

PLAYING GAMES USING HAND GESTURES

Submitted in partial fulfillment of the requirements for
the award of
Bachelor of Engineering degree in Computer Science and Engineering

by

K. B. SHREE BALA MUKESH (Reg. No. 37110709)

P. SUNDARAKKUMAR (Reg. No. 37110745)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade "A" by NAAC
JEPPIAAR NAGAR, RAJIV GANDHI
SALAI, CHENNAI – 600 119**

MARCH – 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with "A" grade by NAAC
Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119

www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this project report is the bonafide work of **K.B. SHREE BALA MUKESH (Reg. No. 37110709)** and **P. SUNDARAKKUMAR (Reg. No. 37110745)** who carried out the project entitled "**PLAYING GAMES USING HAND GESTURES**" under my supervision from **August 2020** to **March 2021**.

Internal Guide

Dr. R. M. Gomathi, M.E., Ph.D.,

Head of the Department

Dr. S. Vigneswari, M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I **K. B. SHREE BALA MUKESH** and **P. SUNDARAKKUMAR** hereby declare that the Project Report entitled “**PLAYING GAMES USING HAND GESTURES**” is done by us under the guidance of **Dr. R. M. GOMATHI, M.E., Ph.D.**, Department of Information and Technology at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala, M.E., Ph.D., Dean**, School of Computing, **Dr. S. Vigneswari, M.E., Ph.D.**, and **Dr. L. Lakshmanan, M.E., Ph.D.**, **Heads of the Department** of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. R. M. Gomathi, M.E., Ph.D.**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Now a days there are many developments in computer vision, the interaction between computers and human beings are more extensive. The concept of detecting color and shape helps us to detect the objects via video or an image. This technology can be used by computers for various purposes like facial reorganization, object detection, hand gestures etc. This paper is about hand gesture segmentation by capturing specific colors by basic web cam which is built in laptops, which helps us to monitor the movements of hand. By monitoring the movements, we can achieve by controlling the movements of game characters like person or cars in a game with hand gesture. But also, some games require mouse control for playing we can also able to control the movements of mouse as well as clicking operation of the mouse via hand movements. This feature gives a fantastic experience of playing games instead of just sitting and using keyboards to play games. In mobile phones we have gyroscope sensor to play games by tilting phones as we can't tilt or use gyroscope sensors in laptops or pc, we need to use webcam for playing with physical motion.

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE
No		No
	ABSTRACT	5
	LIST OF FIGURES	8
1.	INTRODUCTION	9
	1.1 HAND GESTURE	9
	1.2 MOTION CONTROL	11
2.	LITERATURE SURVEY	14
	2.1 LITERATURE REVIEW	14
	2.2 OPEN PROBLEMS	17
3.	AIM AND SCOPE OF THE PRESENT INVESTIGATION	19
	3.1 AIM	19
	3.2 SCOPE	19

4.	METHOD AND ALGORITHMS	21
	4.1 MODULES	21
	4.1.1 PYTHON	21
	4.1.2 COMPUTER VISION	22
	4.1.3 PYAUTOGUI	23
	4.2 METHODOLOGY	24
	4.2.1 COLOR DETECTION	24
	4.2.2 KEYBOARD AND MOUSE CONTROL	27
5.	RESULT AND DISCUSSION, PERFORMANCE ANALYSIS	38
	5.1 RESULT	38
6.	CONCLUTION AND FUTURE WORK	46
	6.1 RESULT	46
	6.2 FUTURE WORK	47
	REFERENCES	48
	APPENDIX	49
	A SOURCE CODE	49
	B SCREENSHOTS	60

LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE NO
1.1	FLOW CHART	13
4.1	RGB COLOR MODULE	24
4.2	HSV MODULE	25
4.3	RGB TO HSV CONVERSION FOLMULA	26
4.4	COLOR DETECTION FLOW DIAGRAM	27
4.5	INITIAL COLOR DETECTION WITH A NOISY IMAGE	30
4.6	NOISELESS CLEAR DETECTION	31
5.1	INITIAL DETECTION WITH A NOISEY VIDEO	38

5.2	FORWARD OPERATION	39
5.3	APPLYING BREAK OR REVERSE	40
5.4	TURNING LEFT	41
5.5	TURNING RIGHT	42
5.6	MOUSE MOVEMENTS	43
5.7	LEFT CLICK OPERATION	44
5.8	RIGHT CLICK OPERATION	45

CHAPTER 1

INTRODUCTION

1.1 HAND GESTURE

Gesture recognition is an active research field which tries to integrate the gestural channel in Human Computer Interaction. It has applications in virtual environment control, but also in sign language translation, robot remote control or musical creation. Recognition of human gestures comes within the more general framework of pattern recognition. In this framework, systems consist of two processes: the representation and the decision processes. The representation process converts the raw numerical data into a form adapted to the decision process which then classifies the data.

Hand gesture recognition system can be used for interfacing between

computer and human using hand gesture. Proper hand segmentation is the primary requirement from the background and other body parts of the image frame for the design of a hand gesture-based application. These image frames can be captured from a low-cost webcam (camera) for use in a vision-based gesture recognition technique. This paper discusses about the continuous hand gesture recognition. The aim of this paper is to report a robust and efficient hand segmentation algorithm where a new method, wearing glove on the hand is utilized. After that a new idea called “Finger-Pen”, is developed by segmenting only one finger tip from the hand for proper tracking. In this technique, only the tip of the finger is segmented instead of whole hand (palm). Problems such as skin color detection, complexity from large numbers of people in front of the camera, complex background removal and variable lighting conditions are found to be efficiently handled by the system. Noise present in the A Colored Finger Tip-Based Tracking Method for Continuous Hand Gesture segmented images due to dynamic background can be removed with the help of this adaptive technique which is found to be effective for the application conceived.

Color detection is the process of detecting the name of any color. Simple isn't it? Well, for humans this is an extremely easy task but for computers, it is not straightforward. Human eyes and brains work together to translate light into color. Light receptors that are present in our eyes transmit the signal to the brain. Our brain then recognizes the color. Since childhood, we have mapped certain lights with their color names. We will be using the somewhat same strategy to detect color names.

Sufficient differences in wavelength cause a difference in the perceived hue; the just-noticeable difference in wavelength varies from about 1 nm in the blue-green and yellow wavelengths to 10 nm and more in the longer red and shorter blue wavelengths. Although the human eye can distinguish up to a few hundred hues, when those pure spectral colors are mixed together or diluted with white light, the number of distinguishable chromaticities can be quite high.

In very low light levels, vision is scotopic: light is detected by rod cells of the retina. Rods are maximally sensitive to wavelengths near 500 nm and play little, if any, role in color vision. In brighter light, such as daylight, vision is photopic: light is detected by cone cells which are responsible for color vision. Cones are sensitive to a range of wavelengths, but are most sensitive to wavelengths near 555 nm. Between these regions, mesopic vision comes into play and both rods and cones provide signals to the retinal ganglion cells. The shift in color perception from dim light to daylight gives rise to differences known as the Purkinje effect.

The perception of "white" is formed by the entire spectrum of visible light, or by mixing colors of just a few wavelengths in animals with few types of color receptors. In humans, white light can be perceived by combining wavelengths such as red, green, and blue, or just a pair of complementary colors such as blue and yellow.

During the project, four gestures were chosen to represent four navigational commands for the robot, namely Move Forward, Move Left, Move Right, and Stop. A simple computer vision application was written for the detection and recognition of the four gestures and their translation into the corresponding commands for the robot. The appropriate OpenCV functions and image processing algorithms for the detection and interpretation of the gestures were used. Thereafter, the program was tested on a webcam with actual hand gestures in real-time and the results were observed.

1.2 MOTION CONTROL

Motion detection is the process of detecting a change in the position of an object relative to its surroundings or a change in the surroundings relative to an object. It can be achieved by either mechanical or electronic methods. When it is done by natural organisms, it is called motion perception.

Motion detection is an important tool for securing your business or building. It alerts you when someone is on your property that isn't authorized. Understanding how this technology helps you set up better motion detection regions and alerts, but do you actually know how motion detection works? To understand motion detection, you first need to understand how a camera works. Inside the camera is an image sensor, which the camera lens directs light to - when light hits the image sensor each individual pixel records how much light it's getting. That pattern of light and dark areas on the pixels becomes the complete video image you see. You can read a more in-depth explanation of image sensors [here](#).

When you set up motion detection, you select a region or area to monitor, say a doorway. The way it works is to compare sequential images from your video and if enough of the pixels have changed between those frames, the camera software determines something moved and sends you an alert. How "enough" change is determined depends on the sensitivity level or percentage level settings you put on your motion detection. The sensitivity level looks at the changing light/dark levels in the pixels. Higher sensitivity will pick up on more changes while lower sensitivity will require a large level of change, such as the lights going on in a dark room, to set off the alerts. Percentage, on the other hand, looks to the percentage of pixels that have registered a change.

If you are monitoring a door, setting the percentage at 50% will alert you when something large walks into the room, such as a person, but ignore something smaller. Getting the best balance in your motion detection settings to prevent too many false alerts may take some time. While setting up your camera you can test the motion detection by having someone walk in the motion detection region and adjust the settings as needed.

Implementation of new game control settings that is playing games using the hand movements. This can be done in several ways, first we have to detect the hand for monitoring its movements. There are many ways to achieve it, by using CNN hand data sets, by edge detection, by sensors, and by color detection. Cui

and Weng proposed a hand tracking and sign recognition method using appearance-based method. We tried to achieve this by using the concept of color detection and edge detection. This can also be done with the help of data sets but if we use data sets for game settings then the space complexity of the program will be high. So, we didn't go for it.

We don't require and hardware requirements for this implementation, we are using pre built web cam for detection hand through video format. In terms of hand gesture tracking, it is about real time location and hand gesture tracking in video according to some features of them, so it is the key step for hand gesture recognition. These gestures are the definition of intuitive and natural gesture, where every gesture object can be denoted as sequence of contiguous sub gestures of a continuous action done by the user. The hand can be detected by converting RGB video to HUE saturation, so that specific color can be detected as a depth color of the video.

After detecting the hand, we have to draw contours for the hand so the we could get the area. By restricting the area of the contour size, we can detect only the hand so the face or other colors can be ignored. Then we could get the center point of the area of the hand which helps to detect the location of the hand. And this location is used for monitoring the movements of the hand for our purpose of the project. When movements are monitored, this will be used by the user defined control functions which will invoke certain keys in keyboard so that the objects in the game can be controlled. But also, this will be useful for controlling the mouse so that some games which requires mouse control can also be played.

