Junghun Byun: 301605293
Ines Machinho Rodrigues: 301426807
Simran Vig: 301554993

CMPT 310 Final Project Report:

Hand Gesture Controlled Space Invaders

By: Junghun Byun, Ines Machinho Rodrigues, Simran Vig

Introduction:

This report explains, analyzes, and discusses our final project: Hand Gesture Controlled Space Invaders. This project uses AI to detect certain hand gestures, which determine whether the player's spaceship is shooting and/or moving. GitHub was used to collaborate on the files necessary for this project. To access the project's repository, please follow this link:

https://github.com/HoonieB/310_project/tree/feature/game-interface

The dataset used to train our AI model was created by hand. To see how this was accomplished, please see our How-To Guide on how to build a Hand Gesture Detection Dataset.

All game assets (sounds, sprites, and music) are copyright free.

System Explanation:

Goal: The goal of this system is to revolutionize the classic arcade game, Space Invaders, by making it more immersive via hand gesture controls. AI should be used and trained to recognize two different hand gestures: a gun, and a fist. The exact gestures used to train the model are as follows:

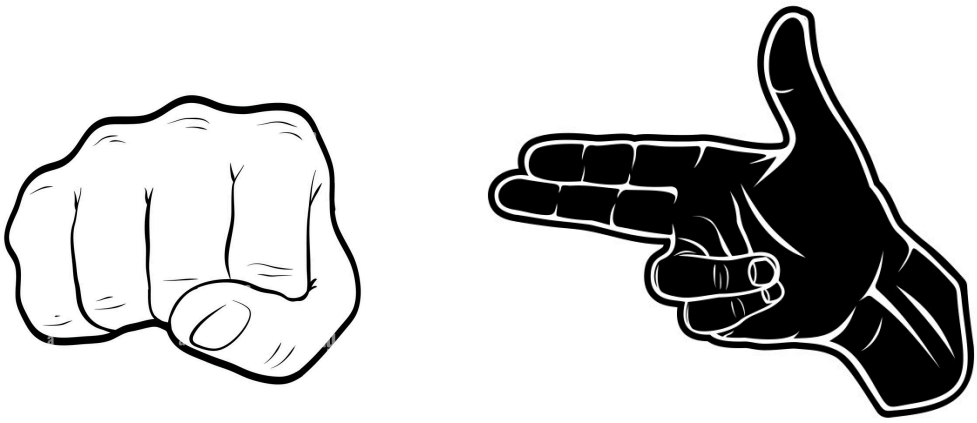

Figure 1: The recognized hand gestures for gun and fist in our system

Junghun Byun: 301605293
Ines Machinho Rodrigues: 301426807
Simran Vig: 301554993

When the player makes a fist hand gesture, the spaceship should not shoot any bullets. When the player makes a gun hand gesture, the spaceship should shoot bullets. When either gesture is made, the spaceship should move left or right in accordance with the player's hand in front of the camera.

System Pipeline:

1. **Data Input:** Live video feed from the webcam is captured by OpenCV. This video stream is processed frame-by-frame, and the landmarks of the user's hand are extracted with MediaPipe.

2. **Pre-processing:** MediaPipe extracts the user's hand into 21 (x, y, z) tuples as landmarks. These coordinates are then flattened into a 63-coordinate feature vector, which is then passed to the trained KNN model.

3. **AI Model:** The model used in this system is a K-Nearest Neighbours Classifier, which was implemented from scratch (see knn_model.py). This model computes the Euclidean distance between the input gesture and all training samples (see dataset.csv). It then selects the top K (we used K = 5) nearest samples, and uses majority voting to classify the gesture as either a "fist" or a "gun."

4. **Output:** While the predicted gesture is "gun," the spaceship fires bullets. While the predicted gesture is "fist," the spaceship does not fire bullets. Regardless of the gesture, the horizontal position of the hand (specifically, the position of the knuckles), is used to update the x-coordinate of the spaceship on screen.

These steps are repeated at each frame of the video feed, until the player either loses the game (ie, their spaceship runs out of lives), or they exit the game/program.

Junghun Byun: 301605293
Ines Machinho Rodrigues: 301426807
Simran Vig: 301554993

Flowchart:

