Zip, Zap, Boing: The Hermeneutics of Play

Kaleb Gezahegn, Alex Wardle-Solano, Keoni Spencer

1 ABSTRACT

The use of multiple training modalities to improve an agent's performance in reinforcement learning tasks is an open area of research. In this paper, we investigate the merits of combining 'learning by observation' and 'learning by correction' whilst training an agent to perform a simple, multi-player, turn-based game called 'Zip, Zap, Boing'. The testing and implementation of our models were done in a virtual game environment, seeking to model the actions of humans in a real game of 'Zip, Zap, Boing'. Two models were considered: a simple tabular approach, and a deep learning-based approach. Our models are then deployed in a real-world setting where our robot, "Shutter", detects and classifies moves from human players in its field of view and actively participates in the game through a series of move outputs. We seek to investigate the effect of imperfect or erroneous demonstrations by human 'experts' during the phase of 'learning by observation' and attempt to demonstrate the merit of using multiple training modalities whilst training an agent to perform a task. It was found that our agent was relatively robust in the face of erroneous moves by human experts, provided the moves were correctly labeled as errors. However, it was found that when erroneous moves went unnoticed the performance of our models quickly degenerated. It was also found that our model's performance improved far more quickly when combining observation with correction.

2 INTRODUCTION

The task in this project was to create an agent which, through observation of real-world gameplay, could infer the rules of and participate in a game of 'Zip, Zap, Boing'. The task has been split broadly into two subtasks: the first focused on the reinforcement learning aspects of building a game-playing agent, and the second focused on pose detection and interfacing through Shutter.

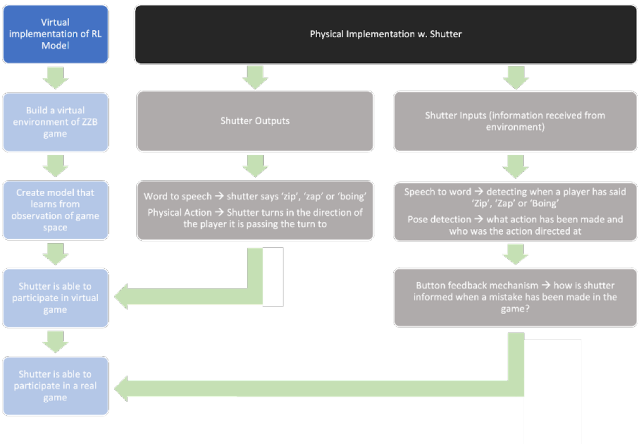

Figure 1: Workflow for 'Zip, Zap, Boing' task

The use of multiple training modalities to improve an agent's performance in reinforcement learning tasks is an open area of research. In this paper, we investigate the merits of combining 'learning by observation' and 'learning by correction' whilst training an agent to perform a simple, multi-player, turn-based game called 'Zip, Zap, Boing'.

'Zip, Zap, Boing' is an interactive, multi-player, turn-based game. A random player is chosen to initiate the game. When it is a player's turn, they are able to pass on the turn to another player using one of three moves: to pass the turn to the player on your left you simultaneously point left and say 'Zip'; to pass the turn to the player on your right, you simultaneously point right and say 'Zap'; to pass the turn back to the player who gave it to you, you simultaneously raise both arms and say 'Boing'. If a mistake is made, the game is reset and the player who made the mistake begins a new round. You can only play a move on your turn, and the legality of the moves you may play is context-dependent. The rules for legal moves are as follows:

(1) If you are selected to initiate the game, you can play either a 'Zip' or a 'Zap'
(2) If you receive the turn from a 'Zip', you can play either a 'Zip' or a 'Boing'
(3) If you receive the turn from a 'Zap', you can play either a 'Zap' or a 'Boing'
(4) If you receive the turn from a 'Boing' there are two scenarios
    - You receive the 'Boing' from the player to your left, in which case you can pass the turn right with a 'Zap'
    - You receive the 'Boing' from the player to your right, in which case you can pass the turn left with a 'Zip'

