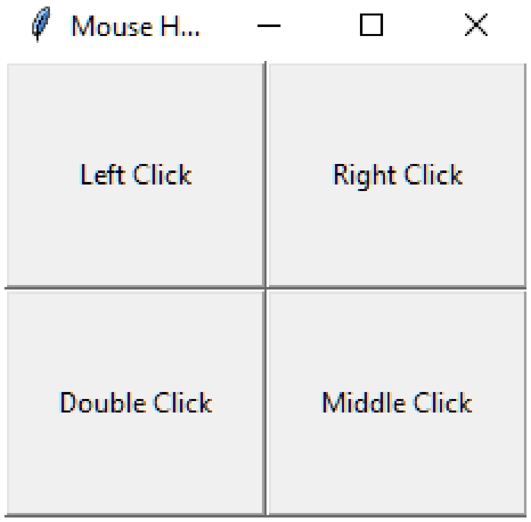
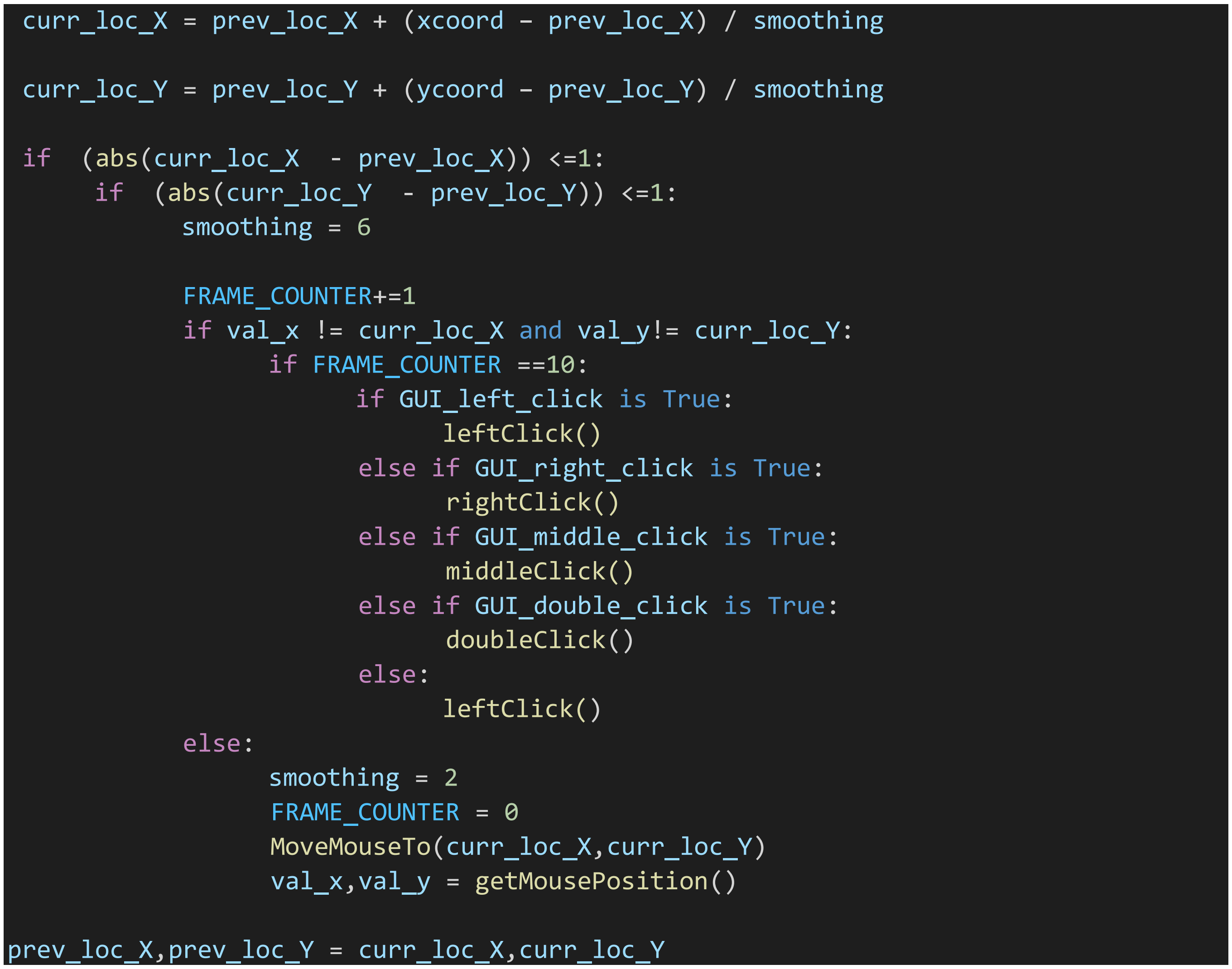


Figure 7. The small helper GUI in “Hover Mode”, where the user can choose their preferred mouse function

In Particular, we have to ensure that when the user stays still, the cursor has to be very stable, ignoring any very small movement that the user might do when trying to stay still. This can be achieved by applying smoothing techniques to the X and Y coordinates of the mouse, which will give very precise mouse movement using the head but will sacrifice responsiveness in result. Keeping that in mind, we keep the smoothing value at 2, and only increase the smoothing value at 6 when we identify that the mouse is not moving. We have found that a constant smoothing value of 2 gives ideal results. Below is the pseudocode of this process:



```
 curr_loc_X = prev_loc_X + (xcoord - prev_loc_X) / smoothing

 curr_loc_Y = prev_loc_Y + (ycoord - prev_loc_Y) / smoothing

 if  (abs(curr_loc_X  - prev_loc_X)) <=1:
     if  (abs(curr_loc_Y  - prev_loc_Y)) <=1:
           smoothing = 6

           FRAME_COUNTER+=1
           if val_x != curr_loc_X and val_y!= curr_loc_Y:
                 if FRAME_COUNTER ==10:
                       if GUI_left_click is True:
                             leftClick()
                       else if GUI_right_click is True:
                             rightClick()
                       else if GUI_middle_click is True:
                             middleClick()
                       else if GUI_double_click is True:
                             doubleClick()
                       else:
                             leftClick()
           else:
                 smoothing = 2
                 FRAME_COUNTER = 0
                 MoveMouseTo(curr_loc_X,curr_loc_Y)
                 val_x,val_y = getMousePosition()

prev_loc_X,prev_loc_Y = curr_loc_X,curr_loc_Y
```

Figure 8. The Pseudocode of the "Hover-mode" functionality

Firstly, we apply smoothing to the interpolated coordinates by getting the location of the mouse in the previous frame and summing it with the difference of the coordinates given by the LK algorithm, and the previous location of the mouse. We then divide it by 2 (or a variable smoothing value), which gives us a more accurate position of the mouse, but with less responsiveness. As to not sacrifice responsiveness in the overall system, we identify when the mouse is still or is very slightly moving. This can be achieved by finding the absolute difference between the mouse location in the current frame, and the location of the mouse in the previous frame. If the result is equal to or less than one pixel, then we can assume that the mouse is still. Hence, we increase the smoothing value to 6, to achieve a more accurate mouse estimation on small movements. A frame counter is initiated where it counts the number of frames. In this stage, if the mouse remains still for 10 consecutive frames (or a variable frame number), we assume that the user wants to perform a mouse event on a point of interest. The mouse event is then initiated depending on which button the user has clicked on the helper GUI (figure 7). When the user suddenly performs rapid movement to go to another point of interest, the smoothing value goes back to 2 and the process repeats as needed.

