HOME AUTOMATION USING GESTURE RECOGNITION

## A PROJECT REPORT

***Submitted by***

# DEVARASAN C (1920106018)

# KARAN SAMBHAJI BABAR (1920106034)

# DHARUNKUMAR R(1920106019)

***in partial fulfilment for the award of the degree***

***of***

# BACHELOR OF TECHNOLOGY

**IN**

**INFORMATION TECHNOLOGY**

## SONA COLLEGE OF TECHNOLOGY

# ANNA UNIVERSITY: CHENNAI 600 025

**NOVEMBER** **2023**

ANNA UNIVERSITY: CHENNAI 600 025

# BONAFIDE CERTIFICATE

Certified that this project report “**Gesture Automation for Sign Language”** is the bonafide work of **“DEVARASAN (1920106018), KARAN SAMBHAJI BABAR (1920106034),DHARUNKUMAR R(1920106019)”** who carried out the project work under my supervision.

**SIGNATURE SIGNATURE**

Dr. J. AKILANDESWARI Ms. J. DEEPIKA

**HEAD OF THE DEPARTMENT SUPERVISOR**

Professor Assistant Professor

Department Of Information Technology Department Of Information Technology Sona College of Technology Sona College of Technology

Salem- 636 005. Salem- 636 005.

Submitted for Mini Project -I viva voce examination held on…………

|  |  |
| --- | --- |
| **INTERNAL EXAMINER** | **EXTERNAL EXAMINER** |

# ABSTRACT

Gesture Detection Using Machine Learning: The application of machine learning techniques for robust and real-time gesture recognition. Leveraging computer vision and deep learning, we develop an efficient model capable of accurately identifying and interpreting human gestures from video feeds. Our approach harnesses convolutional neural networks (CNNs) to extract intricate spatial features and for temporal context modeling. Through extensive training on diverse gesture datasets, our model achieves high precision and generalizability. The proposed system finds utility in various domains, including human-computer interaction, virtual reality, and gaming, enhancing user experiences and opening avenues for innovative interfaces

# ACKNOWLEDGEMENT

First and foremost, we thank to **power of almighty** for showing us inner peace and for all blessings. Special gratitude to our parents, for showing their support and love always.

We express our sincere thanks to chairman **Sri.C.VALLIAPPA** and principal **Dr.S.R.R.SENTHILKUMAR** for providing adequate facilities to complete the project.

We are immensely grateful to our Head of Information Technology Department, **Dr.J.AKILANDESWARI for** the continue encouragement to complete the project.

We express our heartfelt thanks to our project supervisor **Ms.J.DEEPIKA** for his valuable guidance and fruitful discussions throughout the course of the project work.

We feel proud in sharing this success with all our department Faculty, Staff members and friends who helped directly and indirectly to completing this project successfully.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
|  | **ABSTRACT** | **iii** |
|  | **LIST OF FIGURES** | **vii** |
|  | **LIST OF SYMBOLS AND ABBREVIATION** | **vii** |
| **1.** | **INTRODUCTION** 1.1 About the Project | **1**  **1** |
| **2.** | **LITERATURE REVIEW** | **2** |
| **3.** | **METHODOLOGY** | **4** |
| **4.** | **PROJECT DESCRIPTION**  **4.1 System Architecture**  **4.2 Workflow of system** | **5**  **5**  **6** |
|  | **4.3 Image Collection** | **7** |
|  | 4.3.1 Data Gathering | 7 |
|  | **4.4 Dataset Creation** | **7** |
|  | 4.4.1 Data Preprocessing | 8 |
|  | 4.4.2 Dataset Structure | 8 |
|  | 4.4.3 Data Variety | 9 |
|  | 4.4.4 Data Annotation | 9 |
|  | **4.5 Model Training** | **9** |
|  | 4.5.1 Data Preparation | 10 |
|  | 4.5.2 Model Selection | 10 |
|  | 4.5.3 Training Process | 11 |
|  | 4.5.4 Model Evaluation | 11 |
|  | **4.6 Real – Time Gesture Detection** | **12** |
|  | 4.6.1 Video Input | 13 |
|  | 4.6.2 Computer Vision Techniques | 13 |

|  |  |  |
| --- | --- | --- |
|  | 4.6.3 Model Application | 14 |
|  | 4.6.4 Visualization | 14 |
| **5.** | **EVALUATION** | **15** |
|  | **5.1 Model Accuracy** | 15 |
|  | 5.1.1 Model Robustness | 15 |
|  | **5.2 Generalization** | **16** |
|  | 5.2.1 Usability and Accessibility | 16 |
|  | **5.3 Real-time Performance** | 17 |
|  | 5.3.1 Accuracy and Reliability | 17 |
|  | **5.4 Inclusivity** | 18 |
|  | 5.4.1 User-Focused Design | 18 |
| **6.** | **RESULTS** | **19** |
|  | 6.1 Visual Repository | 19 |
|  | 6.2 Constructing the Dataset | 20 |
|  | 6.3 Model Training Phase | 21 |
|  | 6.4 Gesture Recognition | 22 |
| **7.** | **CONCLUSIONS AND FUTURE WORK** | **24** |
|  | **APPENDICES** | **24** |
|  | Appendix 1 Sample Script | 24 |
|  | **REFERENCES** | **50** |

# LIST OF FIGURES

|  |  |
| --- | --- |
| **FIGURE NO. TITLE** | **PAGE NO** |
| Fig 3.1 Hand Gestures | 4 |
| Fig 3.2 System Architecture | 5 |
| Fig 3.3 Workflow of System | 6 |
| Fig 6.1.1 Preprocessing images | 20 |
| Fig 6.1.2 Image Capture | 20 |
| Fig 6.1.3 List of Images in Folder | 21 |
| Fig 6.2.1 Dataset Created | 22 |
| Fig 6.3.1 Training of data | 22 |
| Fig 6.4.1 Execution of Trained Dataset | 23 |
| Fig 6.4.2 gesture recognition and actuator | 24 |

**LIST OF SYMBOLS AND ABBREVIATIONS**

|  |  |
| --- | --- |
| **CHAPTER 3** | **PAGE NO** |
| Table 3.1 List of Modules | 3 |

# LIST OF SYMBOLS AND ABBREVIATIONS

# ABBREVIATIONS

**API:** Application Program Interface **CNN:** Convolutional Neural Network **GUI:** Graphical User Interface

**NLP:** Natural Language Processing

**RGB:** Red, Green Blue

**H, W:** Height and Width of the frame

**D:** Depth of the frame

**MP:** Media pipe Library **SVM:** Support Vector Machine **F, G:** Finger number

**X, Y:** Coordinates of hand landmarks

**C:** Number of classes

**DIR:** Directory

**Ret:** Return value.

**Cap:** Video Capture object **CV2:** OpenCV library **Dict:** Dictionary

**Model:** Machine Learning model

**Pred:** Prediction

**LB:** Label Binarize

**LFW:** Label Face Words

**LSTM:** Long Short Term Memory

**Path:** File path

# CHAPTER 1 INTRODUCTION

In our pursuit of advancing gesture automation, we have initiated a Gesture Recognition Project. This forward-thinking initiative is designed to streamline communication by automating the interpretation of various gestures. Leveraging the capabilities of computer vision and machine learning, we've successfully crafted a real-time system that can identify and interpret a range of gestures, translating them into actionable commands. This technology has significant implications for automation in sectors such as manufacturing, smart homes, and interactive technologies. Our dedication to innovation propels us to develop solutions that enhance efficiency and connectivity through automated gesture recognition.

# 1.1 About the Project

The Gesture Automation Initiative utilizes state-of-the-art technology to enhance communication through automated gesture recognition. By deploying advanced computer vision techniques, we can precisely identify and interpret an extensive array of gestures. This real-time system incorporates machine learning algorithms to seamlessly convert these gestures into actionable commands, streamlining interactions for various applications. From manufacturing processes to smart environments, this project marks a substantial stride toward a more automated and efficient future. Our commitment is to empower individuals by providing them with a means to effortlessly convey information through automated gesture recognition in diverse contexts..

# CHAPTER 2

**LITERATURE REVIEW**

**Gesture based home automation system(P.N.Arathi et al ,2017)**

**Methodology:** The gesture is captured by the camera and handled by MATLAB software. If the preset gesture matches the gesture which is already there, the data is transferred to the microcontroller, which then controls the household appliances. A, PIC microcontroller, light, fan, camera,power supply, LED, and GSM module combinely make up the hardware module. A USB to serial converter bus, which comes with driver software, is used to connect this hardware module to simulation software

**Observation** : The image collected by the camera may be processed quickly using gesture detection which is based on the MATLAB simulation software tool. However, exact recognition is difficult to do since matching stored Some network architectures, including LeNet, InceptionResNetV2, VGG16, VGG19, ResNet50, and DenseNet201, have already been defined in the literature. The CNN input in image classification is an image represented by a random colour model. In the CNN layer, every neuron has a kernel window which is convolved during training of CNN with input image. It's customary to apply a pooling layer after a convolution layer. This is significant because pooling lessens the dimensionality of feature maps, resulting in faster network training. Convolution and pooling are alternated in certain topologies; for example, GoogLeNet has 5 CNN layers and then by a pooling layer, lines, and texture.In subsequent layers, the retrieved features are further tuned. It is critical to stress that the values of the features, like edges, circles. movements to existent gestures is a difficult operation. The object detection method identifies the object immediately with high precision. This technology is more accurate than using a hand glove to recognise gestures. The Arduino compatible with MATLAB simulation tool is more expensive and difficult to integrate with a PC than the MATLAB compatible with PIC microcontroller

**Recognizing hand gestures for controlling home appliances with mobile sensors**

**(Khanh Nguyen Trong et al,2019)**

**Methodology:** Smart watches containing accelerometers sensors and gyroscopes sensors, smartphones with finite storage, and popular smart home platforms were employed as devices and equipment in genuine smart home configurations. A proper hand gesture vocabulary based on an investigation of real smart homes is provided using which users can recall and control their house effectively. To recognise hand movements, two deep neural networks are used: one is a DeepConvLSTM, which is a convolutional and recurrent network, and the other is a DeepConvLSTM, which is a combination of baseline deep convolutional neural networks (CNN).