We chose the 'Zip, Zap, Boing' game task for two primary reasons. Firstly, 'Zip, Zap, Boing' presents us with a good 'toy RL' problem, in which we would expect, a priori, for an RL-based agent to perform well. In fact, we might more aptly characterize this as a 'conceptual bandit problem': the relative 'goodness' of a given move is context-dependent, but the agent's actions in the game do not change the environment or the sum of expected rewards. However, for the sake of consistency, we will frame this as an RL problem in this paper. Secondly, the state and action spaces for this problem are relatively small (10^3), leading to computationally tractable problems.

With the spirit of parallelizing work in mind, the workload for this project was split into three tracks which were developed in tandem. (1) Virtual Implementation of the RL Model, (2) Physical Implementation of Shutter Outputs, (3) Physical Implementation of Shutter Inputs. The execution of task (1) involved the creation of a virtual game environment for the creation of synthetic data and the creation of tabular and deep learning-based models. In task (2) Shutter was configured to display a combination of verbal and motion-based outputs on its turn. In task (3) shutter was programmed to identify moves made by players based on simultaneous verbal and visual cues. Both supervised learning-based and

logic-based methods for move classification were considered. The latter was found to work more consistently.

3 RELATED WORK

Our project fits within a long trajectory of imitation learning and inverse reinforcement learning-based approaches. We are choosing to frame our problem as an IRL problem in the sense that we are using observations to model a reward function as opposed to the traditional imitation learning framework in which models compute a policy without attempting to model the underlying reward. Examples of similar tasks are numerous, for example, Google DeepMind's use of Deep Q Learning to train agents on Atari games. A number of works have been relevant to our approach:

i) "Algorithms for Inverse Reinforcement Learning", Stuart Russell, Andrew Ng[3] - this is an early paper in inverse reinforcement learning in which the issue of 'degeneracy' in reinforcement learning problems is addressed. This is the problem of having multiple suitable policies for a given set of observations - this is likely to be relevant to our project in which there are often multiple possible policies/actions for a given state.

ii) "Apprenticeship Learning via Inverse Reinforcement Learning", Pieter Abbeel, Andrew Ng[2] - another early paper in IRL which goes into many of the implementation details for an IRL algorithm. Though our current agent is very basic, for the final milestone it is hoped that a more sophisticated model, akin to the one outlined in this paper, could be tried for this problem.

iii) "Attention is All You Need", Ashish Vaswani, Noam Shazeer [4] - the attention model allows a model to retain 'memory' of past time-steps. The simple version of the 'Zip, Zap, Boing' has no need for memory because it can be thought of as a Markov model in which a given state is only dependent on the previous state and action, however, it might be interesting to incorporate an attention model which would be capable of playing more complicated versions of our game with temporal dependencies.

iv) Reward-rational (implicit) choice: A unifying formalism for reward learning.", Hong Jun Jeon, Smitha Milli, Anca D. Dragan[1] - discuss the importance of agents learning from the rich pool of 'implicit feedback' that humans produce: this feedback could be intentional or unintentional. The approach taken in this paper is slightly different from ours as they model human behavior via a Boltzmann-Rational policy and then use a Bayesian approach to update a belief over possible rewards, something which we do not do. However, the paper repeatedly stresses the importance of building agents that learn based on multiple types of implicit feedback simultaneously and proposes a formalism for doing so.

4 TECHNICAL APPROACH

4.1 Virtual Game Environment

The virtual game environment is an attempt to virtually model the Shutter robot playing a real game of 'Zip, Zap, Boing'. It allows the game to generate thousands of synthetic data points in a controlled and time-efficient manner and to tentatively evaluate the performance of our models before deploying them into a real game environment. Moves made in the virtual game environment are printed to the command line to help with visualization.

