Nitte Meenakshi Institute Of Technology

Govindapur Bangalore 560064

(An Autonomous Institution, affiliated to VTU Belgaum, and Approved By AICTE)

PROJECT REPORT

ON

"VIRTUAL PAINT"

Submitted in Partial fulfilment of the requirements of the degree of

MASTER'S OF COMPUTER APPLICATION

IN

COMPUTER APPLICATION

Submitted by

VISHAL 1NT22MC118

Under the guidance of

MS.SOWMYA K

Department of MCA

Nitte Meenakshi Institute Of Technology

Govindapur Bangalore 560064

(2023-2024)

NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

GOVINDAPUR BANGALORE 560064

(An Autonomous Institution, affiliated to VTU Belgaum, and Approved By AICTE)

(Department of MCA)

This is to certify that, **VISHAL (1NT22MC118)** of 4th Sem of Master's of Computer Application has submitted project report entitled **VIRTUAL PAINT** in partial fulfilment for the award of the degree of Master's of Computer Application in session 2023-24, under the guidance of **MS. SOWMYA K .** It has been found to be satisfactory and hereby approved for the submission.

Ms. Sowmya K	**Ms. Sowmya K**	**Dr. Shrikant PhD**
(Project Guide)	*(Project Co-Ordinator)*	*(HOD Dept. of MCA)*

EXAMINERS:

1.
2.

Acknowledgement

The satisfaction that accomplished the successful completion of any work would be incomplete without the mention of people who made it possible and whose constant guidance and encouragement crown all the effort with success. This acknowledgement transcends the reality of formality when I would like to express deep gratitude and respect to all the people who have inspired and helped me for the completion of my PROJECT.

My special gratitude to my guide Ms. SOWMYA K for his inspiration, guidance, constant supervision, direction and discussions in successful completion of the Project Report. I would like to showcase my gratitude to the project co-ordinator Ms. SOWMYA K for his valuable guidance, keen increase and encouragement at various stages of my project.

It's my great pleasure to express my heartiest thankfulness and gratitude to my H.O.D DR. SHRIKANT PhD who has been the inspiration right from the starting till it's completion of the project successfully.

Also I would like to express my sincere thanks to DR. H C NAGRAJ principal NMIT, for his valuable support during the development of this project.

Last but not least I express my deep sense of gratitude to the teaching and non-teaching staff members of the department who have cooperated and supported me in or the way during the completion of the project.

Vishal (1NT22MC118)

Table of Contents

Abstract

Nowadays, the interaction between people and the machines is mainly completed through the mouse, keyboard, remote control, touch screen, and other direct contact manner, while the communication between people is basically achieved through more natural and intuitive non-contact manner, such as sound and physical movements. The communication by natural and intuitive non-contact manner is usually considered to be flexible and efficient; many researchers have thus tried efforts to make the machine identify other intentions and information through the non-contact manner like people, such as sound, facial expressions, physical movements, and gestures. Among them, gesture is the most important part of human language, and its Gestures that play very important roles in human communication also. They are considered as the easiest means of communication between humans and computers gesture recognition has wide applications including sign language recognition, robotics and so on. Gesture recognition can be simply categorized into two methods based on devices which are used to capture gestures: wearable sensor-based methods and optical camera-based methods. The example of device used in the wearable sensor-based method is the data glove which is capable of exactly capturing the motion parameters of the user's hands and it can achieve high recognition performance. These devices used in wearable sensor method affect the naturalness of the user interaction and they are also expensive. In optical camera-based method, optical cameras are used which record a set of images to capture gesture movements from a distance. These optical cameras recognize gestures by analyzing visual information extracted from the captured images so they are also called visionbased methods. optical cameras are easy to use and also inexpensive but the quality of the captured images is sensitive to lighting conditions and clutter backgrounds so it is difficult to detect and track the hands properly. As mentioned above the two methods i.e., wearable sensor-based method and optical camera-based method come with their own set of advantages and disadvantages, but as the goal of this project was to develop a cost effective and easy way to improve on HCI, the project focuses on optical camera-based method.

List of tables

List of figures

List of abbreviations

- HCI – Human Computer Interaction
- HMD – Head Mounted Display

1. INTRODUCTION

Image caption generation has emerged as a challenging and important research area following advances in statistical language modelling and image recognition. The generation of captions from images has various practical benefits, ranging from aiding the visually impaired, to enabling the automatic and cost-saving labelling of the millions of images uploaded to the Internet every day. The field also brings together state-of-the-art models in Natural Language Processing and Computer Vision, two of the major fields in Artificial Intelligence.

There are two major ways of gesture detection, Wearable sensor-based methods and optical camera-based methods.

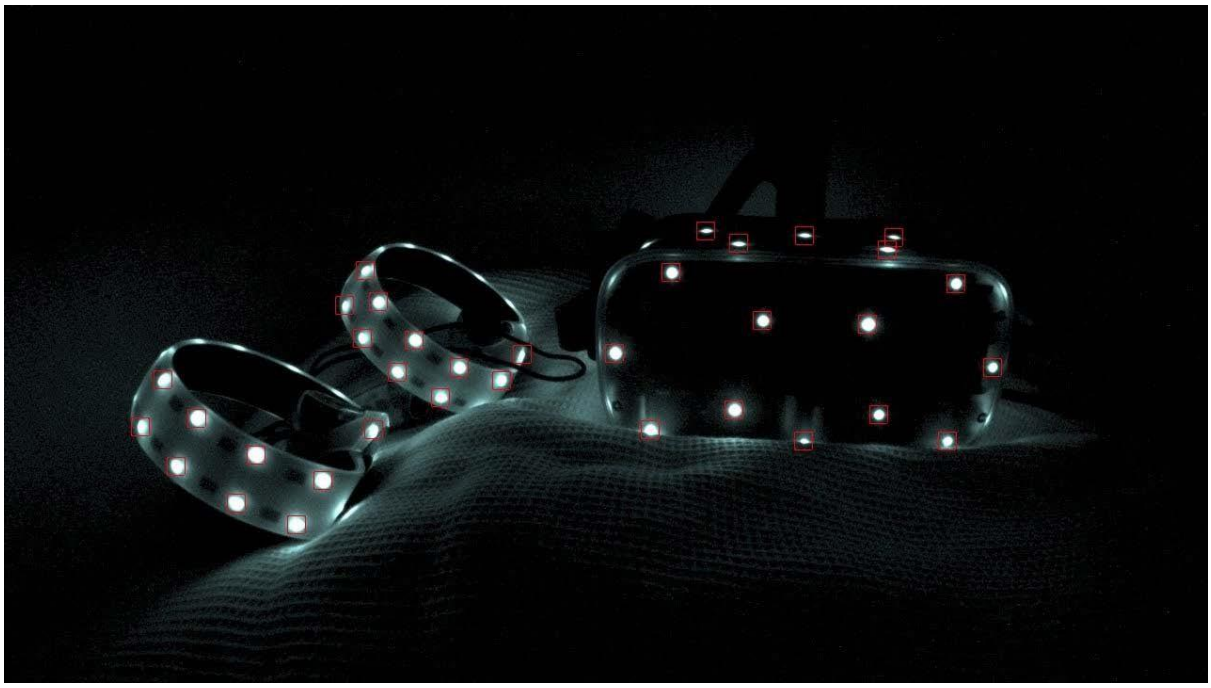


Figure 1: Oculus quest tracking system

Wearable sensor-based methods are already in use in systems like HTC vive and the oculus rift. These are virtual reality entertainment systems. The tracking systems adopted by these systems typically consist of a device capable of creating a signal and a sensor capable of reading it. These can take various forms including optical, electromagnetic signals, acoustic signals and mechanical systems. In optical systems emitted light is captured by cameras in various formats. Electromagnetic tracking can be used where small electrified coils

affect others can position them in space. Acoustic `systems use ultrasonic sound waves to identify position and orientation of objects. Mechanical tracking can use articulated arms/limbs/joysticks/sensors connected to headsets or inside them, this is often made possible by accelerometers and gyroscopes. As for device such as the rift it uses constellations of infrared LED's built into the HMD and controllers. These are subsequently picked up by the two or more desktop sensors designed to recognize the LED's specific glow and convert their placement into positional data. There is also a magnetometer, gyroscope and accelerometer in the headset, combined these allow for accurate tracking across all three dimensions of the 3D world. But these forms of tracking come with two major disadvantages, cost and complexity. Developing and integrating multitude of sensors to work in harmony and give accurate data is expensive. Just a pair of controllers can start from $200 (12ndpoin. Rs.16000) fust for the handheld sensors.

Optical camera-based methods are not widely used as the accuracy of date provided by such systems is less compared to Wearable sensor-based systems, because the only source of information is the camera. With advancements in the fields of machine learning, artificial intelligence, neural networks and modern image processing techniques have increased possibility of creating optical camera-based systems that provide accurate information. Optical camera-based systems eliminate the drawbacks of wearable sensor-based systems. As the only sensor needed is a camera the cost and complexity is reduced drastically and improves the accessibility of such a system. For all the reasons stated in the above paragraphs our focus in this project will be on optical camera-based systems.

2. PRODUCT SURVEY

2.1 BMW iDrive

Figure 2: BMW gesture control

Gesture control isn't a new concept. Automakers have been talking about moving beyond touchscreens to implement touchless infotainment controls since the early 2010s, though few have actually followed through with the talk.

BMW gesture control is BMW's latest feature, that allows drivers to control select iDrive functions with the use of hand gestures captured by a 3D camera. Making simple moves, you can accept or decline a phone call, turn up or down the audio volume, set navigation destinations or change the angle in the 360-degree view of the vehicle. BMW gesture control technology was made with helping drivers to safely interact with the iDrive system BMW gesture control sensors are located in the roof lining of the car, next to the rear-view mirror. They translate a set of predefined hand gestures made in front of the center console, above the gearshift lever. For the system to work properly, the driver should gesticulate close to the screen.

2.2 Oculus rift / HTC vive

Figure 3: VR headset

Virtual reality (VR) is now the fastest-growing content segment in the world. Research by PwC found that VR content will grow at a compound annual rate of 30 percent between 2021 and 2025, outstripping over-the-top (OTT) video, video games, and even traditional cinema. VR headsets can allow users to consume VR content by providing them with an immersive, three-dimensional experience. A VR headset is a head-mounted device that includes a display screen, stereo sound, sensors, and compatible controllers to deliver an immersive and interactive audiovisual experience. When a user puts on a VR headset, they can no longer see the world around them, but instead only see VR content projected on the display screen such as 360-degree videos and VR games, workspaces, or meeting rooms for other activities Unlike 2D video, virtual reality is not a passive experience. Users interact with virtual worlds, which adapts according to the user's continuous inputs. To achieve this, VR headsets come with a number of sensors, and some devices even have a six degrees of freedom (6DoF) system for head tracking.

Using gyroscopes, accelerometers, and other sensors, a 6DoF system tracks head movements and repositions the display accordingly. Some headsets also have eyetracking sensors that can understand when eyes focus on a VR object or location. Two images are passed through the lens, one for each eye, similar to how our eyes perceive and process visuals in the real world. Additionally, images in VR headsets appear to move side-to-side to recreate a 360-degree experience and is achieved by subtly moving the display content in response to head tracking data Two images are passed through the lens, one for each eye, similar to how our eyes perceive and process visuals in the real world. Additionally, images in VR headsets appear to move side-to-side to recreate a 360degree experience and is achieved by subtly moving the display content in response to head tracking data

2.3 Outcomes of product survey

- Gesture recognition systems are currently used in very niche markets.
- Current implementations of gesture recognition technology yields high accuracy
- Current implementations of gesture recognition technology works in most environments
- Current implementations of gesture recognition technology require that each sensor has a direct connection to the controlling module/PC.
- The array of sensors used has very low vertical field of view
- Current implementations of gesture recognition technology are very expensive to produce and integrate
- To register the hand movements the systems, make use of base stations that need to be mounted on walls and need to be spaced accurately.
- Reflective surfaces in the room may cause glitches

3. LITERATURE SURVEY

1. Sushmita Ray, "**A quick review of machine learning algorithms**": - This paper published on IEEE xplore. This paper gives us a basic yet fundamental idea of machine learning and explains a few of the fundamental and most widely used algorithms like gradient descent, linear regression, multivariate regression analysis, decision tree, support vector machine, Bayesian learning, naïve bayes, K nearest algorithm, K means clustering and more. The advantages and disadvantages of all the algorithms along with some comparisons between some algorithms and their performance and learning rate have been discussed.

2. Pramila P. Shinde and Dr. Seema Shah, "**A review of machine learning and deep learning applications**": - This paper is published on IEEE xplore. This paper explains the need to study machine learning and deep learning. The evolution of machine learning and deep learning along with their application over the past few decades have been covered.

3. J. Francis and A. B K, "**Significance of hand gestures recognition systems in vehicular automation – A survey**", International journal of computer applications, 2014: This paper comprises of the existing methods in detecting and recognizing hand gestures and a detailed study on their performances, accuracy, convenience, operational range and design challenges etc

4. Radhika Bhatt, Nikita Fernandes, Archana Dhage "**Vision based hands gesture recognition for human computer interaction**" University of Mumbai, 2013: This paper presents an approach to develop a real-time hand gesture recognition enabling human-computer interaction. It is "Vision Based" that uses only a webcam and Computer Vision (CV) technology, such as image processing that can recognize several hand gestures.

5. Jonas robin, Mehul R S, Rishabh Dubey, Nimish Datkhile, Jyoti Kolap "**Computer vision for hand gestures**", International conference on convergence

of digital world, 2020: The model developed here is used to detect specific items from the environment. The desired objects to be detected from the environment is hands(gestures)Here the recognition of the image is done by using Convolution neural network (CNN) algorithm, the gesture is predicted and this predicted result is shown on the screen connected or an audio device connected.

6. **Which is Better For Your Machine Learning Task, OpenCV or TensorFlow?(** Link: https://towardsdatascience.com/which-is-better-for-your-machine-learning-taskopencv-or-tensorflow-ed16403c5799): Provides comparison between OpenCV and TensorFlow for machine learning and gesture recognition tasks

7. **A Review of Google's New Mobile-Friendly AI Framework: Mediapipe** (Link: https://medium.com/swlh/a-review-of-googles-new-mobile-friendly-ai-frameworkmediapipe-25d62cd482a1): Introduces mediapipe framework and explains the different modules in the mediapipe framework

8. **Hand tracking and gesture recognition using AI: Applications and challenges** (Link: https://intellias.com/hand-tracking-and-gesture-recognition-using-aiapplications-and-challenges/) :- This article outlines the advantages of gesture control. How, in the 21st century gesture control system can change systems in the automotive, consumer electronic, healthcare Etc. information about the market of gesture recognition technology and how the market is expected to reach $32.3 billion by 2025 up from $9.8 billion in 2020.

9. **On-Device, Real-Time Hand Tracking with MediaPipe** (Link: https://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html): This web article provides a basic implementation of the hand tracking mediapipe module.

4. FEASIBILITY STUDY

We conducted a feasibility study to examine the project parameters, and ensure that the project is possible to be carried out with the technology available to us. The primary objective of the feasibility study is to check the technical, operational and economic feasibility of the project proposed. The preliminary investigation of the feasibility study has 3 aspects.