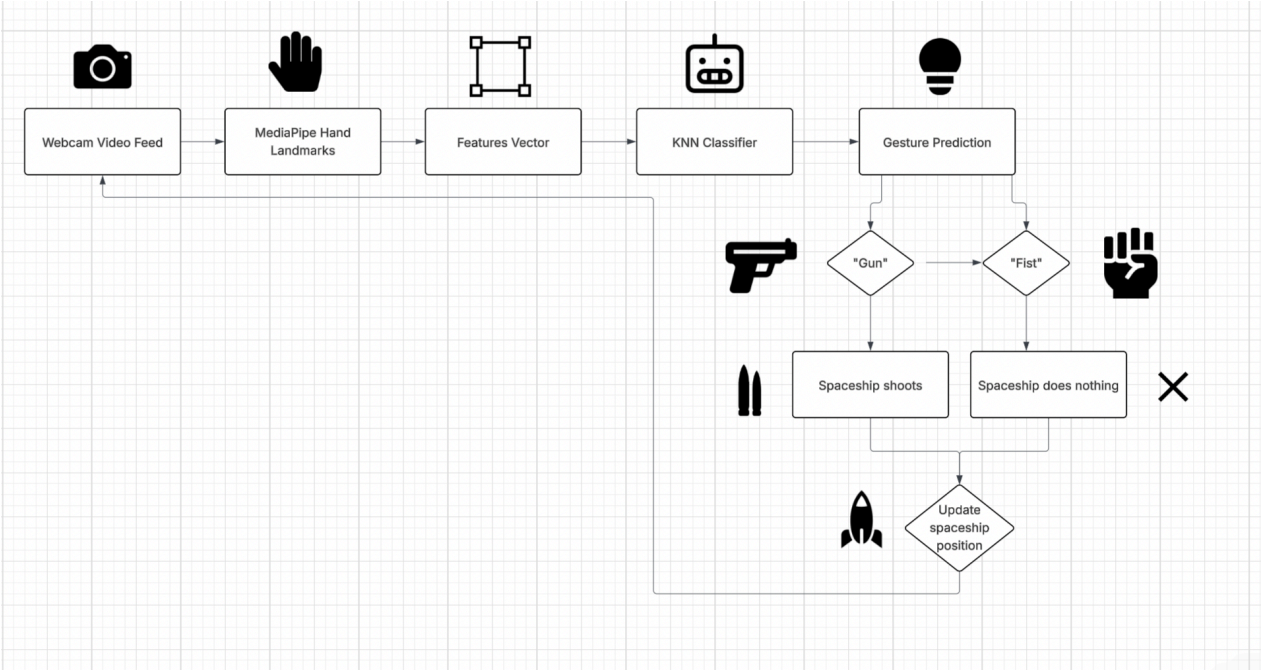

Figure 2: Flowchart of the system's pipeline

Evaluation Metrics: Using our dataset (dataset.csv), our KNN model with K = 5 has the following results:

```
GUN:
Avg Precision:  0.966904156575961

Avg Recalls:  0.9262756831637295

Avg Specificity:  0.9636043143494554

FIST:
Avg Precision:  0.9188712810002995

Avg Recalls:  0.9636043143494554

Avg Specificity:  0.9262756831637295

Overall Avg Accuracy:  0.9436324887520721
```

Figure 3: Evaluation metrics of the KNN model

These results are given in the main.py file.

The results from the evaluation metrics show that while the KNN model has a high overall accuracy of approximately 94%, there are differences in the precision, recall, and specificity of the “gun” and “fist” hand gestures.

The “gun” gesture has a high precision of around 97%, which indicates that there are few false positives - ie, the system rarely mistakes “fist” for “gun.” However, the “fist” gesture has a lower precision of around 92%, so we expect the system to mistake “gun” for “fist” more often than vice versa.

The “gun” gesture has a recall of around 92%, which means that the system is fairly good at detecting all true “gun” gestures. However, the “fist” recall is higher at around 96%, so the system is better at detecting all true “fist” gestures.

The “gun” gesture has a specificity of around 96%, which indicates that the system is good at detecting non-gun gestures (ie, “fist” gestures) as not being “gun” gestures. The system is slightly worse at doing so for the “fist” gesture with a specificity of around 94%.

The overall accuracy, for both gestures, is approximately 94%. This means that 94%, the KNN model accurately predicts the given hand gesture, making it a fairly strong model.

Limitations:

The system’s main limitation is that it sometimes mistakes a “fist” for a “gun.” To mitigate this, we added temporal smoothing to the main game loop (game_interface.py). This adds another round of majority voting to the predicted gesture. It records 7 of the model’s predictions for the given gesture, and then only outputs the gesture which has at least 4 out of 7 predictions.

This temporal smoothing not only helped with the false positives for the “gun” gesture, but with misclassifications of gestures overall.

Junghun Byun: 301605293
Ines Machinho Rodrigues: 301426807
Simran Vig: 301554993

Feature Table:

Description	Platform	Completeness	Code	Authors	Notes
Hand Landmark Extraction	Local	5	Python	Simran	Extracted 21 key points from each the hand's coordinates (x,y,z) to be fed into the AI model
KNN classifier	Local	4	Python	Simran	The AI model, built by our original KNN algorithm, predicts hand gestures with 94% accuracy; still room to achieve higher accuracy
Data set creation	Local	4	Python	Simran, Junghun, Ines	Recorded our own hands to create the dataset; less robust dataset than desired since we were manually recording our own hands
Webcam Video Feed	Local	5	Python	Simran, Ines	Captures live hand motion and sends it as input to the AI model for prediction
Game Interface	Local	4	Python	Junghun, Ines	Game interface and game logic; game difficulty adjustment isn't dynamic enough

Junghun Byun: 301605293
Ines Machinho Rodrigues: 301426807
Simran Vig: 301554993

External Tools and Libraries:

1. Scikit-learn
    a. Used to build the KNN-based AI model and evaluate its performance
2. OpenCV
    a. Used to capture the live video feed from the webcam
3. MediaPipe
    a. Used to detect and extract 21 key coordinates from the hands
4. Pygame
    a. Used to develop the game interface and handle core game logic
5. Custom Dataset
    a. We all recorded our own hands for dataset

Dependencies and Installation

- This project is designed for Windows, MacOS, and Linux
- This project requires to be run on Python 3.10 or below to be compatible with Scikit-learn, OpenCV, and MediaPipe
- Before try running the game, the external libraries (Scikit-learn, OpenCV, Mediapipe, and Pygame) must be installed on your local machine
    - In order to install these libraries, you will need a package manager like pip
    - E.g., “pip install pygame” to install pygame, follow the same format “pip install *library name*” to install other libraries

References

1. The following GIT repository was used to see a working example of how MediaPipe is used to track hands when we began the project: https://github.com/Sousannah/hand-tracking-using-mediapipe/blob/main/hand_tracking.py

2. The images in **Figure 1** are taken from PowerPoint clip-art

3. **Figure 2** was created by Simran Vig

4. **Figure 3** was taken after being generated from dataset.py in the submission zip file.