Certain assumptions about the ways in which real players participate in this game were necessary for the construction of the model. Namely, the model accounts for the fact that human participants will occasionally make mistakes, and that on occasion these mistakes will go unnoticed, and the game will continue. Furthermore, the model assumes that human participants are likely to make mistakes in a predictable or plausible manner and that these plausible moves may go unspotted. From observation, it was noticed that humans would frequently confuse verbal and action-based responses. For example, they might point left and say 'Zap' rather than 'Zip' or point right and 'Zip'. In cases when these moves went unnoticed by other players, the erroneous move was interpreted as being the 'closest' legal move. In the case where a player erroneously points left and says 'Zap', the game might proceed as if the player had played the legal move left + 'Zip'.

```
#define three legal moves in the game
Zip = Move( (self.player - 1) % N, "Zip")
Zap = Move( (self.player + 1) % N, "Zap")
Boing = Move(self.previous_sender, "Boing")
```

Figure 2: Definitions of "Zip", "Zap" and "Boing" moves

The game environment is defined using three classes: a 'State' class which defines the state of the game, a 'Move' class which defines moves within the game, and the main 'ZipZapBoing' class in which gameplay takes place. Due to the Markov nature of the game, the state space is fully specified by three variables, the index of the current player, the index of the player who passed the turn to the current player and the 'action' that the previous player made: "Zip", "Zap", "Boing" or "Reset" which indicates that the previous turn was a mistake, and the game is being reinitialized. If there are N players, then the total number of states is N x N x 4. The set of possible actions in each state has a size of N x 3: N players that the turn can be passed to and 3 ways in which the turn can be passed. Thus, the total size of the state-action space is N x N x 4 x N x 3. This is relatively small.

'Move' is the class we use to define the moves in our game and takes two arguments: 'pointer' and 'action'. Pointer is an integer value that specifies the index of the player that the turn has been passed to, and action specifies the thing that was said when the turn was passed (i.e. "Zip", "Zap" or "Boing"). Thus the three primary moves in our game are dynamically defined as follows, where 'self.player' is the index of the current player, 'self.previous_sender' is the index of the previous sender and N is the number of players.

The 'ZipZapBoing' class defines the actual workings of the game. It requires three arguments: the number of players in the game, the type of gameplay (either computer-generated moves or user-inputted moves) and 'error' which specifies the proportion of the time that our program generates an incorrect move. There are also several class variables that are updated throughout the game: the

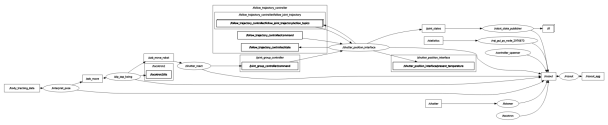

Figure 3: System Diagram

current player, the index of the previous player, and the move made by the previous player, all of which are updated after each move. The entire gameplay then stores the state in a class variable called 'ledger', a dictionary of timestamped actions taken. Each log in the ledger is a three-element array defining the state of the game, a two-element array defining the action taken and an integer value defining the inferred reward for that state-action pair.

In order to mimic 'observational' and 'participatory' styles of learning, the virtual environment comes pre-configured with two 'gameplay' settings: one in which "virtual Shutter" simply observes a group of "virtual experts" playing a game, and another in which "Virtual Shutter" inserts itself into the game and actively participates. In the participatory gameplay mode, when a player in the game space takes an action and gameplay continues, we assume that the action was a good one and ascribe that state-action to a positive reward of +1: this is the reward-rational assumption. Another way of framing this is to say that when a move is made and gameplay continues, there was an implicit choice by the participants between stopping the game or allowing it to continue. The fact that they allow it to continue implies that the move was legal and thus should be given a positive reward. If a move is flagged as an error, then we ascribe the state-action pair a reward of -10. This provides us with the data on which we train our models in an online fashion.

To better visualize the proceedings of the game, each move is printed onto the console as shown in Figure 4.

In order to play the game with Shutter, beyond the simulation environment, the 'zip_zap_boing.py' script will switch from generating moves to receiving them. Here, the game takes in an interpreted player pose and ID as a String and uses it to update the game state. The ID passed corresponds to the player's ID and position in the game array. Array indices update when their corresponding real-world player takes their turn, and when the player index becomes zero, then Shutter predicts an action.