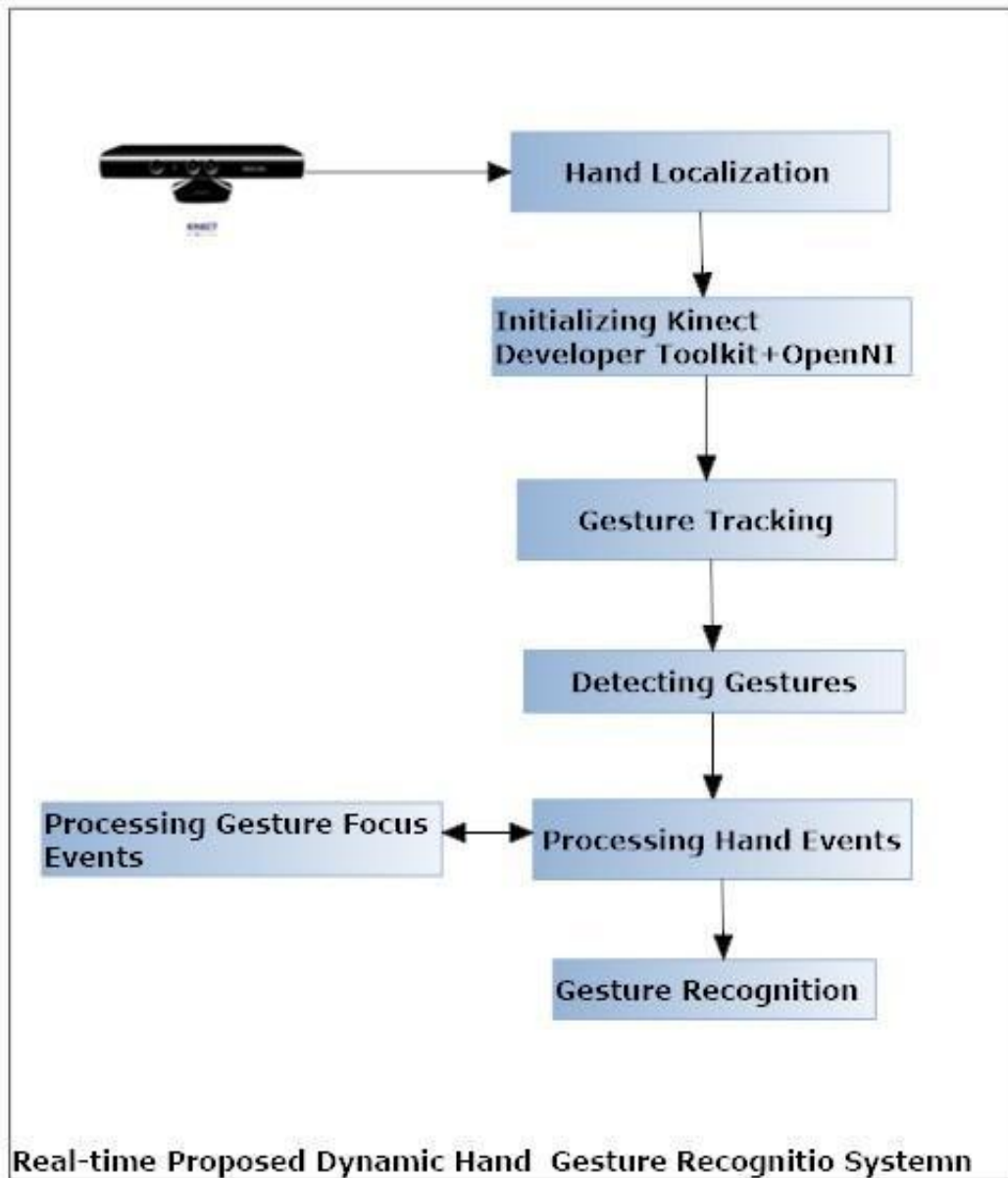


FIG 1.1 FLOW CHART OF PROJECT

CHAPTER 2

LITERATURE SURVEY

2.1 LITERATURE REVIEW

In Y. Cui and J. Weng, "View-based hand segmentation and hand sequence recognition with complex backgrounds" [1], A method to determine whether a pixel is the edge point of a color image is presented. First, according to the characteristic of RGB color space, a color triangle of a pixel is constructed with its color coordinates RGB, and then the perimeter and the internal angle of the color triangle which can be seen as the information value of a pixel is calculated. Last, comparing the value with that in its neighborhood, a pixel is in the edge point or not is determined. This method took the correlation of every color component into account reasonably, and converted the computation of a vector space to scalar quantity in a natural way. Comparing it with traditional method, experimental result shows that more legible edges can be detected with this method and can be realized easily.

In X. W. Li and X. R. Zhang, "A perceptual color edge detection algorithm" [3], Edge detection is one of the most powerful image analysis tools for enhancing and detecting edges. Indeed, identifying and localizing edges are a low level task in a variety of applications such as 3-D reconstruction, shape recognition, image compression, enhancement, and restoration. This paper introduces a new algorithm for detecting edges based on color space models. In this RGB image is taken as an input image and transforming the RGB image to color models such as YUV, YCbCr and XYZ. The edges have been detected for each component in color models separately and compared with the original image of that particular model. In order to measure the quality assessment between images, SSIM (Structural Similarity Index Method) and VIF (Visual Information Fidelity) has been calculated. The results have shown that XYZ color model is having high SSIM value and VIF value. In the previous papers, edge detection based on RGB color

model has low SSIM and VIF values. So, by converting the images into different color models shows a significant improvement in detection of edges. Keywords: Edge detection, Color models, SSIM.

In M. Panwar and P. S. Mehra, "Hand Gesture Recognition for Human Computer Interaction" [5], Actually gesture technique is used who can understand it and is being used today to control the computer instead of mouse or keyboard. This paper presents three new interactive methods which will make the system more reliable. The paper also describes an effective hand-based interaction method that needs preprocessing means conversion of the user's hand image from one form to another form for implementation that makes a connection with the next step which leads to hand detection. As many new technologies have been arrived to do our work and among them hand gesture-based technology is the latest, so its usage is very limited. This technique is being used in television, computer, and robots without using remote control, joystick to complete the least job. It will also be used in some applications such as holographic technology, gadgets like phone, editing and copying data, creating programs etc.

In H. L. Zhang, H. Y. Li, M. Zhou, Hand posture recognition based on multi-feature and compressive sensing [7], Vision-based hand gesture recognition methods commonly use a single feature of hand gesture for classification. There are some problems as follows: recognition inaccuracy, system instability and ambiguous recognition results. This paper proposes a hand gesture recognition method based on multi-feature fusion and template matching. The method detects hand-shaped contour region and obtains the maximum contour according to skin color feature, by extracting angle count, skin color angle, and non-skin color angle in combination with Hu invariant moments features of the largest hand-shaped region for sample training. Euclidean distance template matching technique is applied for hand gesture classification and recognition. In order to remove the impact of the arm, images are transformed from RGB space into HSV space for skin color detection. Because of H (hue) are greatly different from the palm and arm, the use of this transformation can effectively locate the hand shaped region excluding the impact of the palm in HSV space. After skin detection, next

extraction of the contour of the hand gesture region, we can obtain the maximum contour of the hand shaped area by comparing to ensure the integrity of the hand gesture in contour when hand gesture recognition. Experiments show that the method is effective for extracting features of different hand gestures and real-time recognition of ten kinds of hand gestures. This paper proposed a multi-feature fusion-based hand gesture recognition method which solves problems of low rate of recognition accuracy, system instability and multiple recognition results when using a single hand gesture feature for recognition. The method can also achieve real-time hand gesture recognition. The use of angle count, skin color angle, non-skin color angle and Hu invariant moments features are easy to understand, and can effectively characterize the different hand gesture patterns. Experiments prove that the recognition method of cascade template matching using the Euclidean distance classification is accurate.

In Oana Saman, Loredana Stanciu, "Hand Mobility Recovery Through Game Controlled by Gestures" [9], Motivation is one of the key factors when having to do exercises for hand mobility, because they have to be done consistently. This paper proposes a motivational application for people who have suffered hand trauma and aims to help them to do exercises for regaining hand mobility. There are two modules in this application: one for gesture recognition using a web camera and one for a gestures-controlled Tetris game. OpenCV, a computer vision library is used for recognizing nine different gestures, and the game is developed in a visual programming language. The recognition rate is high enough in order to fluidly play Tetris. The application is visually attractive, simple to use and may help in increasing motivation to do hand exercises. The proposed application aims to contribute to the recovery process by means of a game controlled by hand movements. In playing the game, the hand movements are actually hand exercises, so the recovery of hand mobility becomes a relaxing activity, not an obligation. The application is divided into two components: a gesture recognition component which is developed using artificial vision algorithms and a gesture-controlled game, developed in a visual programming language. The primary purpose of the application is to incite the user to perform as many hand exercises as possible, as exercise improves blood circulation, allowing the

affected area to heal faster, restore flexibility and strengthen the muscles of the hand, increasing chances of recovering hand functionality. In addition, these exercises can be done by anyone who wants to get the benefits mentioned above, or just play a game controlled by hand movements. The application aims to be as user-friendly and as intuitive as possible and to use the available hardware, without inducing extra costs. Nine gestures that correspond to exercise recommended for regaining hand mobility are implemented. The user is motivated to do hand exercises by playing a game of Tetris. The game is visually appealing and has components such as the highest score and total playing time, which account for the total time doing exercises. Thus, the user is urged to obtain the highest score and play as much as possible, this being the motivational component of the application. For the gesture recognition, a simple USB camera or built-in web camera are used. The developed application consists of a gesture recognition module and a Tetris game, which can be controlled by a series of nine gestures. Recognition of gestures was possible by using artificial vision algorithms of the OpenCV library for hand detection, and by developing a gesture recognition algorithm which uses the detected features and matches them against predefined features representing the nine gestures. The Tetris game was implemented in Construct2, which uses a visual programming language, and contains elements to motivate the user to play, such as the highest score and total game time. This application aims to make the recovery process of people who have suffered various hand traumas to be easier and at the same time more fun. The program is simple to use because it only needs a webcam and a background which is different from the skin color. The application works under different brightness conditions, as it allows the user to take samples with the color of the skin from time to time.

2.2 OPEN PROBLEMS

1. Non-standard backgrounds

Gesture recognition should bring great results no matter the background: it

should work whether you're in the car, at home, or walking down the street. Machine learning gives you a way to teach the machine to tell the hand apart from the background consistently.

2. Movement

Common sense suggests that gesture is rather a movement than a static image. Gesture recognition should thus be able to recognize patterns, ex. instead of recognizing just an image of an open palm, we could recognize a waving movement and identify it e.g., as a command to close the currently used application.

3. Combination of movements

What is more, gestures could consist of several movements, so we need to provide some context and recognize patterns like moving fingers clockwise and showing a thumb could be used to mark some limited number of files or some limited area.

4. Diversity of gestures

There is much diversity in how humans perform specific gestures. While humans have a high tolerance for errors, this inconsistency may make the detection and classification of gestures more difficult for machines. This is also where machine learning helps.

