CP468 Artificial Intelligence Final Project: Generating Human-Like Mouse Trajectories in a 3D Environment

Bassil Ahmad

200450090

Introduction

There has been a heavy focus on having deep learning agents perform better than humans in behavioural tasks, however, an interesting area is in having deep learning agents explicitly behave like humans. There are many agents which perform as well as humans, or have the goal of performing at least as well as humans, but it is rare that their true goal is to behave as similar to a human as possible. This goal could be considered a variation of the Turing Test. The most well-known example of the Turing Test is related to NLP, but this can be generalized to any situation where an agent has the ability to act like a human.

With this idea of a generalized Turing Test in mind, this paper will focus on human movement. The goal of this paper is to create an agent which can mimic how a human moves their mouse (mouse trajectory) while playing 3D video games, such as Quake or Valorant. Such an agent would be highly valuable in both gaming and robotics. The agent could detect cheaters in gaming that are using aiming assistance software, or it could be used to train a robot to move and look around like a human, allowing for more realistic human-AI interactions.

In an attempt to accomplish the goal outline above, a small game was created in Unreal Engine 5 to collect data, allowing for easy data collection. The data that was collected was then cleaned and used to train a convolutional neural network autoencoder, which would generate human-like mouse trajectories from an input vector. The following subsections will go into details of the construction of the data collector and autoencoder, and then discuss the results and future potential.

Project Description

Overview

The goal of the project is to create an artificial intelligence which can construct human-like mouse trajectories which are as indistinguishable from real human mouse trajectories as possible. The environment will be in a 3D video game, where the human can control where the player looks by moving the mouse, and the goal of the game will be to aim at and click on targets that continuously spawn in front of the player. The AI will learn from the data extracted from humans playing this game. The next section will explain what data was collected and the process to attain it.

Data Collection

To collect data, a 3D video game was created (Virk, B, 2022) [2] using the Unreal Engine 5 game engine, which is a very popular, free, open source game engine. The game that was created allows the player to control where they are looking in-game by moving the mouse. A target will spawn in front of the player, where the goal will be to aim at the center of the screen at the target and click on it. After they click, a new target will spawn, and the process will repeat for 5 minutes. As the game is played, the following is recorded in a CSV file at a rate of 60 times per second: current time, current rotation of the player, location of the current target in spherical coordinates relative to the player, whether the player has clicked or not, and whether they hit the target if they clicked.

The dataset consists of 7 of the sessions described above, which adds up to 35 minutes of mouse trajectory data. There was total of 3453 mouse trajectories. The dataset is very small due to time constraints. The data was collected from participants who play 3D video games, such as CSGO and Valorant, on a regular or semi-regular basis. All participants were 20 years old.

Data Analysis

A single mouse trajectory is defined as the "path" taken by the player to aim at the given target. The dataset consists of many of these individual mouse trajectories, each with their own target. A mouse trajectory is composed of a sequence of player rotations that occur over time. This will be represented as: $T = (\{p_0,\ y_0\},\{p_1,\ y_1\}, \dots ,\ \{p_{n-1},\ y_{n-1}\},\ \{p_n,\ y_n\})$. Where p and y are the pitch and yaw respectively. As mentioned before, the player rotations was recorded at 60 Hz, so every rotation, or element, in a mouse trajectory will be approximately one 60th of a second apart. So, every mouse trajectory consists of a sequence of a player rotations whose length is defined by how long the player takes to aim at the target, and the location of the center point of the target in spherical coordinates. The general trend that is expected to be seen is that the sequence will start at some point relatively far from the target, and will approach the target location towards the end of the sequence. This can be seen in Fig. 1.

