DIPARTIMENTO
DI INFORMATICA
SAPIENZA
UNIVERSITÀ DI ROMA

Hand Gesture Recognition and Mouse Control System

Institution: Sapienza University of Rome

Course Name: Applied Computer Science and Artificial Intelligence

Authors:

Bekka Ayele Debebe (2111636)

Efekan Menemencioglu (2109122)

Date: 04/06/2025

Abstract

PURPOSE: To build a real-time mouse control interface using a webcam for hand tracking and gesture recognition.

METHODS: Implemented by mainly using the MediaPipe python library to work on hand recognition and then utilizing a model trained on an ASL dataset by the K-Nearest Neighbor algorithm (from Scikit-learn) along with the PyAutoGUI library to simulate various mouse and windows' shortcut actions.

RESULTS: We were able to make a functioning mouse controlled by input from the webcam along with a few intuitive shortcuts deployed through gesture recognition and also added a GUI for easy configuration.

Introduction

In recent years, natural human-computer interaction (HCI) has become a significant area of research in computer science, especially with the rise of touchless technologies and intelligent interfaces. Among these, gesture recognition has emerged as a powerful method for enabling more intuitive and accessible interaction between humans and digital devices.

This project focuses on recognizing gestures using machine learning and integrating this capability into a real-time hand gesture-controlled mouse system. The goal is to bridge gesture recognition with practical control over a graphical interface, creating an assistive tool that enhances usability and accessibility.

The system is developed in two major components. The first involves training a K-Nearest Neighbors (KNN) classifier on a labeled dataset of ASL hand gestures. This model is then used in the second component, a real-time application that uses a webcam and the MediaPipe framework to detect hand landmarks, recognize gestures, and trigger corresponding mouse or system actions.

Additionally, the system includes a graphical user interface (GUI) to allow dynamic configuration of gesture functionality and pointer sensitivity. This project demonstrates the integration of machine learning, computer vision, and GUI development to build a responsive, user-configurable gesture recognition tool with practical applications in daily computing environments.

Methods

The development of this project involved two main phases: training a machine learning model to classify hand gestures, and implementing a real-time gesture-controlled mouse interface.

1. Dataset Preparation and Preprocessing

The ASL dataset used consists of numerical gesture samples, where each gesture is represented by 21 hand landmarks (each with x, y, and z coordinates). The CSV files

containing each gesture's landmarks was merged into a single dataset. The dataset was then loaded using pandas, and the features (X) and labels (y) were extracted by separating the landmark coordinates from the gesture class labels.

The data was then split into training and testing sets using the train_test_split() function from scikit-learn, with 60% of the data used for training and 40% for testing. No additional preprocessing was required as the features were already normalized landmark coordinates.

2. Model Training and Evaluation

A K-Nearest Neighbors (KNN) classifier was selected for this task due to its simplicity and effectiveness for multi-class classification problems. The model was trained using scikit-learn's KNeighborsClassifier, with the number of neighbors (k) set to 70 after basic tuning and testing.

After training, the model's performance was evaluated on the test set using:

- Accuracy score
- Classification report
- Confusion matrix(visualised using Seaborn)

The trained model was saved to disk using joblib for later use in the real-time interface.

3. Real-Time Gesture Recognition System

The gesture recognition system was implemented using OpenCV and MediaPipe.

3.1 Hand Tracking with MediaPipe

MediaPipe Hands was used for real-time hand detection and tracking via the webcam. For each detected hand, the 21 landmarks were extracted and converted into a feature vector matching the format used during model training.

3.2 Gesture Classification

The extracted landmark vector was passed into the saved KNN model to predict the corresponding ASL gesture. Predictions were made continuously on each video frame to ensure responsiveness.

4. Mouse Control and Gesture Actions

Mouse movement was controlled using the middle finger's position, scaled to match screen resolution. To improve stability, a smoothing algorithm was applied using a moving average of the cursor coordinates.

Three specific gestures were programmed to trigger system actions using PyAutoGUI.:

- Pinch Gesture (thumb and index finger close): Simulates a left mouse click. (Implemented using landmark distances, not the gesture recognition model.)

- Peace Gesture (index and middle finger up and apart (2)): Triggers the “Show Desktop” shortcut (Win + D).
- Point Gesture (only index finger up (1)): Triggers the ESC key.

To avoid false triggers, the system required the gesture to be detected consistently over a few frames before executing the corresponding command.

5. GUI Configuration

A lightweight GUI was created using Tkinter to allow users to adjust cursor speed and enable or disable specific gestures (Peace, Pinch, Point)

This GUI ran in a separate thread to ensure it did not block the main gesture recognition loop.