5. Fighting the lag

The gesture detection system must be designed to eliminate the lag between performing a gesture and its classification. Adoption of hand gestures can only be encouraged by showing how consistent and instantaneous it can be. There is really no other reason to start using gestures if they don't make your

interaction faster and more convenient. Ideally, we're looking for a negative lag (classification even before the gesture is finished) to truly instantaneously give the feedback to the user. If you want to read more about gestures classification, there is a great article on our blog on [Classifying unknown gestures](#).

6. Machine learning data sets

One of the most common challenges in applying machine learning in gesture recognition projects is the lack of a rich and meaningful data set. For machine learning to work, you will need to feed it with data to train our models. The data set has to be adjusted to your needs, but some data sets can be very helpful.

CHAPTER – 3

AIM AND SCOPE OF THE PRESENT INVESTIGATION

3.1 AIM

The aim of the project is to create a cost-effective solution for playing games using hand movements which:

1. Does not require any external hardware implementation or any kind of expensive sensors.
2. But also, it does not require any data sets which helps to identify the posture of hand so that the location of hand can be detected.
3. So, ignorance of using data sets reduce the space complexity of the program so that this feature does not affect the gaming speed or space.
4. The detection of hand is been monitored by the basic built in web cams,

so the user can play as usual.

This would make the overall gaming experience a very exciting, convenient, modern, inexpensive and practical approach.

3.2 SCOPE

The video gaming industry has been ever increasing since its inception. The early video gaming platforms for commercial purposes became to be successful in the 1970s. With the improvements on the computer industry, especially the rise of desktop computers led to humongous improvements in the gaming industry. Today there are various platforms on which video games are developed such as the PC, mobile, arcade, web browser, virtual reality, blockchain, etc. In the year 2015, the estimated video gaming industry net worth was around \$91 billion. Nowadays we have lots of game controllers that are improving the gaming experience over the years. Gaming experience can be improved to a good extent with the help of controllers but these controllers are quite expensive. This feature gives a fantastic experience of playing games instead of just sitting and using keyboards to play games. In mobile phones we have gyroscope sensor to play games by tilting phones as we can't tilt or use gyroscope sensors in laptops or pc, we need to use webcam for playing with physical motion. The results indicate the potential in terms of user engagement and training efforts for short-time experiences. However, the study results also indicate usability issues. The experiences with gesture-based controls are rated as exhausting after about 20 min. While the suitability for traditional video games is thus described as limited, users see potential in gesture-based controls as training and rehabilitation tools.

Computer vision is the development of interaction between computer and human, the interaction between human and machine is becoming more frequent in many fields like face lock in mobiles and object detection etc... Among them, one of the feature developments is hand gestures and are commonly used in this

project. This hand gesture developments are used in various aspects like motion capture movies like Kochadaiyaan movie, machinery control, robotic control and many.

Advantages of this project is, as we are not using any external hardware or any costly sensors, the cost of the implementation is NONE. And also, we are not using any data sets of hands the space complexity of the program is also very effective so that when this is implemented in a game control setting it won't be result in increase in memory for the game. And as playing or controlling the pc with hands and without touching anything gives a fantastic and exciting experience.

CHAPTER - 4

METHODS AND ALGORITHMS

4.1 MODULES:

4.1.1 PYTHON

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application

Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace.

A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

4.1.2 COMPUTER VISION

Computer vision is a field of artificial intelligence that trains computers to interpret and understand the visual world. Using digital images from cameras and videos and deep learning models, machines can accurately identify and classify objects — and then react to what they “see”. Computer vision is one of the most remarkable things to come out of the deep learning and artificial intelligence world.

The advancements that deep learning has contributed to the computer vision field have really set this field apart.

The human vision is a very intricate piece of organic technology that involves our eyes and visual cortex, but also takes into account our mental models of objects, our abstract understanding of concepts and our personal experiences through billions and trillions of interactions we've made with the world in our lives. Digital equipment can capture images at resolutions and with detail that far surpasses the human vision system.

Computers can also detect and measure the difference between colours with very high accuracy. But making sense of the content of those images is a problem that computers have been struggling with for decades. To a computer, the above picture is an array of pixels, or numerical values that represent colours.

Computer vision is the field of computer science that focuses on replicating parts of the complexity of the human vision system and enabling computers to identify and process objects in images and videos in the same way that humans do. Until recently, computer vision only worked in limited capacity. Thanks to advances in artificial intelligence and innovations in deep learning and neural networks, the field has been able to take great leaps in recent years and has been able to surpass humans in some tasks related to detecting and labelling objects.

4.1.3 PyAutoGUI

PyAutoGUI lets your Python scripts control the mouse and keyboard to automate interactions with other applications. The API is designed to be as simple. PyAutoGUI works on Windows, macOS, and Linux, and runs on Python 2 and 3. PyAutoGUI has several features:

- Moving the mouse and clicking or typing in the windows of other applications.
- Sending keystrokes to applications (for example, to fill out forms).
- Take screenshots, and given an image (for example, of a button or checkbox), find it on the screen.
- Locate an application's window, and move, resize, maximize, minimize, or close it (Windows-only, currently)
- Display message boxes for user interaction while your GUI automation script runs.

Like the enchanted brooms from the Sorcerer's Apprentice programmed to keep filling (and then overflowing) the bath with water, a bug in your program could make it go out of control. It's hard to use the mouse to close a program if the mouse cursor is moving around on its own.

As a safety feature, a fail-safe feature is enabled by default. When a PyAutoGUI function is called, if the mouse is in any of the four corners of the primary monitor, they will raise a `pyautogui.FailSafeException`. There is a one-tenth second delay after calling every PyAutoGUI functions to give the user time to slam the mouse into a corner to trigger the failsafe. Locations on your screen are referred to by X and Y Cartesian coordinates. The X coordinate starts at 0 on the left side and increases going right. Unlike in mathematics, the Y coordinate starts at 0 on the top and increases going down. The pixel at the top-left corner is at coordinates 0, 0. If your screen's resolution is 1920 x 1080, the pixel in the lower right corner will be 1919, 1079 (since the coordinates begin at 0, not 1).

The screen resolution size is returned by the `size()` function as a tuple of two integers. The current X and Y coordinates of the mouse cursor are returned by the `position()` function.

4.2 METHODOLOGY

4.2.1 COLOUR DETECTION

Color detection is the process of detecting the name of any color. Simple isn't it? Well, for humans this is an extremely easy task but for computers, it is not straightforward. Human eyes and brains work together to translate light into color. Light receptors that are present in our eyes transmit the signal to the brain. Our brain then recognizes the color. Since childhood, we have mapped certain lights with their color names. We will be using the somewhat same strategy to detect color names.

Colors are made up of 3 primary colors; red, green, and blue. In computers, we define each color value within a range of 0 to 255. So, in how many ways we can define a color? The answer is $256 \times 256 \times 256 = 16,581,375$. There are approximately 16.5 million different ways to represent a color. In our dataset, we need to map each color's values with their corresponding names. But don't worry, we don't need to map all the values. We will be using a dataset that contains RGB values with their corresponding names. RGB is a color model with three dimensions – red, green, and blue – that are mixed to produce a specific color. When defining colors in these dimensions, one has to know the sequence of colors in the color spectrum, e.g. that a mix of 100% red and green produces yellow. The RGB color model is often depicted as a cube by mapping the red, green, and blue dimensions onto the x, y, and z axis in 3D space. This is illustrated in the interactive example below, where all possible color mixes are represented within the bounds of the cube.

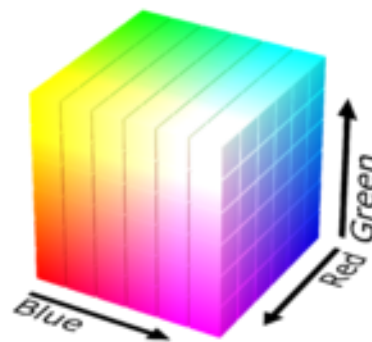


FIG 4.1 RGB COLOR MODULE

HSV is a cylindrical color model that remaps the RGB primary colors into dimensions that are easier for humans to understand. Like the Munsell Color System, these dimensions are hue, saturation, and value.

- *Hue* specifies the angle of the color on the RGB color circle. A 0° hue results in red, 120° results in green, and 240° results in blue.
- *Saturation* controls the amount of color used. A color with 100% saturation will be the purest color possible, while 0% saturation yields grayscale.
- *Value* controls the brightness of the color. A color with 0% brightness is pure black while a color with 100% brightness has no black mixed into the color. Because this dimension is often referred to as brightness, the HSV color model is sometimes called HSB, including in P5.js.

It is important to note that the three dimensions of the HSV color model are interdependent. If the value dimension of a color is set to 0%, the amount of hue and saturation does not matter as the color will be black. Likewise, if the saturation of a color is set to 0%, the hue does not matter as there is no color used. Because the hue dimension is circular, the HSV color model is best depicted as a cylinder. This is illustrated in the interactive example below, where all possible color mixes are represented within the bounds of the cylinder.

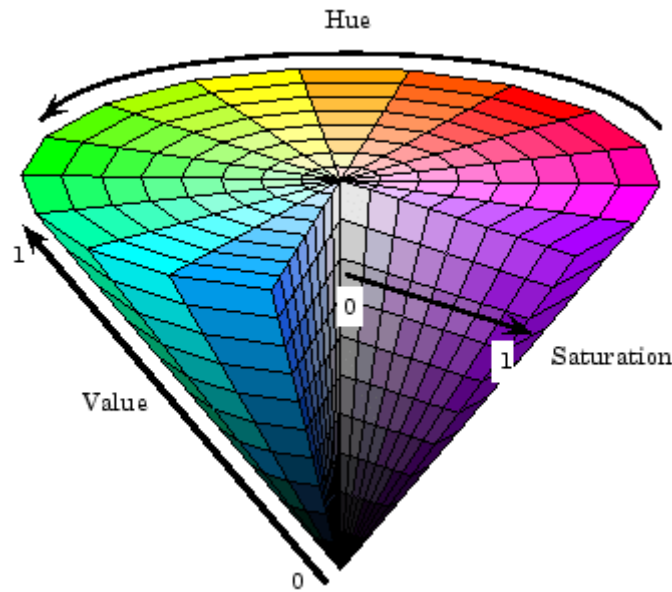


FIG 4.2 HSV MODEL

Once the webcam is on then we will be receiving input as a live video format. This video format will be in Red Green Blue format abbreviated as RGB format. The RGB format is where the color models include a little of all three colours-red, green and blue like an add-on which gives a broad array of different colours when put together. The name of this model is derived from the initials of these three primary colours red, green, and blue.

After the detection of specific colour, we should convert RGB model to HSV model. Hue, saturation, value (HSV) also known as Hue, saturation, brightness (HSB) is most particularly and most often used by the artists because when compared to RGB it is more natural when it comes to hue and saturation other than an add-on, additive or subtractive colour components. HSV is derived from RGB also like a transformation of it, and its colorimetry and components are very relative to the RGB colour space from which it was derived. If we have to detect the object of the specific colour then we have to give the HSV value of that specific colour only. The HSV value of any colour can be derived from the below formula given in figure 4.3.