- Technical feasibility
- Economic feasibility
- Social feasibility

4.1 Technical feasibility

While analyzing technical feasibility, a number of questions were raised which need to be answered to fulfill the technical feasibility

- oes the technology to implement the planned project exist?
- Can the system be upgraded, if developed?
- Does the system guarantee accuracy of output, simple access, and information security?
- Does the proposed system require hardware and software that are difficult acquire?

As we plan to use optical camera-based detection techniques, technologies like computer Vision, object detection, machine learning algorithms, artificial intelligence exist and make it possible to achieve high accuracy tracking. The proposed system can be easily upgraded with faster processors, better resolution cameras and better optimized tracking algorithms in the future as the current models are improved with more data over time. The accuracy of the model cannot be as high as systems that use multitude of sensors, but with the help of current computer vision technology accuracy of the results are high and reliable. As the proposed system will have an GUI interface it will be easy to use. As previously pointed the software does not require any specialized software and hardware components hence making it cost effective and easy to implement and upgrade.

4.2 Economic feasibility

Economic feasibility is a kind of cost-benefit analysis of the examined project, which assesses whether it is possible to implement it. This term means the assessment and analysis of a project's potential to support the decision-making process by objectively and rationally identifying its strengths, weaknesses, opportunities and risks associated with it, the resources that will be needed to implement the project, and an assessment of its chances of success.

A major part of economic feasibility is market study. As the project focuses on contactless/gesture based HCI, its of interest in the current market. After the COVID-19 pandemic the importance of contactless HCI was brought in a spotlight, in order to better the public health by avoiding touch-based systems like ATM, Ticket dispensers etc. If such services can be availed without actually touching the screen, then it could turn out to be a wise investment.

If the proposed system is developed and all the planned features are implemented, it should still be an honest investment if it is adopted by an organization. The system is economically feasible, it doesn't need proprietary hardware or code, and since it is developed using prevailing technologies and resources there is nominal expenditure and economically safe.

4.3 Social feasibility

A social feasibility study explores the impact of a project on society and of society on the project. For example, this type of study might look at how ambient social structure in the area will affect the number of qualified employees that may be available, or the compatibility of local residents with the project.
The proposed project looks to be useful in the impact it may have on the society. If implemented at a mass scale then the general public may need to adapt to the new machines and methods to avail services. But on the upside, it may play a role in improving the public health by prevent contact with commonly touched surfaces.

5. SYSTEM ANALYSIS AND REQUIREMENTS

5.1 System study and environment

Computer Vision can be defined as a discipline that explains how to reconstruct, interrupt, and understand a 3D scene from its 2D images, in terms of the properties of the structure present in the scene. It deals with modeling and replicating human vision using computer software and hardware. In order to carry out computer vision we use computer vision and OpenCV framework. Other frameworks like TensorFlow, OpenGL, PyTorch, OpenCL are available but we decided to use OpenCV for its ease of use and large array of supported features like object detection, pattern recognition etc.

5.2 Computer vision

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 labeling objects.

One of the driving factors behind the growth of computer vision is the amount of data we generate today that is then used to train and make computer vision better.Along with a tremendous amount of visual data (more than 3 billion images are shared online every day), the computing power required to analyze the data is now accessible. As the field of computer vision has grown with new hardware and algorithms so has the accuracy rates for object identification. In less than a decade, today’s systems have reached 99 percent accuracy from 50 percent making them more accurate than humans at quickly reacting to visual inputs.

Early experiments in computer vision started in the 1950s and it was first put to use commercially to distinguish between typed and handwritten text by the 1970s, today

the applications for computer vision have grown exponentially. On a certain level Computer vision is all about pattern recognition. So one way to train a computer how to understand visual data is to feed it images, lots of images thousands, millions if possible that have been labeled, and then subject those to various software techniques, or 21ndpoint21ms, that allow the computer to hunt down patterns in all the elements that relate to those labels.

For example, if we feed a computer thousands of images of Abraham Lincoln, it will subject them all to algorithms that le them analyze the colors in the photo, the shapes, the distances between the shapes, where object border each other and so on. So, it identifies a profile of what "Abraham Lincoln" means. The sample figure 4 illustrates the grayscale image buffer which stores our image of Abraham Lincoln, each pixels brightness is represented by a single 8-bit number, whose range is from 0 to 255

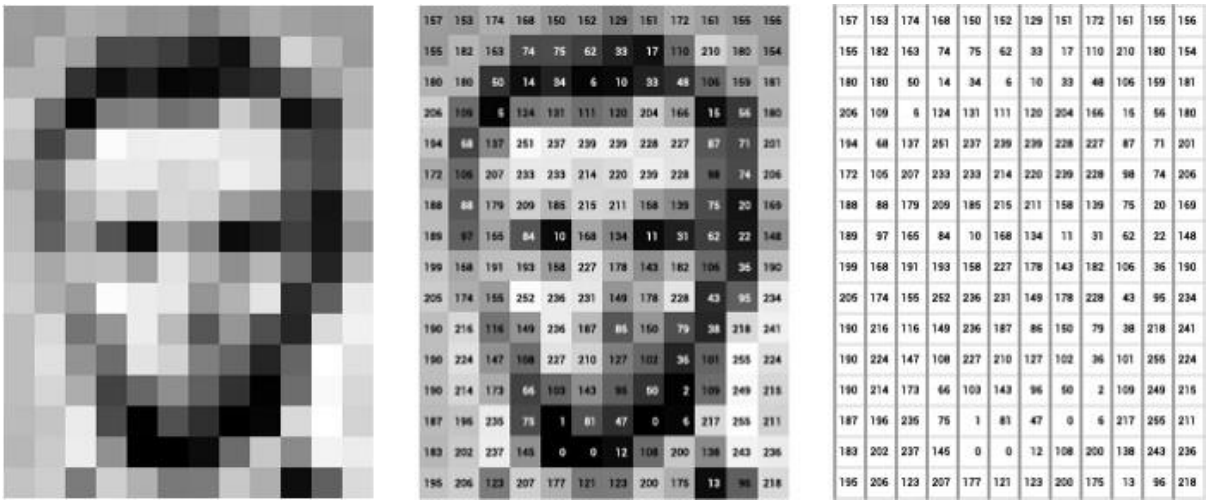


Figure 4: Abraham Lincoln Pixel data diagram

Applications Of Computer Vision

Computer vision is one of the areas in Machine Learning where core concepts are already being integrated into major products that we use every day.

CV In Self-Driving Cars: But it's not just tech companies that are leverage Machine Learning for image applications. Computer vision enables self-driving cars to make sense of their surroundings. Cameras capture video from different angles around the car and feed it to computer vision software, which then processes the images in real-

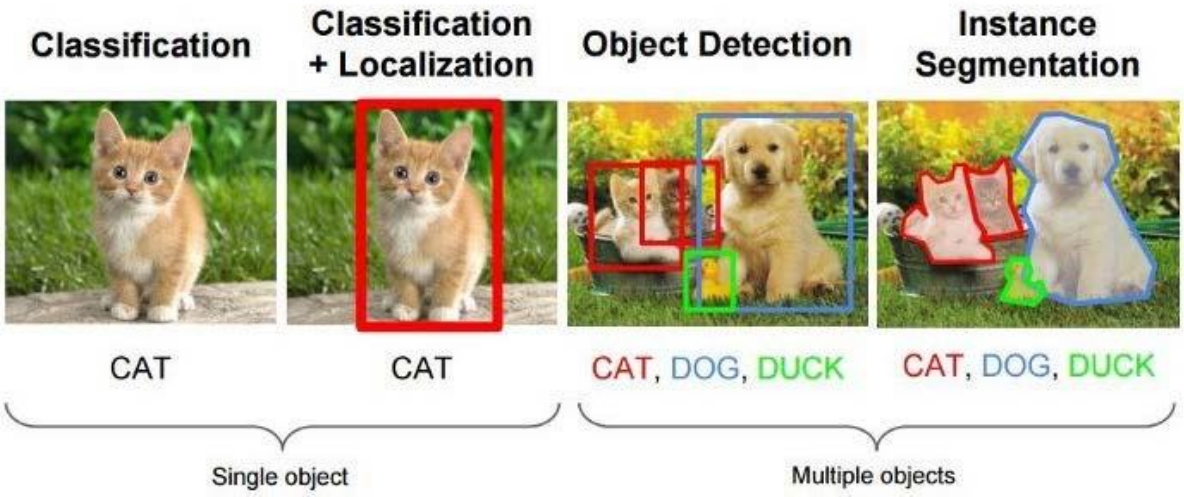
time to find the extremities of roads, read traffic signs, detect other cars, objects and pedestrians. The self-driving car can then steer its way on streets and highways, avoid hitting obstacles, and (hopefully) safely drive its passengers to their destination.

CV In Facial Recognition: Computer vision also plays an important role in facial recognition applications, the technology that enables computers to match images of people's faces to their identities. Computer vision algorithms detect facial features in images and compare them with databases of face profiles. Consumer devices use facial recognition to authenticate the identities of their owners. Social media apps use facial recognition to detect and tag users. Law enforcement agencies also rely on facial recognition technology to identify criminals in video feeds.

CV In Augmented Reality & Mixed Reality: Computer vision also plays an important role in augmented and mixed reality, the technology that enables computing devices such as smartphones, tablets and smart glasses to overlay and embed virtual objects on real world imagery. Using computer vision, AR gear detect objects in real world in order to determine the locations on a device's display to place a virtual object. For instance, computer vision algorithms can help AR applications detect planes such as tabletops, walls and floors, a very important part of establishing depth and dimensions and placing virtual objects in physical world.

CV In Healthcare: Computer vision has also been an important part of advances in health-tech. Computer vision algorithms can help automate tasks such as detecting cancerous moles in skin images or finding symptoms in x-ray and MRI scans.

Computer Vision tasks



Many popular computer vision applications involve trying to recognize things in photographs; for example:

- **Object Classification**: What broad category of object is in this photograph?
- **Object Identification**: Which type of a given object is in this photograph?
- **Object Verification**: Is the object in the photograph?
- **Object Detection**: Where are the objects in the photograph?
- **Object Landmark Detection**: What are the key points for the object in the photograph?
- **Object Segmentation**: What pixels belong to the object in the image?
- **Object Recognition**: What objects are in this photograph and where are they?

5.3 OpenCV

OpenCV stands for Open-Source Computer Vision. To put it simply, it is a library used for image processing. In fact, it is a huge open-source library used for computer vision applications, in areas powered by Artificial Intelligence or Machine Learning algorithms, and for completing tasks that need image processing. As a result, it assumes significance today in real-time operations in today's systems. Using OpenCV, one can process images and videos to identify objects, faces, or even the handwriting of a human.

Originally developed by Intel, OpenCV was later supported by Willow Garage then Itseez, in turn later acquired by Intel. The first OpenCV version was 1.0. Released under a BSD license, this cross-platform library is free for both academic and commercial use under the open-source Apache 2 License. It has C++, C, Python, Java and MATLAB (a proprietary multi-paradigm programming language that offers a good numeric computing environment) interfaces and the API for these interfaces can be found in the online documentation. OpenCV also supports Windows, Linux, Mac OS, iOS and Android. Initially, the main aim of creating OpenCV was real-time applications for computational efficiency. Since 2011, OpenCV also offers GPU acceleration for real-time operations. Upon integration with other libraries, such as NumPy, Python can process the OpenCV array structure for analysis. Identifying image patterns and its several features needs use of vector space and carrying out mathematical operations on these feature

Applications of OpenCV

There are lots of applications which rely on the use of the Open-Source Computer Vision library. Let's look at them.

- **Face detection / face recognition/ facial recognition system:** A facial recognition system is a technology that can match a human face from a digital image or a video frame against a database of faces. It is used to authenticate users through ID verification services, and works by pinpointing and measuring facial features from a given image.

- **Egomotion estimation:** Egomotion refers to the 3D motion of a camera within an environment. In the context of computer vision, egomotion deals with estimating a camera's motion relative to a rigid scene. An example of egomotion estimation would be estimating a car's moving position with respect to lines on the road or street signs being seen from the car itself. The estimation of egomotion is important in autonomous robot navigation applications.

- **Gesture recognition:** A subdiscipline of computer vision, gesture recognition works with the goal of interpreting human gestures via mathematical algorithms. Gestures can stem from any bodily motion or state but commonly originate from the face or hand.

- **Human–computer interaction (HCI):** Human-computer interaction (HCI) is a field of research in the design and the use of computer technology, which studies in detail the interfaces between people (users) and computers. HCI researchers focus on the ways in which humans interact with computers and design technologies allowing humans to interact with computers in novel ways.

- **Mobile robotics:** A mobile robot is a robot that can move in its surroundings. Thus, mobile robotics is a subfield of robotics and information engineering.

- **Motion understanding:** As the term suggests, motion understanding refers to understanding, interpreting and overall analysing the motion of objects in certain environments, such that we are able to better predict their movements and and interact better with them.

- **Object detection:** Object detection is a computer technology related to computer vision and image processing that focuses on identifying instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos.

- **Image segmentation and recognition:** In digital image processing and computer vision, image segmentation is the process of partitioning a digital image into

multiple segments (sets of pixels, also known as image objects). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze.

- **Stereopsis stereo vision:** Here stereopsis refers to the perception of depth and 3-dimensional structure formed on the basis of visual information gained from two cameras.

- **Structure from motion (SFM):** Structure from motion (SFM) is a photogrammetric range imaging technique for estimating threedimensional structures from two-dimensional image sequences that can be combined with local motion signals. It is studied in the fields of computer vision and visual perception.

- **Motion tracking**: Also known as video tracking, motion tracking is the process of locating a moving object (or multiple objects) over time using a camera.

- **Augmented Reality (AR):** Augmented Reality (AR) is basically an interactive experience of a real-world environment where the objects in the real world get enhanced by computer-generated perceptual information, often across several sensory faculties, be it visual, auditory, haptic, somatosensory and olfactory

5.4 Mediapipe

MediaPipe is a Framework for building machine learning pipelines for processing timeseries data like video, audio, etc. This cross-platform Framework works in Desktop/Server, Android, iOS, and embedded devices like Raspberry Pi and Jetson Nano. MediaPipe Toolkit comprises the Framework and the Solutions. The following diagram shows the components of the MediaPipe Toolkit.