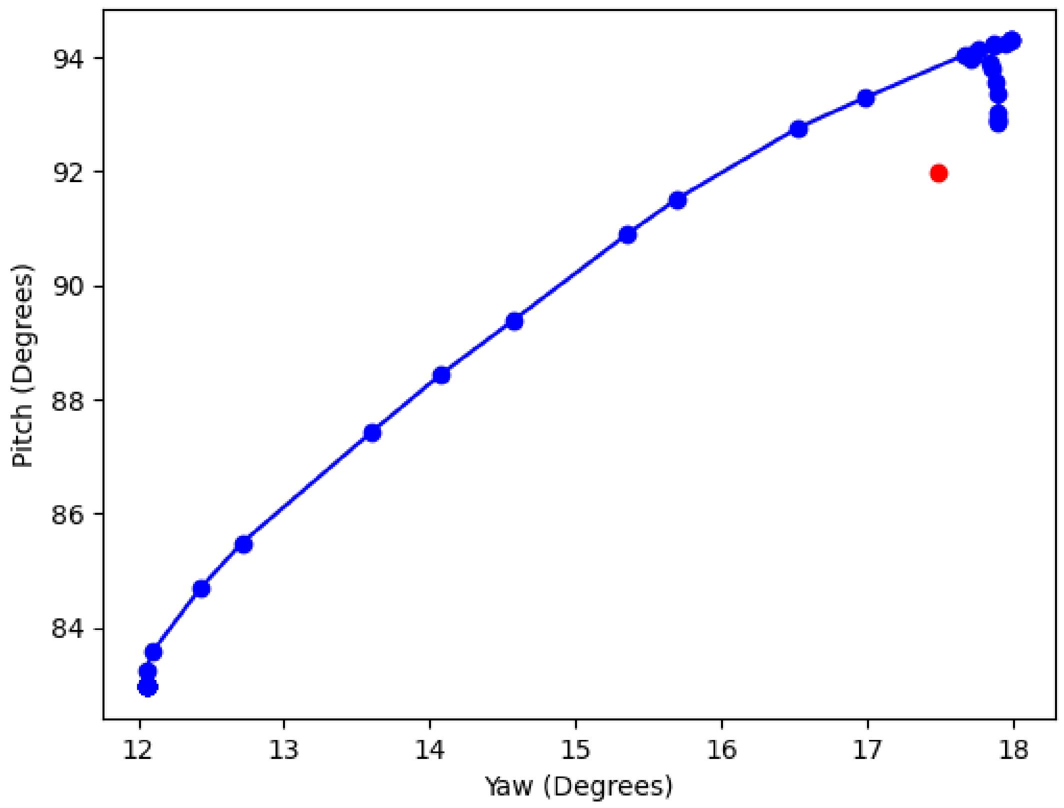

Fig.1: A mouse trajectory sequence (blue) and the corresponding target (red). Every point represents a position in spherical coordinates without the radius. Every point is approx. 16.66 milliseconds apart in time. The path starts at the bottom right, and moves towards the target. It can be seen that there is a slight overshoot towards the end, followed by a correction downwards.

From Fig.1 it can be seen that the data is particular to the situation at the time. The first point in Fig.1 is not at the origin angle (0, 0), it is at the angle that the player was looking at the beginning of the trajectory. To make the data more general, every mouse trajectory should be normalized by subtracting the value of the first point from every point, including the target. This will normalize all trajectories to the origin angle, allowing any trajectory to represent a movement from the origin, and thus making the data more general to any movement. Fig.2 shows this transformation on Fig.1.

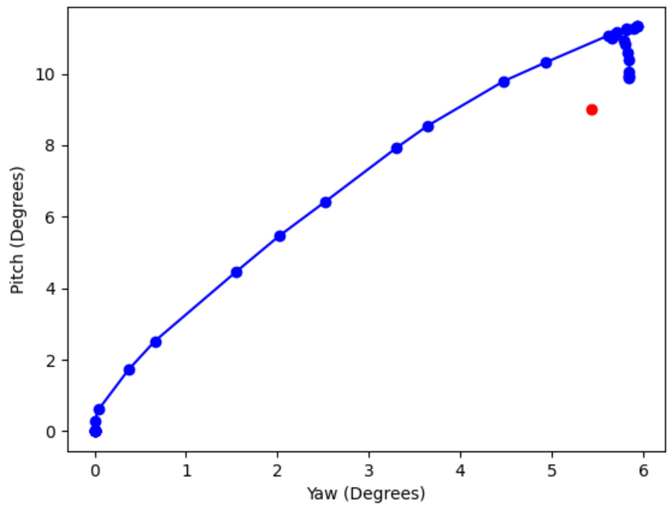

Fig. 1: Normalization of mouse trajectory from Fig.1. The path itself does not change, but the path now starts from the origin angle (0, 0).

While the normalized trajectory data is useful for visualization of the path, it is not the best representation of how the angle of rotation changes throughout the sequence. The change in angle with respect to time is best for this. Fig.3a and Fig.3b show change in pitch (dp) and yaw (dy) with respect to time.