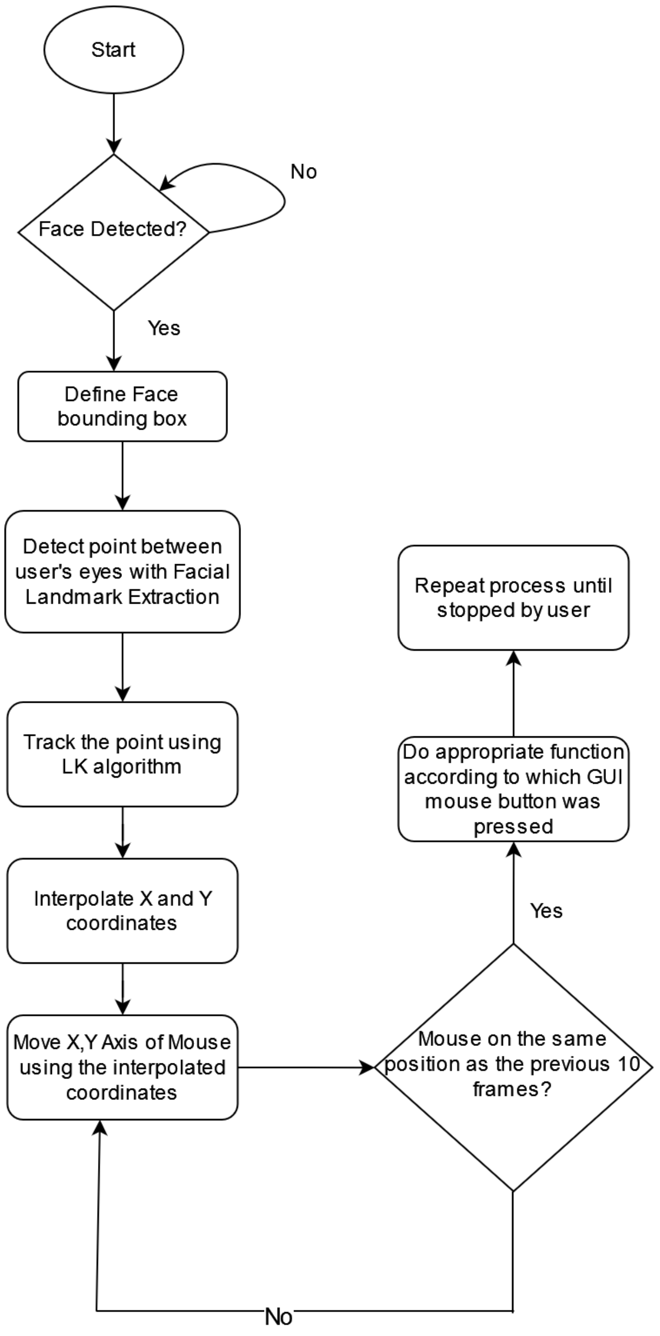

Start
Face Detected?
No
Yes
Define Face bounding box
Detect point between user's eyes with Facial Landmark Extraction
Track the point using LK algorithm
Interpolate X and Y coordinates
Move X,Y Axis of Mouse using the interpolated coordinates
Mouse on the same position as the previous 10 frames?
Yes
Do appropriate function according to which GUI mouse button was pressed
Repeat process until stopped by user
No

Figure 9. The high-level flowchart of "Hover Mode"

3.3 Voice Mode

An alternative method to using the proposed system for triggering mouse events is with the use of "Voice Mode". Voice mode, as the name entails, uses voice recognition to detect various mouse events that the user has said in their microphone. Below is the table with voice commands that correspond to mouse events.

Voice Command	Function
"Left"	Left mouse click
"Right"	Right mouse click
"Scroll"	Middle mouse click (Can be used for scrolling)
"Double"	Double left mouse click
"Exit"	Exits the application

Table 1. The voice commands that the user can say to operate certain mouse functions

Voice recognition can be a challenging task and various issues can arise, such as background noise, voice distortion, different accents, and word ambiguity. To circumvent these issues as best as possible in this proposed system, some solutions have been implemented.

Firstly, we identify the default microphone of the user's PC. Using the default microphone, we then adjust a threshold to remove ambient noise. Particularly, this will filter out audio that is detected to be background noise and will only focus on the voice of the user. The system then gives a 3-second window for the user to say their preferred voice command. If a voice command is not recognized by the user, the system will then repeat this process as needed. We then circumvent the word ambiguity problem by providing multiple words for each command that can be a homonym, meaning a word that is said the same way as another but has a different meaning altogether. For example, the system recognizes the word "write" as the "right" command. However, this solution for word ambiguity is not entirely bulletproof. Some other requirements are also needed for this mode. Particularly, this mode requires a constant internet connection since audio is sent to Google's cloud server for processing and subsequently posts back the result of the voice command. Another notable requirement is that this mode works optimally with a headset microphone since it is directly situated near the user's mouth, resulting in clearer and more interpretable audio. A high-level flowchart of "Voice Mode" can be seen in figure 10.

This concludes the implementation of "Voice Mode" which aims to provide to the users an alternative way to use the system according to their preference. The third mode called "Motion-only Mode" will not be discussed in detail since it only uses the motion of the system. Specifically, "Motion-only Mode" was implemented for individuals that may already have other assistive mouse peripherals, such as a foot-controlled mouse clicker for triggering events and want to use those specific peripherals in conjunction with the mouse motion of the system using their head movement. Additionally, this mode can give the freedom of choice to the user, as to be able to use it with their preferred voice recognition software.