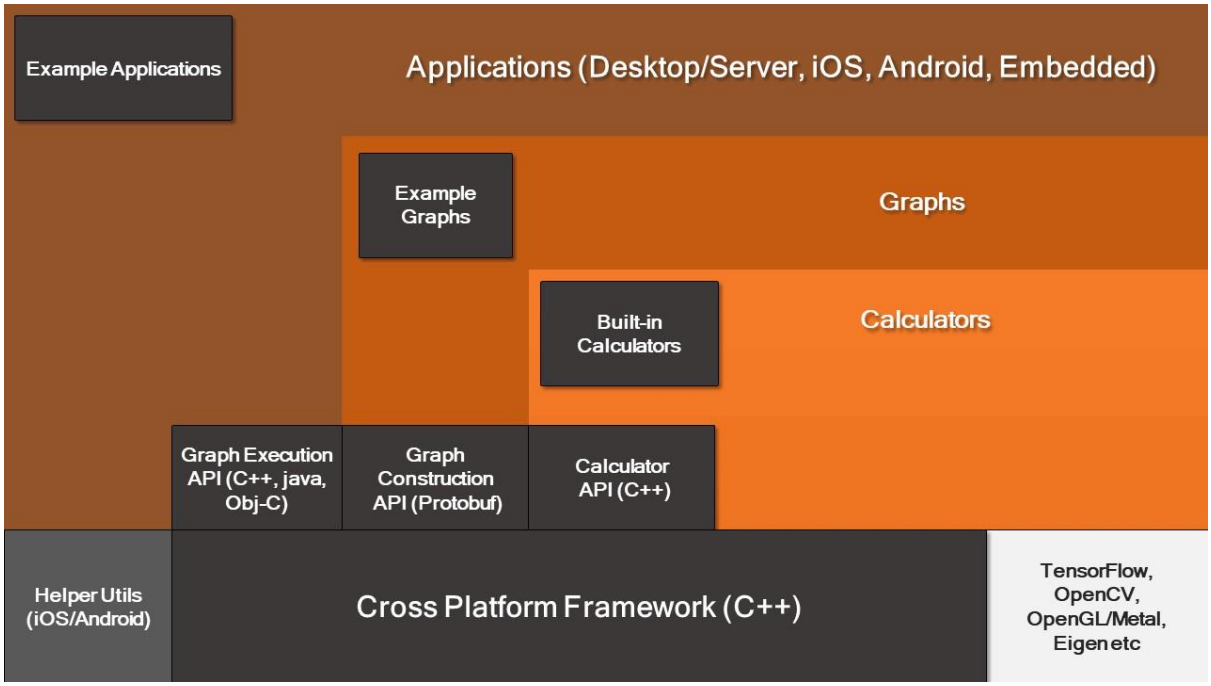


Figure 5: Mediapipe Toolkit

5.5 Mediapipe Hands

MediaPipe Hands is a high-fidelity hand and finger tracking solution. It employs machine learning (ML) to infer 21 3D landmarks of a hand from just a single frame. Whereas current state-of-the-art approaches rely primarily on powerful desktop environments for inference, our method achieves real-time performance on a mobile phone, and even scales to multiple hands. We hope that providing this hand perception functionality to the wider research and development community will result in an emergence of creative use cases, stimulating new applications and new research avenues.

5.5.1 ML pipeline

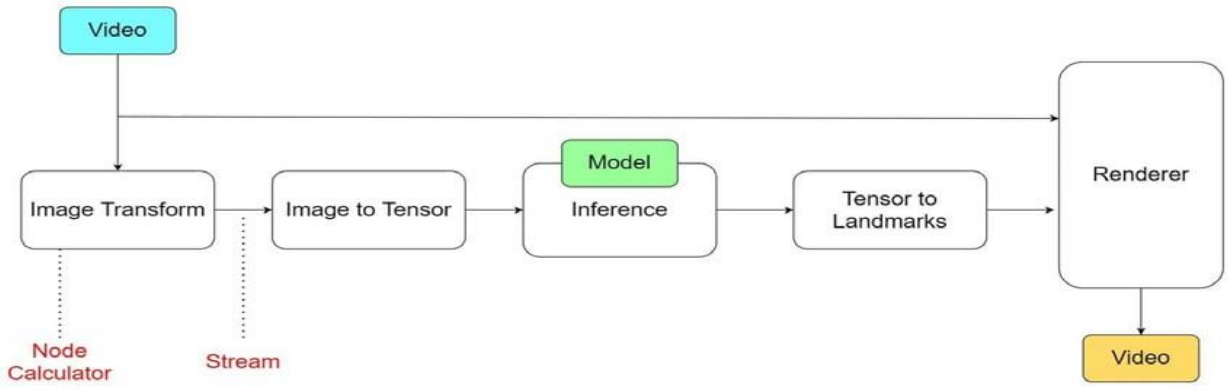


Figure 6: Mediapipe hands solution graph

MediaPipe Hands utilizes an ML pipeline consisting of multiple models working together: A palm detection model that operates on the full image and returns an oriented hand bounding box. A hand landmark model that operates on the cropped image region defined by the palm detector and returns high-fidelity 3D hand key points. This strategy is similar to that employed in our MediaPipe Face Mesh solution, which uses a face detector together with a face landmark model.

Providing the accurately cropped hand image to the hand landmark model drastically reduces the need for data augmentation (e.g., rotations, translation and scale) and instead allows the network to dedicate most of its capacity towards coordinate prediction accuracy. In addition, in our pipeline the crops can also be generated based on the hand landmarks identified in the previous frame, and only when the landmark model could no longer identify hand presence is palm detection invoked to relocalize the hand.

The pipeline is implemented as a MediaPipe graph that uses a hand landmark tracking subgraph from the hand landmark module, and renders using a dedicated hand renderer subgraph. The hand landmark tracking subgraph internally uses a hand landmark subgraph from the same module and a palm detection subgraph from the palm detection module.

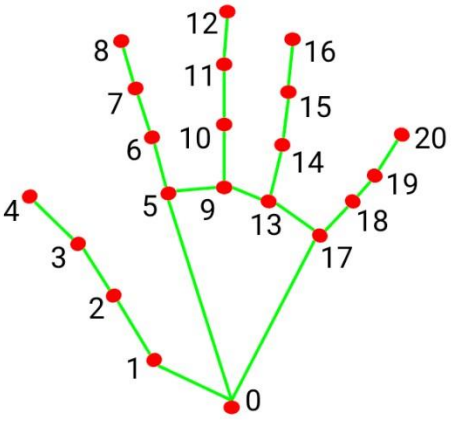
5.5.2 Models

Palm Detection Model: To detect initial hand locations, we designed a single-shot detector model optimized for mobile real-time uses in a manner similar to the face detection model in MediaPipe Face Mesh. Detecting hands is a decidedly complex task: our lite model and full model have to work across a variety of hand sizes with a largescale span (~20x) relative to the image frame and be able to detect occluded and selfoccluded hands. Whereas faces have high contrast patterns, e.g., in the eye and mouth region, the lack of such features in hands makes it comparatively difficult to detect them reliably from their visual features alone. Instead, providing additional context, like arm, body, or person features, aids accurate hand localization. Our method addresses the above challenges using different strategies. First, we train a palm detector instead of a hand detector, since estimating bounding boxes of rigid

objects like palms and fists is significantly simpler than detecting hands with articulated fingers. In addition, as palms are smaller objects, the non-maximum suppression algorithm works well even for two-hand self-occlusion cases, like handshakes. Moreover, palms can be modelled using square bounding boxes (anchors in ML terminology) ignoring other aspect ratios, and therefore reducing the number of anchors by a factor of 3-5. Second, an encoder-decoder feature extractor is used for bigger scene context awareness even for small objects (similar to the RetinaNet approach). Lastly, we minimize the focal loss during training to support a large number of anchors resulting from the high scale variance.With the above techniques, we achieve an average precision of 95.7% in palm detection. Using a regular cross entropy loss and no decoder gives a baseline of just 86.22%.

Hand Landmark Model: After the palm detection over the whole image our subsequent hand landmark model performs precise key point localization of 21 3D hand-knuckle coordinates inside the detected hand regions via regression, that is direct coordinate prediction. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions.

To obtain ground truth data, we have manually annotated ~30K real-world images with 21 3D coordinates, as shown below (we take Z-value from image depth map, if it exists per corresponding coordinate). To better cover the possible hand poses and provide additional supervision on the nature of hand geometry, we also render a high-quality synthetic hand model over various backgrounds and map it to the corresponding 3D coordinates.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figure 7: Hand landmarks

5.6 Existing systems

While trying to search for existing systems we searched in the Indian patent office website, and the USPTO (United states patent and trademark office), we came across a few patents/trademarks that are slightly related to our work, they are

- (Application no: 202241024169) Hand gesture recognition using raspberry Pi: Hindustan college of engineering and technology (INDIA)
- Application no: 202241025825) Hand gesture controlled robotic arm: Hindustan college of engineering and technology (INDIA)

The patents/trademarks above make use of different technologies and some use hardware like raspberry pi to achieve the desired results. What all the results have in common is that that are designed for a specific purpose i.e., control robotic arm etc. Our proposed project is just to demonstrate the power of gestures using computer vision, and hence can be adopted easily to multitude of solutions like ATM, kiosks and even systems like sign language detection, IoT device controls etc.

5.7 Proposed System

We propose a solution for hand recognition system for virtual paint application. The application makes use of a camera to capture the gestures, the Opencv and mediapipe analyse every frame of the video. While analysing the frame mediapipe first tracks the palm of the hand and then proceeds to mark the 21 landmarks. The observed landmarks are matched with gesture library to identify the gesture and send input to the system/application. Our proposed system differentiates itself from the existing system by not using sensors like gyroscopes, accelerometers etc. and not using special IR or UV cameras/ sensors to detect gestures and promises high and reliable performance. Since the virtual paint application is just to prove the effectiveness of the system, the gesture recognition system can be easily adopted to fit different systems and functions.

5.8 System requirements

5.8.1 Hardware requirements

- CPU/processor: Minimum dual core intel/AMD processor at 2Ghz
- RAM: 4GB minimum
- Storage: 500MB minimum
- Camera: Minimum 720p or 1080p camera

5.8.2 Software requirements

- Programming language: Python3
- IDE: PyCharm
- Operating system: OS independent
- Tools/modules/frameworks: OpenCV, NumPy, MediaPipe, Cvzone

6. SYSTEM DESIGN

6.1 System architecture

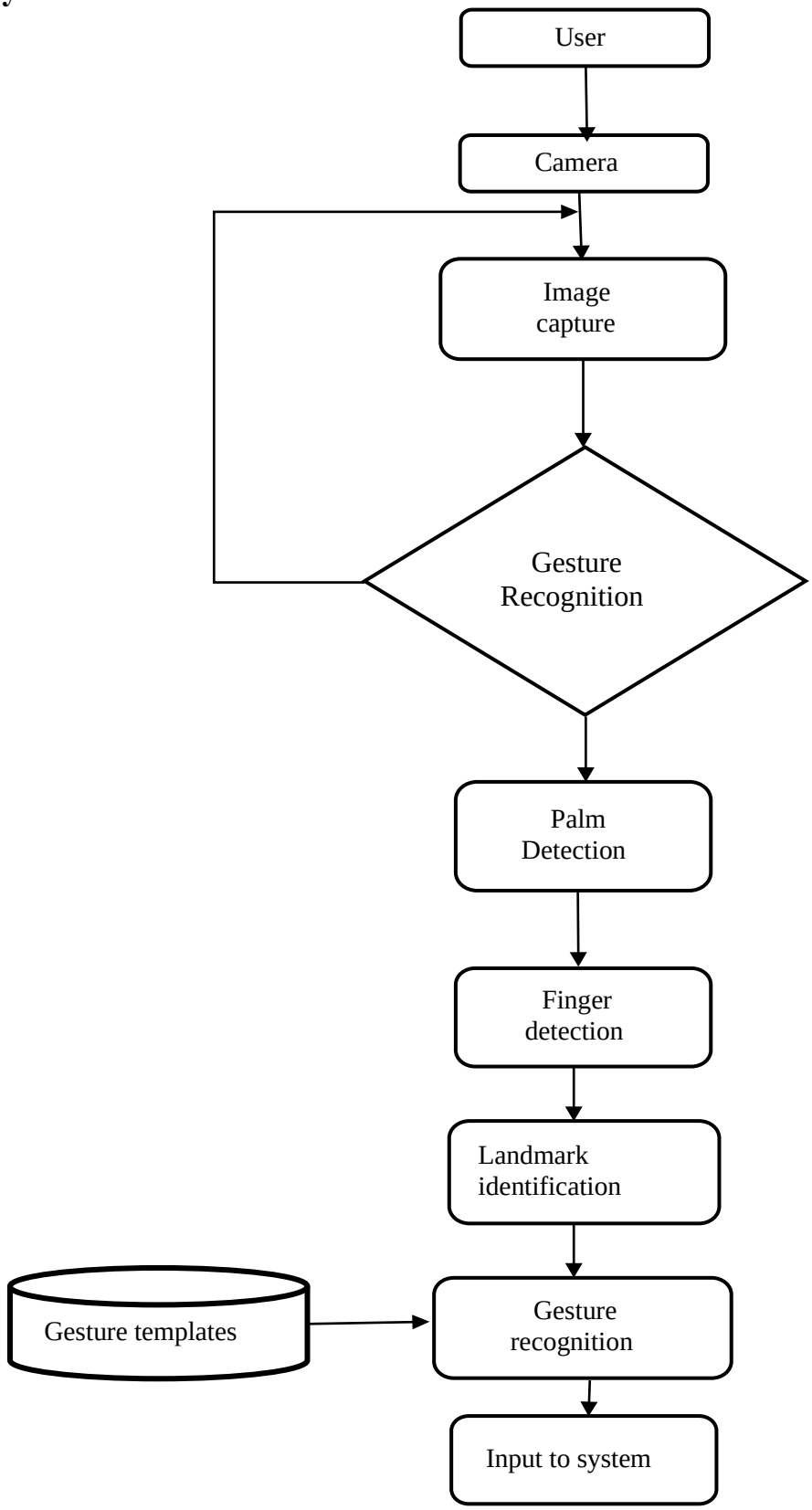


Figure 8: System architecture

6.2 UML Diagrams

The Unified Modeling Language permits the technologist to specific AN analysis model mistreatment the modeling notation that's ruled by a group of grammar linguistics and pragmatic rules. A UML system is diagrammatical mistreatment 5 completely different views that describe the system from clearly different perspective. Every read is outlined by a group of diagrams, that is as follows. It represents the dynamic of behavioral as elements of the system, portrayal the interactions of assortment between varied structural components delineated within the user model and structural model read. Use case Diagrams represent the practicality of the system from a user's purpose of read. Use cases are used throughout needs induction and analysis to represent the practicality of the system. Use cases specialize in the behavior of the system from external purpose of read. Actors are external entities that move with the system. Samples of actors embody users like administrator, bank client …etc., or another system like central info. **6.2.1 Use case diagram**