**Observation** : Hand gesture recognition can be completed entirely on a standard smartphone. As a result, users don’t require to install more resources, making this system simple to use.

**Hand Gesture Recognition and Interface via a Depth Imaging Sensor for Smart Home Appliances(Dong-Luong Dinha et al,2014)**

**Methodology**: A synthetic hand database is created, which includes hand part annotated maps. The information in this database is used to train RFs (Random Forests). The image is collected and a hand depth silhouette is extracted during the recognition stage. A synthetic hand database is created, which includes hand part annotated maps. Finally, hand motions were identified using the extracted features, resulting in interface commands.

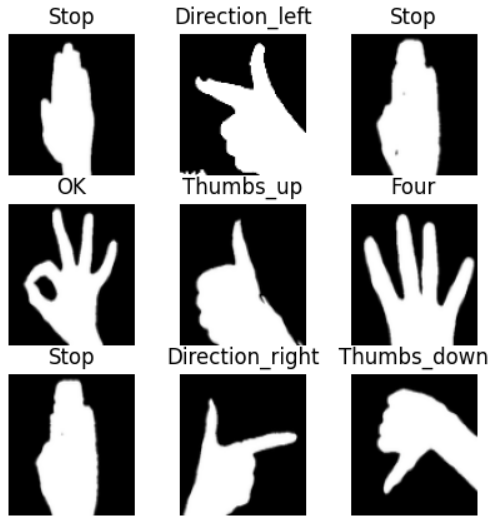
**Observation** : A unique hand gesture recognition system for device management in smart homes is provided in this paper, which uses identified hand parts and trained random forests from a hand depth silhouette. Over the four hand motions from five respondents, the average identification rate was 98.50 percent.

# CHAPTER 3

# METHODOLOGY

|  |  |
| --- | --- |
| **S.NO** | **MODULES** |
| **1.** | Image Collection |
| **2.** | Dataset Creation |
| **3.** | Training the Dataset |
| **4.** | Inference Classifier |

**Table 3.1** List of Modules



**Fig 3.1** Hand Gestures

# CHAPTER 4

**PROJECT DESCRIPTION**

The proposed system detects the gestures given as input by the user and controls the home appliance. The main objective is to provide portability and enable blind, deaf and dumb people to control various appliances with ease and comfort. Also, methods of control are needed due to the increase in the number of industrial and home appliances that must be controlled. The gesture input from the user is captured using an python file and sent to the raspberry pi (which acts as a microcontroller) and the raspberry pi then operates on the respective functionality of the appliances. The system uses a Convolutional Neural Network algorithm for image classification. CNN is used in problems such as pattern and image recognition. They have several advantages compared to other techniques. Using the standard neural network that is equivalent to a CNN, will have a much higher number of parameters thus the training time would also increase proportionately.

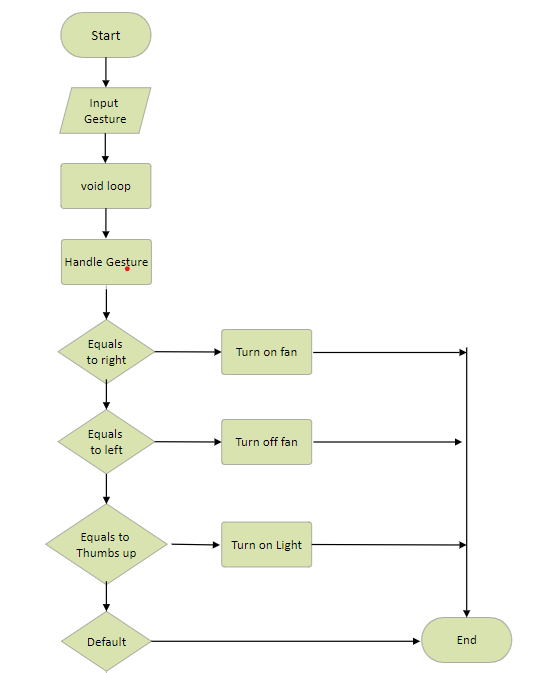
# 4.1 System Architecture

# A diagram of a hand extractor Description automatically generated

**Fig 3.2** System Architecture Diagram

Our system is divided into three modules as shown in the fig consisting of Client Interface, Backend Processing, and Home Interface. Client Interface is responsible for capturing the input gesture from the user and uploading it on the raspberry pi server. Backend Processing involves image preprocessing and training the CNN model and prediction of image class category of input gesture image. Based on this predicted class of image, respectively assigned actions to take place at the home interface

**4.2 Work Flow of the System**

****

**Fig 3.2** System Architecture Diagram

Initially, the user captures the image of a gesture using the camera using python file. This input image is further preprocessed. The images of each gesture are used for training and to create a CNN model the input image acts as a testing image and based on the trained model the class category of testing image(input image) is predicted. If the prediction is precise then the assigned action according to the image class is carried out (for ex. Bulb on, fan off, etc)

## Image Collection:

Image collection plays a pivotal role in our gesture automation project. To establish a robust system, we recognize the critical importance of curating a diverse and extensive dataset of gesture images. The quality and diversity of these images directly influence the system's ability to accurately interpret various gestures and hand positions. It is crucial to ensure that our image collection spans a wide spectrum of gesture expressions, representing diverse hand gestures. This inclusivity is key to developing a system that can effectively cater to a broad user base, promoting accessibility and inclusiveness.

## 4.3.1Data Gathering:

In the context of gesture automation, the process of data collection involves capturing images of various gestures using a camera. It is crucial to curate a dataset that encompasses a broad spectrum of gestures, including diverse hand shapes and movements performed by individuals with varied styles. Consideration of lighting conditions paramount to ensure the system's adaptability to different environments. The collected images should meet high-quality standards, characterized by sharp focus, optimal lighting, and minimal background distractions. To enhance the system's generalization, it is essential to include images with different backgrounds and orientations.

## Dataset Creation:

The establishment of a gesture dataset is a foundational step in developing a resilient and precise automation system. This dataset serves as the training ground for the machine learning model, enabling it to comprehend and interpret diverse gestures accurately. It plays a crucial role in ensuring the model's effectiveness in recognizing and translating the nuanced and varied hand movements inherent in gesture automation. To construct a thorough dataset, it is imperative to include a broad spectrum of gestures and expressions representing various forms of automated gestures.

## 4.4.1 Data Preprocessing:

In the context of gesture automation, data preprocessing is a crucial step before integrating images into the dataset. This process involves normalizing gesture landmarks and ensuring they conform to a consistent format. Normalization is essential to enable the model to accommodate variations in hand size, position, and orientation. Standardizing the scale and orientation of all gesture landmarks reduces the model's sensitivity to individual differences in hand anatomy, particularly relevant when dealing with diverse user inputs in automation scenarios. Additionally, padding the gesture landmarks to a uniform size contributes to more efficient and streamlined processing by the neural network. This not only enhances computational efficiency but also simplifies the model architecture, making it more manageable and interpretable. The data preprocessing stage is instrumental in optimizing the dataset for training the model, ultimately improving the overall performance and robustness of the automated gesture recognition system.

## 4.4.2 Dataset Structure:

In gesture automation, the dataset is organized into distinct training and testing sets. The training set is employed to educate the model, facilitating its learning and adaptation to the inherent patterns and features within the data. This training phase is instrumental for the model to make informed predictions and decisions based on the acquired information. Conversely, the testing set serves as a crucial evaluation stage, examining the model's accuracy and its capacity to generalize beyond the training data.By subjecting the model to unseen data during testing, we can gauge its performance and ensure its ability to make reliable predictions in new, real-world automation scenarios. The segregation of data into training and testing sets is a foundational practice in machine learning, playing a pivotal role in constructing robust and effective models for gesture automation.

## 4.4.3 Data Variety:

In the context of gesture automation, the dataset must encompass a broad spectrum of gestures, including those related to automation commands, symbols, and common actions. It should account for variations in gesture execution speed, style, and hand positions relevant to automation scenarios. This diversity is crucial to ensure that the model, when trained on the dataset, can adeptly recognize and interpret gestures across various automation contexts and scenarios. The dataset should incorporate gestures from different automation languages and dialects, acknowledging that regional variations can significantly impact communication in gesture-based automation.

**4.4.4 Data Annotation:**

In the domain of gesture automation, each image in the dataset is meticulously annotated with the corresponding automation command or symbol. Accurate labeling is imperative for effective model training and assessment in the automation context. These annotations serve not only to provide crucial context to the images but also to facilitate the development and evaluation of gesture automation recognition and translation models. The annotation process involves collaboration with automation experts who ensure that the symbols are correctly identified and labeled, maintaining the integrity of the dataset. Moreover, precise annotation enhances the dataset’s accessibility and usability for researchers and practitioners in the field of gesture automation. This annotated dataset becomes a valuable resource for advancing gesture automation technology and promoting inclusivity in communication by facilitating the development of more accurate and efficient automation models.