- colour cone

- $H = \text{hue} / \text{colour in degrees} \in [0, 360]$
- $S = \text{saturation} \in [0, 1]$
- $V = \text{value} \in [0, 1]$

- conversion $RGB \rightarrow HSV$

- $V = \max = \max(R, G, B), \quad \min = \min(R, G, B)$
- $S = (\max - \min) / \max \quad (\text{or } S = 0, \text{ if } V = 0)$
- $H = 60 \times \begin{cases} 0 + (G - B) / (\max - \min), & \text{if } \max = R \\ 2 + (B - R) / (\max - \min), & \text{if } \max = G \\ 4 + (R - G) / (\max - \min), & \text{if } \max = B \end{cases}$
 $H = H + 360, \text{ if } H < 0$

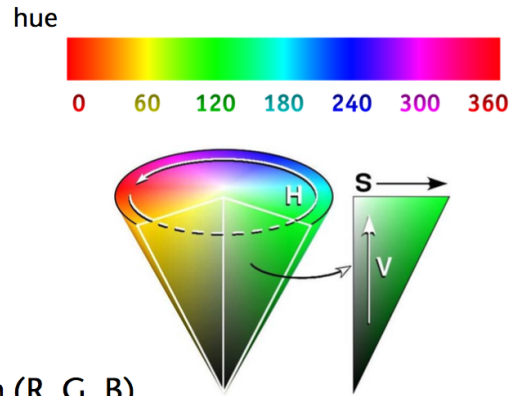


FIG 4.3 RGB TO HSV CONVERSION FORMULA

The flow chart for detecting objects in a particular and track the movements of the object can be done with above color conversion techniques is given below. Since hand is also an object, we can track the hand movements via web cam using colour detection.

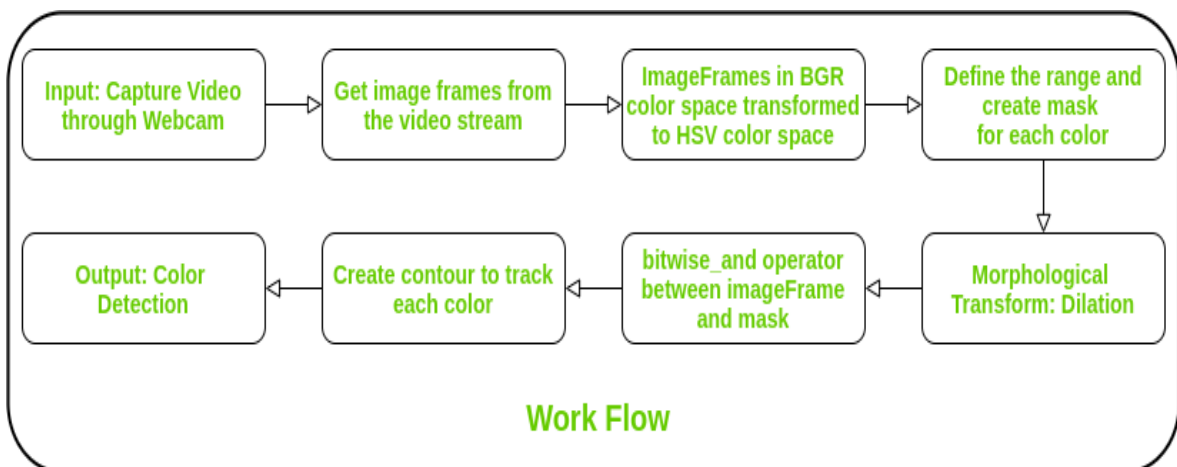


FIG 4.4 COLOR DETECTION FLOW DIAGRAM

4.2.2 KEYBOARD AND MOUSE CONTROL

Once the colour conversion is done then we have to draw contours for the object so that we can see what are the objects detected in the video. In background subtraction modeling uses the new video frame to calculate and update a background model. This application is written in Python 3.6 and it uses the very famous OpenCV library. OpenCV is a computer vision and machine learning software library that includes many common image analysis algorithms that will help us build custom, intelligent computer vision applications. This background model provides a statistical description of the entire background scene.

Foreground detection then identifies pixels in the video frame that cannot be adequately explained by the background model, and outputs them as a binary candidate foreground pixel. During data validation eliminates those pixels that do not correspond to actual moving objects and mark the fore ground pixels which are marked as moving object. Preprocessing of the video is done by converting the input video stream in gray scale video sequencing. To perform video tracking, an algorithm analyses sequential video frames and outputs the movement of targets between the frames.

There are a variety of algorithms, each having strengths and weaknesses. Considering the intended use is important when choosing which algorithm to use. There are two major components of a visual tracking system: target representation and localization, as well as filtering and data association.

Video tracking is the process of locating a moving object (or multiple objects) over time using a camera. It has a variety of uses, some of which are: human-computer interaction, security and surveillance, video communication and

compression, augmented reality, traffic control, medical imaging and video editing.

The *blue Lower* and the *blue Upper* NumPy arrays help us in finding the blue coloured cap. The kernel helps in smoothing blue cap once found. The *bpoints*, *gpoints*, *rpoints* and *ypoints* dequeues are used to store the points drawn on the screen of color blue, green, red and yellow respectively.

As there is little difference in adjacent video frames some of the sequences are skipped in order to better utilization of memory available. To determine the foreground regions, a stabilization algorithm is used to align the current frame with a preceding frame and with a succeeding frame. After alignment, the current frame is subtracted from the two other frames.

Now we use the OpenCV function ***cv2.VideoCapture()*** method to read a video, frame by frame (using a while loop), either from a video file or from a webcam in real-time. In this case, we pass 0 to the method to read from a webcam. Once we start reading the webcam feed, we constantly look for a blue color object in the frames with the help of ***cv2.inRange()*** method and use the *blue Upper* and *blue Lower* variables initialized in Step 0. Once we find the contour, we do a series of image operations and make it smooth. They just make our lives easier. If you want to know more about these operations — erode, morph and dilate, check this out.

Once we find the contour (the ***if*** condition passes when a contour is found), we use the center of the contour (blue cap) to draw on the screen as it moves. Now we start tracking coordinates of each and every point the center of the contour touches on the screen, along with its color. We store these set of points of different colors in different dequeues (*bpoints*, *gpoints* etc.).

When the center of the contour touches one of the colored boxes we put on the screen in Step 1, we store the points in its respective color deque. So far we

stored all the points in their respective color dequeues. Now we just join them using a line of their own color. The OpenCV function **`cv2.line()`** comes in handy for us to do that. The following code does the same.

This is a simple demonstration of the image processing capabilities of OpenCV. If you find this either interesting or easy or boring or think that there is nothing challenging about it, I urge you to try adding more paint features like an eraser, different types of brushes, color-fill options etc. to really get a feel of how fun OpenCV can be.

The result of each subtraction is thresholded to form a binary image that represents the locations of foreground objects in the two subtracted frames. To know the locations of the foreground objects in the current frame, the results of the two subtractions are combined by an AND operation.

Foreground pixel detection compares the input video frame with the background model and identifies candidate foreground pixels from the input frame. The most commonly used approach for foreground detection is to check whether the input pixel is significantly different from the corresponding background estimate.



FIG 4.5 Initial color detection with a noisy image

Contours is to draw lines, outlines or the shapes of something, which can be also defined as the line which joins all the points along the boundary of an image identical or same intensity. In shape analysis, contours are very convenient to use, to find the size of the object, and to detect the object. OpenCV has a function called `findContour()` which helps in removal of the contours from the given image. Thresholding techniques, sober edges are initially applied because it works the best in binary images.

After the detection of blue colour in the input image, a median filter has been used to filter out the noise. Median filtering basically is a nonlinear operation mostly used in image processing to reduce noise ("salt and pepper"). When the goal is to simultaneously reduce noise and preserve edges, a median filter is better and effective than convolution. By using the above techniques, we can eliminate the noises around the image. So, the resultant image will be as shown below.



FIG 4.6 NOICELESS CLEAR DETECTION

Python provides a library named keyboard which is used to get full control of the keyboard. It's a small Python library which can hook global events, register hotkeys, simulate key presses and much more.

- It helps to enter keys, record the keyboard activities and block the keys until a specified key is entered and simulate the keys.
- It captures all keys; even onscreen keyboard events are also captured.
- Keyboard module supports complex hotkeys.
- Using this module, we can listen and send keyboard events.
- It works on both windows and Linux operating system.

To ignore other objects with same color we have to limit the range of areas of the object to be detected so, when we give the area range of the hands only hands are detected. After detection, the location of the hands in the window the x coordinates and y coordinates of the contours of the hand are passed as an argument using Python built in functions. By using the contours in the graph window, we can get the area of the hand and the center coordinates of the hand area. After getting the center point of two hands the distance between them is used for calling the user defined acceleration() function is called, if the value of distance is TRUE .This acceleration function invokes the keyboard and gives an input forward key to move to object character in the game. The center points of

two hands, they are used again to find the slope between those two points by the given formula. If the slope value is negative then the user defined function Turning() is called which invokes the character to turn left and if it is positive then it invokes to turn right. And OpenCV provides us various algorithms to draw rectangles 3 and drawing lines in the result window. This can be done by a python library known as keyboard. This provides us to undertake the whole keyboard control via the program.

The end points for this type of problem will be given in written format or on a graph. The written format looks like (x1, y1) and (x2, y2). The numbers in this example are simply subscripts, meaning they're just in place to differentiate between the two end points (end point 1 and end point 2). If there is a graph, use that to determine the end points. It doesn't matter which end point is used, as end point 1 and which is used as end point 2. They should simply be labelled like this to make it easier to tell which one is which in the next part of determining the distance.

The next step in determining the distance between two points is to use the aptly named distance formula. This formula looks like this: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. This might look like a complex formula but, by following the order of operations, it can be easy to solve. Once the equation is solved, the student will know what the exact distance is between the two end points.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

As distance we can also find the slope between two points by slope formula. Basically, the point-slope form is derived from the concept of finding the

slope of a line when two points are given. In other words, if you can remember the slope formula then the point-slope form should naturally come to you. The slope, m , of a line passing through two arbitrary points $\left(\{x_1, y_1\}\right)$ (x_1, y_1) and $\left(\{x_2, y_2\}\right)$ (x_2, y_2) is calculated as follows.

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

If we slightly tweak how the two points “look”, say for instance

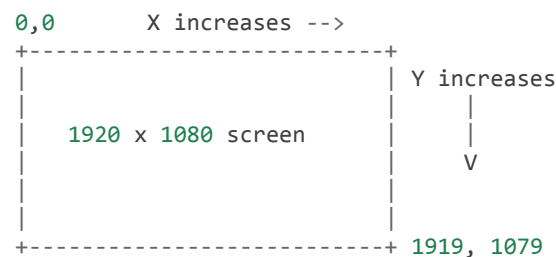
- Let $\left(\{x_1, y_1\}\right)$ (x_1, y_1) be a **specific point** on a line
- Let $\left(\{x, y\}\right)$ (x, y) be **any other point** on the line

Generally, in this theory any line which is x_1 and y_2 values are higher than x_2 and y_1 values or vice versa, the resulting value will be negative. In simple any line slanting towards leftwards down will have a negative slope value. Likewise any line slanting towards rightwards down will have a positive slope value.