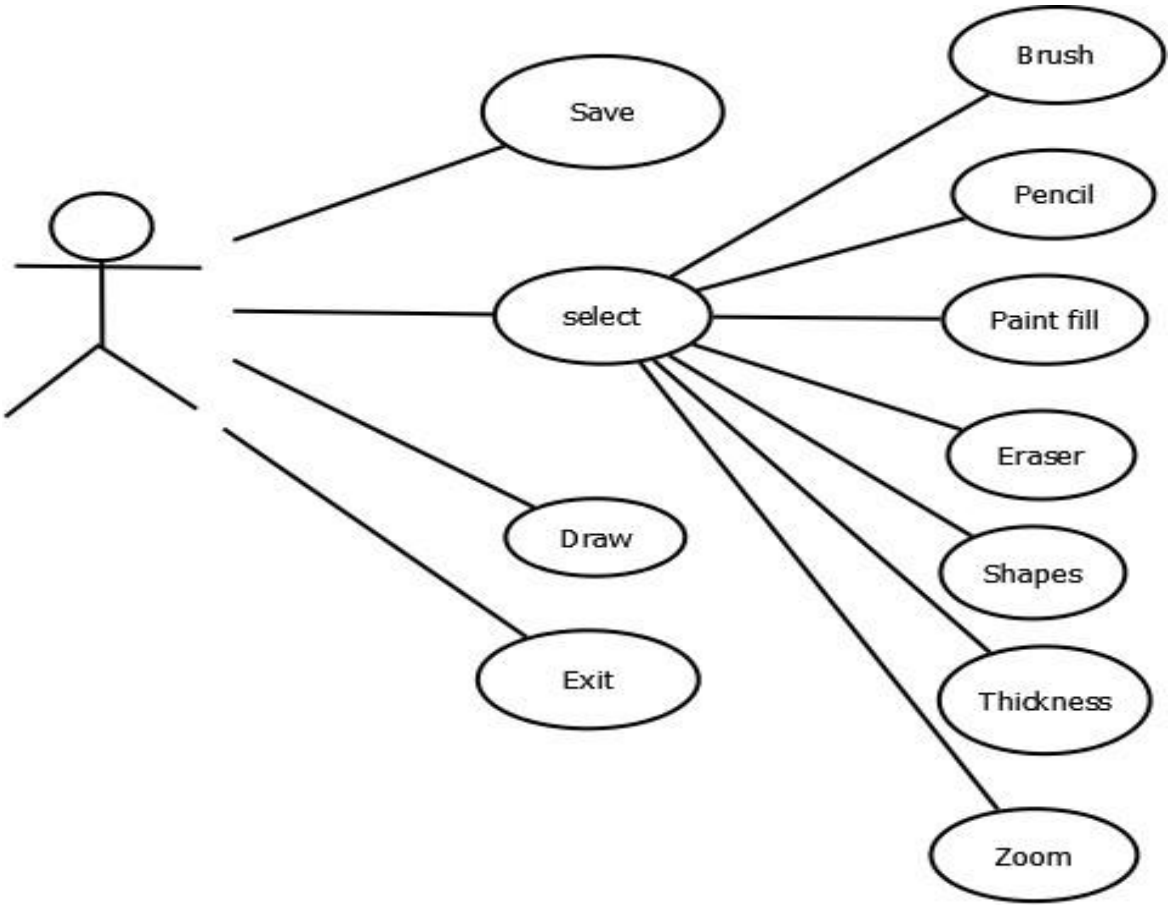


Figure 9: Use case diagram

6.2.2 Component diagram

An element diagram describes the organization of the physical elements in a very system. An element could be a physical building block of the system. It's pictured as a parallelogram with tabs.

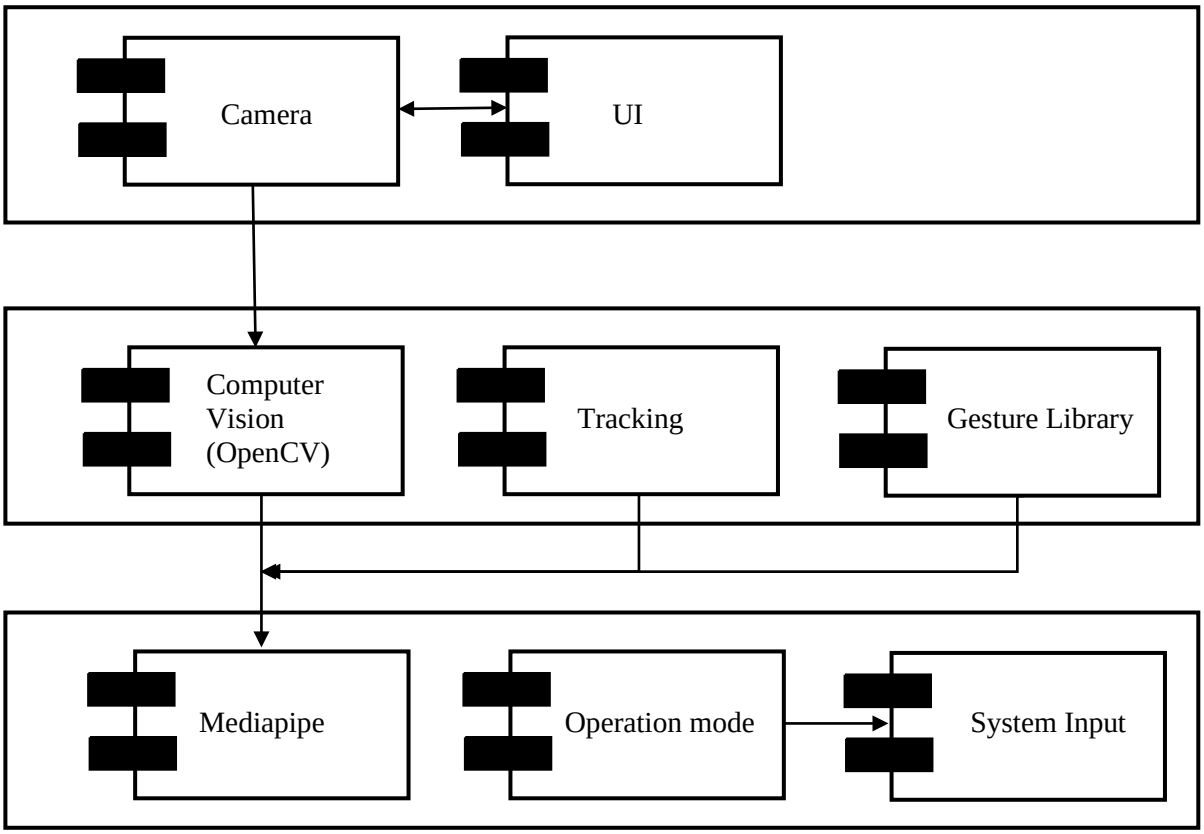
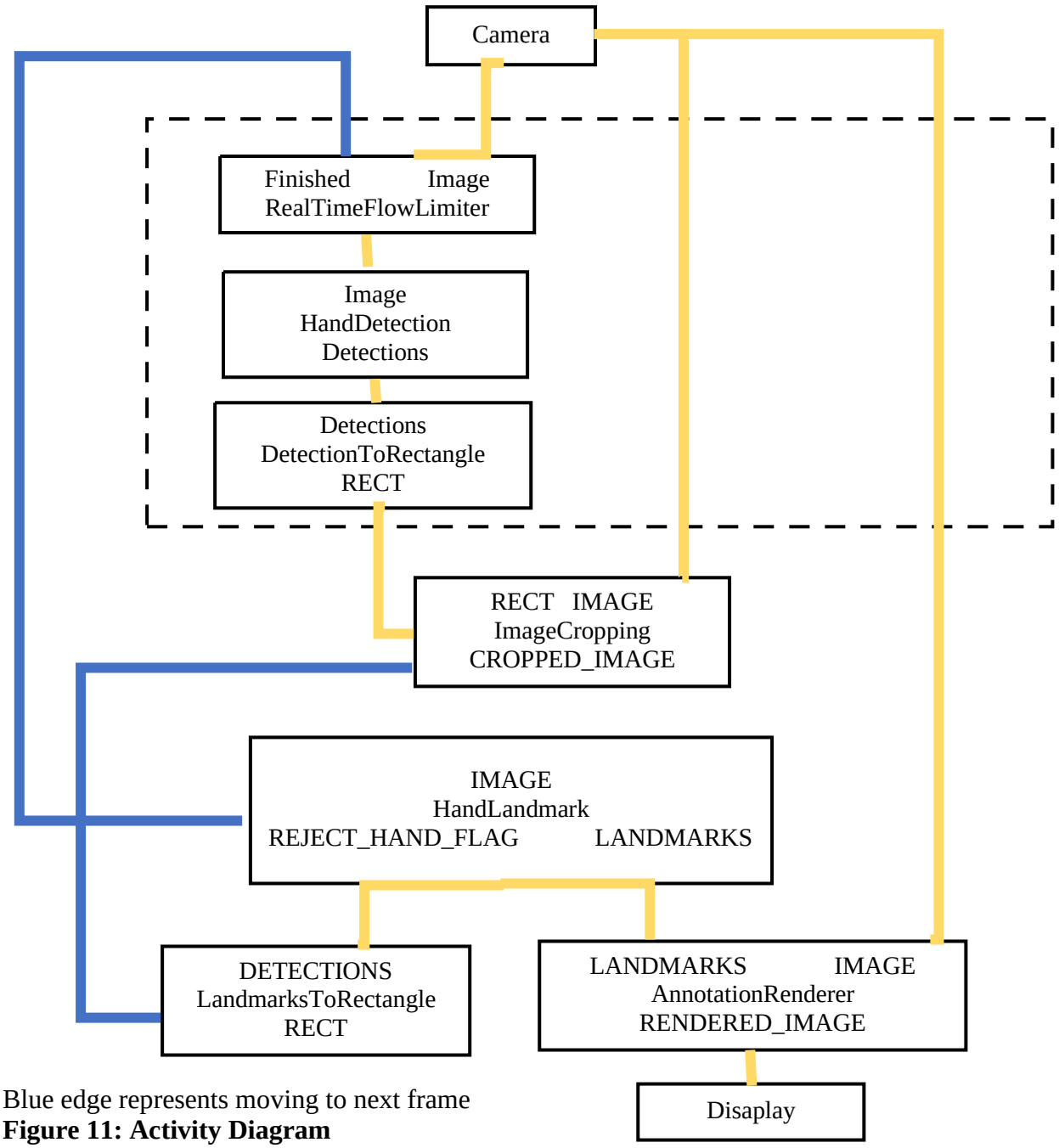


Figure 10: Component diagram

6.2.3 Module diagram

An module diagram illustrates the dynamic nature of a system by modeling the flow of management from activity to activity. An activity represents AN operation on some category within the system that leads to an amendment within the state of the system. Typically, activity diagrams are accustomed model progress or business processes and internal operation. As a result of AN activity diagram may be a special quite state chart diagram, it uses a number of constant modelling conventions.



Blue edge represents moving to next frame

Figure 11: Activity Diagram

6.2.4 Data flow diagram

A data-flow diagram is a way of representing a flow of a data of a process or a system The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops.

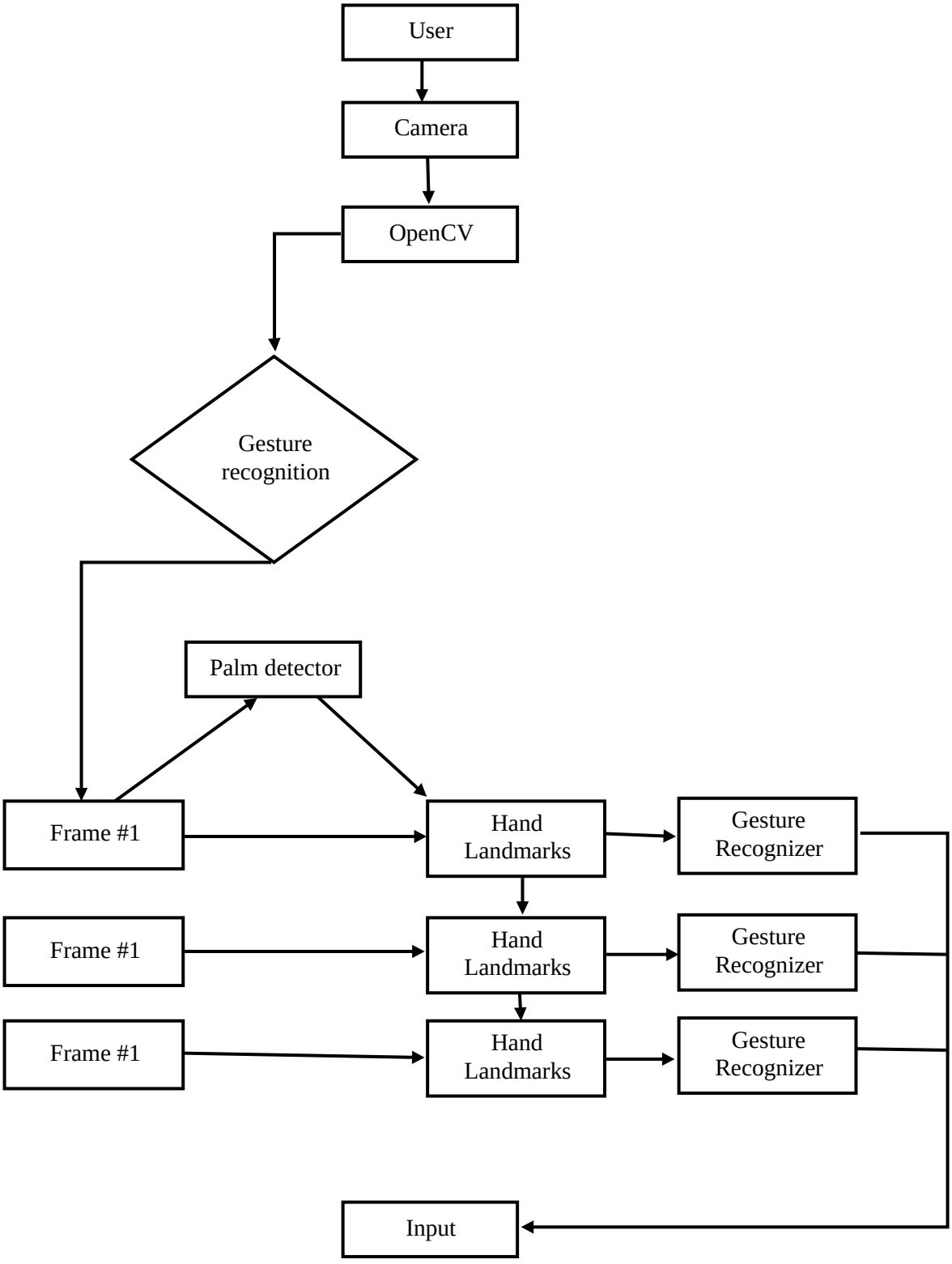


Figure 12: Data flow diagram

7.SYSTEM IMPLEMENTATION

7.1 Modules

7.1.1 Hand tracking

In this module OpenCV and mediapipe use the camera to detect and track the hand as well as mark the 21 landmarks.