## 4.5 Model Training:

Model training is a critical stage in the development of our gesture automation recognition system. This phase involves instructing the machine learning model to comprehend and classify gestures accurately for automation commands and symbols. A substantial dataset of various gestures is utilized to train the model, enabling it to discern the nuanced details of different gestures and their corresponding meanings in the automation context. The model undergoes multiple iterations, fine-tuning its internal parameters to enhance accuracy and minimize errors. This iterative process is guided by the expertise of data scientists and automation specialists, ensuring that the model captures the diverse nuances of gestures relevant to automation scenarios. Ultimately, the success of our gesture automation recognition system relies significantly on the effectiveness of this training process in refining the model's ability to interpret and respond to a wide array of automation gestures. This progress aims to bridge the gap in communication and interaction, particularly in the context of gesture-based automation, fostering a more inclusive and accessible technological landscape.

## Data Preparation:

The model is trained using the meticulously pre-processed gesture automation dataset. This dataset comprises images of various gestures associated with automation commands and symbols, with each image meticulously labeled with the corresponding automation sign. These images have undergone careful curation and processing to ensure high quality and consistency in both lighting and background conditions, aligning with the demands of automation scenarios. Furthermore, the dataset incorporates a diverse range of automation gestures to provide the model with a comprehensive understanding of the nuances in gesture-based communication relevant to automation commands. To bolster the model's performance, noise reduction techniques have been applied, minimizing unwanted artifacts or distractions in the dataset. Additionally, data augmentation methods, such as rotation, scaling, and translation, have been employed to increase the dataset's size and enhance the model's adaptability in recognizing signs from various angles and hand positions within the automation context.

## Model Selection:

During the model selection process in gesture automation, a pivotal decision is made to choose the most appropriate machine learning algorithm that aligns with the project's objectives. An algorithm that frequently demonstrates effectiveness in this context is the Sequential Classifier, renowned for its versatility and robust performance across various automation scenarios. The model selection phase also entails meticulous consideration of hyperparameters and initialization values, ensuring that the chosen algorithm is finely tuned to achieve optimal results in the context of gesture-based automation. Moreover, the decision on the machine learning algorithm is contingent upon the nature of the automation data, the intricacy of the automation command recognition problem, and the available computational resources. This thoughtful selection process contributes to the development of a highly effective and efficient model for gesture automation recognition, enhancing the system's overall performance in diverse automation scenarios.

## Training Process:

In the context of gesture automation, the machine learning model undergoes a comprehensive training process, where it learns from the pre-processed dataset, capturing intricate patterns and relationships between images and their corresponding automation commands or symbols. The overarching goal is to equip the model with the capability to make accurate predictions during real-time gesture detection in automation scenarios. Throughout the training process, the model undergoes multiple iterations, continuously adjusting its internal parameters through a process known as "backpropagation." This involves fine-tuning the model's weights and biases to minimize the disparity between its predictions and the actual automation signs present in the dataset.

## Model Evaluation:

The trained model's accuracy and performance are rigorously assessed using the testing dataset in the context of gesture automation. This evaluation is instrumental in ensuring the model's reliability and its ability to generalize effectively in various automation scenarios. Through a meticulous comparison of the model's predictions with the actual data in the testing dataset, we gain insights into how well it performs on unseen automation gestures, a critical measure of its practical utility. Beyond accuracy, this evaluation process may encompass various other metrics such as precision, recall, F1-score, and the area under the ROC curve, depending on the specific requirements of the automation command recognition problem. This comprehensive assessment is essential for identifying potential issues, enhancing the model's robustness, and making well-informed decisions about its deployment in real-world gesture-based automation applications.

## Real-time Gesture Detection:

Real-time gesture detection stands as the fundamental functionality of our gesture automation recognition system, acting as a pivotal link between individuals using gesture commands and the automated system. This cutting-edge technology seamlessly captures the nuanced movements and expressions involved in gesture automation, providing an immediate translation of these gestures into actionable commands. This functionality is a crucial tool for facilitating effective communication and enhancing accessibility across various automation scenarios. Whether in manufacturing processes, smart environments, or interactive technologies, our system empowers individuals to interact and convey commands in their preferred mode of gesture-based communication.

## Video Input:

The system seamlessly captures video input from a camera, operating in real-time to ensure a continuous flow of data. This video stream is the lifeblood of our gesture automation recognition technology, providing the visual information necessary to detect and interpret automation gestures with precision and accuracy. Through sophisticated algorithms and machine learning models, the system meticulously processes each frame of the video, recognizing not only the gestures themselves but also the subtleties and nuances that convey the rich and diverse vocabulary of gesture-based automation. Our commitment to this video input extends beyond mere recognition; it encompasses understanding the context and emotions behind each automation gesture.

## Computer Vision Techniques:

In the context of gesture automation, computer vision techniques play a pivotal role in identifying and tracking key points associated with hand gestures within each frame. This process is fundamental for precise and real-time recognition of automation gestures. Utilizing advanced algorithms and image processing methods, these techniques extract and analyze crucial features, enabling the system to comprehend and interpret intricate hand movements accurately. The application of computer vision enhances the interaction between humans and automated systems, providing a foundation for diverse applications, ranging from automation gesture recognition to intuitive control of devices. This technology offers a seamless and natural user experience across various automation scenarios.

## Model Application:

The trained machine learning model is actively applied to the processed data, generating real-time predictions about the automation gestures being performed. These predictions are then translated into actionable commands or symbols, facilitating seamless communication for users interacting through gestures. By leveraging advanced computer vision and pattern recognition techniques, the system can accurately identify and interpret complex automation gestures. This technology holds the potential to bridge the communication gap between users interacting through gestures and the automated systems they engage with, offering a more inclusive and accessible environment in various automation scenarios. Moreover, the real-time nature of the predictions allows for immediate interaction, fostering more natural and efficient communication between gesture-based automation users and those who may not be proficient in this mode of interaction.

## Visualization:

The outcomes of real-time gesture detection are dynamically presented on the screen, offering users a seamless and interactive experience. Through innovative technology, the system can visually represent the recognized automation gesture on the screen, creating a dynamic visualization of the communicated command or symbol. Additionally, the corresponding text associated with the recognized gesture is displayed, facilitating clear and efficient communication. This dual-mode approach not only enhances inclusivity but also ensures that the information is readily accessible to a broader audience. In this dynamic visualization process, the system utilizes advanced graphics to highlight and emphasize the recognized gestures, simplifying communication for both the sender and receiver of automation commands.

# CHAPTER 5 EVALUATION

In evaluating our gesture recognition system, we emphasize a thorough assessment encompassing accuracy, adaptability, real-time responsiveness, user interaction, and inclusivity. This holistic evaluation approach aims to provide a comprehensive understanding of our system's strengths and weaknesses. Our goal is to consistently refine and improve our technology to cater to a broad spectrum of gestures, ensuring accessibility and inclusivity for diverse users. This commitment is aimed at fostering a more inclusive and seamless interaction environment for gesture-based communication.

## Model Accuracy:

We place a premium on the precision of our machine learning model in recognizing and categorizing gestures accurately. The comparison between predicted gestures and the actual data, often referred to as ground truth data, is pivotal in ensuring high accuracy. This accuracy is crucial for fostering seamless and error-free gesture-based interactions. It not only enhances communication but also significantly enriches the user experience. Furthermore, a highly precise model lessens the cognitive burden on users, thus promoting a more accessible and user-friendly system. Consistent efforts directed towards refining data, optimizing algorithms, and rigorous testing constitute our ongoing endeavor to boost the accuracy of our model. Our unwavering dedication to precision is the cornerstone of creating a dependable and empowering tool for gesture recognition, benefiting a wide range of users.

## Model Robustness and Adaptability:

Beyond accuracy, it's pivotal to gauge the resilience and adaptability of our machine learning model in the context of gesture recognition. Assessing the model's performance should encompass its capacity to generalize across various hand gesture styles, lighting conditions, and backgrounds. This evaluation ensures its ability to sustain high accuracy in a spectrum of real-world scenarios. Additionally, it's crucial to appraise the model's flexibility in accommodating new signs or gestures, along with its efficiency in retraining when needed. This evaluation plays a crucial role in ensuring the model's long-term efficacy in facilitating seamless communication for individuals reliant on sign language. This comprehensive evaluation framework extends beyond mere accuracy, focusing on the model's practical utility and its impact on user experience.

## Generalization:

The capability of our system to manage a broad spectrum of gesture variations, encompassing differences in speed, style, and hand positions, is fundamental. We conduct rigorous evaluations to ensure its adaptability across diverse conditions. Recognizing the richness and dynamism of gesture-based communication, we emphasize the need for our technology to bridge communication divides for a wide user base. Whether facing a rapid or distinct gesturer, our system is engineered to interpret and respond accurately. Moreover, we acknowledge the significant variability in hand positions and movements within various gestures and regional styles. Our continuous commitment to enhancing adaptability signifies the ongoing evolution of our system to cater to the diverse requirements of gesture-based communication. It's not solely about recognizing gestures; it's about fostering inclusivity and accessibility for all users.

## Usability and Accessibility:

In addition to assessing the system's generalization, it is equally essential to evaluate its usability and accessibility. An effective Gesture recognition system should be user-friendly and accessible to a wide range of individuals, including those with varying levels of gesture proficiency. Therefore, our evaluation extends to user feedback and user testing, focusing on how intuitively the system can be operated and how well it serves its target users. This includes considering the user interface design, the system's response time, and any potential barriers that might affect its accessibility for people with disabilities. By combining these usability and accessibility assessments with rigorous generalization testing, we aim to develop a comprehensive understanding of the system's overall performance and its suitability for diverse user populations.