This means of game input limited the number of changes needed between playing the game in simulation and the real world and enabled the model Shutter trained on to accurately apply to playing with user input.

4.1.1 Tabular Model. In the tabular approach, we create an N x N x 4 x N x 3 'reward matrix' which stores a reward value for each of the possible state-action pairs. After each observation, this reward matrix is updated by adding the reward value for that observation point to the appropriate index in the reward matrix. This was ultimately the model that we chose to use.

4.1.2 Deep Learning Model. In the deep-learning approach, observations are treated as labeled inputs to a neural network where the state is the input, and the reward is the label. The indexes used to

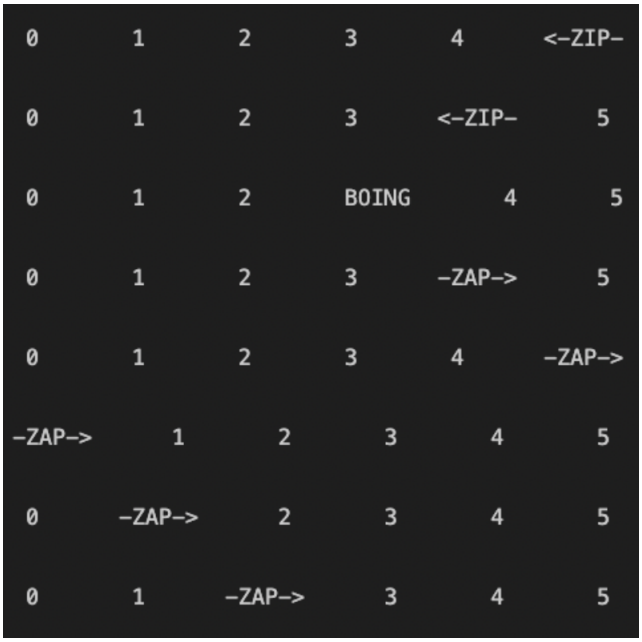

Figure 4: Visualisation of moves

define the state of the game are converted to categorical values and flattened to create an input vector of dimension N + N + 4. The neural network maps to an output vector of size N x 3 which represents probabilities over the next move in the game. The output is of size N x 3 because there are three possible actions that can be taken and N players that the action can be directed at. The neural network is trained online using an online gradient descent algorithm.

Though the deep learning model did work, it was ultimately found to be an overly heavy-handed approach for such a simple problem. Because of the tractability of this problem, it was not necessary to approximate the function that maps from states to actions using a neural network, we could just represent this function directly using the tabular approach. In fact, even a very modest neural network architecture had parameters of orders greater than 10^4. Given that our tabular approach only has entries on the order of 10^3, from a memory and computational perspective there is no motivation for taking a deep learning approach for this task.

4.1.3 Model Evaluation. The performance of our model was evaluated by feeding it a set of all possible legal contexts as inputs and considering the proportion of the time that the model's prediction of the most likely next move was correct. The set of legal contexts is simply the set of all states in the game that could be legally arrived at. To illustrate, an accuracy of 85% means that the model correctly predicts a legal next move for 85% of the legal contexts.

4.2 Body Tracking

In order for Shutter to participate in gameplay outside of a simulation, the robot needs to be able to detect people in a real environment and interact with them. For the detection component, either the Kinect camera on Shutter was used.

The Kinect is a very fast tool for feature detection on people, capable of full-body pose tracking. It does not do hand data, which meant a pivot from our original approach, but no less effective. This device was chosen for the project because of its ability to classify and locate a number of human features in real-time, the fact that it already had a ROS wrapper, and because MediaPipe was incapable of fulfilling the system's needs. The Kinect pose interpretation script is in the 'interpert_pose.py' script of the project, where the code receives a marker array of human features, published by the Kinect, and translates them into a 'Zip', 'Zap', or 'Boing'. The script runs as a node and publishes a String for the core game node to use. Attached to the String is also an ID suffix, denoting which player is performing the action.