Handtracking.py

```
import cv2
import numpy as np
import time
import os
import mediapipe as mp
import math
from cvzone.HandTrackingModule import HandDetector


class HandDetector:
  def __init__(self, mode=False, maxHands=2, detectionCon=0.5, minTrackCon=0.5):
        self.mode = mode
        self.maxHands = maxHands
        self.detectionCon = detectionCon
        self.minTrackCon = minTrackCon
        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(static_image_mode=self.mode,
max_num_hands=self.maxHands,
                                        min_detection_confidence=self.detectionCon,
                                        min_tracking_confidence=self.minTrackCon)
        self.mpDraw = mp.solutions.drawing_utils
        self.tipIds = [4, 8, 12, 16, 20]
        self.fingers = []
```

```
        self.lmList = []


def findHands(self, img, draw=True, flipType=True, count=1):
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        self.results = self.hands.process(imgRGB)
        allHands = []
        h, w, c = img.shape
        if self.results.multi_hand_landmarks:
                for handType, handLms in zip(self.results.multi_handedness,
        self.results.multi_hand_landmarks):
                        myHand = {}
                        mylmList = []
                        xList = []
                        yList = []
                        for id, lm in enumerate(handLms.landmark):
                                px, py, pz = int(lm.x * w), int(lm.y * h), int(lm.z * w)
                                mylmList.append([px, py, pz])
                                xList.append(px)
                                yList.append(py)


                        xmin, xmax = min(xList), max(xList)
                        ymin, ymax = min(yList), max(yList)
                        boxW, boxH = xmax - xmin, ymax - ymin
                        bbox = xmin, ymin, boxW, boxH
                        cx, cy = bbox[0] + (bbox[2] // 2), bbox[1] + (bbox[3] // 2)


                        myHand["lmList"] = mylmList
                        myHand["bbox"] = bbox
                        myHand["center"] = (cx, cy)
```

```
                    if flipType:
                        if handType.classification[0].label == "Right":
                            myHand["type"] = "Left"
                        else:
                            myHand["type"] = "Right"
                        else:
                            myHand["type"] = handType.classification[0].label
                        allHands.append(myHand)


## draw
if draw:
    if count == 1:
        self.mpDraw.draw_landmarks(img, handLms,
    self.mpHands.HAND_CONNECTIONS)
        cv2.rectangle(img, (bbox[0] - 20, bbox[1] - 20),
            (bbox[0] + bbox[2] + 20, bbox[1] + bbox[3] + 20),
            (255, 0, 255), 2)
        cv2.putText(img, myHand["type"],
            (bbox[0] - 30, bbox[1] - 30), cv2.FONT_HERSHEY_PLAIN,
            2, (255, 0, 255), 2)
if draw:
    return allHands, img
else:
    return allHands


def fingersUp(self, myHand):
    myHandType = myHand["type"]
    myLmList = myHand["lmList"]
```

```
        if self.results.multi_hand_landmarks:
                fingers = []
                # Thumb
                if myHandType == "Right":
                        if myLmList[self.tipIds[0]][0] > myLmList[self.tipIds[0] - 1][0]:
                                fingers.append(1)
                        else:
                                fingers.append(0)
                        else:
                                if myLmList[self.tipIds[0]][0] < myLmList[self.tipIds[0] -
                        1][0]:
                                        fingers.append(1)
                                else:
                                        fingers.append(0)

        # 4 Fingers
        for id in range(1, 5):
                if myLmList[self.tipIds[id]][1] < myLmList[self.tipIds[id] - 2][1]:
                        fingers.append(1)
                else:
                        fingers.append(0)
                        return fingers

def findDistance(self, p1, p2, img=None):
        x1, y1 = p1
        x2, y2 = p2
        cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
        length = math.hypot(x2 - x1, y2 - y1)
        info = (x1, y1, x2, y2, cx, cy)
```

```
        if img is not None:
            cv2.circle(img, (x1, y1), 15, (255, 0, 255), cv2.FILLED)
            cv2.circle(img, (x2, y2), 15, (255, 0, 255), cv2.FILLED)
            cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), 3)
            cv2.circle(img, (cx, cy), 15, (255, 0, 255), cv2.FILLED)
            return length, info, img
        else:
            return length, info


def findPosition(self, img, handNo=0, draw=True):
        xList = []
        yList = []
        bbox = []
        self.lmList = []
        if self.results.multi_hand_landmarks:
            myHand = self.results.multi_hand_landmarks[handNo]
            for id, lm in enumerate(myHand.landmark):
            h, w, c = img.shape
            cx, cy = int(lm.x * w), int(lm.y * h)
            xList.append(cx)
            yList.append(cy)
            self.lmList.append([id, cx, cy])
            if draw:
                cv2.circle(img, (cx, cy), 5, (255, 0, 255), cv2.FILLED)


                xmin, xmax = min(xList), max(xList)
                ymin, ymax = min(yList), max(yList)
                bbox = xmin, ymin, xmax, ymax
```

```
        if draw:
            cv2.rectangle(img, (xmin - 20, ymin - 20), (xmax + 20, ymax + 20), (0,
255, 0), 2)
            return self.lmList, bbox
```

5.1.2 Drag

This module enables the drag and drop feature for shapes in the application

Drag.py

```
import cv2
import numpy as np
import time
import os
import mediapipe as mp
import math
from cvzone.HandTrackingModule import HandDetector


class DragShape():
   def __init__(self,posOrigin,size=[150,150],isShape=None,radius = 100,color =
(255,255,255),poly1=[0,0],ecli=[150,100],tri1=[0,0]):
      self.posOrigin=posOrigin
      self.size = size
      self.isShape=isShape
      self.radius=radius
      self.color = color
      self.poly1=poly1
      self.ecli=ecli
      self.tri1=tri1
      self.arrow1=[100,100]
      self.diamond1=[100,100]
      self.five_side=[100,100]
```

```
    self.six_side=[100,100]
    self.star1=[100,100]
    self.star2=[100,100]

def update(self, cursor,drawColor):
    ox, oy = self.posOrigin
    if self.isShape=='circle':
        self.size=100,100
    h, w = self.size

    # Check if in region
    if ox < cursor[0] < ox + w and oy < cursor[1] < oy + h:
        self.posOrigin = cursor[0] - w//2 , cursor[1] - h//2
        self.color=drawColor

def zoom(self,cursor,scale):
    ox, oy = self.posOrigin

    h, w = self.size

    if ox < cursor[0] < ox + w and oy < cursor[1] < oy + h:
        h1, w1= self.size
        #print(scale)
        if scale <10:
            newH, newW = ((h1 - scale) // 2) * 2, ((w1 - scale) // 2) * 2
            #print(newH,newW)
        else:
            newH, newW = ((h1 + scale) // 2) * 2, ((w1 + scale) // 2) * 2
        try:
```

```
    if newH>250 and newW>250:
        pass
    if newH<100 and newW<100:
        pass
    if 100<=newH<250 and 100<=newW<250:
        if self.isShape == 'circle':
            self.radius = scale * 7
        elif self.isShape == 'polygon':
            self.poly1=scale*2,scale*2
        elif self.isShape=='eclipse':
            self.ecli=scale*2+100,scale*2
        elif self.isShape=='triangle':
            self.tri1=scale*2,scale*2
        elif self.isShape=='arrow':
            self.arrow1=scale*2,scale*2
        elif self.isShape=='diamond':
            self.diamond1=scale*2,scale*2
        elif self.isShape=='five_side':
            self.five_side=scale*2,scale*2
        elif self.isShape=='six_side':
            self.six_side=scale*2,scale*2
        elif self.isShape=='star1':
            self.star1=scale*2,scale*2
        elif self.isShape=='star2':
            self.star2=scale*2,scale*2
        elif self.isShape=='rect':
            self.size = newW, newH
except
            pass
```

5.1.3 Pattern

This module implements the insert pattern feature, which enables the user to drag and drop a pattern on the canvas

Pattern.py

```
import numpy as np

class Pattern:

    def Right_arrow(self,startPoint,endPoint):

        a,b = endPoint[0],endPoint[1]


        A= (int(startPoint[0]), int(a / 2 + startPoint[1]))

        B = (int(startPoint[0] + 4 * b / 3), int(a / 2 + startPoint[1]))

        C = (int(startPoint[0] + 4 * b / 3), int(startPoint[1]))

        D = (int(startPoint[0]), int(3 * a / 2 + startPoint[1]))

        E = (int(startPoint[0] + 4 * b / 3), int(3 * a / 2 + startPoint[1]))

        F = (int(startPoint[0] + 4 * b / 3), int(2 * a + startPoint[1]))

        G = (int(startPoint[0] + 2 * b), int(a + startPoint[1]))


        points = np.array([[A, B], [B, C], [C, G],

                           [A,D],[D,E],[E,F],[F,G]], np.int32)

        points = points.reshape((-1, 1, 2))

        return points


    def Diamond(self,startPoint,endPoint):

        a,b = endPoint[0],endPoint[1]


        A= (int(startPoint[0] + b), int(startPoint[1]))

        B = (int(startPoint[0]), int(startPoint[1] + a))

        C = (int(startPoint[0] + b), int(int(startPoint[1]) + 2 * a))

        D = (int(startPoint[0] + 2 * b), int(startPoint[1] + a))
```

```
    points = np.array([[A, B], [B, C], [C, D],
                       [D,A]], np.int32)
    points = points.reshape((-1, 1, 2))
    return points

def triangle(self,startPoint,endPoint):
    a,b = endPoint[0],endPoint[1]

    A = (int(startPoint[0]), int(endPoint[1]))
    B = (int(endPoint[0]), int(endPoint[1]))
    C = (int(startPoint[0] + b), int(startPoint[1]))

    points = np.array([[A, B], [B, C], [C, A]], np.int32)
    points = points.reshape((-1, 1, 2))
    return points

def Polygon_Five(self,startPoint,endPoint):
    a,b = endPoint[0],endPoint[1]

    A = (int(startPoint[0] + b), int(startPoint[1]))
    B = (int(startPoint[0]), int(startPoint[1] + a))
    C = (int(startPoint[0] + b / 2), int(int(startPoint[1]) + 2 * a))
    D = (int(startPoint[0] + 3 * b / 2), int(startPoint[1] + 2 * a))
    E = (int(startPoint[0] + 2 * b), int(startPoint[1] + a))

    points = np.array([[A, B], [B, C], [C, D],[D,E],[E,A]], np.int32)
    points = points.reshape((-1, 1, 2))
    return points
```

```
def Polygon_Six(self,startPoint,endPoint):
    a, b = endPoint[0], endPoint[1]

    A = (int(startPoint[0] + b), int(startPoint[1]))
    B = (int(startPoint[0]), int(startPoint[1] + a / 2))
    C = (int(startPoint[0]), int(int(startPoint[1]) + 3 * a / 2))
    D = (int(startPoint[0] + b), int(startPoint[1] + 2 * a))
    E = (int(startPoint[0] + 2 * b), int(startPoint[1] + 3 * a / 2))
    F = (int(startPoint[0] + 2 * b), int(startPoint[1] + a / 2))

    points = np.array([[A, B], [B, C], [C, D], [D, E], [E, F],[F,A]], np.int32)
    points = points.reshape((-1, 1, 2))
    return points

def star(self,startPoint,endPoint):
    a,b = endPoint[0], endPoint[1]

    A = (int(startPoint[0] + b), int(startPoint[1]))
    A1 = (int(startPoint[0] + 3 * b / 4), int(startPoint[1] + 3 * a / 4))
    A2 = (int(startPoint[0]), int(0.8 * a + startPoint[1]))
    A3 = (int(startPoint[0] + b * 0.65), int(startPoint[1] + 1.25 * a))
    A4 = (int(startPoint[0] + b / 2), int(startPoint[1] + 2 * a))

    B = (int(startPoint[0] + b), int(1.25 * a + startPoint[1]))
    B1 = (int(startPoint[0] + 1.5 * b), int(2 * a + startPoint[1]))
    B2 = (int(startPoint[0] + 1.35 * b), int(1.25 * a + startPoint[1]))
    B3 = (int(startPoint[0] + 2 * b), int(0.8 * a + startPoint[1]))
    B4 = (int(startPoint[0] + 1.25 * b), int(3 * a / 4 + startPoint[1]))
```

```
    points = np.array([[A, A1], [A1, A2],
                       [A2, A3], [A3, A4],
                       [A4, B], [B, B1],
                       [A, B4], [B4, B3],
                       [B3, B2], [B1, B2]
                       ], np.int32)
    points = points.reshape((-1, 1, 2))
    return points

def Star_six(self,startPoint,endPoint):
    a, b = endPoint[0], endPoint[1]

    A = (int(startPoint[0] + b), int(startPoint[1]))
    B = (int(startPoint[0] + 3 * b / 4), int(startPoint[1] + a / 2))
    C = (int(startPoint[0]), int(int(startPoint[1]) + a / 2))
    D = (int(startPoint[0] + b / 2), int(startPoint[1] + a))
    E = (int(startPoint[0]), int(startPoint[1] + 1.5 * a))
    F = (int(startPoint[0] + 3 * b / 4), int(startPoint[1] + 1.5 * a))
    G = (int(startPoint[0] + b), int(startPoint[1] + 2 * a))
    H = (int(startPoint[0] + 1.25 * b), int(startPoint[1] + 1.5 * a))
    I = (int(startPoint[0] + 2 * b), int(startPoint[1] + 1.5 * a))
    J = (int(startPoint[0] + 1.5 * b), int(startPoint[1] + a))
    K = (int(startPoint[0] + 2 * b), int(startPoint[1] + a / 2))
    L = (int(startPoint[0] + 1.25 * b), int(startPoint[1] + a / 2))

    points = np.array([[A, B], [B, C], [C, D],
                       [D, E], [E, F], [F, G],
                       [G, H], [H, I], [I, J],
                       [J, K], [K, L], [L, A]],
```

```
                np.int32)
    points = points.reshape((-1, 1, 2))
    return points
```

5.1.4 Integrate

This is the final module this integrates all the different modules into a single application for execution