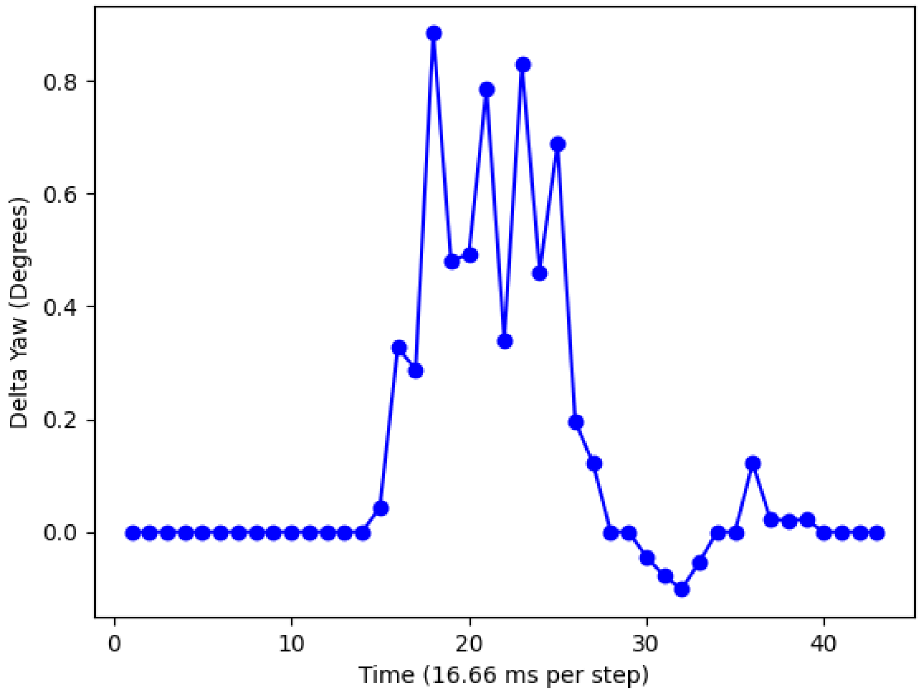

Fig.2a: Delta Yaw with respect to time of Fig.2

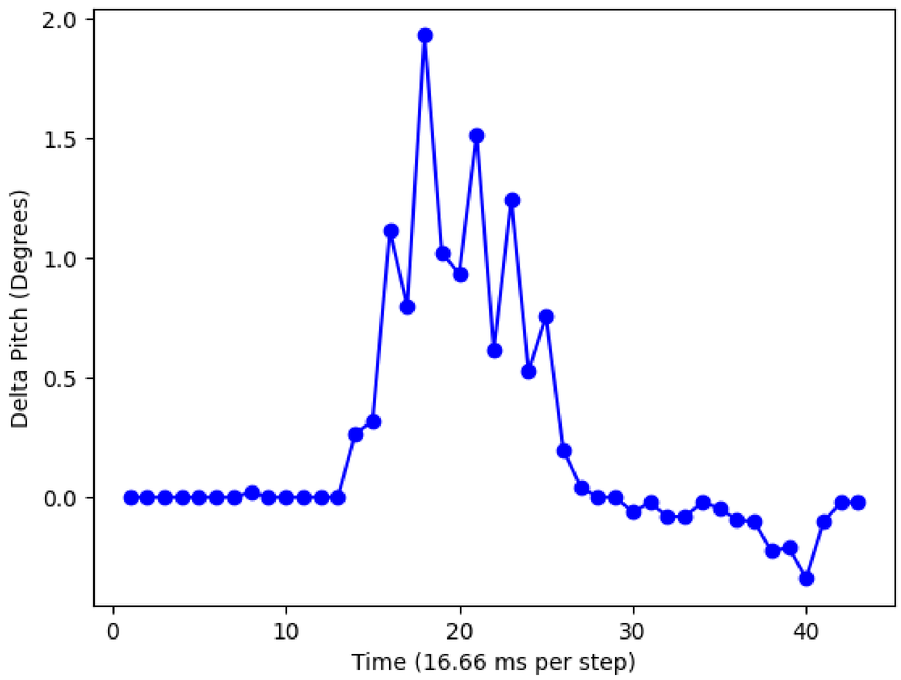

Figure 3b: Delta Pitch with respect to time of Fig.2

Fig.3a and 3b show how the angles change over time. From analyzing other trajectories, it is apparent that every trajectory starts slow, rises to a peak, and then slows down again, potentially creating a smaller peak in the opposite direction of the original peak, to correct an overshoot.