## Real-time Performance:

Prompt detection and translation of gestures stand as a cornerstone in our mission for seamless communication. Recognizing the imperative for swift outcomes, we prioritize near-instant responses to not only facilitate smooth interactions but also to preempt any potential delays that might impede the user experience. Real-time performance is pivotal, enabling users to communicate seamlessly, overcoming geographical and linguistic barriers effortlessly. To attain this, we consistently fine-tune our systems and technologies, relentlessly pushing the boundaries of real-time responsiveness to augment the usability and accessibility of our services. Our unwavering commitment to swift outcomes underscores our dedication to furnishing top-tier communication tools for our users in gesture recognition.

## Accuracy and Reliability:

In addition to real-time performance, the accuracy and reliability of prompt gesture detection and translation are paramount. Ensuring that the system correctly interprets and translates gestures is crucial for effective communication. Any errors or misinterpretations can lead to confusion and misunderstandings. Therefore, continuous monitoring, testing, and improvement of the algorithm's accuracy are essential. Reliability also plays a significant role, as users should be able to trust that the system will consistently provide accurate translations, fostering confidence

in the technology. Regular updates and quality control measures should be implemented to maintain and enhance both accuracy and reliability, aligning with the goal of delivering a seamless and dependable user experience.

## Inclusivity:

Inclusivity forms the bedrock of our mission in home automation through gesture recognition. Our aim is to bridge the gap between gesture-based control and the wider community by fostering an environment that transcends linguistic and cultural differences. We are committed to developing a system that doesn't just enable control but also cultivates mutual understanding and empathy. To achieve this, we rigorously evaluate the system's impact across various home contexts, ensuring that our technology not only empowers gesture users but also enriches the lives of all household members. Our goal is to create an environment where gesture-based control not only facilitates home management but also fosters unity and inclusivity among all inhabitants. We believe that creating an inclusive and seamless control environment is key to establishing a more equitable and compassionate home, and we're dedicated to realizing this vision.

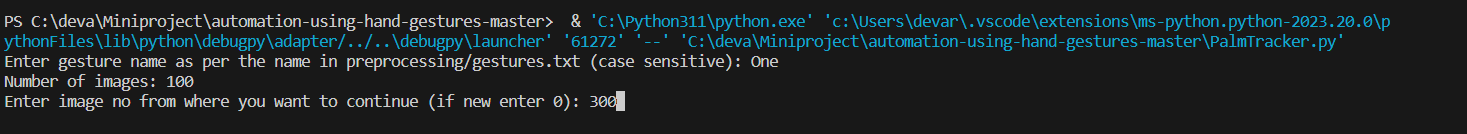
## User-Focused Design:

In creating our gesture-based home automation system, we center our approach on the user. We actively involve homeowners, including individuals using gesture controls, family members, and those involved in home automation, to gather feedback and ensure that the system's design and functionalities match their needs and preferences. This ongoing exchange allows us to consistently enhance the system, aligning it more closely with the varied and evolving requirements of users, ultimately delivering a more inclusive and efficient means of controlling and managing home devices through gestures.

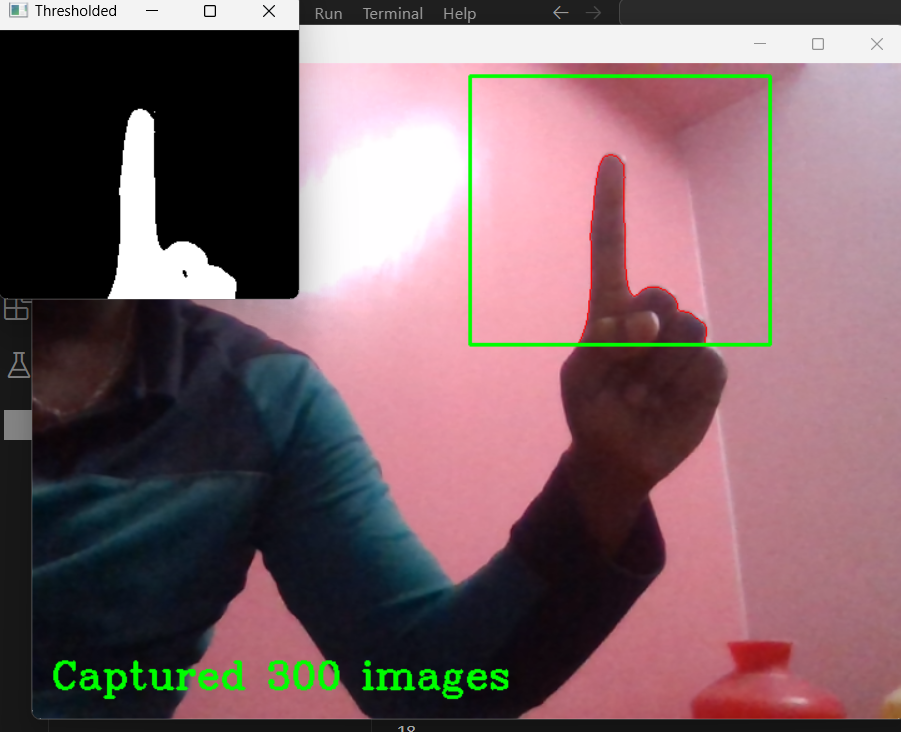
# CHAPTER 6 RESULTS

## VISUAL REPOSITORY:

This module captures hand gestures via a camera feed. It records a diverse dataset, encompassing various hand positions and signs, ensuring high-quality images with proper lighting. These collected images lay the foundation for subsequent dataset creation and model training, making it a critical step in the sign language translation project.

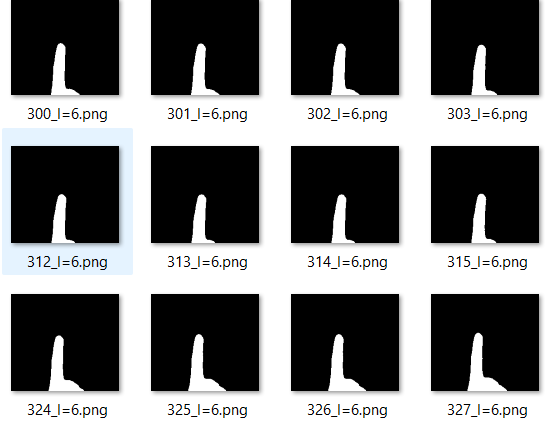


**Fig 6.1.1** preprocessing images



**Fig 6.1.2** Image capturing

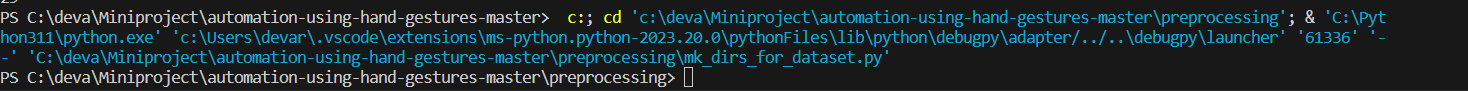
During the image capture process, a high-resolution camera hand gestures with precision, focusing on various hand positions and signs performed by a diverse group of individuals. Each image is meticulously reviewed to ensure optimal lighting, clarity, and quality. The collected images are then organized into a well-structured directory, making it easy to access and manage the dataset. This organized 'List of Images in Folder' serves as a valuable resource for researchers, facilitating efficient data retrieval and enabling seamless integration into the subsequent stages of the sign language translation project. The meticulous approach to image capture and dataset organization ensures that the foundation for this project is solid, setting the stage for effective model training and meaningful results.



**Fig 6.1.3** List of Images in Folder

## CONSTRUCTING THE DATASET:

The "Dataset Creation" module is responsible for structuring the collected sign language gesture images into a well-organized dataset. It involves categorizing the images into different classes or labels, ensuring a balanced representation of gestures. Additionally, the module preprocesses the data by normalizing and converting it to grayscale to a uniform format, making it suitable for machine learning.

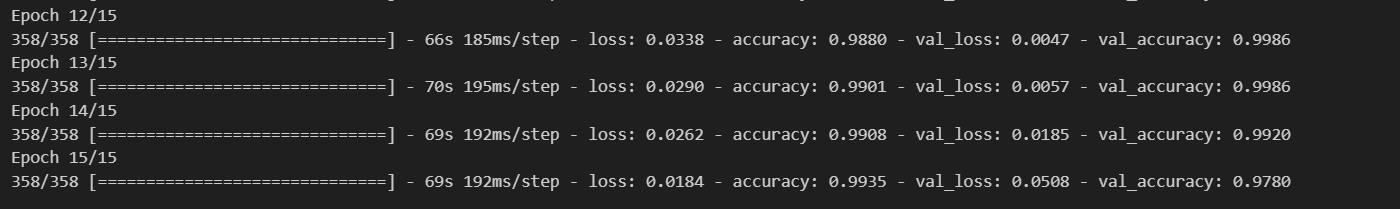


**Fig 6.2.1** Dataset Created

## MODEL TRAINING PHASE:

During the Model Training Phase, the dataset is utilized in training a specialized machine learning architecture, employing Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks. This combination allows for the creation of a robust model tailored for gesture recognition. The CNN and LSTM architectures learn intricate patterns and relationships within the gesture data, facilitating precise recognition.