Integrate.py

```
import cv2
import numpy as np
import time
import os
import HandTrackingModule as HT
import Pattern as pt
import Drag as DT

def Draw_shapes(listImg,img,pattern):
  for rect1 in listImg:
    if rect1.isShape == 'arrow':
      startPoint = rect1.posOrigin[0], rect1.posOrigin[1]
      endPoint = rect1.arrow1
      points = pattern.Right_arrow(startPoint, endPoint)
      cv2.fillPoly(img, pts=[points], color=rect1.color)

    elif rect1.isShape == 'diamond':
      startPoint = rect1.posOrigin[0], rect1.posOrigin[1]
      endPoint = rect1.diamond1
      points = pattern.Diamond(startPoint, endPoint)
      cv2.fillPoly(img, pts=[points], color=rect1.color)

    elif rect1.isShape == 'rect':
      cx, cy = rect1.posOrigin[0], rect1.posOrigin[1]
      w, h = rect1.size
      cv2.rectangle(img, (cx - w // 2, cy - h // 2),
                (cx + w // 2, cy + h // 2), rect1.color, cv2.FILLED)

    elif rect1.isShape == 'triangle':
      cx, cy = rect1.posOrigin[0], rect1.posOrigin[1]
      w, h = rect1.size
      x, y = rect1.tri1
```

```
        pt1 = (cx, cy)
        pt2 = (cx - 50, cy + 100 + y)
        pt3 = (cx + 50 + x, cy + 100 + y)
        triangle_cnt = np.array([pt1, pt2, pt3])
        cv2.drawContours(img, [triangle_cnt], 0, rect1.color, -1)

    elif rect1.isShape == 'five_side':
        startPoint = rect1.posOrigin[0], rect1.posOrigin[1]
        endPoint = rect1.five_side
        points = pattern.Polygon_Five(startPoint, endPoint)
        cv2.fillPoly(img, pts=[points], color=rect1.color)

    elif rect1.isShape == 'six_side':
        startPoint = rect1.posOrigin[0], rect1.posOrigin[1]
        endPoint = rect1.six_side
        points = pattern.Polygon_Six(startPoint, endPoint)
        cv2.fillPoly(img, pts=[points], color=rect1.color)

    elif rect1.isShape == 'eclipse':
        cx, cy = rect1.posOrigin[0], rect1.posOrigin[1]
        w, h = rect1.size
        a, b = rect1.ecli
        center = cx, cy
        axes = a, b
        angle = 15
        cv2.ellipse(img, center, axes, angle, 0, 360, rect1.color, cv2.FILLED)

    elif rect1.isShape == 'star1':
        startPoint = rect1.posOrigin[0], rect1.posOrigin[1]
        endPoint = rect1.star1
        points = pattern.star(startPoint, endPoint)
        cv2.fillPoly(img, pts=[points], color=rect1.color)

    elif rect1.isShape == 'star2':
        startPoint = rect1.posOrigin[0], rect1.posOrigin[1]
        endPoint = rect1.star2
        points = pattern.Star_six(startPoint, endPoint)
        cv2.fillPoly(img, pts=[points], color=rect1.color)

    elif rect1.isShape == 'circle':
        cx, cy = rect1.posOrigin[0], rect1.posOrigin[1]
        w, h = rect1.size
        cv2.circle(img, (cx + w // 2, cy + h // 2), rect1.radius, rect1.color, cv2.FILLED)
```

```
def main():
    header = cv2.imread(f"Img.png")
    header1=cv2.imread(f"Img4.png")
    #cv2.imshow("jei",header)
    cap = cv2.VideoCapture(0)
    cap.set(3, 1280)
    cap.set(4, 720)
    brushThickness = 5
    pattern = pt.Pattern()
    eraserThickness = 100
    drawColor = (255, 0, 255)
    xp, yp = 0, 0
    mylist = []
    ox, oy = 300, 300
    listImg = []
    path2 = False
    imgCanvas = np.zeros((720, 1280, 3), np.uint8)
    detector = HT.HandDetector(detectionCon=0.8, maxHands=2)
    x = 1
    a = False
    save = False
    b12 = False
    y3 = 0
    p1 = False

    arrow = False
    diamond = False
    rect = False
    triangle = False

    five_side = False
    six_side = False
    eclipse = False

    star1 = False
    star2 = False

    increase_size = False
    decrease_size = False

    polygon = False
    circle = False
    while True:
```

```
# 1. Import image
success, img = cap.read()
img = cv2.flip(img, 1)
hands, img = detector.findHands(img, count=1)
if hands:
    hand1 = hands[0]
    lmList1 = hand1["lmList"]
    handType1 = hand1["type"]
    x1, y1 = lmList1[8][0:2]
    x2, y2 = lmList1[12][0:2]
    fingers1 = detector.fingersUp(hand1)

    if len(hands) == 2:
        # Hand 2
        hand2 = hands[1]
        lmList2 = hand2["lmList"]
        handType2 = hand2["type"]
        x1, y1 = lmList2[8][0:2]
        x2, y2 = lmList2[12][0:2]
        fingers2 = detector.fingersUp(hand2)

    # 4. If Selection Mode - Two finger are up
    if fingers1[1] and fingers1[2]:
        # xp, yp = 0, 0
        # print("Selection Mode")
        # # Checking for the click
        if 35 < y1 < 110:
            if 20 < x1 < 70:
                drawColor = (0, 255, 0)
            elif 124 < x1 < 190:
                drawColor = (255, 0, 255)
            elif 240 < x1 < 290:
                drawColor = (255, 0, 0)
            elif 340 < x1 < 400:
                drawColor = (0, 0, 255)
            elif 445 < x1 < 510:
                drawColor = (0, 0, 0)

            elif 560 < x1 < 620:
                if decrease_size == False:
                    if brushThickness > 1:
                        brushThickness -= 1
                        decrease_size = True
```

```
                print(brushThickness)
        elif 665 < x1 < 730:
            if increase_size == False:
                if brushThickness < 15:
                    brushThickness += 2
                    increase_size = True
                    print(brushThickness)

        elif 779 < x1 < 870:
            drawColor = (255, 0, 0)
            mylist.append('arrow')
            arrow = True
            a = True
            path2 = True

        elif 920 < x1 < 980:
            drawColor = (0, 0, 0)
            mylist.append('star2')
            star2 = True
            a = True
            path2 = True

        elif 1030 < x1 < 1120:
            drawColor = (255, 0, 255)
            eclipse = True
            mylist.append('eclipse')
            a = True
            path2 = True

        elif 1170 < x1 < 1260:
            save = True
            drawColor = (0, 0, 0)
    if 10 < x1 < 80:
        if 140 < y1 < 180:
            drawColor = (255, 255, 255)
            mylist.append('diamond')
            diamond = True
            a = True
            path2 = True
        elif 215 < y1 < 250:
            drawColor = (0,0,128)
            mylist.append('rect')
            rect = True
            a = True
```

```
            path2 = True
        elif 295 < y1 < 335:
            drawColor = (128,0,128)
            mylist.append('six_side')
            six_side = True
            a = True
            path2 = True
        elif 385 < y1 < 440:
            drawColor = (128,128,0)
            mylist.append('circle')
            circle = True
            a = True
            path2 = True
        elif 480 < y1 < 520:
            drawColor = (0, 165, 255)
        elif 555 < y1 < 600:
            drawColor = (0, 255, 255)
        elif 635 < y1 < 680:
            drawColor = (144, 238, 144)

    elif y1 > 125:
        increase_size = False
        decrease_size = False
    cv2.rectangle(img, (x1, y1 - 25), (x2, y2 + 25), drawColor, cv2.FILLED)

if fingers1[1]:
    if not fingers1[0] and not fingers1[2]:
        if fingers1[1] == True:
            cv2.circle(img, (x1, y1), 15, drawColor, cv2.FILLED)
            # print("Drawing Mode")
            if xp == 0 and yp == 0:
                xp, yp = x1, y1

            cv2.line(img, (xp, yp), (x1, y1), drawColor, brushThickness)
            if drawColor == (0, 0, 0):
                cv2.line(img, (xp, yp), (x1, y1), drawColor, eraserThickness)
                cv2.line(imgCanvas, (xp, yp), (x1, y1), drawColor, eraserThickness)
            else:
                cv2.line(img, (xp, yp), (x1, y1), drawColor, brushThickness)
                cv2.line(imgCanvas, (xp, yp), (x1, y1), drawColor, brushThickness)
xp, yp = x1, y1
if (fingers1[0] and fingers1[1]):
    try:
        lmList = hands[0]['lmList']
```

```
length, info, img = detector.findDistance([lmList[4][0], lmList[4][1]],
                                          [lmList[8][0], lmList[8][1]], img)
pat1 = 'Img'
x = 1
if path2 and a:
    if rect:
        r = DT.DragShape([50 + x * 450, 300], isShape='rect')
        listImg.append(r)
        rect = False

    elif circle:
        r = DT.DragShape([50 + x * 450, 300], isShape='circle')
        listImg.append(r)
        circle = False

    elif polygon:
        r = DT.DragShape([50 + x * 450, 300], isShape='polygon')
        listImg.append(r)
        polygon = False

    elif eclipse:
        r = DT.DragShape([50 + x * 450, 300], isShape='eclipse')
        listImg.append(r)
        eclipse = False

    elif triangle:
        r = DT.DragShape([50 + x * 450, 300], isShape='triangle')
        listImg.append(r)
        triangle = False

    elif arrow:
        r = DT.DragShape([50 + x * 450, 300], isShape='arrow')
        listImg.append(r)
        arrow = False

    elif diamond:
        r = DT.DragShape([50 + x * 450, 300], isShape='diamond')
        listImg.append(r)
        diamond = False

    elif five_side:
        r = DT.DragShape([50 + x * 450, 300], isShape='five_side')
        listImg.append(r)
        five_side = False
```

```
                elif six_side:
                    r = DT.DragShape([50 + x * 450, 300], isShape='six_side')
                    listImg.append(r)
                    six_side = False

                elif star1:
                    r = DT.DragShape([50 + x * 450, 300], isShape='star1')
                    listImg.append(r)
                    star1 = False

                elif star2:
                    r = DT.DragShape([50 + x * 450, 300], isShape='star2')
                    listImg.append(r)
                    star2 = False
                a = False

            if length <= 50:
                cursor = lmList[4]
                for imgObject in listImg:
                    imgObject.update(cursor, drawColor)

            elif length > 50:
                cursor = lmList[4]
                scale = int(np.interp(length, [50, 300], [0, 60]))
                for imgObject in listImg:
                    imgObject.zoom(cursor, scale)
        except:
            pass

    '''if drawColor == (0, 0, 0):
        cv2.line(img, (xp, yp), (x1, y1), drawColor, eraserThickness)'''
    # cv2.line(imgCanvas, (xp, yp), (x1, y1), drawColor, eraserThickness)

    Draw_shapes(listImg,img,pattern)

    if save:
        Draw_shapes(listImg, imgCanvas, pattern)

    if len(hands) == 2:
        if detector.fingersUp(hands[0]) == [1, 1, 0, 0, 0] and detector.fingersUp(hands[1]) ==
[1, 1, 0, 0, 0]:
            lmList1 = hands[0]["lmList"]
            lmList2 = hands[1]["lmList"]
```

```
            length, info = detector.findDistance([lmList1[8][0], lmList1[8][1]], [lmList1[2][0],
lmList1[2][1]],
                                                 )
            length1, info1 = detector.findDistance([lmList1[2][0], lmList1[2][1]],
[lmList2[2][0], lmList2[2][1]],
                                                   )
            length2, info2 = detector.findDistance([lmList2[2][0], lmList2[2][1]],
[lmList2[8][0], lmList2[8][1]],
                                                   )
            length3, info3 = detector.findDistance([lmList2[8][0], lmList2[8][1]],
[lmList1[8][0], lmList1[8][1]],
                                                   )
            x3 = info3[0]
            x4 = info1[0]
            y3 = info3[1]
            y4 = info1[1]
            cv2.rectangle(img, (x3, y3),
                          (x4, y4), (0, 255, 0), 2)
            b12 = True
        if b12 and y1 != 0:
            if y3 == y4 or x3 == x4:
                pass
            if x3 > x4:
                roi = imgCanvas[y3:y4, x4:x3]
            else:
                roi = imgCanvas[y3:y4, x3:x4]
            p1 = True

        imgGray = cv2.cvtColor(imgCanvas, cv2.COLOR_BGR2GRAY)
        _, imgInv = cv2.threshold(imgGray, 50, 255, cv2.THRESH_BINARY_INV)
        imgInv = cv2.cvtColor(imgInv, cv2.COLOR_GRAY2BGR)
        img = cv2.bitwise_and(img, imgInv)
        img = cv2.bitwise_or(img, imgCanvas)

        img[0:125, 0:1280] = header
        img[125:705, 0:100] = header1
        cv2.imshow("Image", img)
        if save:
            cv2.imwrite("b.png", imgCanvas)
        if cv2.waitKey(1):
            if p1 and save:
                r12 = cv2.imread("b.png")
                if x3 > x4:
```

```
            roi = r12[y3:y4, x4:x3]
        else:
            roi = r12[y3:y4, x3:x4]

        cv2.imwrite("bx.png", roi)


if __name__ =='__main__':
    main()
```

8 . SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

The test process is initiated by developing a comprehensive plan to test the general functionality and special features on a variety of platform combinations. Strict quality control procedures are used.

The process verifies that the application meets the requirements specified in the system requirements document and is bug free. The following are the considerations used to develop the framework from developing the testing methodologies.

8.1 Unit Testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input produces valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

8.2 Functional Testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must ber ejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised. Systems/Procedures: interfacing systems or procedures must be invoked.

8.3 System Testing

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

8.4Performance Testing

The Performance test ensures that the output be produced within the time limits, and the time taken by the system for compiling, giving response to the users and request being send to the system for to retrieve the results.

8.5 Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact withouterror.

8.6 Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

8.7 Test Cases

Test case number	1
Test case name	Unit testcase for draw functionality
Feature to be tested	Draw functionality
Description	As the user moves the finger the the brush should follow the finger and draw in the selected color
Sample input	User dragging finger across the screen
Expected output	A line on the canvas
Actual output	A line was drawn across the canvas
Remarks	Success

Table 1: Test case 1

Test case number	2
Test case name	Unit test case for change color functionality
Feature to be tested	Change color
Description	When the user changes the color of the brushes, the color of the brush should change
Sample input	User selects a different color than the currently selected one
Expected output	Color of the brush stroke should change
Actual output	The color of the stroke changed on the canvas
Remarks	Success

Table 2: Test Case 2

Test case number	3
Test case name	Unit test case for increasing thickness of the brush
Feature to be tested	Increase brush thickness
Description	When the user increases the thickness of the brush, the thickness should increase
Sample input	User increases the brush thickness
Expected output	The thickness of the brush stroke increased
Actual output	The brush stroke thickness increased
Remarks	Success

Table 3: Test case 3

Test case number	4
Test case name	Unit test case for decreasing thickness of the brush
Feature to be tested	Decrease brush thickness
Description	When the user Decreases the thickness of the brush, the thickness should increase
Sample input	User decreases the brush thickness
Expected output	The thickness of the brush stroke decreased
Actual output	The brush stroke thickness decreased
Remarks	Success

Table 4: Test case 4

Test case number	5
Test case name	Unit test case for eraser
Feature to be tested	Eraser
Description	The user should be able to erase drawing on the canvas
Sample input	User gestures over the part of the drawing that needs to be erased
Expected output	The drawing is erased
Actual output	The drawing is successfully erased
Remarks	Success

Table 5: Test case 5

Test case number	6
Test case name	Unit test case for inserting shape
Feature to be tested	Insert shape
Description	The user should be able to insert shape from the shape list
Sample input	The user gestures over a shape to insert it
Expected output	The shape is inserted to the desired location
Actual output	The shape is inserted
Remarks	Success

Table 6: Test case 6

Test case number	7
Test case name	Unit test case for drag feature
Feature to be tested	Drag feature
Description	The use should be able to drag a shape to different location
Sample input	User used the drag gesture to change the location of the shape
Expected output	The shape is moved to the desired location
Actual output	The shape is successfully moved
Remarks	Success

Table 7: Test case 7

9. RESULTS AND DISCUSSION

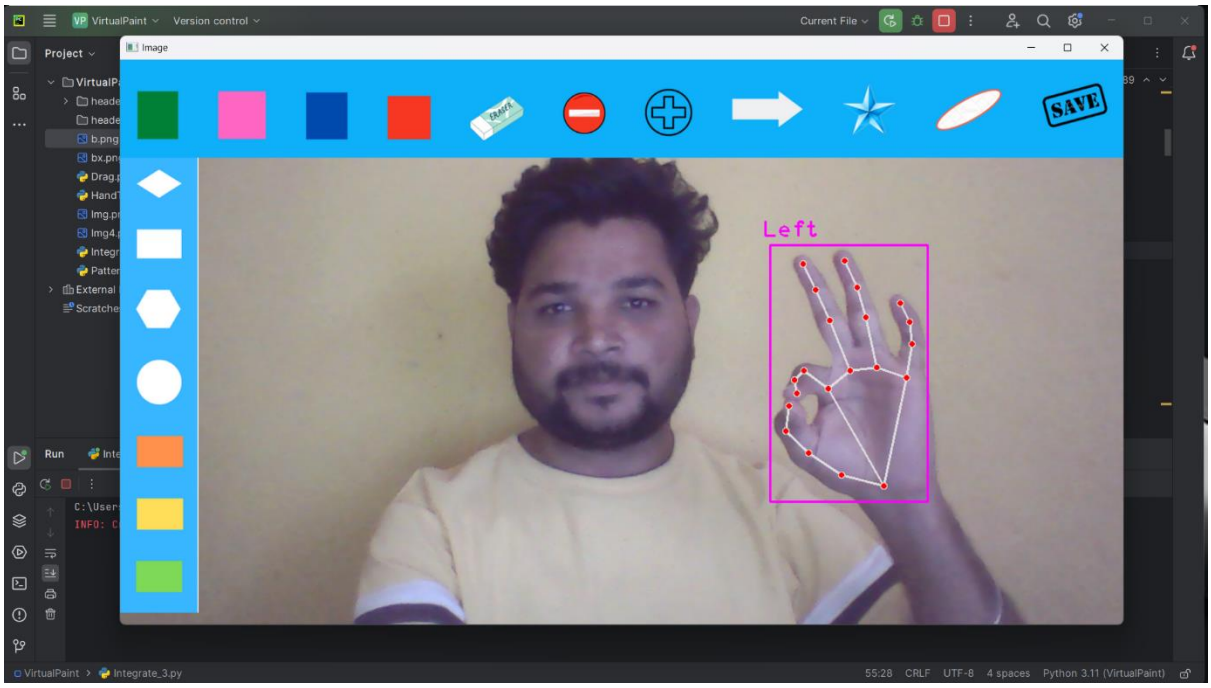


Figure 13: Working of handtracking

The above figure 13., shows the hand tracking working model where all the assigned nodes get recognised and these nodes are used to tracks the hand for gesture recognition .