So, we perform turning left operation while the slope is negative and perform turning right operation while slope is positive. In all game the basic movements for the gaming object or avatar are WASD. “W” is for forward, “S” is for reverse or brake, “A” is for turning left and “D” is for turning right. So, when acceleration function is called it invokes “W” key from keyboard, and while break function is called “S” key will be invoked, but also while turning function is called the “A” and “D” will be called as per the slope values.

Some game needs mouse control to play games. In order to make a virtual mouse we need the help of the python library PyAutoGUI which provides various functions to control mouse movements and operations. Locations on your screen

are referred to by X and Y Cartesian coordinates. The X coordinate starts at 0 on the left side and increases going right. Unlike in mathematics, the Y coordinate starts at 0 on the top and increases going down.



The pixel at the top-left corner is at coordinates 0, 0. If your screen's resolution is 1920 x 1080, the pixel in the lower right corner will be 1919, 1079 (since the coordinates begin at 0, not 1).

The screen resolution size is returned by the `size()` function as a tuple of two integers. The current X and Y coordinates of the mouse cursor are returned by the `position()` function.

```
>>> pyautogui.size()
(1920, 1080)
>>> pyautogui.position()
(187, 567)
```

The `moveTo()` function will move the mouse cursor to the X and Y integer coordinates you pass it. The `None` value can be passed for a coordinate to mean “the current mouse cursor position”. For example:

```
>>> pyautogui.moveTo(100, 200) # moves mouse to X of 100, Y of 200.
>>> pyautogui.moveTo(None, 500) # moves mouse to X of 100, Y of 500.
```

```
>>> pyautogui.moveTo(600, None) # moves mouse to X of 600, Y of 500.
```

Normally the mouse cursor will instantly move to the new coordinates. If you want the mouse to gradually move to the new location, pass a third argument for the duration (in seconds) the movement should take.

The click() function simulates a single, left-button mouse click at the mouse's current position. A "click" is defined as pushing the button down and then releasing it up. For example:

```
>>> pyautogui.click() # click the mouse
```

To combine a moveTo() call before the click, pass integers for the x and y keyword argument:

```
>>> pyautogui.click(x=100, y=200) # move to 100, 200, then click the left mouse button.
```

To specify a different mouse button to click, pass 'left', 'middle', or 'right' for the button keyword argument:

```
>>> pyautogui.click(button='right') # right-click the mouse
```

To do multiple clicks, pass an integer to the clicks keyword argument. Optionally, you can pass a float or integer to the interval keyword argument to specify the amount of pause between the clicks in seconds. For example:

```
>>> pyautogui.click(clicks=2) # double-click the left mouse button
```

```
>>> pyautogui.click(clicks=2, interval=0.25) # double-click the left mouse button,  
but with a quarter second pause in between clicks
```

```
>>> pyautogui.click(button='right', clicks=3, interval=0.25) ## triple-click the right  
mouse button with a quarter second pause in between clicks
```

As a convenient shortcut, the `doubleClick()` function will perform a double click of the left mouse button. It also has the optional `x`, `y`, `interval`, and `button` keyword arguments. For example:

```
>>> pyautogui.doubleClick() # perform a left-button double click
```

There is also a `tripleClick()` function with similar optional keyword arguments. The `rightClick()` function has optional `x` and `y` keyword arguments.

The primary keyboard function is `write()`. This function will type the characters in the string that is passed. To add a delay interval in between pressing each character key, pass an int or float for the `interval` keyword argument.

For example:

```
>>> pyautogui.write('Hello world!') # prints out "Hello world!" instantly
>>> pyautogui.write('Hello world!', interval=0.25) # prints out "Hello world!" with a
quarter second delay after each character
```

You can only press single-character keys with `write()`, so you can't press the Shift or F1 keys, for example.

To press these keys, call the `press()` function and pass it a string from the `pyautogui.KEYBOARD_KEYS` such as `enter`, `esc`, `f1`. See `KEYBOARD_KEYS`. For example:

```
>>> pyautogui.press('enter') # press the Enter key
>>> pyautogui.press('f1') # press the F1 key
>>> pyautogui.press('left') # press the left arrow key
```

The `press()` function is really just a wrapper for the `keyDown()` and `keyUp()` functions, which simulate pressing a key down and then releasing it up. These functions can be called by themselves. For example, to

press the left arrow key three times while holding down the Shift key, call the following:

```
>>> pyautogui.keyDown('shift') # hold down the shift key
>>> pyautogui.press('left')    # press the left arrow key
>>> pyautogui.press('left')    # press the left arrow key
>>> pyautogui.press('left')    # press the left arrow key
>>> pyautogui.keyUp('shift')   # release the shift key
```

To press multiple keys similar to what write() does, pass a list of strings to press(). For example:

```
>>> pyautogui.press(['left', 'left', 'left'])
Or you can set how many presses left:
>>> pyautogui.press('left', presses=3)
```

To add a delay interval in between each press, pass an int or float for the interval keyword argument.

To make pressing hotkeys or keyboard shortcuts convenient, the hotkey() can be passed several key strings which will be pressed down in order, and then released in reverse order. This code:

```
>>> pyautogui.hotkey('ctrl', 'shift', 'esc')
...is equivalent to this code:
>>> pyautogui.keyDown('ctrl')
>>> pyautogui.keyDown('shift')
>>> pyautogui.keyDown('esc')
>>> pyautogui.keyUp('esc')
>>> pyautogui.keyUp('shift')
>>> pyautogui.keyUp('ctrl')
```

To add a delay interval in between each press, pass an int or float for the interval keyword argument.

CHAPTER-5

RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS

5.1 RESULT

In the experiment, two approaches are proposed. One is controlling keyboard elements for playing games and another one is controlling mouse for playing games. When I start the implementation, I went with a simple approach of detecting color. This color detection results in lot of noise in the color or object. When movements occur noise of the object increases even more. So that even the object is detected we can't monitor its movements accurately due to the noise occur. This will affect the whole concept of this project. So, we have to find a way to avoid the noise occurs during video monitoring. So, we have to use blurring concepts and edge smoothening to get a clear-cut detecting of object.



FIG 5.1 INITIAL DETECTION WITH A NOISY VIDEO

So, we used a white glove as white has an extreme unique color value and we used smoothing and blurring the contours we got a result of very clear noise less image.

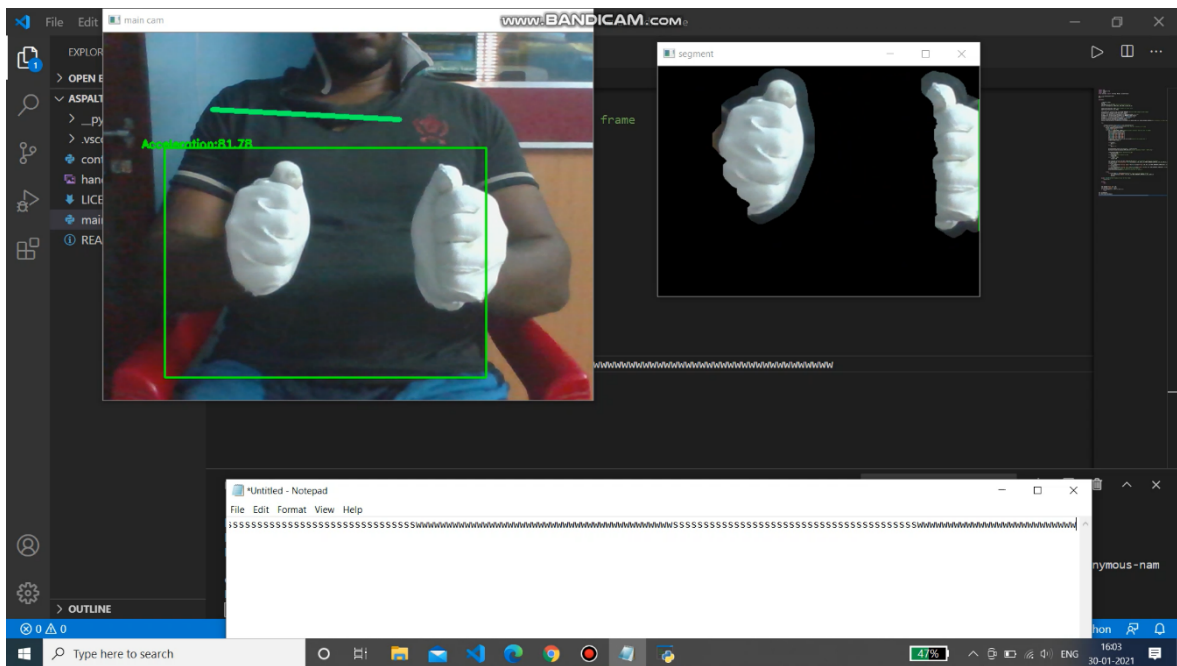


FIG 5.2 FORWARD OPERATION

And after detections of hand with a white glove the implementation of controls by calculating distance and slope of the white glove gave us a clear-cut detection and monitoring the movements occurred without any flaw. If there is a distance value of hands then it will invoke “W” key which helps to move a car or an object in a game forward. The primary keyboard function is write(). This function will type the characters in the string that is passed. To add a delay interval in between pressing each character key, pass an int or float for the interval keyword argument.

To press these keys, call the press () function and pass it a string from the pyautogui. KEYBOARD_KEYS such as enter, esc, f1. See KEYBOARD_KEYS.

For example:

```
>>> pyautogui.press('W') # press the w key
```

Then when the two hands come closer and intersect with each other. Then the contours of the two objects are joined. And so the distance between them will be zero, hence it will call break function which will invoke “S” key.

To press these keys, call the `press ()` function and pass it a string from the `pyautogui`. `KEYBOARD_KEYS` such as `enter`, `esc`, `f1`. See `KEYBOARD_KEYS`.