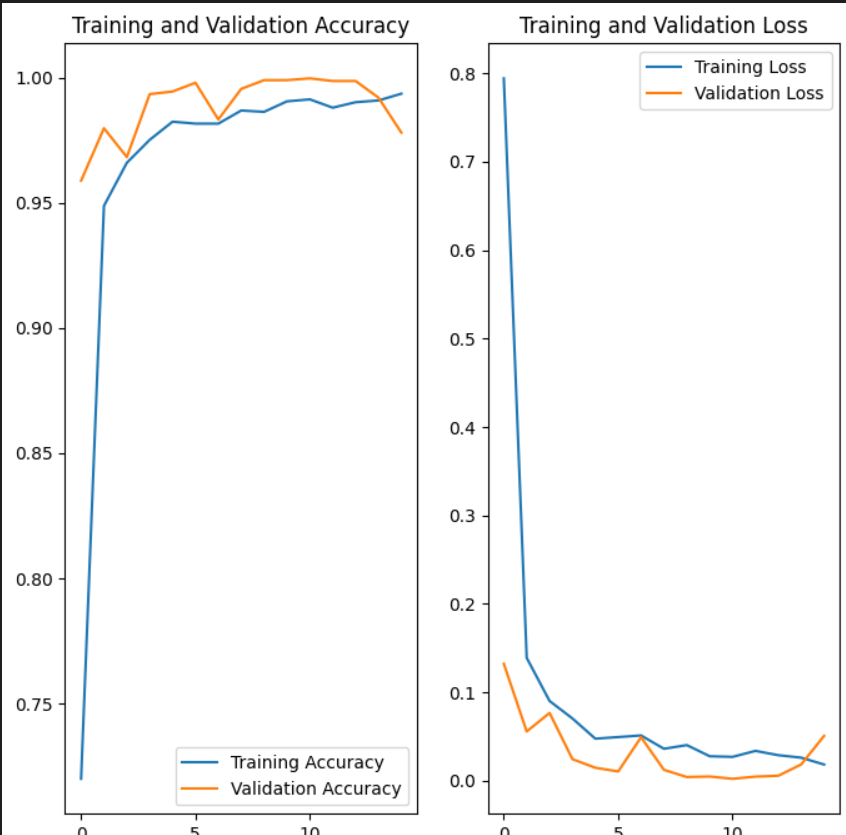
This training phase involves not only feeding the dataset into the model but also refining and adjusting the network's parameters to optimize its performance. These adjustments prepare the system for real-time inference, ensuring swift and accurate gesture recognition. This phase serves as the central component of the gesture recognition system, empowering it to comprehend and interpret various gestures effectively.



**Fig 6.3.1** Training of Data

## GESTURE RECOGNITION:

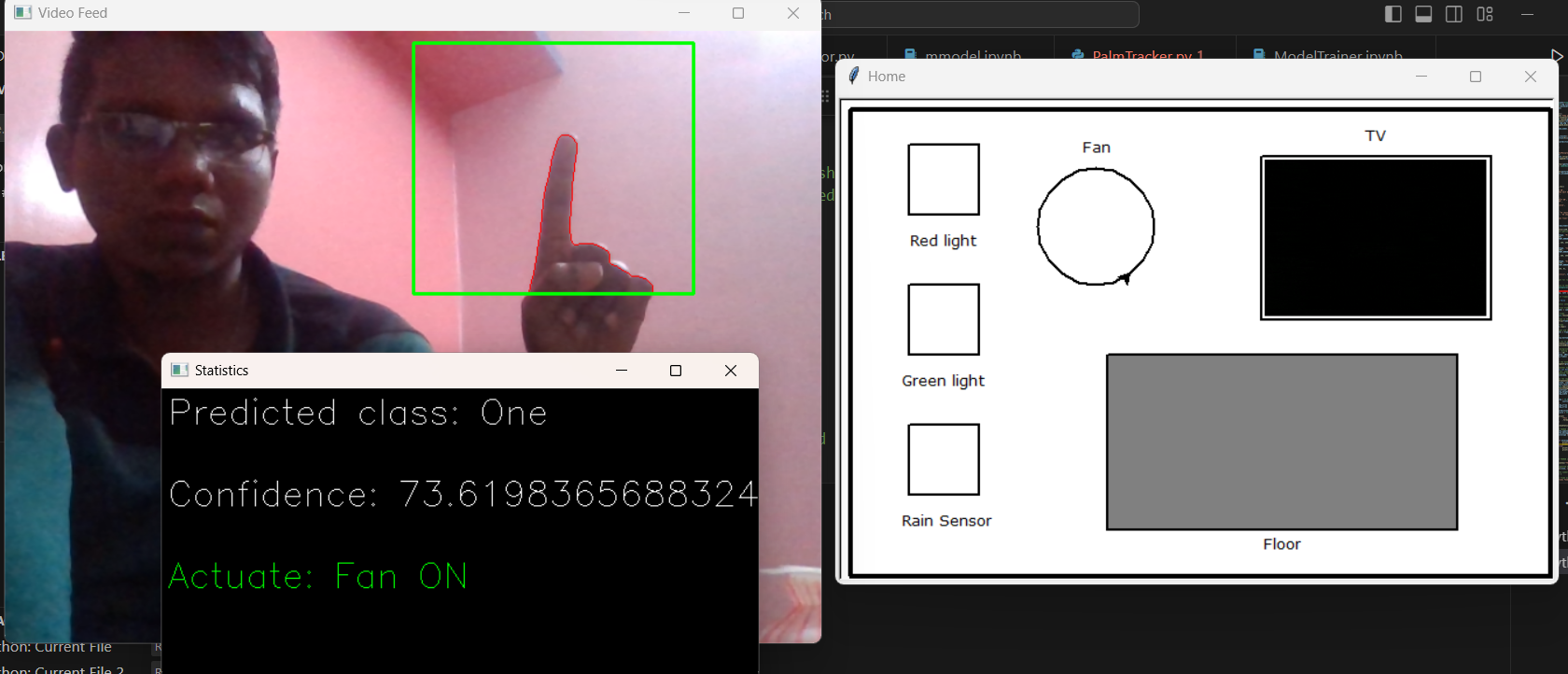
This module utilizes machine learning algorithms to train a model on the collected sign language gesture data. It learns and internalizes patterns, allowing it to recognize and classify gestures accurately. The trained model is a crucial component for real-time gesture Recognition.



**Fig 6.4.1** Execution of Trained Dataset

Once the machine learning model is trained on the compiled gesture dataset, it emerges as a pivotal tool for real-time control in home automation. The operational process involves two primary stages: label assignment and actuation. Label assignment refers to the system's identification of specific gestures made by the user, captured via cameras and sensors. These recognized gestures are then processed in real-time by the trained CNN and LSTM model.

Utilizing a sophisticated blend of gesture recognition techniques, the system interprets the recognized gestures and associates them with predefined commands, such as "turn on fan" or "turn off." This streamlined fusion of label assignment and actuation allows seamless control of home devices based on recognized gestures. By integrating gesture recognition with home automation, individuals can effortlessly execute various commands, effectively managing household devices. This not only simplifies home control but also enhances accessibility and user experience within domestic settings

****

**Fig 6.4.2 gesture recognition and actuator**

Through continual processing and knowledge updates, the inference classifier in home automation with gesture recognition refines its understanding of diverse gestures and their variations. It goes beyond recognizing basic gestures, adapting to the distinct styles of individual users, thus becoming a versatile tool for a broad spectrum of residents. Its ability to learn and evolve ensures its relevance and effectiveness, even with new gestures and individual-specific variations.

Moreover, this inference classifier isn't limited to a single system; it can seamlessly integrate into various home automation devices and applications. This integration transforms it into a valuable asset for creating inclusive and accessible control solutions within homes, offering a user-friendly approach to managing household devices through gesture recognition.

# CHAPTER 7

**CONCLUSION AND FUTURE WORK**

The Gesture Recognition in Home Automation project marks a significant stride in utilizing machine learning and computer vision to streamline control mechanisms for homeowners. Enabling real-time gesture recognition and execution, this project not only simplifies home management but also contributes to a deeper understanding of gesture-based control. Its potential applications in optimizing household tasks, refining lifestyle conveniences, and ensuring efficient interactions within a home setting render it a promising solution.

Looking forward, there are avenues for further refinement and expansion. Enhancing accuracy assessment could involve developing a tailored pixel-level evaluation method customized for gesture recognition in home automation. This method would precisely compare recognized gestures to ideal or "ground truth" values.

The challenge of recognizing multiple gestures within a single frame, especially when gestures differ in size, requires careful attention. Future improvements should focus on fine-tuning the model to accurately manage complex gesture scenarios.

Moreover, adapting to variations in gestures specific to different regions or languages and expanding the system's vocabulary can elevate its inclusivity and usability. Continual research and development in these areas will refine the project's performance, ultimately benefiting homeowners and their interaction with home automation technology.