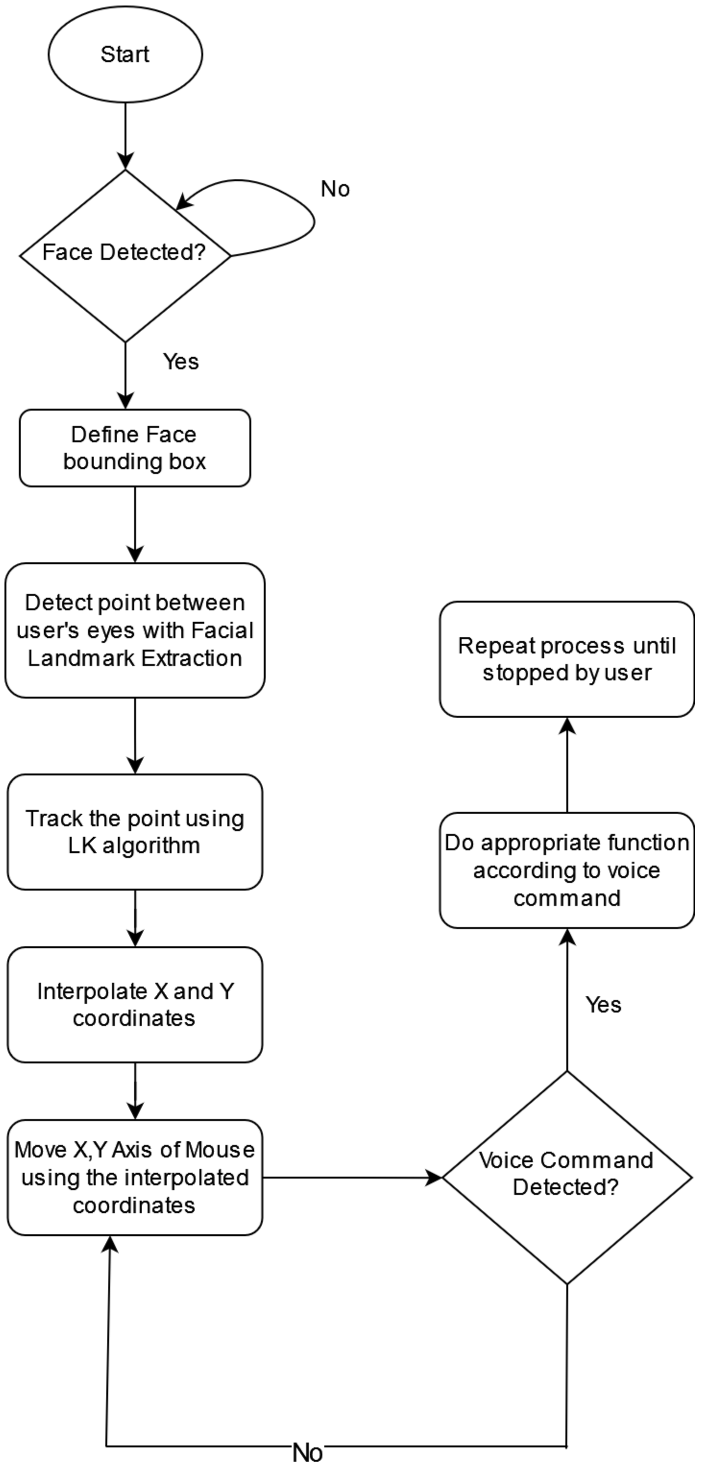
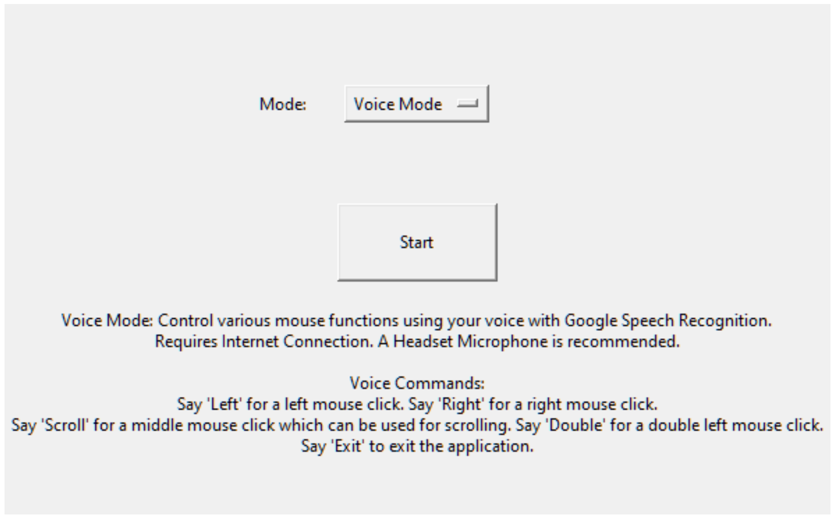

Start
Face Detected?
No
Yes
Define Face bounding box
Detect point between user's eyes with Facial Landmark Extraction
Track the point using LK algorithm
Interpolate X and Y coordinates
Move X,Y Axis of Mouse using the interpolated coordinates
Voice Command Detected?
Yes
Do appropriate function according to voice command
Repeat process until stopped by user
No

Figure 10. The high-level flowchart of "Voice Mode"

3.4 Technologies Used

This sub-section will describe the tools and libraries that were used to implement the system. Nowadays, a plethora of tools, libraries and plugins are available and open-source, which provides the opportunity to implement a task or an algorithm that was considered to be challenging previously, to be implemented relatively easy.

The system was developed using Python 3.8. This is because the Python programming language and its ecosystem are considered powerful and versatile, and opportunely offer an abundance of AI and CV libraries that are essential for this system's implementation. For image processing tasks, the OpenCV library for Python has been used, since it offers everything related to capturing and manipulating an image from a web camera [26]. Moreover, Dlib, a C++ library that also offers a Python wrapper was used for the implementation of face detection and facial landmark extraction [27]. The reason Dlib was used rather than OpenCV for these tasks is that Dlib offers the HOG face detection algorithm, where OpenCV's face detection is using the VJ algorithm, which its drawbacks were discussed in the background section. Facial landmark extraction was implemented with Dlib as well, using a pre-trained 68 facial landmarks model, to identify the point between the user's eyes [25]. For the implementation of the LK sparse optical flow tracker, OpenCV offers a robust function for this algorithm, which also implements its pyramidal method [28]. After the X and Y coordinates have been interpolated, PyautoGUI, a scripting library is used to move the mouse programmatically and was also used to trigger relevant mouse events [29]. For the implementation of voice recognition, Google's speech recognition API and engine have been used to provide a robust and real-time transcription of the user's voice, to execute mouse events as rapid as possible [30]. Finally, for the creation of the GUI in "Hover Mode" and the creation of the main User Interface where users can choose their preferred mode, Tkinter has been used, which is the definitive GUI library used within Python [31].



Voice Mode: Control various mouse functions using your voice with Google Speech Recognition.
Requires Internet Connection. A Headset Microphone is recommended.

Voice Commands:
Say 'Left' for a left mouse click. Say 'Right' for a right mouse click.
Say 'Scroll' for a middle mouse click which can be used for scrolling. Say 'Double' for a double left mouse click.
Say 'Exit' to exit the application.

Figure 11. The main User Interface of the system, where users can choose their preferred mode using a dropdown menu

4. Evaluation

After describing the implementation of the proposed system in the section above, an essential step is to evaluate the usability of the system, including its robustness, speed, and precision, to judge whether the system can be a viable option for real use.

The system is evaluated on a standard Desktop PC, using an Intel i7 920 2.6 GHz CPU and 8GB of RAM. It uses the Logitech C270 web camera mounted on top of the screen, which is capable of 720p resolution at 30 Frames per second, under optimal lighting conditions. The PC uses the Windows 10 Operating system. The system processes each frame at 420x200 resolution. Under optimal lighting conditions, the system at full-use processes at around 24 Frames per second. The CPU usage during the use of the system is 22-24% on average.

Similar to an experiment in [22], we first test the system in extreme scenarios, to measure its robustness. Specifically, the system is tested on poor lighting conditions, odd poses, hand occlusion, blurriness due to fast movement, and partially hidden faces. The green square represents the face detection, and the red point represents the tracking using the LK algorithm. Figure 12 demonstrates some frames captured from this experiment.