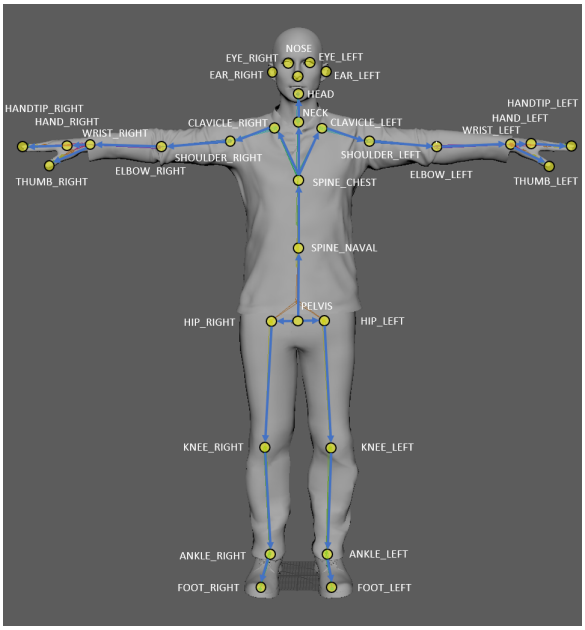

Figure 5: Kinect features

The Kinect pose detection system locates 32 landmarks on the human body and outputs each of their locations as a point. It is a very robust means of detection and can identify landmarks even when partially obscured.

The project implements pose tracking to isolate features on a player's arm and pelvis. Marker array message type is decomposed into the 'right shoulder', 'left shoulder', 'right wrist', 'left wrist', and 'pelvis' points. From there, vectors are made from the shoulders to their corresponding wrist points, and from the shoulders to the pelvis. With these vectors, the angle between the two is calculated and an action is assigned depending on whether the angle exceeds the action activation threshold.

This action is then published to the 'zip_zap_boing.py' script to update the game state.

4.3 Auxiliary Features

There were some additional auxiliary features added to increase Shutter's interactiveness with other people. For the purposes of making Zip Zap Boing with Shutter more realistic, Shutter moves and speaks its actions, much like a human would do if playing the game. In order to move, each prediction of Shutter's is published and mapped to a corresponding pose. When Shutter chooses an action, the correct pose is also published to its servos.

Additionally, players speak their actions aloud. When Shutter chooses an action, it also publishes its action as a String to Tacotron, a ROS wrapped library that translates text to speech and then outputs it using any connected speakers (the computer's in this case). This greatly improved how realistic the game was.

5 EXPERIMENTS

5.1 Imperfect Demonstration by Experts

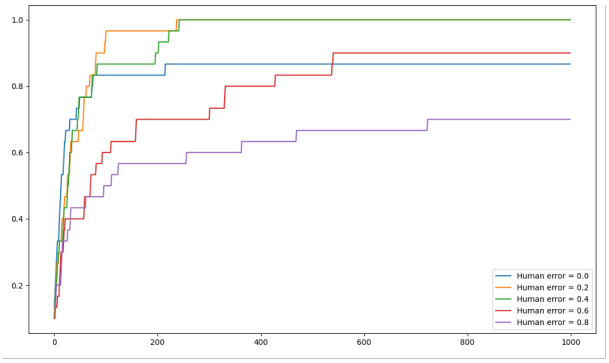

Figure 6: Accuracy Rate vs Observed Moves Given Different Human Error Rates (Players = 10)

One of the main observations made whilst watching people play 'Zip, Zap, Boing' was that players would often make mistakes and that these mistakes would often go unnoticed. We wanted to investigate the impact that these two error cases would have on model accuracy.