Feature Selection

From the data analysis above, the most important features of a mouse trajectory are delta yaw, delta pitch, and time. The original pitch and yaw are not needed because they can be attained by adding the deltas. As such, the most important features to feed to a machine learning algorithm are the target location, the angle deltas, and time.

Methodology

The first idea that was thought of was to use reinforcement learning by having the AI generate a path for a target, and then punish or reward the AI based on how similar the path is to the original path made by a human. The problem with this idea is that the AI will likely over-fit itself to one batch of data, and be ineffective on other batches of data.

Instead, the methodology of this project was to train an autoencoder based on a convolutional neural network. The purpose of an autoencoder is to essentially replicate the input on the output. The neural network essentially learns how to copy the output. This may seem useless at first, however, the autoencoder is rarely able to perfectly replicate the input and tends to focus on specific features due to the dimensionality reduction, resulting in an approximation of the input. This is very useful for the goals of this project for the following reason: The shortest path from one point to another is a straight line, and humans tend to approximate this line, with some imperfections. An autoencoder does this exactly.

To train the autoencoder as described above, the input will need to be the path that the autoencoder is attempting to approximate. The easiest solution is to divide the straight line between the origin angle and the target angle into n distinct and equal points. Fig.4 shows what this would look like for the target from Fig.1 if n = 64. Fig.4 was generated from the original target point from Fig.1, and is an example of a mouse trajectory that would be used as input to the autoencoder. The problem with this solution is that the output that the autoencoder is compared against must have n points as well. Therefore, the size of every mouse trajectory was limited to n = 64 movements. 64 was chosen because 99.3% (3428 of 3453) of all mouse trajectories had a size that fell at or below this value. Any trajectories which were larger were cut off at 64, and any which were less than 64 were padded with pairs of 0s - (0, 0).

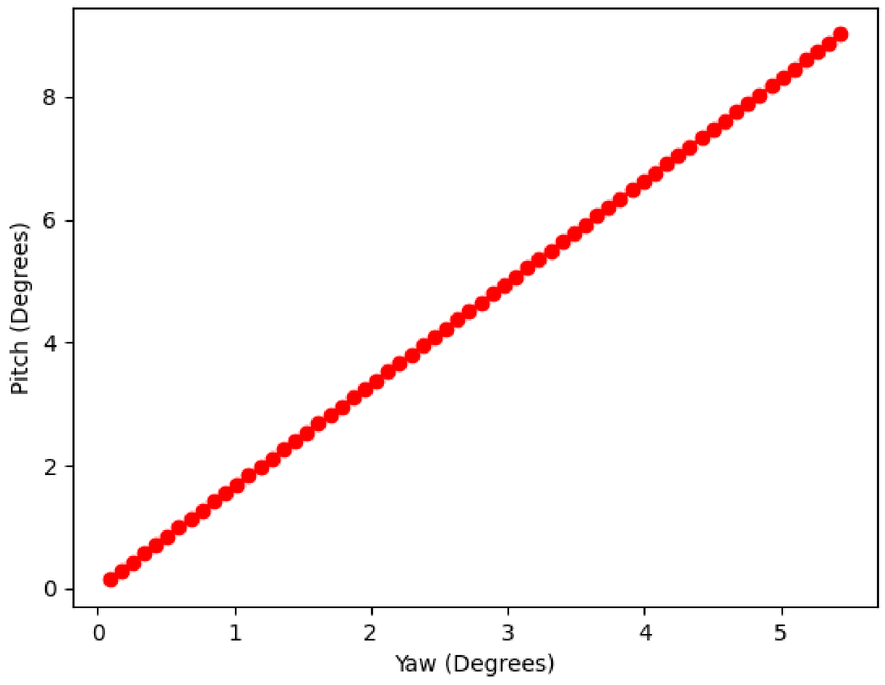

Fig.3: Evenly divided, straight line path to target, which will be fed as input to the autoencoder.