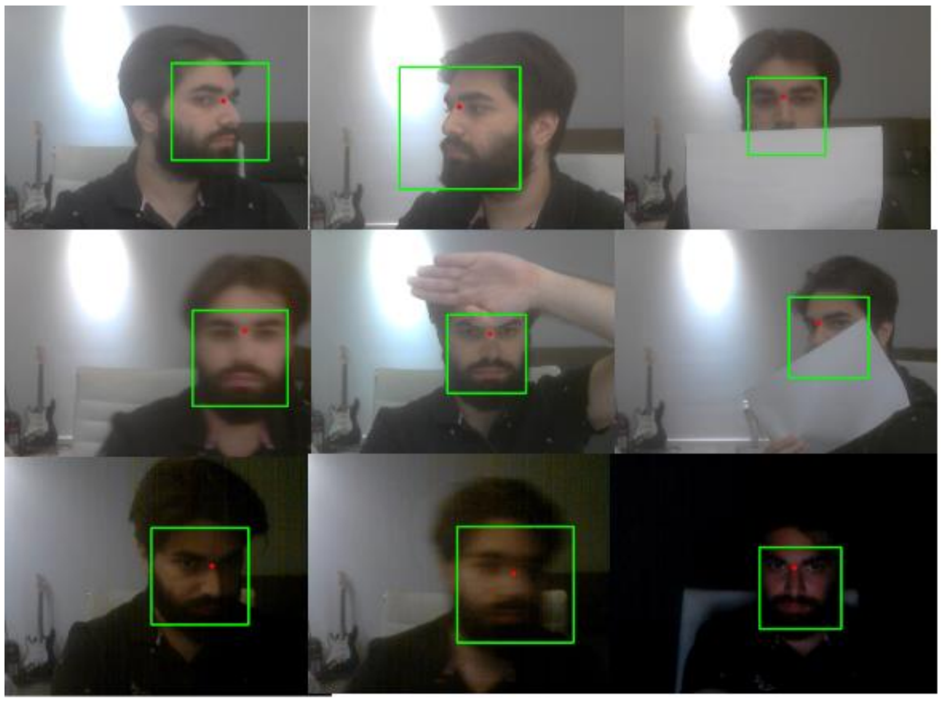


Figure 12. The system tested in extreme scenarios, such as hand and object occlusion, extreme head rotation, blurriness due to rapid movement and poor lighting conditions

As seen in figure 12, the system succeeds at maintaining reliable head tracking performance even when tested against non-ideal scenarios. The system will only fail at tracking the point when the face is fully occluded, or the user has turned more than 90 degrees from the web camera. If the face is lost from the frame and the system stops tracking the head, the reinitialization of the head tracking when a face is found is instantaneous. The suggestion drawn from this experiment is that the system can be used in a variety of conditions since it provides robust tracking in different illumination settings. However, it was observed that the system might suffer in performance on frames with a very bright background.

The second experiment aims to evaluate the usability speed and fluency of the system between various users, against a conventional PC mouse. Specifically, this experiment consists of a set of six numbered rectangles in ascending order, which is shown on the screen. The rectangles are in a pre-defined position, which is unknown to the participant. Different participants are tasked to hover over the numbered rectangles to make them disappear as fast as possible. However, participants have to hover over the rectangles in an ordered manner to make them disappear. For example, the rectangle labelled “2” cannot disappear if the rectangle labelled “1” has not. This experiment will be repeated five times for each participant (five times using a regular mouse and five times using the proposed system), using rectangles with a different position each time. However, the order of the different positions of the rectangles within these five times will be the same for all participants. Ultimately, this experiment aims to compare the time the participant took to make all the rectangles disappear using a regular PC mouse versus using the proposed system. Figure 13 shows an example of this experiment.

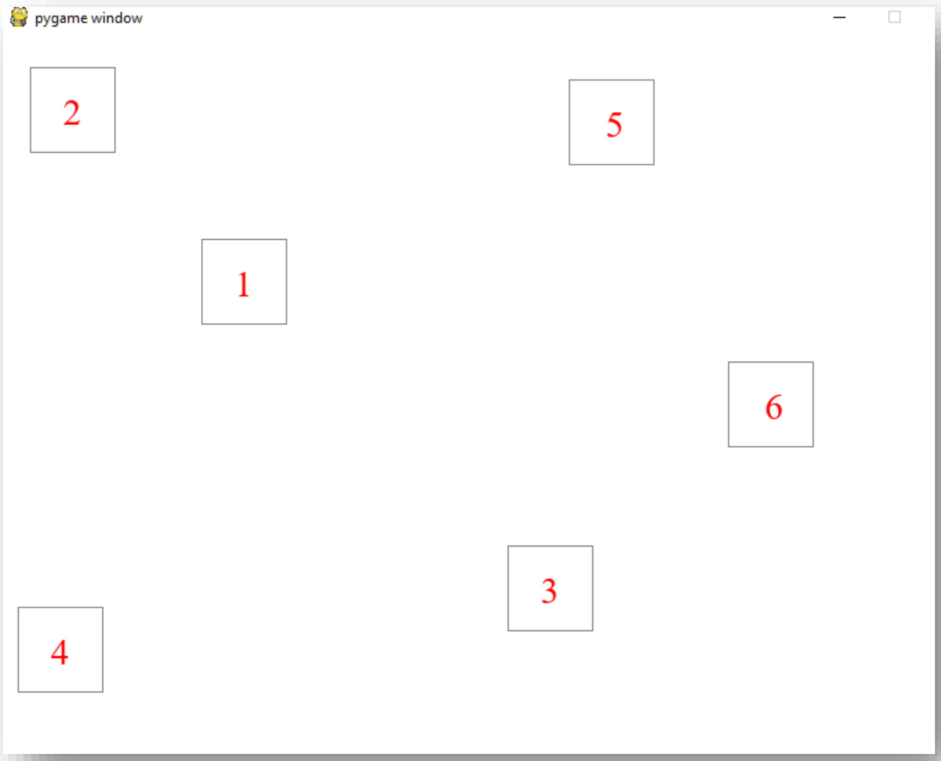


Figure 13. An example of the experiment measuring usability speed, consisting of six rectangles, where a participant is timed to make them disappear by hovering over them as fast as possible

The experiment was conducted on six different non physically impaired participants. Moreover, we have used the “Motion-only” mode of the system for this experiment. Figure 14 illustrates the mean time of all participants for each phase (different positions of rectangles).