In considering the case when mistakes were made but were correctly flagged by players as mistakes, the virtual game environment generates a plausible erroneous move some proportion of the time specified by a 'human_error' variable Interestingly, it was found that when the error rate was 0 the model never reached perfect accuracy (see figure 6) as it never observed the initialization states of the game because this state only occurs after a player makes a mistake. Furthermore, it was found that the model was robust in the face of erroneous moves and converged to 100% for error rates as high as 0.5 (see figure 6).

It was found that unnoticed errors by demonstrators were a much bigger problem for model accuracy. This experiment was conducted by fixing the error rate at 0.2 (i.e. erroneous moves are made 20% of time) and then defining a second variable 'errorpass' which defines the proportion of the time that erroneous moves go on unnoticed. On the occasions where an error goes unnoticed, the 'nearest legal move' to the erroneous move is calculated and the game continues as if it was the nearest legal move that was played. Even for error pass rates as low as 0.2, the model does not reach full accuracy after 1000 observations. For error pass rates of 0.6 the model degenerates to near 0 accuracy.

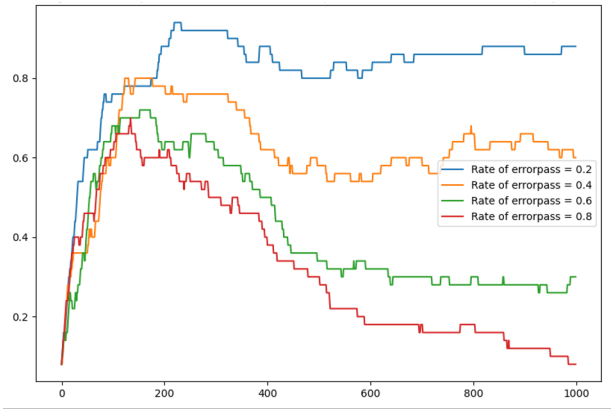

Figure 7: Accuracy Rate vs Observed Moves Given Different Human Error-Pass Rates (Players = 10, Error rate = 0.2)

This result is perhaps intuitive: making a mistake as an expert demonstrator is acceptable, so long as we identify the action as being a mistake.

5.2 Learning Using Multiple Modalities

We evaluated the merits of learning using multiple modalities by comparing the performance of such regimes against a benchmark in which the model only learns from correction. In the benchmark case, our agent plays 'Zip, Zap, Boing' without observing any of the moves made by other players. At each timestep, the agent predicts the next move using an epsilon-greedy policy and receives a 'correction' from the environment if it makes a mistake. The number of moves required for the benchmark model to reach 100% accuracy was in the order of thousand of moves, compared to an order of hundreds when trained using a combination of observation and correction.

5.3 Hand Tracking

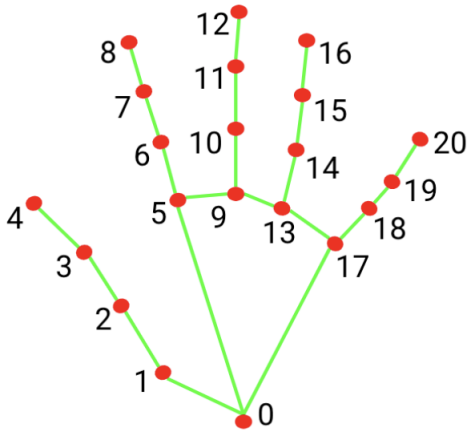

Figure 8: MediaPipe's Hand Tracking Landmarks

Initially, rather than utilizing the Kinect camera to detect player pose and gestures, MediaPipe was our choice feature extraction library. The project aimed to detect distinct hand gestures, and map those to 'Zips', 'Zaps', and 'Boings'.

MediaPipe hands is an ML pipeline model mainly composed of a Palm Detection model that detects initial hand locations and a hand landmark model that takes image frames and localizes twenty-one three-dimensional coordinates on the hand. These coordinates correspond to predefined landmarks that allow us to create very realistic representations of the hand even when certain fingers or features of the hand are occluded.