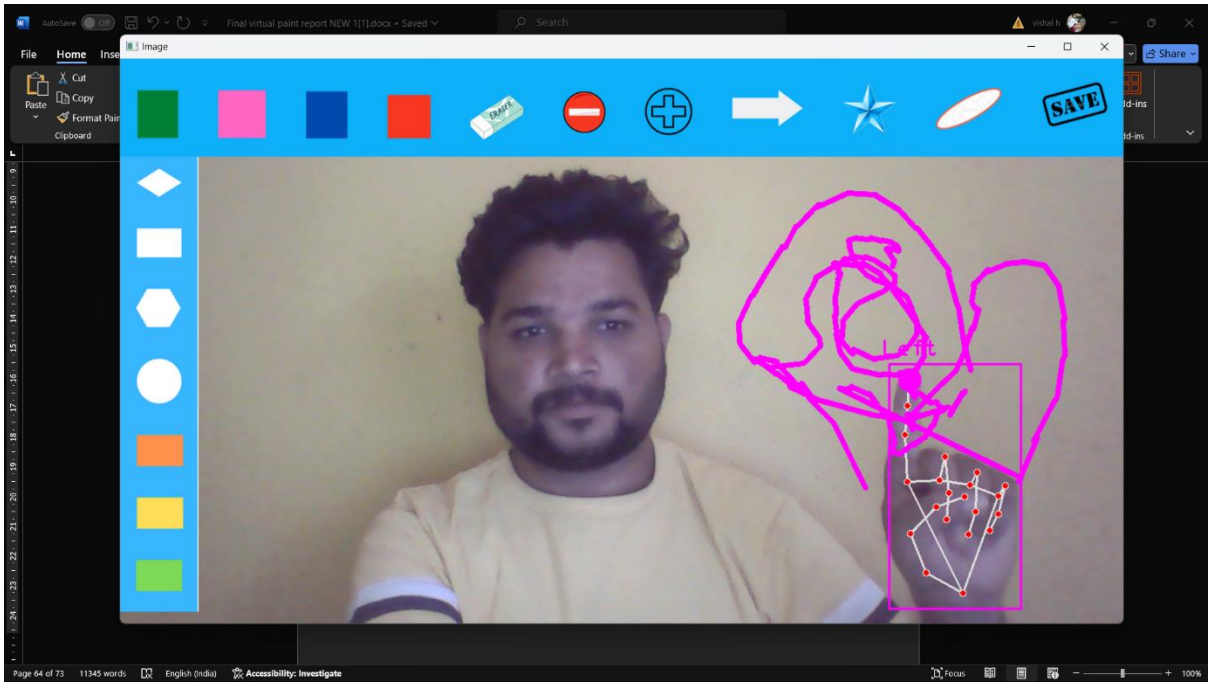


Figure 14: Working of brush

the above figure 14. Shows the working of brush were the hand movement get tracked and the same is drawn. We can draw or paint anything on the screen when the brush is selected.

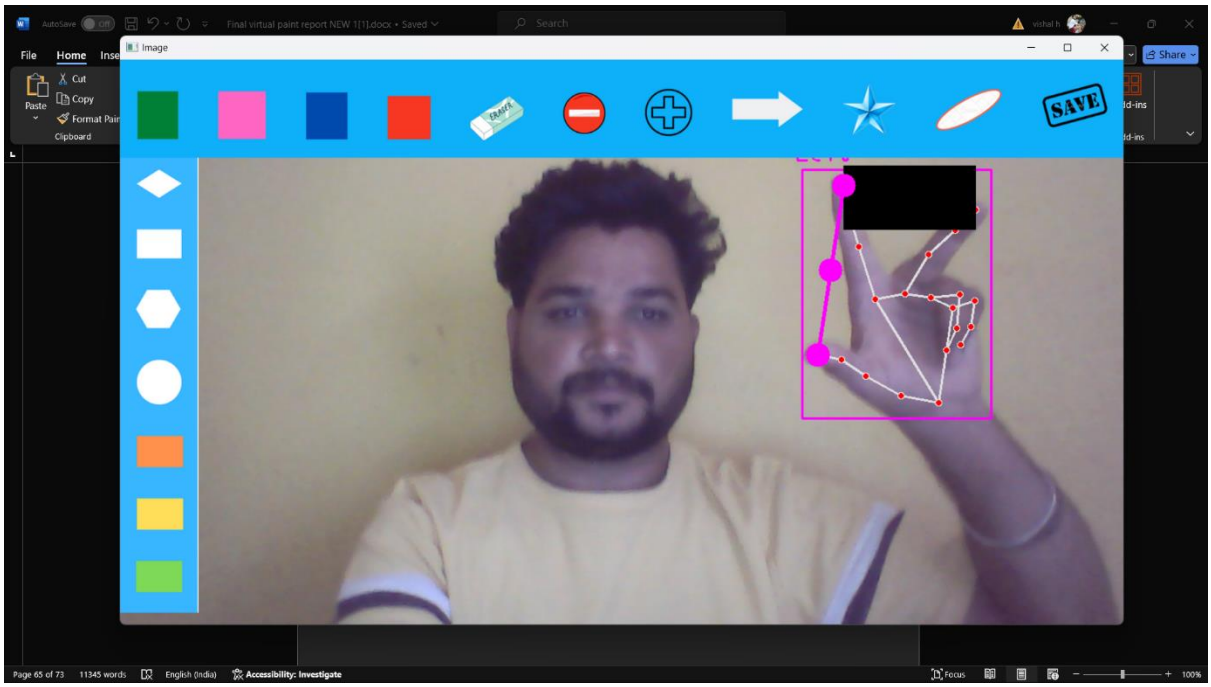


Figure 15: Working of thickness adjust

The above figure 15., shows the working of thickness adjustment. To adjust the thickness of the brush we need to pinch out and pinch in two fingers to increase and decrease the thickness respectively

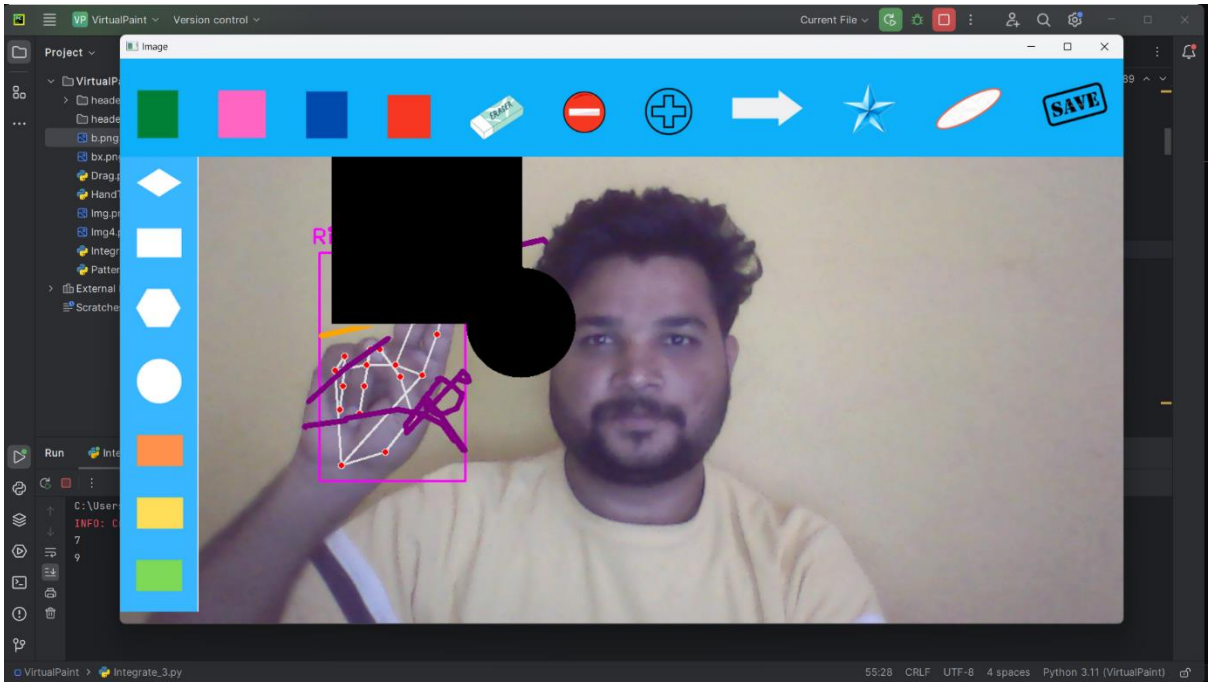


Figure 16: Working of shape insertion

The above figure 16., shows the working of shape insertion where we use two fingers to select. These two fingers are moved towards the shape we need then the shape will will display on screen.

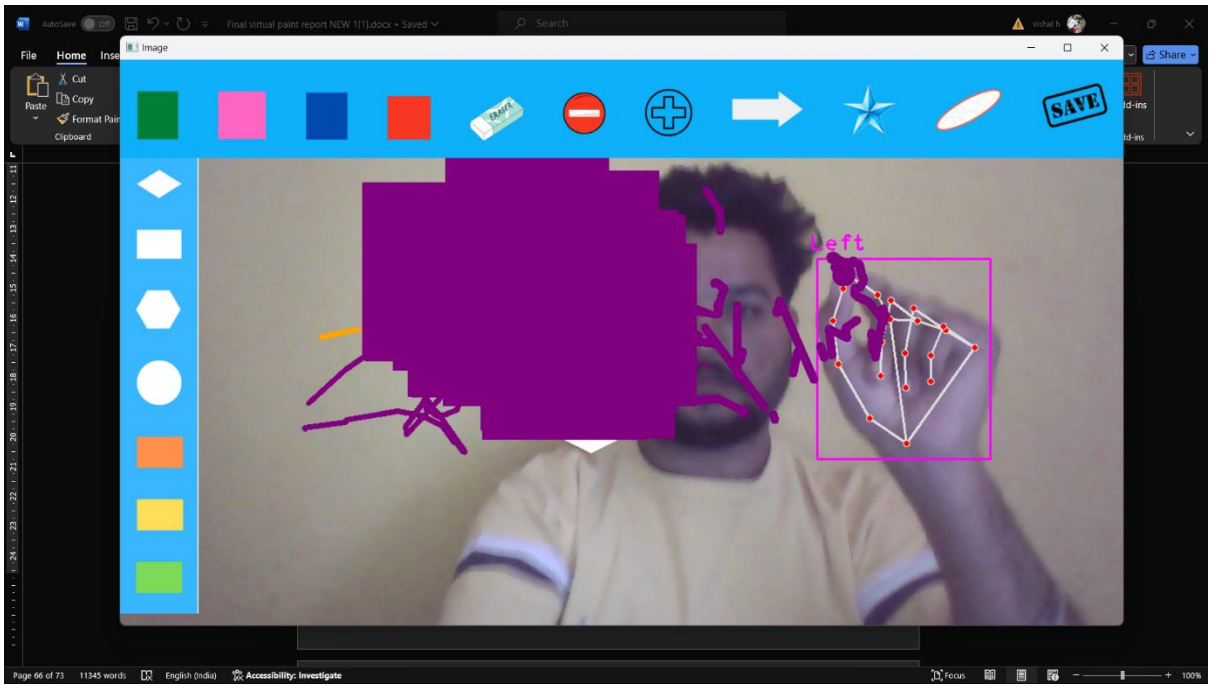


Figure 17: Working of drag and drop shape

The above figure 17 shows how the particular(any) shapes can be selected from the box of shapes and dragged to the central part of screen and droped.

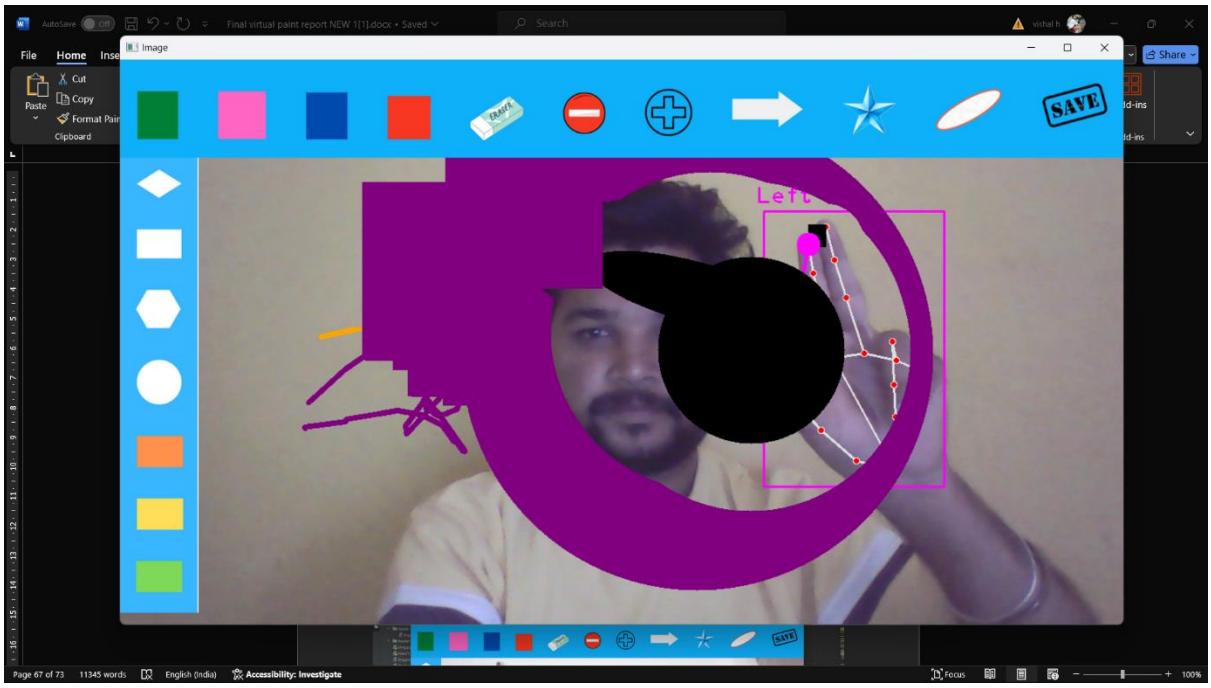


Figure 18: Working of change shape

The above figure 18 shows how the shapes size and dimensions can be changed and altered.