For example:

```
>>> pyautogui.press('S') # press the S key
```

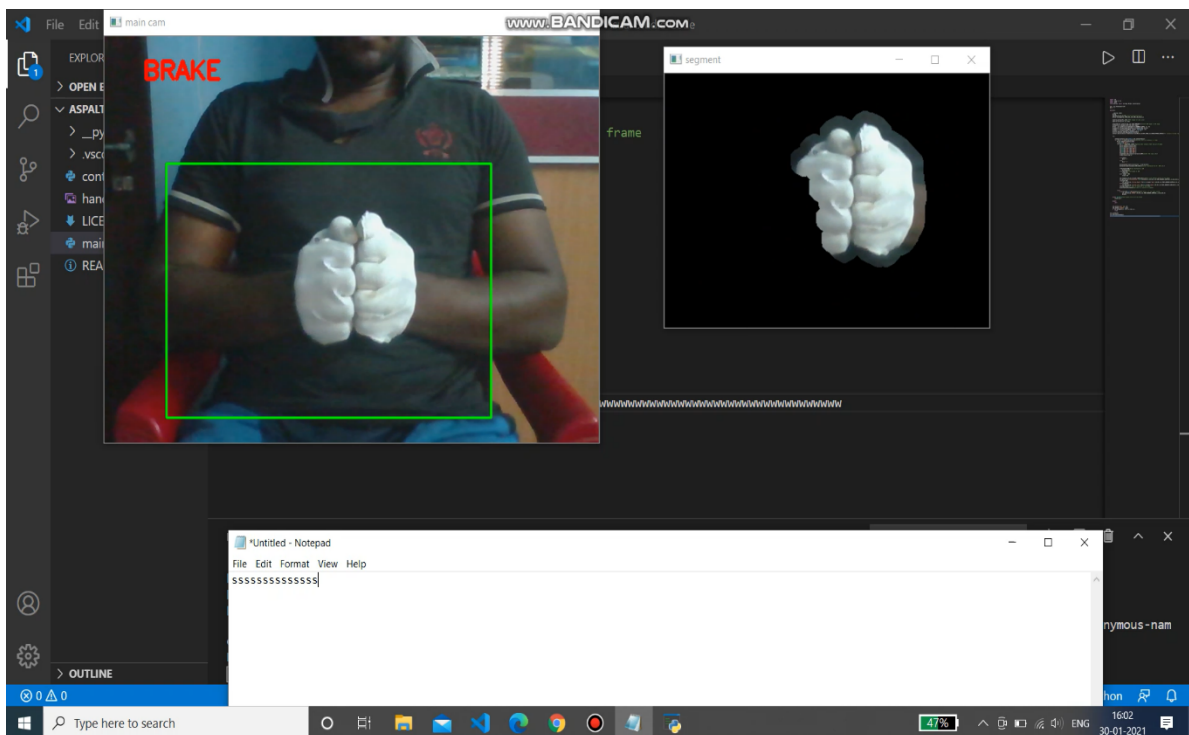


FIG 5.3 APPLING BREAK OR REVERSE

And as told before we are calculating slope for turning function. As a fact if any line is slanting towards leftwards down then the slope will be always negative and if slanting toward rightwards down then the slope will be always positive.

So if the slope is less than -50 then car will turn left by invoking “A” key and if slope is greater the 50 then the car will be turning left by invoking “D” key.

To press these keys, call the `press ()` function and pass it a string from the `pyautogui`. `KEYBOARD_KEYS` such as `enter`, `esc`, `f1`. See `KEYBOARD_KEYS`.

For example:

```
>>> pyautogui.press('D') # press the D key
```

```
>>> pyautogui.press('A') # press the A key
```

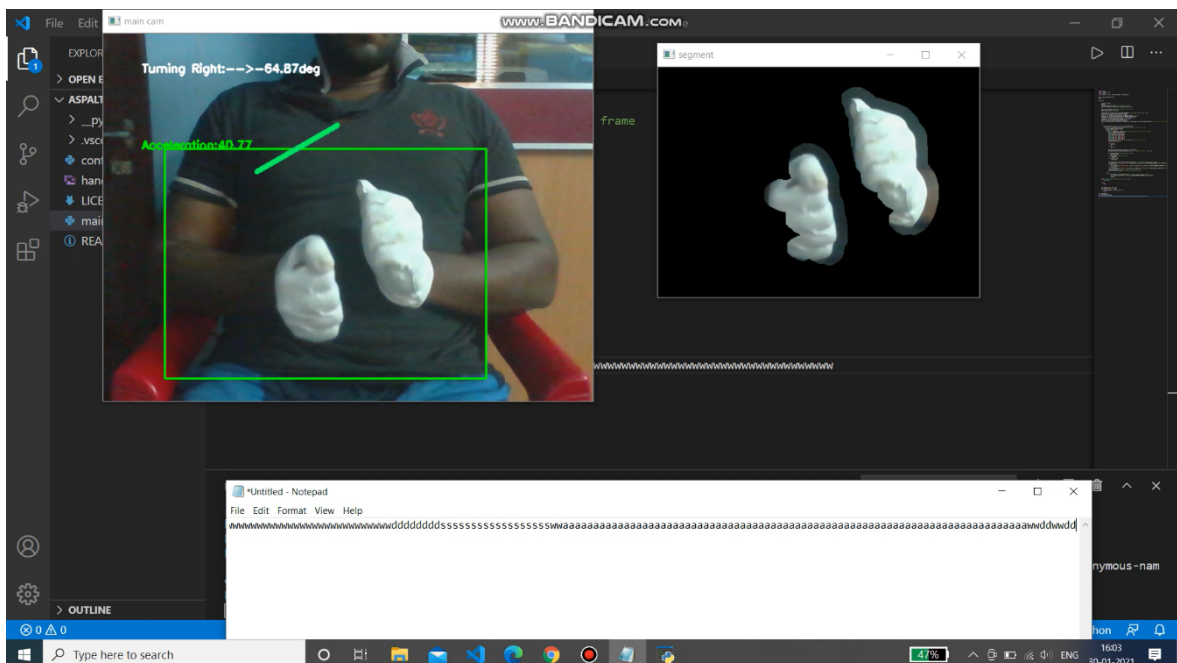


FIG 5.4 TURNING RIGHT

Likewise for turning left using the same concept is achieved when the slope value is less than -50. Hence the major keyboard controls for playing the game is successfully achieved.

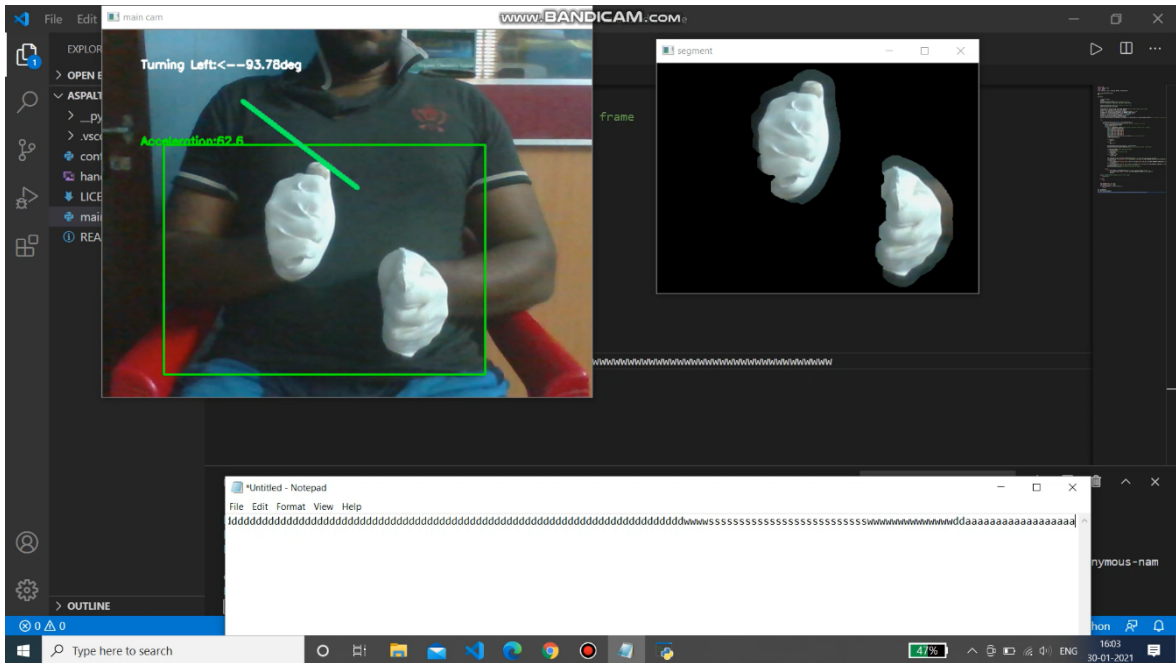


FIG 5.4 TURNING LEFT

Then coming to mouse control, it is more complicate than keyboard control. For performing mouse click operations we cannot detect just one color as done before. So, I used two extreme colors RED and BLUE. We can use a marker cap or a duct tape of these colors at our fingertip to use for detection. These two different colors are detected separately with different HSV values. As with the above discussed implementation with a fixed range of distance between red and blue with the help of mid-point the mouse curser is moved according to the graph point location of the window screen.

It will be continuously running in the while loop until the program is closed every point will be passed to the move function and the cursor will be moved accordingly. The moveTo() function will move the mouse cursor to the X and Y integer coordinates you pass it. The None value can be passed for a coordinate to mean "the current mouse cursor position". For example:

```
>>>while(){ pyautogui.moveTo( new_x, new_y)) # moves mouse to X of 100, Y of 200.
```

Normally the mouse cursor will instantly move to the new coordinates. If you want the mouse to gradually move to the new location, pass a third argument for the duration (in seconds) the movement should take.

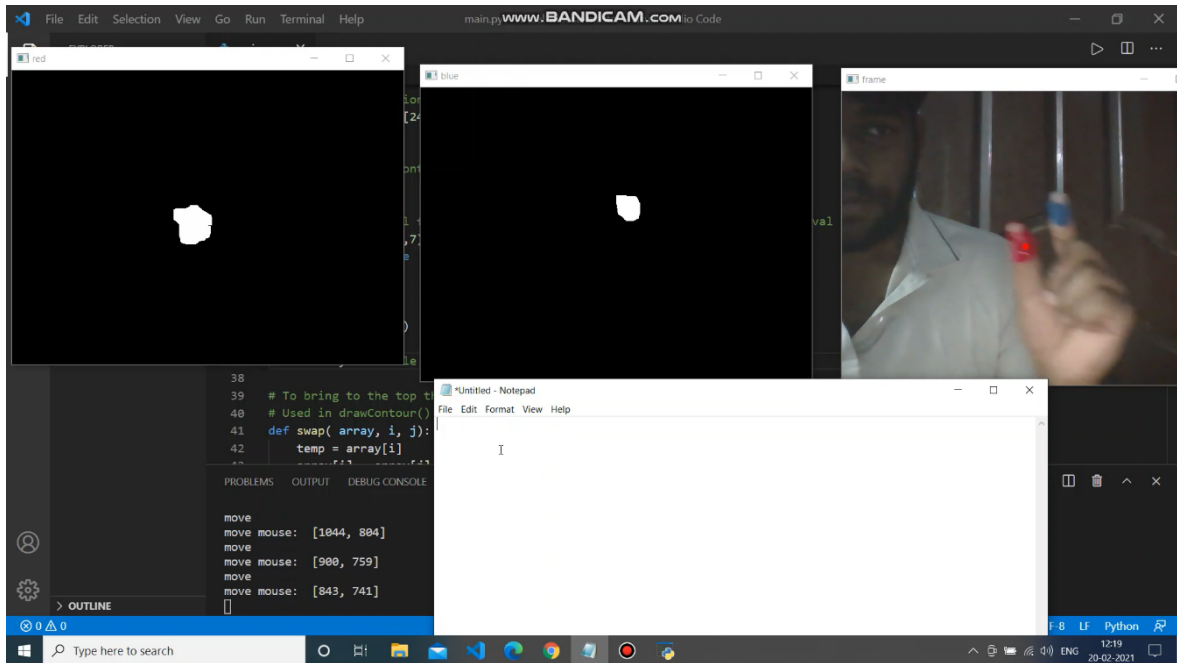


FIG 5.5 MOUSE MOVMENTS

Then for left click operation if the distance is zero then it will perform LEFT click event. In figure.5.6 we have typed random words and we are trying to select it using LEFT click operation.