## APPENDICES APPENDIX 1 SAMPLE SCRIPT

**Code:**

**Image Collection:**

import os

import cv2

import imutils

import numpy as np

from sklearn.preprocessing import LabelEncoder

bg = None

dataset\_dir = 'Dataset'

gestures\_file = 'C:/deva/Miniproject/automation-using-hand-gestures-master/preprocessing/gestures.txt'

def get\_labels():

gestures = []

with open(gestures\_file, 'r') as f:

for line in f:

line = line.strip()

gestures.append(line)

gestures.sort()

gestures = np.array([gestures[i] for i in range(len(gestures))])

integer\_encoded = LabelEncoder().fit\_transform(gestures)

labels = {}

for i in range(len(gestures)):

labels[gestures[i]] = integer\_encoded[i]

# print(labels)

return labels

def run\_avg(image, aWeight):

global bg

# initialize the background

if bg is None:

bg = image.copy().astype("float")

return

# compute weighted average, accumulate it and update the background

cv2.accumulateWeighted(image, bg, aWeight)

def segment(image, threshold=25):

global bg

# find the absolute difference between background and current frame

diff = cv2.absdiff(bg.astype("uint8"), image)

# threshold the diff image so that we get the foreground

thresholded = cv2.threshold(diff,

threshold,

255,

cv2.THRESH\_BINARY)[1]

# get the contours in the thresholded image

(cnts, \_) = cv2.findContours(thresholded.copy(),

cv2.RETR\_EXTERNAL,

cv2.CHAIN\_APPROX\_SIMPLE)

# return None, if no contours detected

if len(cnts) == 0:

return

else:

# based on contour area, get the maximum contour which is the hand

segmented = max(cnts, key=cv2.contourArea)

return (thresholded, segmented)

def main():

# initialize labels dictionary

gesture = input(

'Enter gesture name as per the name in preprocessing/gestures.txt (case sensitive): ')

num\_of\_images = int(input('Number of images: '))

continue\_image\_no\_from = int(

input('Enter image no from where you want to continue (if new enter 0): '))

current\_gesture\_dir = f'{dataset\_dir}/{gesture}'

try:

labels = get\_labels()

label\_id = labels[gesture]

except Exception as e:

print('Make sure you pass in the correct parameters.')

quit()

# initialize weight for running average

aWeight = 0.5

# get the reference to the webcam

camera = cv2.VideoCapture(0)

# region of interest (ROI) coordinates

top, right, bottom, left = 10, 350, 225, 590

# initialize num of frames

num\_frames = 0

image\_num = continue\_image\_no\_from

start\_recording = False

# keep looping, until interrupted

while(True):

# get the current frame

(grabbed, frame) = camera.read()

if (grabbed == True):

# resize the frame

frame = imutils.resize(frame, width=700)

# flip the frame so that it is not the mirror view

frame = cv2.flip(frame, 1)

# clone the frame

clone = frame.copy()

# get the height and width of the frame

(height, width) = frame.shape[:2]

# get the ROI

roi = frame[top:bottom, right:left]

# convert the roi to grayscale and blur it

gray = cv2.cvtColor(roi, cv2.COLOR\_BGR2GRAY)

gray = cv2.GaussianBlur(gray, (7, 7), 0)

# to get the background, keep looking till a threshold is reached

# so that our running average model gets calibrated

if num\_frames < 30:

run\_avg(gray, aWeight)

print(num\_frames)

else:

# segment the hand region

hand = segment(gray)

# check whether hand region is segmented

if hand is not None:

# if yes, unpack the thresholded image and

# segmented region

(thresholded, segmented) = hand

# draw the segmented region and display the frame

cv2.drawContours(

clone, [segmented + (right, top)], -1, (0, 0, 255))

if start\_recording:

# Mention the directory in which you wanna store the images followed by the image name

# cv2.imwrite(current\_gesture\_dir + str(image\_num) + '.png', thresholded)

cv2.imwrite(

f"{current\_gesture\_dir}/{image\_num}\_l={label\_id}.png", thresholded)

image\_num += 1

cv2.imshow("Thresholded", thresholded)

# draw the segmented hand

cv2.rectangle(clone, (left, top), (right, bottom), (0, 255, 0), 2)

# increment the number of frames

num\_frames += 1

# showing number of images captured

cv2.putText(clone, f"Captured {image\_num} images",

(15, 500), cv2.FONT\_HERSHEY\_COMPLEX, 1, (0, 255, 0), 2)

# display the frame with segmented hand

cv2.imshow(f"Capturing for {gesture}", clone)

# observe the keypress by the user

keypress = cv2.waitKey(1) & 0xFF

# if the user pressed "q", then stop looping

if keypress == ord("q") or image\_num >= (num\_of\_images + continue\_image\_no\_from):

# free up memory

camera.release()

cv2.destroyAllWindows()

break

if keypress == ord("s"):

start\_recording = True

else:

print("[Warning!] Error input, Please check your camera or video")

break

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Create Dataset:**

import os

import shutil

from PIL import Image

# resizes all dataset image

dataset\_dir = 'Dataset'

def resizeImage(imageName):

basewidth = 100

img = Image.open(imageName)

wpercent = (basewidth/float(img.size[0]))

hsize = int((float(img.size[1])\*float(wpercent)))

img = img.resize((basewidth, hsize), Image.ANTIALIAS)

img.save(imageName)

gestures\_dirs = []

for dir in os.listdir(dataset\_dir):

gestures\_dirs.append(f"{dataset\_dir}/{dir.strip()}")

for loc in gestures\_dirs:

for image in os.listdir(loc):

resizeImage(f"{loc}/{image}")

**Train Classifier:**

import matplotlib.pyplot as plt

import numpy as np

import PIL

import tensorflow as tf

from pathlib import Path

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.models import Sequential

dataset\_dir = Path('C:/deva/Miniproject/automation-using-hand-gestures-master/Dataset')

image\_count = len(list(dataset\_dir.glob('\*/\*.png')))

print(image\_count)

batch\_size = 32

img\_height = 100

img\_width = 89

train\_ds = tf.keras.utils.image\_dataset\_from\_directory(

dataset\_dir,

validation\_split=0.2,

subset="training",

seed=123,

image\_size=(img\_height, img\_width),

batch\_size=batch\_size)

val\_ds = tf.keras.utils.image\_dataset\_from\_directory(

dataset\_dir,

validation\_split=0.2,

subset="validation",

seed=123,

image\_size=(img\_height, img\_width),

batch\_size=batch\_size)

class\_names = train\_ds.class\_names

print(class\_names)

AUTOTUNE = tf.data.AUTOTUNE

train\_ds = train\_ds.cache().shuffle(1000).prefetch(buffer\_size=AUTOTUNE)

val\_ds = val\_ds.cache().prefetch(buffer\_size=AUTOTUNE)

data\_augmentation = keras.Sequential(

[

layers.RandomFlip("horizontal",

input\_shape=(img\_height,

img\_width,

3)),

layers.RandomRotation(0.1),

layers.RandomZoom(0.1),

]

)

model = Sequential([

data\_augmentation,

layers.Rescaling(1./255),

layers.Conv2D(16, 3, padding='same', activation='relu'),

layers.MaxPooling2D(),

layers.Conv2D(32, 3, padding='same', activation='relu'),

layers.MaxPooling2D(),

layers.Conv2D(64, 3, padding='same', activation='relu'),

layers.MaxPooling2D(),

layers.Dropout(0.2),

layers.Flatten(),

layers.Reshape((12, -1)),

layers.LSTM(64),

layers.Dense(128, activation='relu'),

layers.Dense(num\_classes, name="outputs")

])

model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=['accuracy'])

model.summary()

epochs = 15

history = model.fit(

train\_ds,

validation\_data=val\_ds,

epochs=epochs

)

**Continuousgestureprediction:**

import tensorflow as tf

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Input, Dropout, Flatten, Dense,Conv2DTranspose,UpSampling2D

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

from keras.models import Sequential

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.models import Sequential

from tensorflow.keras.models import load\_model

from pathlib import Path

saved\_model\_path = 'C:/deva/Miniproject/automation-using-hand-gestures-master/TrainedModel/Gesture12RecognitionModelBest.keras'

loaded\_model = load\_model(saved\_model\_path)

#import tflearn

#from tflearn.layers.conv import conv\_2d, max\_pool\_2d

#from tflearn.layers.core import input\_data, dropout, fully\_connected

#from tflearn.layers.estimator import regression

from tensorflow.python.framework import ops

import numpy as np

from PIL import Image

import cv2

import imutils

import os

from sklearn.preprocessing import LabelEncoder

from PalmTracker import \*

# PUBLISHER SIDE

# PUBNUB

from pubnub.callbacks import SubscribeCallback

from pubnub.enums import PNStatusCategory

from pubnub.pnconfiguration import PNConfiguration

from pubnub.pubnub import PubNub

ENTRY = "GestureControl"

CHANNEL = "Detect"

KILL\_CONNECTION = "exit"

the\_update = None

pnconfig = PNConfiguration()

pnconfig.publish\_key = 'pub-c-75f1104f-f51c-4d31-9f65-a763a31e7dad'

pnconfig.subscribe\_key = 'sub-c-11c98f38-0c15-45a2-949e-f881da6b6a1f'

pnconfig.uuid = "serverUUID-PUB"

pubnub = PubNub(pnconfig)

# PUBNUB

checkpoint\_path = 'C:/deva/Miniproject/automation-using-hand-gestures-master/TrainedModel/Gesture12RecognitionModelBest.keras'

best\_checkpoint\_path = 'C:/deva/Miniproject/automation-using-hand-gestures-master/TrainedModel/Gesture12RecognitionModelBest.keras'

saved\_model\_path = 'C:/deva/Miniproject/automation-using-hand-gestures-master/TrainedModel/Gesture12RecognitionModelBest.keras'

n\_classes = len(os.listdir('C:/deva/Miniproject/automation-using-hand-gestures-master/Dataset'))

dataset\_dir = Path('C:/deva/Miniproject/automation-using-hand-gestures-master/Dataset')