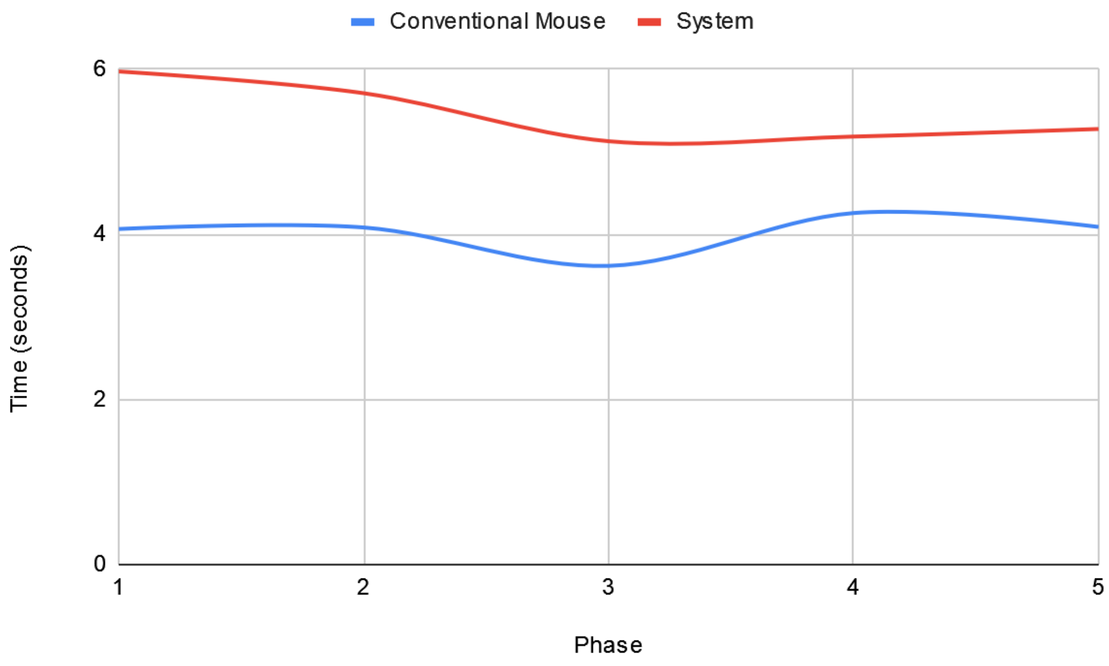


Figure 14. The mean time (in seconds) of all participants for each phase

The total mean time of all participants on all the phases using a conventional mouse was 4.023 seconds, whereas the total mean time of all participants using the system for the same set of experiments was 5.454 seconds. Hence, participants took 1.431 seconds or 35.57% more time to finish the experiments using the system. While it was self-evident that participants would take less time to finish the experiment using a conventional PC mouse, the results between the conventional mouse and the system are comparable. It was also observed that the participants would become more familiar and fluent using the system for the first two phases and as result, participants took less time to complete the following phases using the system. Moreover, it was observed that it took little to no explanation for the participants to understand how the system works, which draws the implication that the system had a perceptual nature for the users who have tried it. In conclusion, based on the results of this experiment, we can acknowledge the fact that the system can be used effectively as an alternative mouse for individuals unable to use a conventional PC mouse.

The third and final experiment will test the controllability of the system and specifically, the precision of the system. As the authors evaluated their system in [21], the precision of the system will be tested using a set of crosshairs with “Hover mode” and “Voice mode”. Specifically, with both modes, we test the precision of the system by trying to click on the absolute centre of the crosshair ten times using the system, and subsequently, measure the mean localization error of the clicked points. Contrary to the resolution used in [21], the resolution of the entire crosshair in this experiment was 400x200, and the experiment was conducted on a 1920x1080 screen, hence a much smaller window than typical UI buttons. The centre of the crosshair (red dot) was 25 pixels in diameter. The localization error can be defined as the difference in pixels between the absolute centre of the crosshair, and the clicked point (Error = location of true value in pixels – location of clicked point in pixels). Subsequently, we get the mean localization error for those ten clicked points. Figure 15 demonstrates the precision of the system for both “Hover mode” and “Voice mode”.

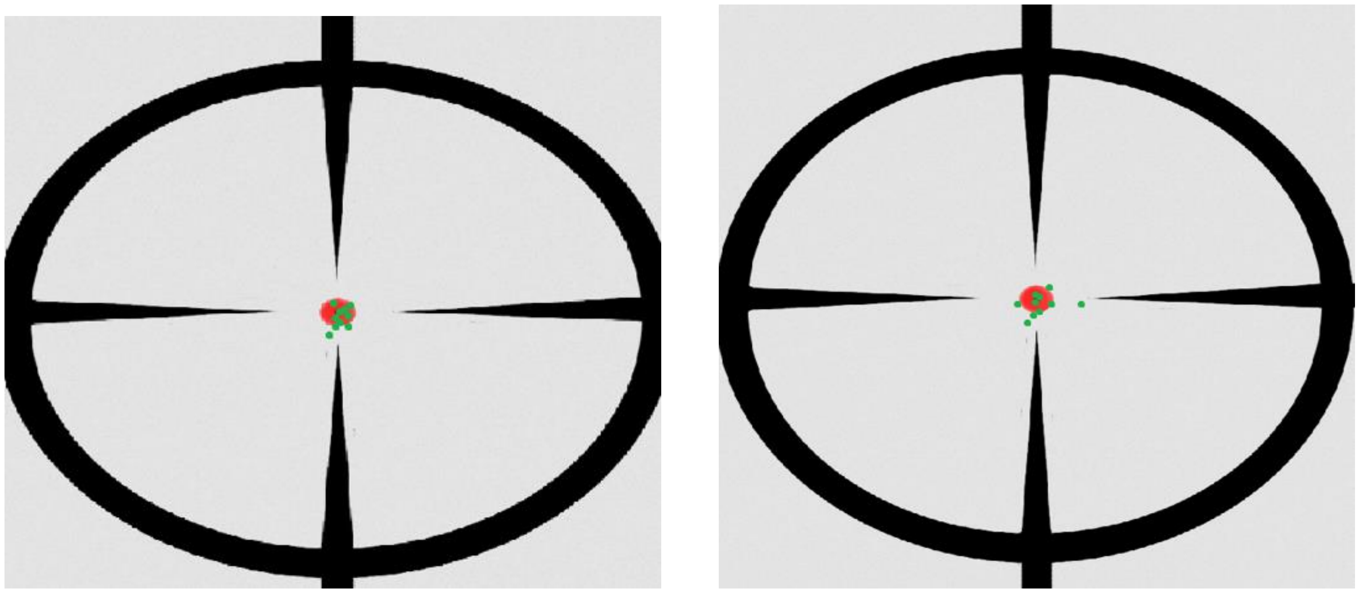


Figure 15. The precision of ten clicked points is illustrated in green colour for “Hover Mode” (left image) and “Voice Mode” (right image)

After experimenting, the localization error for every point and mean localization error for “Hover mode” and “Voice mode” are shown in Table 2 and Table 3 respectively:

Clicked point	Localization error in pixels
1	4 pixels
2	1 pixel
3	2 pixels
4	3 pixels
5	5 pixels
6	10 pixels
7	11 pixels
8	5 pixels
9	15 pixels
10	5 pixels
Mean Error	**6.1 pixels**

Table 2: The localization error for every clicked point in "Hover Mode" and its mean localization error

Clicked point	Localization error in pixels
1	1 pixel
2	2 pixels
3	1 pixel
4	10 pixels
5	13 pixels
6	15 pixels
7	19 pixels
8	15 pixels
9	12 pixels
10	20 pixels
Mean Error	**10.8 pixels**

Table 3: The localization error for every clicked point in "Voice Mode" and its mean localization error

As we can see from the results, the precision of the system is considerable. Based on the results, we can observe that the “Voice Mode” had more errors than the “Hover Mode”. The assumption of why this might have occurred is due to the slight delay between the user saying a certain voice command, and the system sending the audio to Google’s servers to be processed. During that small time frame, the user might slightly move their head, which results in more inaccuracies rather than the “Hover Mode”. However, the overall system achieves significant accuracy even on a 25-pixel diameter target. Moreover, a study in HCI systems determined that User Interface buttons should be at a minimum of 45 pixels in diameter [32]. Hence this experiment demonstrated a more extreme scenario by using a target that is less than 45 pixels in diameter. Ultimately, the conclusion of this experiment indicates that real users of this system will be able to precisely target even the smallest UI buttons of an Operating System, website, or application with relative ease.