To specify a different mouse button to click, pass 'left', 'middle', or 'right' for the button keyword argument:

```
>>> pyautogui.click(button='LEFT') # left-click the mouse
```

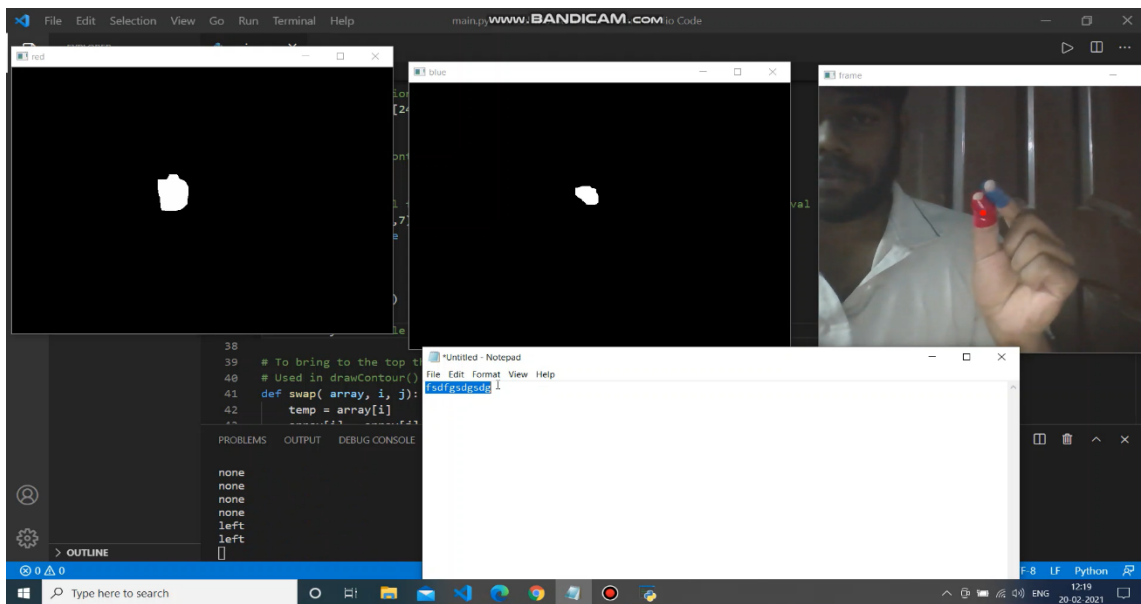


FIG 5.6 LEFT CLICK OPERATION

Then for RIGHT click operation if the distance is far exceeding then it will perform RIGHT click event. In figure.11 we are trying to use RIGHT click operation to open drop down list and then we are performing LEFT click operation to select the delete option. As both keyboard and mouse events can be controlled using hand gesture via color and edge detection this can be used for playing games which gives more realistic feeling and more joyful. With some perfections in this project, it can be used by real time users more efficiently.

To specify a different mouse button to click, pass 'left', 'middle', or 'right' for the button keyword argument:

```
>>> pyautogui.click(button='right') # right-click the mouse
```

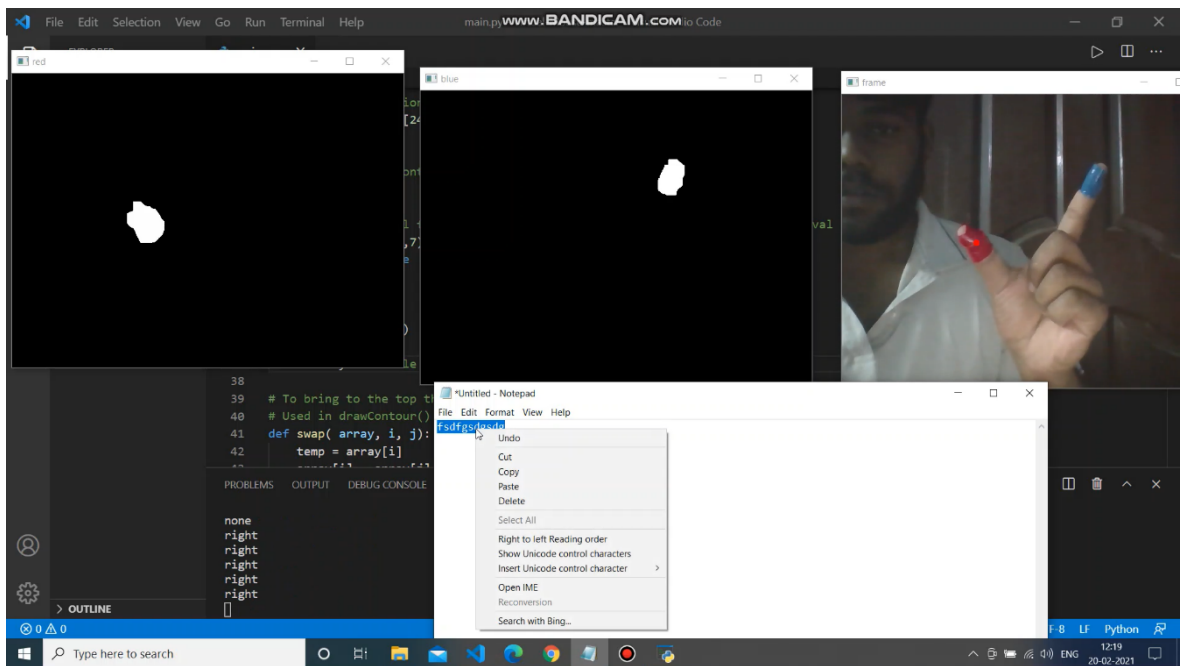


FIG 5.7 RIGHT CLICK OPERATION

CHAPTER - 6

CONCLUSION AND FUTURE WORK

6.1 CONCLUTION

Gesture based interfaces allow human computer interaction to be in a natural and intuitive manner. The most important advantage of the usage of hand gesture based input modes is that using this method the user can interact with the application from a distance without any physical interaction with the keyboard or mouse. This paper develops a hand gesture recognition system for interacting with different applications like image browser; games etc. and provides a fruitful solution towards a user-friendly interface between human and computer. The gesture vocabulary designed can be further extended for controlling different applications like game control etc. As the system provides the flexibility to the users and specifically physically challenged users to define the gesture according to their feasibility and ease of use.

Advantages of this project is, as we are not using any external hardware or any costly sensors, the cost of the implementation is NONE. And also, we are not using any data sets of hands the space complexity of the program is also very effective so that when this is implemented in a game control setting it won't be result in increase in memory for the game. And as playing or controlling the pc with hands and without touching anything gives a fantastic and exciting experience.

Though there are many advantages, we are facing some disadvantages too. As laptop webcams have basic lenses for capturing video, exposure of a bright light or a dim light and also scattering effect may cause some trouble while detecting. And also, many objects with same size while detecting can also be a problem. While controlling mouse we are facing issue like cursor movement is only 45% slower and less stable than using a real mouse. If in future web cam lenses are

modified for this requirement it can be a great technology.

6.1 FUTURE WORK

In this work, we described a design process for the development of a gesture-based game, where manual wheelchair users (and their stakeholders) acted as participants in the sessions. Values from the participants emerged during the design sessions and were explored for the development of the game. Sessions were aligned to Game SCRUM, employing concepts such as user stories, prioritization, and sprints. Creative Process elements were also employed to stimulate the emerging of values.

Our study aims to develop a game focused on entertainment that reflects the values of the wheelchair users with the sole purpose of entertainment. With the possibility of reflection of values in an entertainment game, we will be able, in future works, to carry out studies with physical and educational challenges, as continuity of the research described in the current study.

We believe that the game can further be improved based on the results and testimonials retrieved during the experimentation. Even though we understood that the preproduction prototype of the game reached its objective in representing values of the users, it can be adapted to encompass a broader public. Possibly, the next group of people to be involved in the game content are wheelchair users with more limited functional abilities than the users that participated in the development of this study.

REFERENCES

- [1] Y. Cui and J. Weng, "View-based hand segmentation and hand sequence recognition with complex backgrounds," in Proceedings of 13th ICPR. Vienna, Austria, Aug. 1996, vol. 3, pp. 617– 621.
- [2] T. Lindeberg, "Feature detection with automatic scale selection," IJCV, vol. 30, pp. 77–116, June 2004.
- [3] X. W. Li and X. R. Zhang, "A perceptual color edge detection algorithm," International Conference on Computer Science and Software Engineering, vol. 1, pp.297-300, 2008.
- [4] Yuan Chun-lan, Xiong Zong-long, Zhou Xue-hua, Peng Xiao-hui. "Research on Image Edge Detection Based on Sobel operator ". Laser and Infrared, 2009,39(1):85-87,
- [5] M. Panwar and P. S. Mehra, "Hand Gesture Recognition for Human Computer Interaction", International Conference on Image Information Processing, 2011
- [6] L. Cruz, D. Lucio and L. Velho, "Kinect and RGBD Images: Challenges and Applications", IEEE Computer Society, SIBGRAPI Tutorials, pp. 36- 49, 2012.
- [7] H. L. Zhang, H. Y. Li, M. Zhou, Hand posture recognition based on multi-feature and compressive sensing, Journal of Hunan University, Vol.40, No.3, 87-92, 2013.