We collected data where we recorded these coordinates from videos and then annotated the data to train a ResNet-18 CNN that would take Shutter's Intel RealSense camera feed in real-time, and classify each gesture as a player makes it.

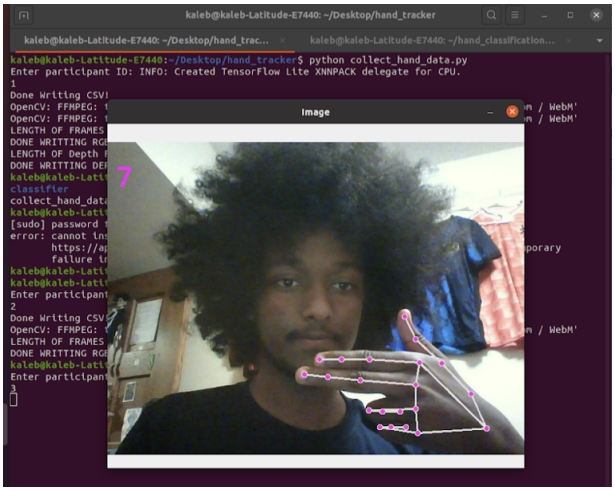

Figure 9: Data Collection Pipeline Demo

Our training pipeline for our gesture classifier consisted mainly of a script that would display shutter's RealSense camera feed and would superimpose 21 hand landmarks using MediaPipe's hand detector model onto the feed. This happened as the user is demonstrating different gestures corresponding to each possible move in the game. After 2-3 minutes of data collection, the script is manually terminated and the image frames are converted to a .webm video and written to disk while the coordinates of the hand landmarks from our model are saved as a csv file. We then take our video and upload it to a **VGG Video Image Annotator** and label the segments where we identify a game gesture with its corresponding move label and save the temporal segments as a csv file.

We were then hoping to time-synchronize the annotated temporal segment csv and the hand landmark csv and feed it into a pre-trained resnet18 model to create a model that can classify our 3 different gestures. Unfortunately, we were running low on time and were able to get moves from body poses using Kinect, so we decided to focus on other aspects of the project.

5.4 Body Tracking

The first body tracking library utilized was the YOLO classification model. YOLO is very modular and easily implemented, and so it was an attractive first choice. However, the script's performance was quite poor with massive amounts of lag. Moreover, the library does not grant human feature segmentation, meaning the hand's position could not easily be extracted explicitly from the detected person's general location. In order to determine the pointing direction of a player, a contour would have to be taken of the person, with their area and extremity locations as deciding factors of their chosen action.

Given the difficulty of implementing this, mediapipe and the Kinect became the most viable methods of determining body pose, specifically because they are capable of returning the location of individual body parts.

MediaPipe was chosen next given that the library is better suited to hand detection and gesture recognition than the Kinect, allowing both the hand and body detection scripts to share a common library. However, initial testing showed mediapipe as less accurate with numerous people, the stock library did not seem to support multiple person detection, and the mediapipe code was not easily discernible.

For the final implementation, Microsoft's Azure Kinect was used. The tool documentation was excellent, and it already had ROS wrappers so that it could be easily integrated into the Zip Zap Boing project.