The concluding remarks regarding the evaluation of the system were mostly positive. Users who have tested the system required little to no explanation on how to use it since the behaviour of the system represents perceptual interaction. Moreover, the experiments that evaluated the robustness, speed and precision of the system suggested that the system can be a practical and inexpensive alternative for real users who are unable to use a conventional mouse otherwise. We believe that the most ideal mode of the system is “Hover Mode”, due to its precise and natural interaction, however, users will have the freedom of choice to use the system however they see fit with the implementation of “Voice Mode” and “Motion-only Mode”.

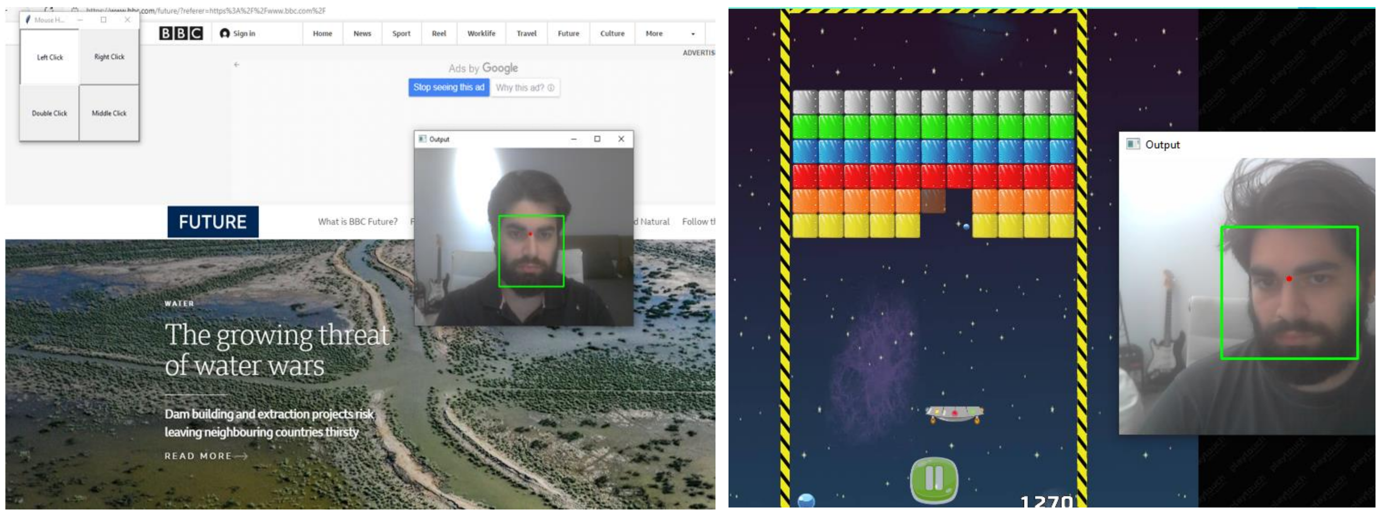
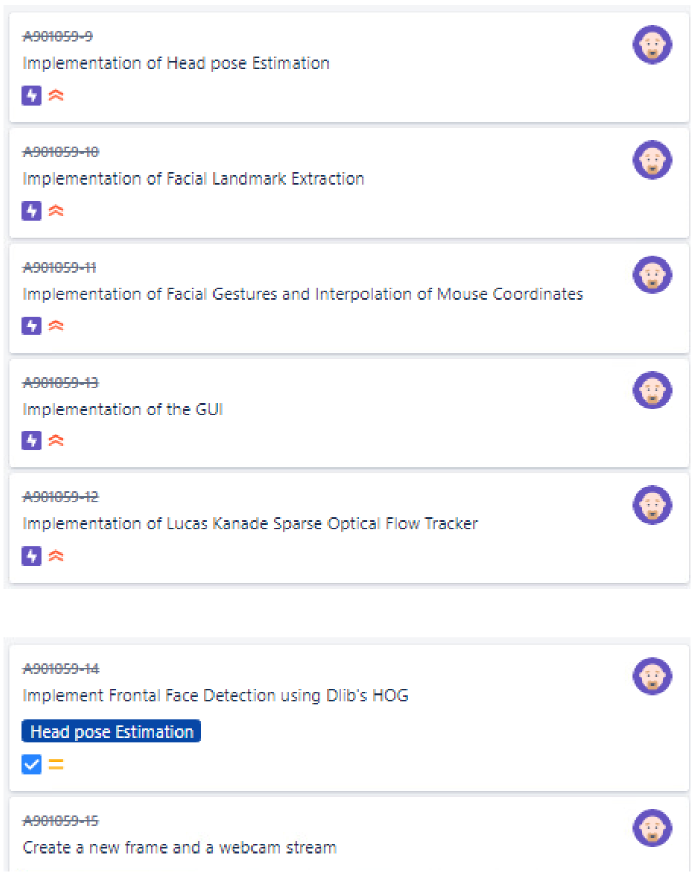


Figure 16. The system can be used for various useful tasks in an OS, such as browsing articles on the web or playing video games

5. Project Plan

5.1 Methodology Used

The correct use of project planning in any project, and particularly for developing software plays an imperative role in the software's success and to meet some criteria in a predetermined time frame. Specifically, this project has used the Agile Methodology throughout its developmental life cycle. The Agile Methodology has an iterative nature, where requirements are created and reviewed within each iteration and subsequently implemented as features [33]. For breaking these large milestones described into smaller individual tasks, the visualization tool Jira has been used in conjunction with the Kanban framework, which does not rely on sprints, but rather individual tasks. Jira has been used to organize these large tasks into smaller ones, including the use of Epics and issues, with a total of 25 tasks. Moreover, Jira has helped the project substantially to meet certain criteria in a prearranged time frame. Lastly, the versioning tool Git and platform Gitlab has been used to keep track and maintain the repository of the source code of the project and its respective versioning history. The source code of the project, including its readme file, can be found in the CSEE's Gitlab [34].