#class\_names = train\_ds.class\_names

batch\_size = 32

img\_height = 100

img\_width = 89

train\_ds = tf.keras.utils.image\_dataset\_from\_directory(

dataset\_dir,

validation\_split=0.2,

subset="training",

seed=123,

image\_size=(img\_height, img\_width),

batch\_size=batch\_size)

class\_names = train\_ds.class\_names

def resizeImage(imageName):

basewidth = 100

img = Image.open(imageName)

wpercent = (basewidth/float(img.size[0]))

hsize = int((float(img.size[1])\*float(wpercent)))

img = img.resize((basewidth, hsize), Image.ANTIALIAS)

img.save(imageName)

def getPredictedClass():

image = cv2.imread('Temp.png')

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

resized\_img = cv2.resize(gray\_image, (100, 89))

rgb\_img = cv2.cvtColor(resized\_img, cv2.COLOR\_GRAY2RGB)

normalized\_img = rgb\_img / 255.0

input\_img = np.expand\_dims(normalized\_img, axis=0)

prediction = loaded\_model.predict([input\_img.reshape(1, 100, 89, 3)])

sum = 0

for i in range(n\_classes):

sum += prediction[0][i]

return np.argmax(prediction), (np.amax(prediction)/sum)

def get\_labels\_rev():

gestures = []

with open('C:/deva/Miniproject/automation-using-hand-gestures-master/preprocessing/gestures.txt', 'r') as f:

for line in f:

line = line.strip()

gestures.append(line)

gestures.sort()

gestures = np.array([gestures[i] for i in range(len(gestures))])

integer\_encoded = LabelEncoder().fit\_transform(gestures)

labels = {}

for i in range(len(gestures)):

labels[integer\_encoded[i]] = gestures[i]

# print(labels)

return labels

llabels = get\_labels\_rev()

func\_map = {

'Thumbs\_up': 'Red ON', # thumbs up

'Thumbs\_down': 'Red half ON', # thumbs down

'Fist': 'Red OFF', # fist

'Two': 'Green ON', # two

'Three': 'Green half ON', # three

'Four': 'Green OFF', # four

'OK': 'Get rain-value', # ok

'One': 'Fan ON', # one

'Stop': 'Fan OFF', # stop

'Direction\_left':'TV channel change',

'Direction\_right': 'TV channel change', # right

'Five-palm': 'Clean floor' # palm-five

}

def showStatistics(predictedClass, confidence):

textImage = np.zeros((250, 512, 3), np.uint8)

#className = labels.get(int(predictedClass), "Unknown")

cv2.putText(textImage, "Predicted class: " + str(predictedClass),

(5, 30),

cv2.FONT\_HERSHEY\_SIMPLEX,

1,

(255, 255, 255),

1)

cv2.putText(textImage, "Confidence: " + str(confidence ) + '%',

(5, 100),

cv2.FONT\_HERSHEY\_SIMPLEX,

1,

(255, 255, 255),

1)

cv2.putText(textImage, "Actuate: " + func\_map[str(predictedClass)],

(5, 170),

cv2.FONT\_HERSHEY\_SIMPLEX,

1,

(0, 255, 0),

1)

cv2.imshow("Statistics", textImage)

def main():

# initialize weight for running average

aWeight = 0.5

# get the reference to the webcam

camera = cv2.VideoCapture(0)

# region of interest (ROI) coordinates

top, right, bottom, left = 10, 350, 225, 590

# initialize num of frames

num\_frames = 0

start\_recording = False

# keep looping, until interrupted

while(True):

# get the current frame

(grabbed, frame) = camera.read()

# resize the frame

frame = imutils.resize(frame, width=700)

# flip the frame so that it is not the mirror view

frame = cv2.flip(frame, 1)

# clone the frame

clone = frame.copy()

# get the height and width of the frame

(height, width) = frame.shape[:2]

# get the ROI

roi = frame[top:bottom, right:left]

# convert the roi to grayscale and blur it

gray = cv2.cvtColor(roi, cv2.COLOR\_BGR2GRAY)

gray = cv2.GaussianBlur(gray, (7, 7), 0)

# to get the background, keep looking till a threshold is reached

# so that our running average model gets calibrated

if num\_frames < 30:

run\_avg(gray, aWeight)

print(num\_frames)

else:

# segment the hand region

hand = segment(gray)

# check whether hand region is segmented

if hand is not None:

# if yes, unpack the thresholded image and

# segmented region

(thresholded, segmented) = hand

# draw the segmented region and display the frame

cv2.drawContours(

clone, [segmented + (right, top)], -1, (0, 0, 255))

if start\_recording:

cv2.imwrite('Temp.png', thresholded)

resizeImage('Temp.png')

sunflower\_url = Path('C:/deva/Miniproject/automation-using-hand-gestures-master/Temp.png')

img\_height = 100

img\_width = 89

img = tf.keras.utils.load\_img(

sunflower\_url, target\_size=(img\_height, img\_width)

)

img\_array = tf.keras.utils.img\_to\_array(img)

img\_array = tf.expand\_dims(img\_array, 0) # Create a batch

predictions = loaded\_model.predict(img\_array)

score = tf.nn.softmax(predictions[0])

print(

"This image most likely belongs to {} with a {:.2f} percent confidence."

.format(class\_names[np.argmax(score)], 100 \* np.max(score))

)

showStatistics(class\_names[np.argmax(score)], 100 \* np.max(score))

# PUBNUB integration

the\_update = str(class\_names[np.argmax(score)])

the\_message = {"entry": ENTRY, "update": the\_update}

envelope = pubnub.publish().channel(CHANNEL).message(the\_message).sync()

if envelope.status.is\_error():

print("[PUBLISH: fail]")

print("error: {}".format(status.error))

else:

print("[PUBLISH: sent]")

print(f"Sent: {the\_update}")

# PUBNUB integration

cv2.imshow("Thresholded", thresholded)

# draw the segmented hand

cv2.rectangle(clone, (left, top), (right, bottom), (0, 255, 0), 2)

# increment the number of frames

num\_frames += 1

# display the frame with segmented hand

cv2.imshow("Video Feed", clone)

# observe the keypress by the user

keypress = cv2.waitKey(1) & 0xFF

# if the user pressed "q", then stop looping

if keypress == ord("q"):

the\_update = KILL\_CONNECTION

break

if keypress == ord("s"):

start\_recording = True

# Model defined

main()

**Actuator:**

from tkinter import PhotoImage

from pubnub.callbacks import SubscribeCallback

from pubnub.enums import PNStatusCategory

from pubnub.pnconfiguration import PNConfiguration

from pubnub.pubnub import PubNub

import os

import threading

import turtle

import random

# import pyfirmata # for hardware (arduino)

# SUBSCRIBER SIDE

# turtle

win\_x = 600

win\_y = 400

wn = turtle.Screen()

wn.bgcolor('white')

wn.title('Home')

wn.setup(width=win\_x+20, height=win\_y+20)

red = (-250, 170, 60, 60)

red\_state = 0

green = (-250, 50, 60, 60)

green\_state = 0

rain = (-250, -70, 60, 60)

rain\_state = False

fan = (-90, 100, 50)

fan\_state = False

stop\_thread = False

floor = (-80, -10, 300, 150)

floor\_filling = False

gif\_dir = 'C:/deva/Miniproject/automation-using-hand-gestures-master/gifs/'

gifs = os.listdir(gif\_dir)

tv = (150, 90)

channel\_count = 1

# pubnub

count = 0

msg\_threshold = 15

msg\_list = []

ENTRY = "GestureControl"

CHANNEL = "Detect"

KILL\_CONNECTION = "exit"

pnconfig = PNConfiguration()

pnconfig.publish\_key = 'pub-c-75f1104f-f51c-4d31-9f65-a763a31e7dad'

pnconfig.subscribe\_key = 'sub-c-11c98f38-0c15-45a2-949e-f881da6b6a1f'

pnconfig.uuid = "serverUUID-SUB"

pubnub = PubNub(pnconfig)

# HARDWARE

# arduino

# board = pyfirmata.Arduino("COM4") # for arduino board

# configuring pins for sensors

# lenPin = board.get\_pin('d:11:p')

# red\_led = board.get\_pin('d:5:p') # digital pin 6

# green\_led = board.get\_pin('d:6:p') # digital pin 7

# gas\_sen = board.get\_pin('a:0:i') # analog input 0

# rain\_sens = board.get\_pin('a:1:i') # analog input 1

# it = pyfirmata.util.Iterator(board)

# it.start()

def draw\_border(l, b):

'''draws border along the turtle window'''

border = turtle.Turtle()

border.speed(0)

border.penup()

border.pensize(4)

border.color('black')

border.setposition(-l//2, -b//2)

border.pendown()

border.hideturtle()

border.fd(l)

border.lt(90)

border.fd(b)

border.lt(90)

border.fd(l)

border.lt(90)

border.fd(b)

def draw\_sq(pos, color='white', win\_length=win\_x, win\_breadth=win\_y):

'''(x, y) correspond to the top left corners for the rectangle'''

x, y, length, breadth = pos

rect = turtle.Turtle()

rect.penup()

rect.pensize(2)

rect.speed(0)

rect.setposition(x, y)

rect.pendown()

rect.hideturtle()

rect.fillcolor(color)

rect.begin\_fill()

for \_ in range(4):

rect.fd(length)

rect.rt(90)

rect.end\_fill()

def draw\_fan(pos):

'''draws fan on the turtle window'''

x, y, r = pos

y -= r

circle = turtle.Turtle()

circle.hideturtle()

circle.speed(0)

circle.pensize(2)

circle.penup()

circle.setpos(x, y)

circle.pendown()

