Mohammed

Kafi Rahman

8/3/2022

**Abstract**

This paper goes into detail to explain how hand gestures were used to control the media displayed on the screen. Using Python as the programming language, several libraries were used to code the project. These libraries are cv2, Mediapipe, pyautogui, and win32gui. Cv2 and Mediapipe are the backbones of the project, and they allowed the hand gestures to be detected. With the help of several functions that were responsible for hand detection, finding the position of the hand, and determining if the fingers of the hand were open or closed it was possible to construct a module. This module can be imported by other python files and then customized so that any hand gesture can be used for any program. The libraries pyautogui and win32gui are responsible for detecting open applications and triggering key presses. The result was a program that gave users the ability to use their hand gestures to improve their viewing experience and accessibility.

Computer programming has been the backbone of the vast amount of technology around us today. Those lines of codes have the power to create software that is possible to make the life of millions of people easier. One area of programming that was tackled in this project is computer automation. Computer automation in the past has been an idea that was depicted in movies. It was only a hope and dream that one day in the future that such technology will be accessible to the public. Such accessibility for the public can give the potential for several creative ideas and ones that can make what was a dream one day into a reality. One area of automation that was tackled in this project was using hand gestures to control media that is displayed on the screen. The capacity to recognize the motion and shape of hands can play a key role in enhancing the user experience across a range of technology platforms and areas. For instance, it can provide the foundation for hand gesture control and sign language comprehension. It can also make it possible for digital information and material to be superimposed on top of the actual environment in augmented reality. Although it comes effortlessly to individuals, strong real-time hand perception is an extremely difficult computer vision problem since hands frequently occlude themselves or each other and lack high contrast patterns. The motivation behind using this technology comes from the ease of accessibility that it brings to the user. With the help of the software that was designed by Google, which is called MediaPipe, the process of hand detection became much simpler. Exploring this sophisticated software was the foundation of the project at hand and allowed the integration of other APIs such as pyautogui and OpenCV much easier to be accomplished. Their description and how they were integrated to allow media to be controlled through hand gesture is explained below.

**MediaPipe**

As stated by Google MediaPipe hands is a hand and finger tracking software. Within a single frame, it pinpoints 21 3D landmarks on the user’s hands. Doing so eliminates the need for a powerful computer to achieve real-time performance. MediaPipe utilizes models for identifying palms that make use of the complete image and generates an oriented hand bounding box as part of the MediaPipe Hands machine learning pipeline. A hand landmark model that generates very precise 3D hand key points while operating on the portion of the image that was clipped by the palm detector. Giving the hand landmark model a properly cropped image of a hand considerably reduces the need for data augmentation (such as rotations, translations, and scaling) and allows the network to concentrate its resources on accurate coordinate prediction. The pipeline further enables the production of crops depending on the preceding frame, and palm detection is only used to re-localize the hand when the landmark model is unable to do so. The pipeline is constructed as a MediaPipe graph using a customized hand renderer subgraph and a hand landmark tracking subgraph from the hand landmark module. The hand landmark tracking subgraph employs both a hand landmark subgraph from the same module and a palm detection subgraph from the palm detection module internally. The code for the hand landmark module, tracking subgraph, hand renderer subgraph, and palm detection module can be found on the Google GitHub page.

**PyAutoGui**

It is an API that is designed to allow Python scripts to control the mouse and keyboard to automate interactions with other applications. Pyautogui has features that were used in the project and that include:

* Dragging the mouse and clicking on the buttons displayed on the screen
* Sending keypresses to other applications
* Locating an application’s window, moving, resizing, maximizing, minimizing, or closing windows.

**OpenCV**

OpenCV is an open-source computer vision. Machine learning and image processing library. It is supported by various programming languages such as Python, C++, and Java. It allows the webcam to process images and videos to identify objects such as facial recognition and detection, license plate reading, photo editing, advanced robotic vision, and optical character recognition, The advantage of this software is that it can be merged with other libraries and models such as MediaPipe.