~~A901059-9~~
Implementation of Head pose Estimation

~~A901059-10~~
Implementation of Facial Landmark Extraction

~~A901059-11~~
Implementation of Facial Gestures and Interpolation of Mouse Coordinates

~~A901059-13~~
Implementation of the GUI

~~A901059-12~~
Implementation of Lucas Kanade Sparse Optical Flow Tracker

~~A901059-14~~
Implement Frontal Face Detection using Dlib's HOG
Head pose Estimation

~~A901059-15~~
Create a new frame and a webcam stream

Figure 17. Various finished tasks of the project in Jira, including Epics and their corresponding smaller tasks

5.2 Work Breakdown Structure

As described above, the project has been organized into important milestones to efficiently track its progress. Figure 18 illustrates the Gantt Chart used to visualize the milestones of the project, which were then broken down into smaller sub-tasks. The first major milestone consisted of detecting the face of the user and their facial features using the HOG algorithm and facial landmark extraction. Subsequently, the second task was to implement the Head pose estimation to track the motion of the head. In the proposal of this research [35], it was mentioned that head pose estimation will be used to track the motion of the head. However, after its implementation, it was found that the end result was not satisfactory. Specifically, as described in the background section, the system was impacted by severe jitter and was not usable. After taking a substantial amount of time from the project to smoothen out this noise using multiple techniques, it was found that it was infeasible to track the head using head pose estimation. After some consideration, the third task thus focused on implementing the LK optical flow tracker and interpolating the mouse coordinates which greatly benefited the project and made the system very usable and robust. The fourth major milestone focused on the implementation of the three modes which were described in Section 3. The fifth milestone implemented the main GUI of the system where the users can choose their preferred method. After the implementation phase, the sixth milestone focused on evaluating the robustness, speed, and precision of the system, collecting data, and testing the system on multiple different users. Finally, the last milestone was writing and presenting the final report of the system.

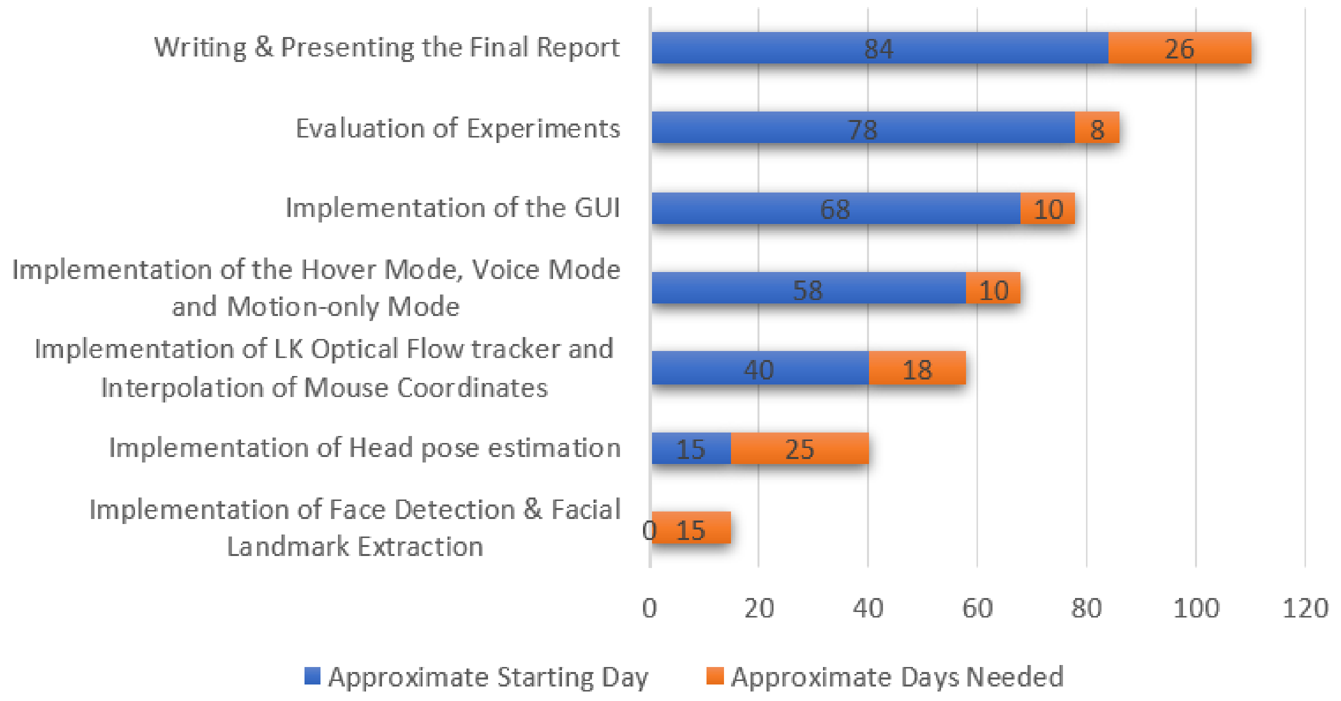


Figure 18. The Gantt Chart of the project, describing its major milestones and their respective duration

6. Conclusion

In this paper, a robust, hands-free camera mouse system has been presented, implemented, and evaluated. It used computer vision techniques, as described in the background section, to detect the face of the user, extract a facial feature between the user's eyes and subsequently track the motion of the head robustly, thus letting the user control their mouse cursor with their head movement using only a consumer-grade web camera. Experimental results showcased that the system achieved good performance on robustness, speed, and precision, and could be a viable option for real users that want a hands-free alternative to a conventional PC mouse, ultimately letting physically impaired users access and use a PC effectively and inexpensively.

Deep learning algorithms were not discussed in this paper. This was due to technological constraints while developing the system. It can be also argued that typical household PCs still do not possess the computational power to infer deep learning methods, thus limiting the availability of the system towards its users. However, it is self-evident that computational power, even for consumer use is rapidly advancing. While this paper focused on staple and computationally low-cost algorithms, an interesting future direction for this system is to apply deep learning methods specifically for face detection and facial landmark extraction, which are more robust compared to traditional algorithms [36] [37]. Additional future work can include the principal implementation of this system for other purposes, such as controlling a wheelchair with head movement alone, or even a car.

7. References

[1] S. Trewin and H. Pain, “Keyboard and mouse errors due to motor disabilities,” *Int. J. Hum. Comput. Stud.*, vol. 50, no. 2, pp. 109–144, 1999, doi: 10.1006/ijhc.1998.0238.

[2] J. Harris, “The use, role and application of advanced technology in the lives of disabled people in the UK,” *Disabil. Soc.*, vol. 25, no. 4, pp. 427–439, Jun. 2010, doi: 10.1080/09687591003755815.

[3] M. Betke, J. Gips, and P. Fleming, “The Camera Mouse: Visual tracking of body features to provide computer access for people with severe disabilities,” *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 10, no. 1, pp. 1–10, 2002, doi: 10.1109/TNSRE.2002.1021581.

[4] N. W. Moon, P. M. Baker, and K. Goughnour, “Designing wearable technologies for users with disabilities: Accessibility, usability, and connectivity factors,” *J. Rehabil. Assist. Technol. Eng.*, vol. 6, p. 205566831986213, Jan. 2019, doi: 10.1177/2055668319862137.

[5] A. Al-Rahayfeh and M. Faezipour, “Eye tracking and head movement detection: A state-of-art survey,” *IEEE J. Transl. Eng. Heal. Med.*, vol. 1, pp. 11–22, 2013, doi: 10.1109/JTEHM.2013.2289879.

[6] E. Hjelmås and B. K. Low, “Face detection: A survey,” *Comput. Vis. Image Underst.*, vol. 83, no. 3, pp. 236–274, Sep. 2001, doi: 10.1006/cviu.2001.0921.