Autoencoder

The structure for the neural network of the autoencoder is directly taken from (Antal, Buza & Fejer, 2021) [1], where it has been shown that it is a very effective structure for the purpose of this project. The structure is best described by them:

> "The original architecture proposed by Wang et al. [41] consists of three convolutional blocks. Each convolutional block contains a convolutional layer followed by ReLu activation. The first block contains 128 filters (kernel size: 8), the second block contains 256 filters (kernel size: 5), and the last block contains 128 filters (kernel size: 3). Each convolution preserves the length of the time series (stride: 1, zero padding). After the third convolutional block, global average pooling is used. We employ this construct in the encoding part of our model. The decoder is built in a similar fashion, but the order of layers is reversed and convolutions are replaced by transposed convolutions (Conv1DTransposed). No activation function was used in the last layer of the decoder, as our model has to generate sequences with positive and negative values." - (Antal, Buza & Fejer, 2021) [1]

It should be noted that the filter sizes for the convolution and convolution transpose blocks have been replaced with the following respectively: 64, 128, 64, 64, 128, 2. These values were replaced to fit the different trajectory size.

The autoencoder was trained with the Adam optimizer, with a batch size of 32, for 150 epochs. The number of epochs is higher than {source}, which was 100 epochs, because there is significantly less data and some of the error needed to be reduced at the risk of some overfitting. A 60%-40% split of the data was used for training and testing, respectively. Fig.5a, Fig.5b, and Fig.5c show the generated pitch deltas, yaw deltas, and path constructed from those deltas, respectively, for the input from Fig.4.

The library that was used to build the autoencoder was keras.

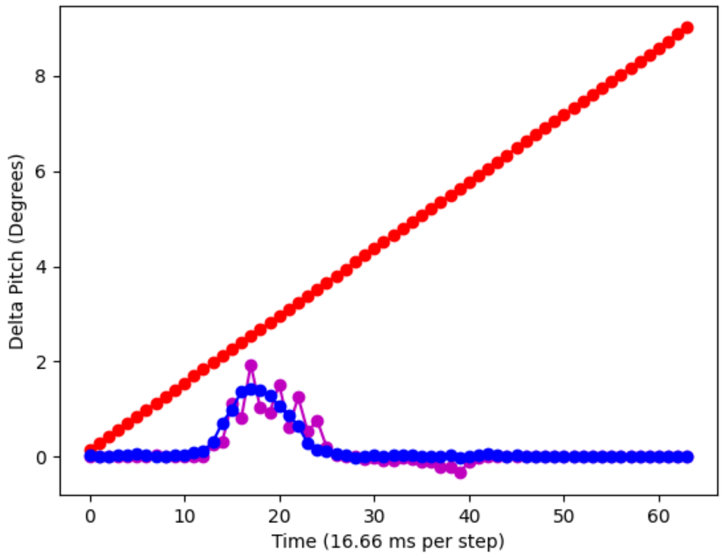

Fig.4a: Delta Pitch of autoencoder output for input from Fig.4. The original pitch input (red), the human delta pitches for this target (magenta), the generated delta pitches (blue).

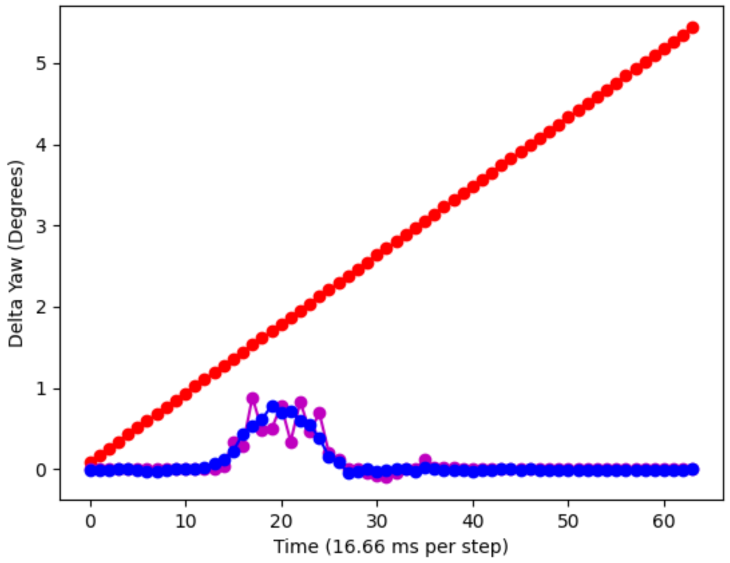

Fig.5b: Delta Yaw of autoencoder output for input from Fig.4. The original yaw input (red), the human delta yaws for this target (magenta), the generated delta yaws (blue).