**Structure of HandTrackingModule.py**

Importing and integrating the cv2 and medipipe libraries was the first step to design the HandTrackingModule.py file. The image is captured frame by frame in a while loop using the VideoCaputure method of the cv2 class. What was required next was to incorporate the medipipe functions which allow whatever is in the frame to be detected. This required the handDetector class to be created. The class had a constructor that set up the values of the parameters which allowed the hand detection to run smoothly. Within the constructor, the maximum hand detection was set to two hands and the detection confidence was set to 0.5. This means that if the confidence level drops below 50% then the hands will not be detected. The rest of the parameters were set as the default values. Within the while loop, the continuous images that are captured by the camera are fed to the functions findHands and findPositon. The findHands functions took the image as the parameter and converted it into an RGB image. The reason for such conversion is because the handDetector class only accepts RGB images and by using the cvtColor method of the handDetector class the conversion was made. The frame is processed, and the image is saved in a result variable. Within the result variable, the hand landmarks are stored and then accessed using results.multi\_hand\_landmarks. Using a for loop that loops through each hand landmark, a dot is drawn using the draw\_landmarks method and then the image is returned to the caller. The returned image shows the landmark drawn on the hand once the camera captures the frame. The findPosition function is one of the important functions in the HandTrackingModule. It returns a list containing the id number of all the 21 landmarks on the hand along with the X and Y coordinates of those landmarks on the frame. Such information is saved in the landmark method of the handDetector class. A for loop is designed to go through the id and coordinates of the landmarks. To convert the coordinates from decimals to pixels, the height and width of the image are used. The product of the decimal X coordinate and width along with the decimal Y coordinates and the height is saved as X coordinate and Y coordinate in pixels. The Id with the addition of the two coordinates is saved in a list and appended to landmark list. The landmark list is returned to the caller. The reason why this function is pivotal for the project is that it allows the ease of accessibility for each point in the hand. The fingersUp function is what checked to see which finger is up and which finger is down. The parameter of the function is a list that stores all the points which are marked on the user’s hand via mediapipe. The way the function works is that it checks the Boolean statement of if the X coordinate of the fingertip dips below the X coordinate of the middle of the finger. If the Boolean statement is true, then it will append 1 to the Open\_fingers list and if it is false then 0 will be appended to the list. The list is then returned to the caller. All these functions are what make up the HandTrackingModule.py. The Module is then imported to Video\_Gestured.py.

**Video\_Gestured.py**

Vide\_Gesutred.py imports three libraries which are cv2, HandTrackingModule, and pyautogui. Pyautogui is the library that provides access to control of the keyboard. The detector object stores the handDetector class of the HandTrackingModule. The detection confidence is set to 0.7. This project was designed to work only with the vlc media player. The video to be played needs to be loaded by the user and must be in the paused state. When the user runs the code pyautogui will press the space button and that will play the video and the state of the video will be stored as play. This allows the program to know whether the video is played or not. The pyautogui.getWindowsWithTitle("vlc")[0].maximize() will display the vlc media player in full screen when it is called. In the while loop, the code checks if the list that is returned by findPosition method of the detector object is empty or not. If it is filled, then the fingersUp method is called. The list returned by fingersUp contains five numbers that represent the five fingers of the hand. The number one represents a finger that is up and zero represents a finger that is down. The five control commands are pause, play, volume down, volume up, move forward, and move backward. The way the hand gestures controlled them are in these ways.

* If the palm is closed, then the state of the video is set to pause, and the video is paused by calling pause\_video().
* If the palm is opened, then the state of the video is set to play, and the video is played by calling play\_video().
* If three fingers are up, then the volume is lowered by calling Volume\_down.
* If four fingers are up, then the volume is increased by calling Volume\_Up.
* If two fingers are up, then the video goes forward by calling move\_forward.
* If one finger is up, then the video goes backward by calling move\_backward.

The pause\_video and play\_video functions contain pyautogui.press('space') which presses the space button. The Volume\_Up and Volume\_down functions contain pyautogui.press(‘volumeup’) and pyautogui.press(‘volumedown’) respectively. Those two functions control the volume. The move\_backward and move\_forward functions contain pyautogui.press(‘left’) and pyautogui.press(‘right’) respectively. Those two functions move the video backward and forward. The program ends when the Q key on the keyboard is pressed by the user. This is done by placing a break statement in the while loop if the key pressed is Q.

**Presentation control.py**

This project imports three libraries pyautogui, win32gui, and HandTrackingModule. To use win32gui the python version needs to be 3.7.x. This code works with Microsoft PowerPoint. For the code to detect an open PowerPoint application, it uses the winEmunHandler. This function checks to see if the application is open and if it is then it will maximize its window. In this project the hand gestures are as follows:

* If only the left-hand thumb is up, then PowerPoint will go to the next slide
* If only the left-hand picky finger is up, then PowerPoint will go to the previous slide

**Conclusion**

This project allowed an enhanced media viewing experience and accessibility. The project successfully utilized mediapipe and cv2 and gave hand gesture controls that can be customized as per the user's need.

Word Cited

“MediaPipe Hands.” *Mediapipe*, 2020, https://google.github.io/mediapipe/solutions/hands.

Mordvintsev, Alexander, and Abid Rahman K. “Introduction to Opencv-Python Tutorials.” *OpenCV*, https://docs.opencv.org/4.x/d0/de3/tutorial\_py\_intro.html.

“Welcome to Pyautogui's Documentation!¶.” *Welcome to PyAutoGUI's Documentation! - PyAutoGUI Documentation*, 2019, https://pyautogui.readthedocs.io/en/latest/.