[7] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001, vol. 1, doi: 10.1109/cvpr.2001.990517.

[8] C. Rahmad, R. A. Asmara, D. R. H. Putra, I. Dharma, H. Darmono, and I. Muhiqqin, “Comparison of Viola-Jones Haar Cascade Classifier and Histogram of Oriented Gradients (HOG) for face detection,” in *IOP Conference Series: Materials Science and Engineering*, Jan. 2020, vol. 732, no. 1, doi: 10.1088/1757-899X/732/1/012038.

[9] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, 2005, vol. I, pp. 886–893, doi: 10.1109/CVPR.2005.177.

[10] M. Bodini, “A Review of Facial Landmark Extraction in 2D Images and Videos Using Deep Learning,” *Big Data Cogn. Comput.*, vol. 3, no. 1, p. 14, Feb. 2019, doi: 10.3390/bdcc3010014.

[11] J. Cech and T. Soukupova, “Real-Time Eye Blink Detection using Facial Landmarks,” *Cent. Mach. Perception, Dep. Cybern. Fac. Electr. Eng. Czech Tech. Univ. Prague*, pp. 1–8, 2016.

[12] M. Nabati and A. Behrad, “Camera mouse implementation using 3D head pose estimation by monocular video camera and 2D to 3D point and line correspondences,” *2010 5th Int. Symp. Telecommun. IST 2010*, pp. 825–830, 2010, doi: 10.1109/ISTEL.2010.5734136.

[13] N. Ruiz, E. Chong, and J. M. Rehg, “Fine-Grained Head Pose Estimation Without Keypoints,” *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Work.*, vol. 2018-June, pp. 2155–2164, Oct. 2017, Accessed: Apr. 14, 2021. [Online]. Available: http://arxiv.org/abs/1710.00925.

[14] "Head Pose Estimation using OpenCV and Dlib | Learn OpenCV." https://learnopencv.com/head-pose-estimation-using-opencv-and-dlib/ (accessed Apr. 14, 2021).

[15] G. Sang, F. He, R. Zhu, and S. Xuan, "Learning toward practical head pose estimation," *Opt. Eng.*, vol. 56, no. 08, p. 1, Aug. 2017, doi: 10.1117/1.OE.56.8.083104.

[16] B. D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," *undefined*, 1981.

[17] "Introduction to Motion Estimation with Optical Flow." https://nanonets.com/blog/optical-flow/ (accessed Aug. 19, 2021).

[18] J. Shi and C. Tomasi, "Good features to track," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600, doi: 10.1109/cvpr.1994.323794.

[19] "What is interpolation?" http://www.geography.hunter.cuny.edu/~jochen/gtech361/lectures/lecture11/concepts/What is interpolation.htm (accessed Aug. 19, 2021).

[20] J. Szabados and P. Vértesi, *Interpolation of Functions*. WORLD SCIENTIFIC, 1990.

[21] J. Tu, T. Huang, and H. Tao, "Face as mouse through visual face tracking," in *Proceedings - 2nd Canadian Conference on Computer and Robot Vision, CRV 2005*, 2005, pp. 339–346, doi: 10.1109/CRV.2005.39.

[22] Y. Fu and T. S. Huang, "HMouse: Head tracking driven virtual computer mouse," in *Proceedings - IEEE Workshop on Applications of Computer Vision, WACV 2007*, 2007, doi: 10.1109/WACV.2007.29.

[23] "This Person Does Not Exist." https://thispersondoesnotexist.com/ (accessed Aug. 20, 2021).

[24] A. V. Subramanyam and S. Emmanuel, "Video forgery detection using HOG features and compression properties," in *2012 IEEE 14th International Workshop on Multimedia Signal Processing, MMSP 2012 - Proceedings*, 2012, pp. 89–94, doi: 10.1109/MMSP.2012.6343421.

[25] "dlib-models/shape_predictor_68_face_landmarks.dat.bz2 at master · davisking/dlib-models · GitHub." https://github.com/davisking/dlib-models/blob/master/shape_predictor_68_face_landmarks.dat.bz2 (accessed Aug. 20, 2021).

[26] "OpenCV - OpenCV." https://opencv.org/ (accessed Apr. 16, 2021).

[27] "dlib C++ Library." http://dlib.net/ (accessed Apr. 16, 2021).

[28] "OpenCV: cv::SparsePyrLKOpticalFlow Class Reference." https://docs.opencv.org/4.5.2/d7/d08/classcv_1_1SparsePyrLKOpticalFlow.html (accessed Aug. 23, 2021).

[29] "Welcome to PyAutoGUI's documentation! — PyAutoGUI documentation." https://pyautogui.readthedocs.io/en/latest/ (accessed Apr. 16, 2021).

[30] "SpeechRecognition · PyPI." https://pypi.org/project/SpeechRecognition/ (accessed Aug. 23, 2021).

[31] “tkinter — Python interface to Tcl/Tk — Python 3.9.4 documentation.” https://docs.python.org/3/library/tkinter.html (accessed Apr. 16, 2021).

[32] Z. X. Jin, T. Piocher, and L. Kiff, “Touch screen user interfaces for older adults: Button size and spacing,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2007, vol. 4554 LNCS, no. PART 1, pp. 933–941, doi: 10.1007/978-3-540-73279-2_104.

[33] M. Cohn, *Agile Estimating and Planning*. Prentice Hall, 2005.

[34] “CE901 / CE901_moussoulides_antonis · GitLab.” https://cseegit.essex.ac.uk/2020_ce901/ce901_moussoulides_antonis (accessed Aug. 27, 2021).

[35] A. Moussoulides, “A Head-Driven Human Computer Interface - Proposal,” 2021.

[36] S. S. Farfade, M. Saberian, and L. J. Li, “Multi-view face detection using Deep convolutional neural networks,” in *ICMR 2015 - Proceedings of the 2015 ACM International Conference on Multimedia Retrieval*, Jun. 2015, pp. 643–650, doi: 10.1145/2671188.2749408.

[37] H. Fan and E. Zhou, “Approaching human level facial landmark localization by deep learning,” *Image Vis. Comput.*, vol. 47, pp. 27–35, Mar. 2016, doi: 10.1016/j.imavis.2015.11.004.

8. Program Listing

- **gui.py**: The main menu of the application, where users can choose their preferred method to use the system
- **head_tracker_dwell.py**: The “Hover Mode” of the system
- **head_tracker_voice.py**: The “Voice Mode” of the system
- **head_tracker_normal.py**: The “Motion-only Mode” of the system
- **optical_flow_tracker.py**: The LK sparse optical flow tracker implementation, which includes functions to update a point for each frame, and add a new point when found from a face
- **model**: The directory where the pre-trained facial extraction model resides
- **shape_predictor_68_face_landmarks.dat**: The pre-trained 68 facial landmarks model used for facial landmark extraction
- **sounds**: The directory used for sounds after triggering a mouse event, to provide auditory feedback to the user
- **venv**: The virtual environment of the system, including all of its dependencies
- **requirements.txt**: The text file containing all of the dependencies of the system
- **README.md**: The readme file of the system, and how to install it