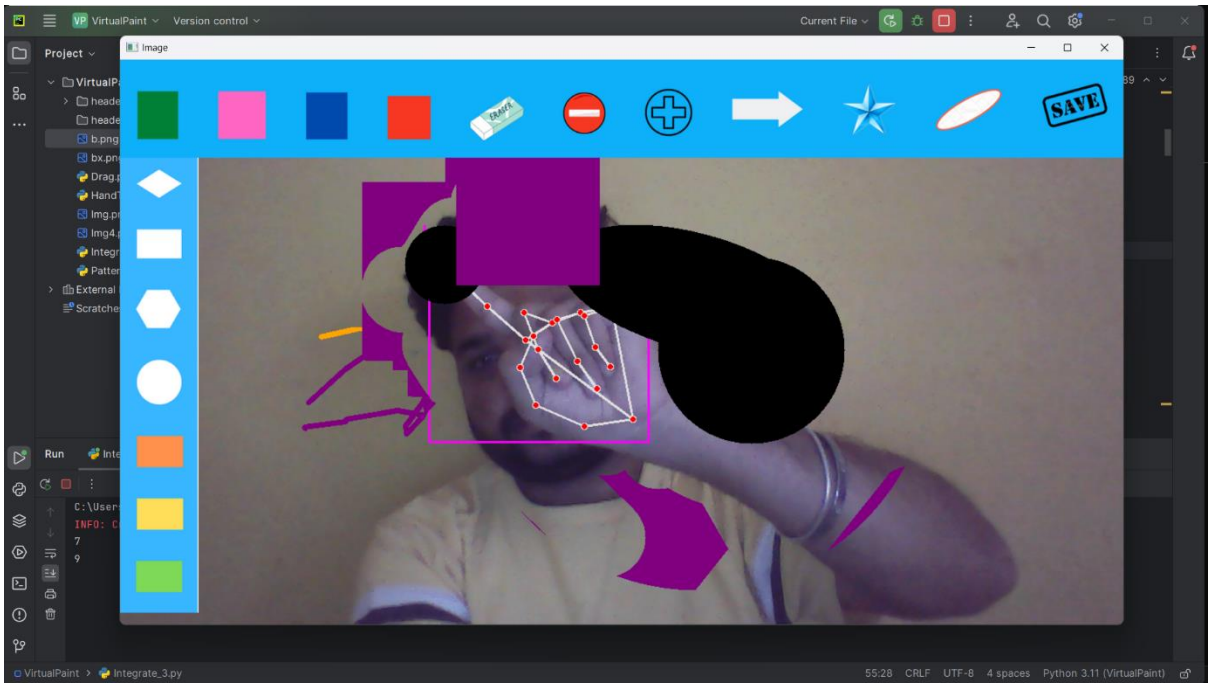


Figure 19: Working of eraser

The above figure 19 shows the working of eraser how the drawn shapes and drawing can be be erased once after selecting the eraser.

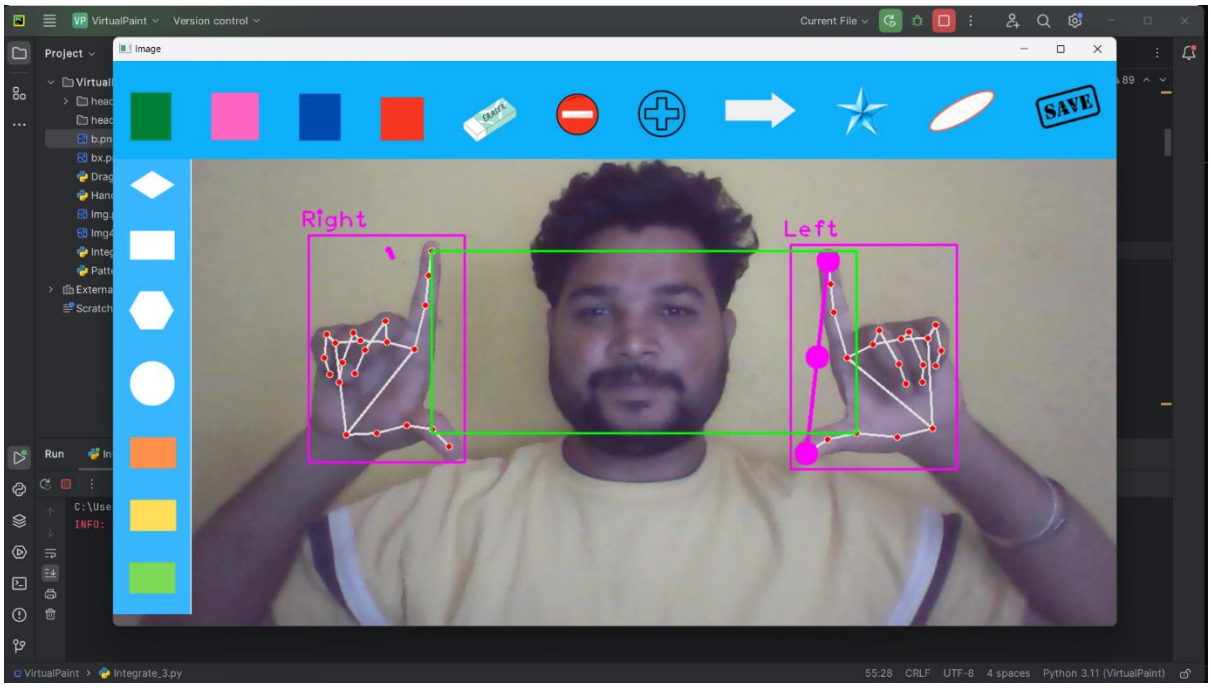


Figure 20: Working of crop tool

The above figure 20 describes the working of crop tools i.e enlarging the size of shapes and minimising the size of shape and adjusting the different shapes and cropping of shapes.

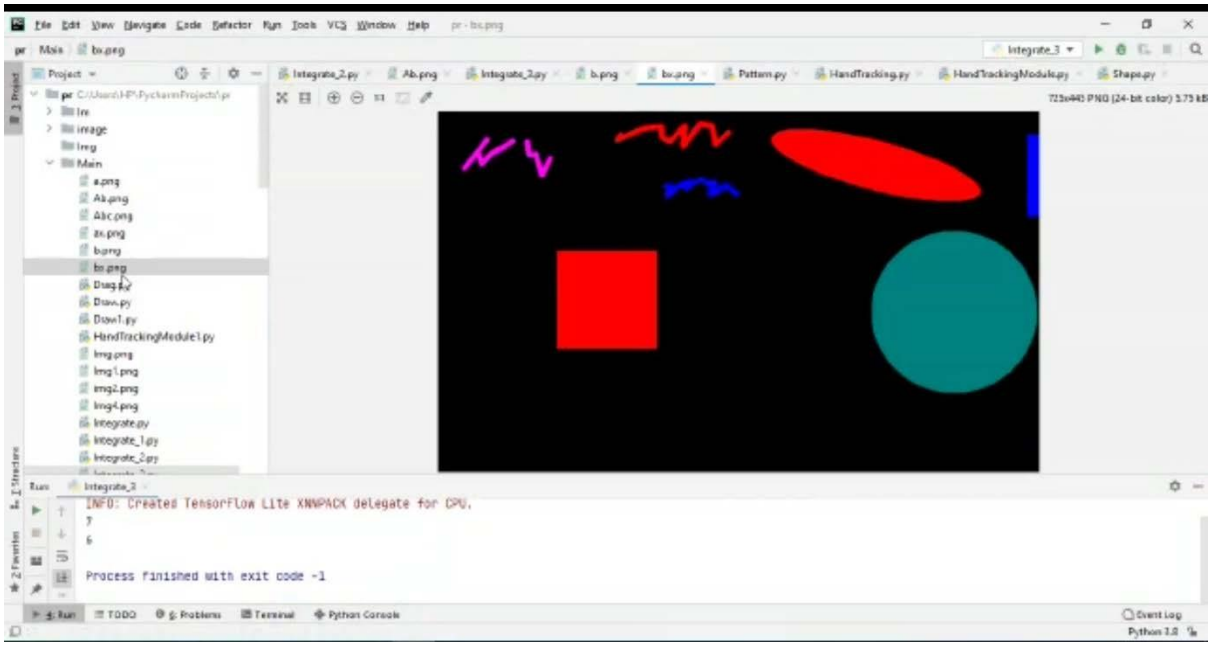


Figure 21: Saved cropped image

The figure 20 shows the saved drawings in the image box.

10. CONCLUSION AND FUTURE SCOPE

In this project we proposed and carried out a project where we tried to prove that gesture based HCI is as good as traditional mouse, keyboard, touch HCI by developing an application that runs entirely on gestures. We made use of existing technologies to develop a cost effective and easily accessible way of integrating gesture-based controls in current and upcoming devices and services. With rapid advancements in the files of camera technology, machine learning this way of interacting with machines will only get easier. With the world pushing for ways to avoid contact with public services like ATM, integrating gesture controls is a great way of enabling the general public to interact with such services without worrying. Going forward free to use models like mediapipe can play a great role in advancing gesture based HCI.

11. REFERENCES

1 J. Francis and A. B K, "Significance of Hand Gesture Recognition Systems in Vehicular Automation-A

Survey", International Journal of Computer Applications, vol. 99, no. 7, pp. 50-55, 2014.

2. R. Bowden, D. Windridge, T. Kadir, A. Zisserman, M. Brady, “A Linguistic Feature Vector for the Visual

Interpretation of Sign Language”, in Tomas Pajdla, Jiri Matas

(Eds), Proc. European Conference on Computer Vision, ECCV04, v. 1: 391-401, LNCS3022, Springer-Verlag,

2004.

3. Radhika Bhatt, Nikita Fernandes, Archana Dhage "Vision Based Hand Gesture Recognition for Human

Computer Interaction" University Of Mumbai (IJESIT) Volume 2, Issue 3, May 2013.

4. Siam, Sayem & Sakel, Jahidul & Kabir, Md. . Human Computer Interaction Using Marker Based Hand

Gesture Recognition., 2016

5. M. Lee and J. Bae, "Deep Learning Based Real-Time Recognition of Dynamic Finger Gestures Using a Data

Glove," in IEEE Access, vol. 8, pp. 219923-219933, 2020, doi: 10.1109/ACCESS.2020.3039401.

6. S. Khan, M. E. Ali, S. Das and M. M. Rahman, "Real Time Hand Gesture Recognition by Skin Color

Detection for American Sign Language," 2019 4th International Conference on Electrical Information and

Communication Technology (EICT), 2019, pp. 1-6, doi: 10.1109/EICT48899.2019.9068809.

7. P. Ramasamy, G. Prabhu and R. Srinivasan, "An economical air writing system converting finger movements

to text using web camera," 2016 International Conference on Recent Trends in Information Technology

(ICRTIT), 2016, pp. 1-6, doi: 10.1109/ICRTIT.2016.7569563.

8. Prajakta Vidhate, Revati Khadse, Saina Rasal, “Virtual paint application by hand gesture recognition system”, 2019 International Journal of Technical Research and Applications, 2019, pp. 36-39.

9.Akira Utsumi,TsutoniuMiyasato, Fumio KishinoandRyoheiNakatsu, "Constant Hand Gesture 2.RecognitionSystem," Proc. of ACCV '95, vol. 11, pp. 249-253, Singapore,1995

10. Attila Licsár, TamásSzirányi University of Veszprém,"Dynamic Training of Hand Gesture Recognition System" Department ofImage Processing andNeuro figuring, H8200 Veszpré,23-26 Aug. 2004

11. L. Bretzner and T. Lindeberg, "Relative direction from broadened groupings of scanty point and line correspondences utilizing the affifine lens tensor," in Proc. fifth Eur. Conf. PC Vision, Berlin, Germany, June 1998, vol. 1406, Lecture Notes in Computer Science, pp.141-157, Springer Verlag

12. Intel Corp, "OpenCV Wiki," OpenCV Library

Available:http://opencv.willowgarage.com/ wiki/

13. Z. Zhang, Y. Wu, Y. Shan, S. Shafer. Visual board: Virtual mouse console and 3d regulator with a normal piece of paper. In Proceedings of Perceptual UIs, 20016.

14.W. T. Freeman and M. Roth, Orientation histograms for hand signal acknowledgment. Global studio on programmed face and signal acknowledgment. 1995, 12: 296-301.

15.G. R. S. Murthy, R. S. Jadon. (2009). "A Review of Vision Based Hand Gestures Recognition," International Journal of Data Technology and Information Management, vol. 2(2), pp. 405410.

16. Mokhtar M. Hasan, Pramoud K. Misra, (2011). "Splendor Factor Matching For Gesture Recognition System Using Scaled Normalization", International Diary of Computer Science

and Information Technology (IJCSIT), Vol. 3(2). knowledge and measurements. 2010: 249-256.

17. S. Belgamwar and S. Agrawal, "An Arduino Based Gesture Control System for Human-Computer Interface," 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), 2018, pp. 1-3, doi: 10.1109/ICCUBEA.2018.8697673.

18.Y. Wu and C. -M. Wang, "Applying hand gesture recognition and joint tracking to a TV controller using CNN and Convolutional Pose Machine," 2018 24th International Conference on Pattern Recognition (ICPR), 2018, pp. 3086-3091, doi: 10.1109/ICPR.2018.8546209.

19.R. Lyu et al., "A flexible finger-mounted airbrush model for immersive freehand painting," 2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS), 2017, pp. 395-400, doi:10.1109/ICIS.2017.7960025.

20. K. K. Ken, S. H. Cho, H. J. Kim, and J. Y. Lee (2005), “Detecting and tracking moving object using an active camera, ” in Proc. of 7th International Conference of Advanced Communication Technology, ICACT , Vol. 2, pp. 817– 820.