Results

1. Model Performance

The K-Nearest Neighbors (KNN) classifier trained to recognize ASL number gestures showed promising performance on the test dataset. The classifier was evaluated using several standard metrics:

- Accuracy Score: The model achieved an accuracy of 99% on the test set.
- Classification Report: The report showed high precision and recall for most gesture classes, indicating that the model can reliably distinguish between different ASL numbers. *(Figure 1)*
- Confusion Matrix: A confusion matrix was generated to visualize the model’s prediction errors. Most predictions were correctly classified along the diagonal, with a few misclassifications occurring between visually similar gestures. *(Figure 2)*

2. Real-Time Gesture Recognition System

Once integrated into the live system, the trained model successfully enabled real-time gesture recognition using webcam input. The system was able to detect hand landmarks continuously and classify gestures with minimal delay.

The following functionalities were tested and worked reliably:

- Pinch Gesture: Consistently triggered left mouse clicks when the thumb and index finger were brought close together.
- Peace Gesture: Correctly recognized and used to minimize or restore all windows (Win + D).
- Point Gesture: Used to simulate the ESC key, helpful for exiting full-screen applications and also used to stop the program.

Smooth and natural mouse control was achieved by tracking the middle finger, with movement smoothing reducing jitter and enhancing accuracy.

3. GUI Configuration Feedback

The Tkinter-based GUI allowed real-time customization of cursor speed and gesture functionality. Users could easily enable or disable specific gestures and adjust movement sensitivity, which improved usability and gave users greater control over the system.

4. Overall System Responsiveness

The system operated in real time, with minimal noticeable lag between gesture performance and system response.

Performance was mostly stable, though gesture recognition accuracy was occasionally affected by poor lighting or bad camera quality.

Discussion

This project demonstrates the effective use of machine learning and computer vision techniques to build a real-time, gesture-based human-computer interaction system. The combination of a trained K-Nearest Neighbors (KNN) model with real-time hand tracking via MediaPipe enabled a functional and responsive interface for controlling mouse actions and triggering system commands using hand gestures.

One of the key advantages of using machine learning for gesture recognition lies in its adaptability and scalability. The flexibility of these models allows for easy expansion. New gestures can be added with additional training data, and the system can be adapted to different users or use cases without rewriting core logic.

However, there are a few limitations to the system. Gesture miscalculation can occur and accidental triggers are possible due to only having the frame counters as checks for actions to occur. Additionally, model performance is inherently limited by the size and diversity of the training dataset.

With a larger and more diverse dataset, improved training techniques, and more robust gesture validation techniques, the system could become significantly more accurate and reliable. These enhancements would position this approach as a strong candidate for future applications in natural and touchless human-computer interaction.

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	211
1	1.00	1.00	1.00	188
2	0.97	1.00	0.98	203
3	1.00	0.98	0.99	209
4	0.98	1.00	0.99	201
5	0.98	0.99	0.99	195
6	1.00	0.96	0.98	192
7	1.00	0.99	0.99	199
8	1.00	1.00	1.00	206
9	1.00	1.00	1.00	196
accuracy			0.99	2000
macro avg	0.99	0.99	0.99	2000
weighted avg	0.99	0.99	0.99	2000

Figure 1. Classification Report

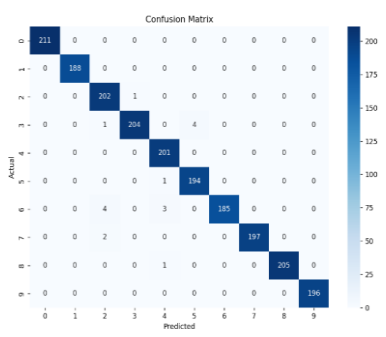

Figure 2. Confusion Matrix

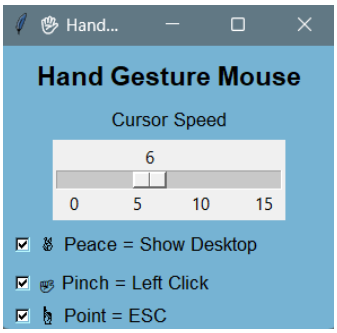

Figure 3. GUI

References

1. ***Scikit-learn:*** *Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.*
2. ***Google MediaPipe Framework****: https://mediapipe.dev*
3. ***OpenCV Library****: https://opencv.org*
4. ***PyAutoGUI Documentation****: https://pyautogui.readthedocs.io*
5. ***Tkinter:*** *Lundh, F. (1999). An introduction to tkinter.*
6. ***Joblib Library****: https://joblib.readthedocs.io*
7. ***NumPy:*** *Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2*
8. ***Matplotlib:*** *J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007*
9. ***Seaborn:*** *Waskom, M. L., (2021). seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021, https://doi.org/10.21105/joss.03021*
10. ***Dataset Source****: Rayeed, S. M. (2021, August 9). American Sign Language Digit Dataset. Retrieved 2025, May 25 from: https://www.kaggle.com/datasets/rayeed045/american-sign-language-digit-dataset*