[8] D. Qurratu'aini, A. Sophian, W. Sediono, H.M. Yusof and S. Sudirman, "Visual-based fingertip detection for hand rehabilitation", *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 9, no. 2, pp. 474-480, 2018

[9] Oana Saman, Loredana Stanciu, "Hand Mobility Recovery Through Game Controlled by Gestures", IEEE 13th International Symposium on Applied Computational Intelligence and Informatics, 2019

APPENDIX

A. SOURCE CODE:

main.py

```
import cv2
import numpy as np
import math
from controls import turning, Brake, acceleration

cap = cv2.VideoCapture(0)
Dir="-->"

while(1):

    _,img=cap.read()
    im=img
    img=img[250:450,150:500] #Define RegionOfInterest(im,(500,150),(80,450),(0,2
55,0),2)
    im=cv2.rectangle(im,(500,250),(150,450),(0,255,0),2)

    lower=np.array([0,0,168]) #HSV ranges for skin color
    upper=np.array([172,111,255])
```

```

#lower=np.array([0,30,60]) #HSV ranges for skin color
#upper=np.array([20,150,255])

converted=cv2.cvtColor(img,cv2.COLOR_BGR2HSV) #convert BGR image to H
SV image
skinMask=cv2.inRange(converted,lower,upper)
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 7))
skinMask=cv2.morphologyEx(skinMask,cv2.MORPH_CLOSE,kernel)
skinMask = cv2.dilate(skinMask, kernel, iterations = 4)
skinMask = cv2.GaussianBlur(skinMask, (7,7), 0)
skin=cv2.bitwise_or(img,img,mask=skinMask)

contours,hierarchy=cv2.findContours(skinMask,cv2.RETR_TREE,cv2.CHAIN_A
PPROX_SIMPLE) #fit contours to hand regions

try:

cnt=max(contours,key=lambda x:cv2.contourArea(x))
for ind,ct in enumerate(contours):#iterate over all contours in frame
    M=cv2.moments(contours[ind])
    area=int(M["m00"])

    if area in range(6000,13000): #Ignore other contours that are not of hands
        m1=cv2.moments(contours[0])
        m2=cv2.moments(contours[1])
        x1=int(m1["m10"]/m1["m00"])
        y1=int(m1["m01"]/m1["m00"])
        x2=int(m2["m10"]/m2["m00"])
        y2=int(m2["m01"]/m2["m00"])
        slope=math.tan(((y2-y1)/(x2-x1)))*100#convert the slope into %
        slope=round(slope,2)

```

```

if slope>0:
    Dir="<--"
else:
    Dir="-->"

distance=math.sqrt(((x2-x1)**2) + ((y2-y1)**2))
distance=round((distance/300)*100,2)#convert distance from 0 - 300 into
%

if(distance>100): #limit distance to 100
    distance=100
if slope>100: #limit angle to 100
    slope=100
elif slope< -100:
    slope=-100

cv2.line(im,(x1,y1),(x2,y2),(100,255,0),5)#plot line between centres of tw
o hands

cv2.putText(im,"Acceleration:"+(str(distance)),(50,150),cv2.FONT_HERS
HEY_SIMPLEX,0.5,(0,255,0),2)
if(slope<-50):
    cv2.putText(im,"Turning Right:"+Dir+str(slope)+"deg",(50,50),cv2.FON
T_HERSHEY_SIMPLEX,0.5,(255,255,255),2)
elif(slope>50):
    cv2.putText(im,"Turning Left:"+Dir+str(slope)+"deg",(50,50),cv2.FONT
_HERSHEY_SIMPLEX,0.5,(255,255,255),2)
    turning(slope)#set xAxis on Joystick to the slope %
    acceleration(distance)#set yAxis on Joystick to speed %

else:
    if area>1300 and len(contours)==1: #If area is of two hands joined

```

```

        cv2.putText(im,"BRAKE",(50,50),cv2.FONT_HERSHEY_SIMPLEX,1,(
0,0,255),4)
        Brake()

```

```

except ValueError: #If hands are out of the frame
    print("pp")

```

```

except:
    pass

```

```

cv2.imshow('main cam',im)
cv2.imshow('segment',skin)
if cv2.waitKey(1) & 0xFF == ord('q'):
    break

```

```

cv2.waitKey(0)
cv2.destroyAllWindows()

```

controls.py

```

from pynput.keyboard import Key, Controller
keyboard = Controller()

```

```

def acceleration(distance):
    if(distance):
        keyboard.press('w')

```

```

def turning(angle):
    if(angle<-50):
        keyboard.press('d')
        cv2.putText(im,"Turning Right:"+Dir+str(slope)+"deg",(50,50),cv2.FONT_HERSHEY_SIMPLEX,0.5,(255,255,255),2)

```

```

elif(angle>50):
    keyboard.press('a')
    cv2.putText(im,"Turning Left:"+Dir+str(slope)+"deg",(50,50),cv2.FONT_HERSHEY_SIMPLEX,0.5,(255,255,255),2)

def Brake():
    keyboard.press('s')

```

Mouse.py

```

import numpy as np
from imutils.video import WebcamVideoStream
import imutils
import cv2
import math
import pyautogui

pyautogui.FAILSAFE=False

# setting the lower and upper range for mask1,[H,S,V], for lighter shades
lower_red = np.array([0,70,70])
upper_red = np.array([10,255,255])

# setting the lower and upper range for mask2, for darker shades
lower_red = np.array([170,70,70])

```

```

upper_red = np.array([180,255,255])

# setting the lower and upper range for mask, for blue
lower_b = np.array([88,78,20])
upper_b = np.array([128,255,255])

# Prior initialization of all centers for safety
one_cen, two_cen = [240,320],[240,320]
cursor = [960,540]

# Area ranges for contours of different colours to be detected
area = [300,1900]

# Rectangular kernal for eroding and dilating the mask for primary noise removal
kernel = np.ones((7,7),np.uint8)
showCentroid = False
cur_pos = [240,320]
contour = -1

dim=pyautogui.size() # Dimension of Screen

sensitivity= 4 #Scale like 1,2,3,4,5 where 1 is lowest sensitivi and 5 max

# To bring to the top the contours with largest area in the specified range
# Used in drawContour()
def swap( array, i, j):
    temp = array[i]
    array[i] = array[j]
    array[j] = temp

def centroid(vid, contour):
    M = cv2.moments(contour)

```

```

if M['m00'] != 0:
    cx = int(M['m10']/M['m00'])
    cy = int(M['m01']/M['m00'])
    center = (cx,cy)
    if showCentroid:
        cv2.circle( vid, center, 5, (0,0,255), -1)

    return center

# Contours on the mask are detected.. Only those lying in the previously set area
# range are filtered out and the centroid of the largest of these is drawn and return
ed
def drawCentroid(vid, color_area, mask, showCentroid):

    ret=False
    center= (-1,-1)

    contour, _ = cv2.findContours( mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_
SIMPLE)

    l=len(contour)
    area = np.zeros(l)

    # filtering contours on the basis of area range specified globally
    for i in range(l):
        if cv2.contourArea(contour[i])>color_area[0] and cv2.contourArea(contour[i])<
color_area[1]:
            area[i] = cv2.contourArea(contour[i])
        else:
            area[i] = 0

    a = sorted( area, reverse=True)

```

```

# bringing contours with largest valid area to the top
for i in range(l):
    for j in range(1):
        if area[i] == a[j]:
            swap( contour, i, j)

if l>0:
    center = centroid(vid, contour[0])
    ret = True
return ret, center

def setCursorPos( yc, pyp):

    if abs(yc[0]-pyp[0])<5 and abs(yc[1]-pyp[1])<5:
        y1 = yc[0] + .7*(pyp[0]-yc[0])
        y2 = yc[1] + .7*(pyp[1]-yc[1])
    else:
        y1 = yc[0] + .1*(pyp[0]-yc[0])
        y2 = yc[1] + .1*(pyp[1]-yc[1])

    return round(y1), round(y2)

# Distance between two centroids
def distance( c1, c2):
    distance = pow( pow(c1[0]-c2[0],2) + pow(c1[1]-c2[1],2) , 0.5)
    return distance

def chooseAction(x):
    out = 'none'
    if x<40:
        out = 'left'

```

```

elif x>150:
    out = 'right'
elif x>60 and x<100:
    out = 'move'
print(out)
return out

def move_mouse(pos):

    #config properly????????????!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! probably change 2.8 to s
    omething else
    cursor[0] = 3*(pos[0])
    cursor[1] = 3*(pos[1])
    if cursor[0]<=dim[0] and cursor[1]<=dim[1]:
        print('move mouse: ', cursor)
        pyautogui.moveTo(cursor[0],cursor[1])

def click_mouse(clicks,btn):
    pyautogui.click(clicks=clicks,button=btn)

def do_action(action, pos):
    if action == 'move':
        move_mouse(pos)
    elif action == 'right':
        click_mouse(1, 'right')
    elif action == 'left':
        click_mouse(1, 'left')
    else:
        print(action)

vs = WebcamVideoStream(src=0).start()

```

```

while True:
    k = cv2.waitKey(10) & 0xFF

    frame = vs.read()
    frame = cv2.bilateralFilter(frame, 5, 50, 100)
    frame = cv2.flip(frame, 1)
    hsv = cv2.cvtColor( frame, cv2.COLOR_BGR2HSV)

    mask1 = cv2.inRange(hsv, lower_red, upper_red)
    mask2 = cv2.inRange(hsv, lower_red, upper_red)
    mask1 = mask1+mask2
    mask1 = cv2.morphologyEx(mask1, cv2.MORPH_OPEN, kernel)
    mask1 = cv2.dilate(mask1, kernel, iterations = 1)

    maskb = cv2.inRange(hsv, lower_b, upper_b)
    maskb = cv2.morphologyEx(maskb, cv2.MORPH_OPEN, kernel)
    maskb = cv2.dilate(maskb, kernel, iterations = 1)

    oldpos = cur_pos

    ret1, one_cen = drawCentroid( frame, area, mask1, showCentroid)
    ret2, two_cen = drawCentroid( frame, area, maskb, showCentroid)

    if ret1==True and ret2==True:
        cur_pos = setCursorPos(one_cen, oldpos)
        cv2.circle( frame, cur_pos, 5, (0,0,255), -1)

        x = distance(one_cen, two_cen)
        do_action(chooseAction(x), cur_pos)

    else:
        print('oops')

```

```
cv2.imshow('blue', maskb)
cv2.imshow('red', mask1)
cv2.imshow('frame', frame)
```

```
if k == 27:
    break
```

```
cv2.destroyAllWindows()
vs.stop()
```

webcamvideostream.py

```
from threading import Thread
import cv2
```

```
class WebcamVideoStream:
    def __init__(self, src=0, name="WebcamVideoStream"):
        # initialize the video camera stream and read the first frame
        # from the stream
        self.stream = cv2.VideoCapture(src)
        (self.grabbed, self.frame) = self.stream.read()
        #self.stream.set(cv2.CAP_PROP_FPS, 10)
        #self.stream.set(cv2.CV_CAP_PROP_FRAME_WIDTH, dimm[0])
        #self.stream.set(cv2.CV_CAP_PROP_FRAME_HEIGHT, dimm[1])

        # initialize the thread name
        self.name = name

        # initialize the variable used to indicate if the thread should
        # be stopped
        self.stopped = False
```

```
def start(self):
    # start the thread to read frames from the video stream
    t = Thread(target=self.update, name=self.name, args=())
    t.daemon = True
    t.start()
    return self
```

```
def update(self):
    # keep looping infinitely until the thread is stopped
    while True:
        # if the thread indicator variable is set, stop the thread
        if self.stopped:
            return

        # otherwise, read the next frame from the stream
        (self.grabbed, self.frame) = self.stream.read()
```

```
def read(self):
    # return the frame most recently read
    return self.frame
```

```
def stop(self):
    # indicate that the thread should be stopped
    self.stopped = True
```

B. SCREENSHOT