# circle.speed(5)

# circle.showturtle()

circle.circle(r)

def draw\_tv(pos, image=gif\_dir+'0.gif'):

'''draws a tv with images in it'''

x, y = pos

larger = PhotoImage(file=image).subsample(2, 2)

wn.addshape("larger", turtle.Shape("image", larger))

tv = turtle.Turtle("larger")

tv.speed(0)

tv.hideturtle()

tv.penup()

tv.setposition(x, y)

tv.stamp()

def turn\_fan():

'''simulates a turning fan'''

x, y, r = fan

y -= r

global fan\_state

global stop\_thread

if fan\_state is False:

fan\_state = True

circle = turtle.Turtle()

circle.hideturtle()

circle.speed(0)

circle.pensize(2)

circle.penup()

circle.setpos(x, y)

circle.pendown()

circle.showturtle()

while True:

if stop\_thread:

stop\_thread = False

break

circle.circle(r)

circle.speed(5)

def fill\_floor():

'''cleans the floor area on the simulation'''

global floor\_filling

if floor\_filling is False:

floor\_filling = True

x, y, w, h = floor

x += 15

y -= 15

w -= 30

step = 20

filler = turtle.Turtle()

filler.hideturtle()

filler.penup()

filler.setposition(x, y)

filler.pendown()

filler.pensize(25)

filler.pencolor('white')

for i in range(3):

filler.fd(w)

filler.setheading(270)

filler.fd(step)

filler.setheading(180)

filler.fd(w)

filler.setheading(270)

filler.fd(step)

filler.setheading(0)

filler.fd(w)

return

# triggering functions for actuations

def red\_on():

global red\_state

if red\_state == 0 or red\_state == 0.5:

red\_state = 1

# red\_led.write(0.999)

draw\_sq(red, 'red')

print("----------------------------->> Red ON")

return

return

def red\_half\_on():

global red\_state

if red\_state == 1 or red\_state == 0:

red\_state = 0.5

# red\_led.write(0.3)

draw\_sq(red, 'orange')

print("----------------------------->> Red half ON")

return

return

def red\_off():

global red\_state

if red\_state == 1 or red\_state == 0.5:

red\_state = 0

# red\_led.write(0.001)

draw\_sq(red, 'white')

print("----------------------------->> Red OFF")

return

return

def green\_on():

global green\_state

if green\_state == 0 or green\_state == 0.5:

green\_state = 1

# green\_led.write(0.999)

draw\_sq(green, 'green')

print("----------------------------->> Green ON")

return

return

def green\_half\_on():

global green\_state

if green\_state == 1 or green\_state == 0:

green\_state = 0.5

# green\_led.write(0.3)

draw\_sq(green, 'yellow')

print("----------------------------->> Green Half ON")

return

return

def green\_off():

global green\_state

if green\_state == 0.5 or green\_state == 1:

green\_state = 0

# green\_led.write(0.001)

draw\_sq(green, 'white')

print("----------------------------->> Green OFF")

return

return

def get\_rainval():

# rainval = 89.0678 #rain\_sens.read()

draw\_sq(rain, color='white')

num = random.randint(1, 100)

t = turtle.Turtle()

t.hideturtle()

t.speed(0)

t.penup()

t.setposition(rain[0], rain[1])

t.pendown()

t.fillcolor('red')

t.begin\_fill()

t.pensize(2)

num = (rain[2]/100) \* num

t.fd(num)

t.rt(90)

t.fd(rain[2])

t.rt(90)

t.fd(num)

t.rt(90)

t.fd(rain[2])

t.end\_fill()

print('----------------------------->> Rainval check')

return

def fan\_on():

f = threading.Thread(target=turn\_fan)

f.start()

print("----------------------------->> Fan ON")

return

def fan\_off():

global stop\_thread

global fan\_state

stop\_thread = True

fan\_state = False

print("----------------------------->> Fan OFF")

return

def change\_channel():

global channel\_count

if channel\_count >= len(gifs):

channel\_count = 0

draw\_tv(tv, gif\_dir + gifs[channel\_count])

print("----------------------------->> Channel change on TV")

channel\_count += 1

return

def clean\_floor():

c = threading.Thread(target=fill\_floor)

c.start()

print("----------------------------->> Floor cleaning")

return

# 2 - 12

func\_map = {

'Thumbs\_up': red\_on, # thumbs up

'Thumbs\_down': red\_half\_on, # thumbs down

'Fist': red\_off, # fist

'Two': green\_on, # two

'Three': green\_half\_on, # three

'Four': green\_off, # four

'OK': get\_rainval, # ok

'One': fan\_on, # one

'Stop': fan\_off, # stop

'Direction\_left': change\_channel, # right

'Direction\_right': change\_channel,

'Five-palm': clean\_floor # palm-five

}

def most\_frequent(List):

return max(set(List), key=List.count)

class MySubscribeCallback(SubscribeCallback):

def presence(self, pubnub, event):

print("[PRESENCE: {}]".format(event.event))

print("uuid: {}, channel: {}".format(event.uuid, event.channel))

def status(self, pubnub, event):

if event.category == PNStatusCategory.PNConnectedCategory:

print("[STATUS: PNConnectedCategory]")

print("connected to channels: {}".format(event.affected\_channels))

def message(self, pubnub, event):

global count

global msg\_list

# print(f"Count = {count}")

print("[MESSAGE received]")

if event.message["update"] == KILL\_CONNECTION:

print("The publisher has ended the session.")

os.\_exit(0)

else:

recvd\_msg = str(event.message["update"])

print(f"Message = {recvd\_msg}")

msg\_list.append(recvd\_msg)

print(msg\_list)

if len(msg\_list) >= msg\_threshold:

actuate = most\_frequent(msg\_list)

msg\_list.clear()

if actuate in func\_map.keys():

func\_map[actuate]()

count += 1

def setup():

# text and other setup for simulation

# red led

turtle.speed(0)

turtle.hideturtle()

turtle.penup()

turtle.setpos(red[0]+2, red[1]-90)

turtle.write('Red light', font=('Verdana', 10, 'normal'))

# green led

turtle.speed(0)

turtle.hideturtle()

turtle.penup()

turtle.setpos(green[0]-5, green[1]-90)

turtle.write('Green light', font=('Verdana', 10, 'normal'))

# rain sensor

turtle.speed(0)

turtle.hideturtle()

turtle.penup()

turtle.setpos(rain[0]-5, rain[1]-90)

turtle.write('Rain Sensor', font=('Verdana', 10, 'normal'))

# fan

turtle.speed(0)

turtle.hideturtle()

turtle.penup()

turtle.setpos(fan[0]-10, fan[1]+60)

turtle.write('Fan', font=('Verdana', 10, 'normal'))

# tv

tv\_border = turtle.Turtle()

tv\_border.speed(0)

tv\_border.hideturtle()

tv\_border.pensize(2)

tv\_border.penup()

tv\_border.setposition(tv[0]-98, 160)

tv\_border.pendown()

tv\_border.fd(197)

tv\_border.rt(90)

tv\_border.fd(140)

tv\_border.rt(90)

tv\_border.fd(197)

tv\_border.rt(90)

tv\_border.fd(140)

turtle.speed(0)

turtle.hideturtle()

turtle.penup()

turtle.setpos(tv[0]-8, tv[1]+80)

turtle.write('TV', font=('Verdana', 10, 'normal'))

# floor

f = turtle.Turtle()

f.hideturtle()

f.speed(0)

f.penup()

f.setposition(floor[0], floor[1])

f.pendown()

f.pensize(2)

f.fillcolor('grey')

f.begin\_fill()

f.fd(floor[2])

f.rt(90)

f.fd(floor[3])

f.rt(90)

f.fd(floor[2])

f.rt(90)

f.fd(floor[3])

f.end\_fill()

turtle.speed(0)

turtle.hideturtle()

turtle.penup()

turtle.setpos(floor[0]+floor[2]//2 - 15, floor[1]-floor[3]-20)

turtle.write('Floor', font=('Verdana', 10, 'normal'))

# main

draw\_border(win\_x, win\_y)

draw\_sq(red, color='white')

draw\_sq(green, color='white')

draw\_sq(rain, color='white')

draw\_fan(fan)

draw\_tv(tv)

setup()

pubnub.add\_listener(MySubscribeCallback())

pubnub.subscribe().channels(CHANNEL).with\_presence().execute()

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\* Waiting for updates to about {}... \*".format(ENTRY))

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

turtle.done()

# REFERENCES

https://www.researchgate.net/publication/275534245\_Hand\_Gesture\_Recognition\_and\_Interface\_via\_a\_Depth\_Imaging\_Sensor\_for\_Smart\_Home\_Appliances

* J. Katti, A. Kulkarni, A. Pachange, A. Jadhav and P. Nikam, "Contactless Elevator Based on Hand Gestures During Covid 19 Like Pandemics," 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS), 2021, pp. 672-676, The number of doi: 10.1109/ICACCS51430.2021.9441827.
* H. Chung, Y. Chung and W. Tsai, "An Efficient Hand Gesture Recognition System Based on Deep CNN," 2019 IEEE International Conference on Industrial Technology (ICIT), Melbourne, Australia, 2019, pp. 853-858, doi: 10.1109/ICIT.2019.8755038.
* P. N. Arathi, S. Arthika, S. Ponmithra, K. Srinivasan and V. Rukkumani, "Gesture based home automation system," 2017 International Conference on Nextgen Electronic Technologies: Silicon to Software (ICNETS2), 2017, pp. 198-201, doi: 10.1109/ICNETS2.2017.8067929.