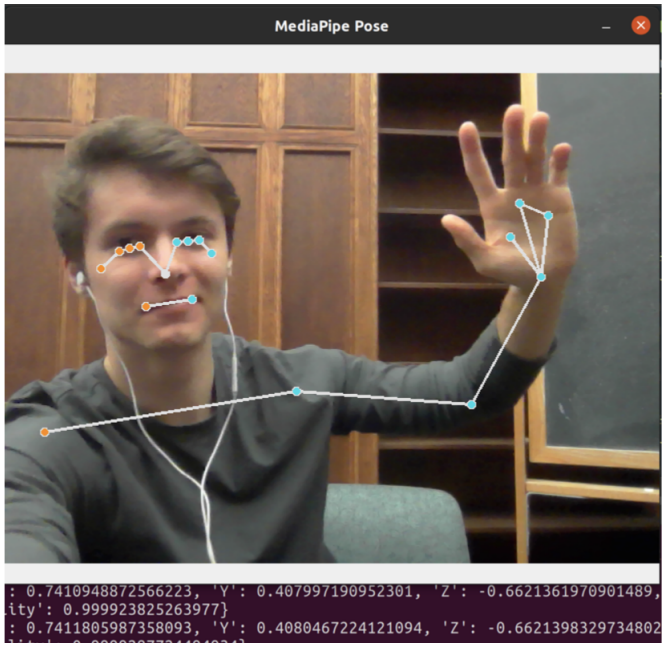

Figure 10: MediaPipe Body Tracking

The Kinect published near real time body tracking data in the form of a marker array. The Kinect segments a person into 32 features, and publishes the pose of each. Using this data, Shutter could use the differences in x and y pose data to reliably compute whether or not a player had their arms raised.

Interestingly, the Kinect would indiscriminately publish human pose data, so retrieving a person's position was sometimes hampered by other people in the frame. The Kinect did not guarantee that after the pose one person was sent, it would send someone else's. However, this did not detract from overall game performance.

It was also found that the Kinect would randomly assign each skeleton an ID. This ID was unique for each person in the frame, but the Kinect would not remember an ID in the case an individual walked out of the camera's view. This meant that in the case a player leaves Shutter's immediate area, the game would need to be restarted to bring them back in.

5.5 Speech to Text

An essential aspect of playing Zip Zap Boing with other people is saying what move you are making out loud. If the move you say out loud doesn't line up with the gesture you make, you lose that around based on the standard rules of the game. In order to capture this feature, we used Google's speech to text API to get audio input from the game participants. After some testing, we were able to generate sets of words that the model associated with Zip Zap and Boing to accurately interpret a user's move or have it output none if the move was not recognized.

6 CONCLUSION & FUTURE WORK

It was found that our agent was relatively robust in the face of erroneous moves by human experts, provided the moves were correctly labeled as errors. However, it was found that when erroneous moves went unnoticed the performance of our models quickly degenerated. It was also found that our model's performance improved far more quickly when combining observation with correction.

After the incorporation of the Kinect and user inputs, as well as Shutter's action outputs, we found that Zip Zap Boing with Shutter was both interactive and viable. Shutter responded to actions in kind with its own, was able to choose accurate actions, and play the game as a real person would. On errors and incorrect moves, the game would start up again quickly as normal and ultimately had a life-like feel to it.

In the future, we would seek to add more feedback to the system in the form of a button or true sentiment analysis so that we experiment on Shutter's ability to learn from feedback. Eventually, by incorporating other learning modalities, speech sentiment analysis and facial mapping, Shutter may learn to play better than humans.

We ran into concurrency issues when attempting to get user input from both poses and audio feedback so we chose to only take input from one stream. We hope to eventually capture this additional aspect of the game by solving the concurrency issues and also having shutter respond when an erroneous move is made.

The mapping code between Kinect IDs and player indices is also rudimentary and does not account for player positioning. Making this framework more adaptable and capable of adding new players would go a long way in improving Shutter's player realism.

We also hope to flesh out our model for detecting hand gestures so that playing the game resembles how the game is played traditionally. This would allow us to create different groupings of gestures corresponding to each game move which allows for additional complexity to how the game is played.

REFERENCES

[1] J., J. H., S., M., and D., . D. A. Reward-rational (implicit) choice: A unifying formalism for reward learning (Version 4)". *NeurIPS 2020* (2020).

[2] P., A., and A., . N. Apprenticeship learning via inverse reinforcement learning. In Twenty-first international conference on Machine learning". *ICML '04. Twenty-first international conference* (2004).

[3] S., R., and A., N. Algorithms for Inverse Reinforcement Learning". *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning* (2000).

[4] Vaswani, A., N., S. N. P., J., U., L., J., N., G. A., L., K., and I., P. Attention Is All You Need". *ICML '04. Twenty-first international conference* (2017).