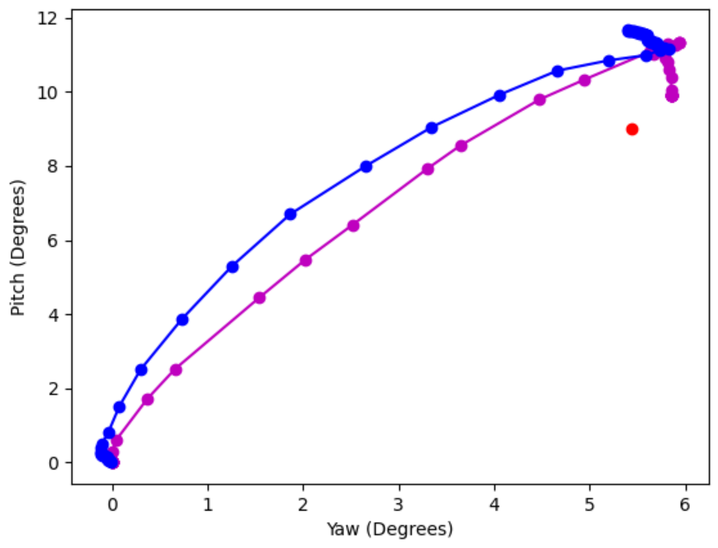

Fig.5c: The generated path from the autoencoder (blue), the human path (magenta), and the target location (red)

Experimental Analysis

By looking at Fig.5c, it is obvious that the autoencoder has done a fairly good job of generating human-like mouse trajectories. The AI has generally learned when to move and by how much. For example, the average visual reaction time for humans is 0.2-0.25 seconds, and the AI has managed to mimic this. The reaction time delay is evident in Fig.3a and Fig.3b, where there are 0.216 seconds (13 time steps) at the beginning with no movement from the human. The same effect can be seen from the AI in Fig.5a and Fig.5b.

However there is a notable issue. Fig.5a and Fig.5b also show that the autoencoder is not able to mimic some of the sudden spikes in the delta pitch and delta yaw that the human is able to. Instead, it smooths the motion out. This resulted in a smoother looking final path, as shown in Fig.5c. This problem is likely exacerbated by the very small dataset. If further data were added to the dataset, the effects of the smoothing by the AI would be lessened, as it would likely identify these spikes as important features.

Conclusion

The method used in this project has shown that there is promise for AI agents to behave like humans in 3D environments. Despite a very small dataset, the autoencoder has been able to generate paths which look almost human, simply from a straight line. If the dataset were to be expanded, there would likely be a drastic increase in how human-like the generated paths are. There might be even larger increases if the dataset is tailored to one person, or one “style” of aiming.

Final Note on Future Uses and Possibilities

A very important use for an AI like the one that was created in this project is to help detect whether or not a bot is controlling a mouse. For example, many sign-up pages will often check that a bot is not creating the account using various visual mechanisms. However, with the rise of deep learning in computer vision, these mechanisms are not working as well. An alternative is to monitor how the mouse is moved to identify whether or not the user is a bot. Synthetic mouse trajectory data can be created with this AI, which can then be used to train another AI on which trajectories are human and which trajectories are bots.

There are also many implications for the gaming industry, the area where much of this project is inspired from. Cheating in video games has been quite complex, with software that can assist players in subtle ways that are nearly impossible to identify with the naked eye. The method described in the above paragraph could be further extended to identify cheaters in video games. However, this AI could also be very useful for creating human-like NPC’s (non-player characters).

The final future possibility to be mentioned is in robotics. Creating robots which move and act like humans is an important step in having simple and easy interactions with robots. This project could easily be extended to work in Virtual Reality. The rotational data that is collected would then be based on how humans looks around and focus on objects. This data could then be used in the same process described in this paper to create an AI that can generate movement trajectories for robots to look around and move their heads like humans.

References

Antal, M., Buza, K., & Fejer, N. (2021). SapiAgent: A Bot Based on Deep Learning to Generate Human-Like Mouse Trajectories. IEEE Access, 9, 124396-124408. doi: 10.1109/access.2021.3111098

Virk, B. (2022). MTDataCollector (Version 1.0.0) [Computer software]. https://github.com/Deprecator16/MTDataCollector