Neural Networks Final Project Writeup

Pranav Akinepalli, Sameer Haniyur, Guy Allouche, Viren Velacheri

December 5th, 2021

1 Introduction

The goal of this final project was to come up with a SuperTuxKart AI agent that would win 2v2 ice hockey game against the game's AI agents as well as those of classmates and instructors. The game is very similar to popular games such as Rocket League. Having played games similar to this, we decided to come up with a strategy that would mimic, as close as possible, to the way we played it.

We chose to use a vision-based model with a hand-built controller, as this seemed closer to the kind of work we'd done in past assignments.

2 Approach

Our big picture approach was to first build a controller for the karts that takes in the true screen position of the puck and executes some reasonable strategy, then use games of this agent to create a training data set for a vision model which predicts the puck's screen position. We could then easily combine the controller and vision model to create a working image-based agent.

3 Controller

Our first controller simply had both karts chase the puck directly, largely adapting the code from previous assignments when the kart would race towards a moving point on the race track. This allowed us to get started quickly, and helped us quickly make some useful general observations. We noticed that 1) having both karts chase the same point meant that they would often collide and interfere with each other 2) the agents would often lose sight of the puck doing this, at which point they could not really make any informed actions.

We made headway at fixing both of these by assigning roles to our karts; one would be an attacker, which chases the puck, while the other would be a defender that stays at the goal. Not only did having a defender mean the karts did not interfere with each other, but the defender could also maintain vision of most of the arena, meaning the puck remained in view even when the attacker lost sight of it.

Our next goal was to improve our attacker agent; clearly, pushing the puck aimlessly would not make it a scoring threat. Instead, we wanted to systematically push the puck towards the goal. We decided having the world position of the puck would be useful for this, since it would allow us to compute vectors between the kart, goal, and puck, so we could develop a more sophisticated navigation algorithm. To do this, we adapted the *_to_image* function, inverting it to create a *screen_to_world* function with which we could convert our screen point to the puck's world position. We include more details about this in section 6 (Projecting Screen to World Coordinates). [NOTE: We are aware the TA's released a similar function *_to_world* on Piazza, but want to be clear that this function is based on our work, not the other way around. The post Tianwei references as the basis for *_to_world* was posted by one of us.] Having the puck's world position also meant that the puck's location could be communicated between our two karts. This meant that only one kart had to see the puck for both agents to know where it is.

The basic idea for improving our attacker agent was to approach in a curve from outside the opponent's goal, such that we hopefully hit the puck towards it. For example, if the puck is on the right side the goal, we'd want to approach in a curve from the puck's right so as to hit towards the opponent's goal on the puck's left. To achieve this, we have our attacker chase an aimpoint which lies on the line from the goal to the puck; the distance from the puck at which this aimpoint lies is determined by an activation function of our distance to the puck and the angle between the puck, goal, and our kart. We visualize this in the figure below. Notice that although at any time the kart is chasing a point with no regard for angle of approach, it ends up forming a nice curved path.

We use a very similar algorithm for our defender agent, using our goal instead of the opponent and curving to hit away rather than towards it, and only going after the ball if it reached some threshold near our goal. We did notice that our defender would struggle to find its way back to our goal, especially if it charged at the ball too far while clearing it, so we decided to have the defender simply become a second attacker if it charged too far away from our goal.

One thing we tried is including a scoring agent that prowls the middle of the field and takes easy shots. The reason we thought this would be effective is that we saw our agents often put the ball in a perfect position right in front of the opponent's goal, then continue to chase it and hit it across from the side. We hoped that this scoring agent could call off the other attacker if it saw an easy goal, then score. Unfortunately, in practice this didn't seem to work out because the scoring agent would often miss the ball anyway, and

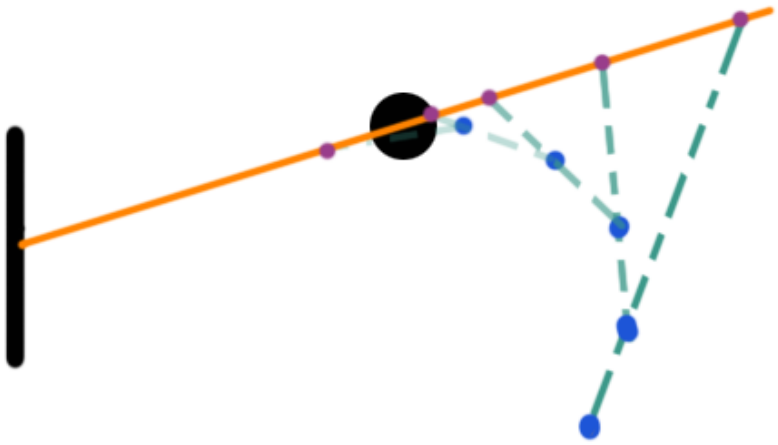

Figure 1: Our attacker route setting. Blue dots are kart locations, purple dots are aim points.

it was not easy to determine when a shot opportunity existed.

Towards the end of the project, we realized that the TA's Jurgen agent relied only on the puck's world position and our own karts' positions. Since we had this data from our *screen_to_world* function, we decided to use the Jurgen agent as our controller. Unfortunately, we quickly realized the Jurgen agent's great performance relied on knowing the puck's location even when off-screen, so having two Jurgen agents would often result in both losing the puck and being completely lost. We avoided this by having one Jurgen agent, and one agent using our defender controller (which becomes an attacker if necessary). Moreover, if both lost sight of the ball, the Jurgen agent would defer to our attacker controller for actions. We found that this combination was highly effective - given the true screen location as input (that is, a theoretical perfect vision model), this controller was able to consistently defeat the AI and TA agents, often scoring 2 or 3 goals per game. Thus, at this stage we were happy with our controller and moved on to the vision model.

4 Network Architecture

Our model is similar to the planner model from homework 5 that is used to predict the aim point, comprised of 4 sequential blocks of batchnorm2d, conv2d, and relu layers. It uses the same encode-decoder structure to predict a heatmap and extracts the corresponding peak using the spatial argmax layer. We experimented with the size of the network reducing it by half or to medium size. When we trained using this smaller network, its

overall weakness was evident since the reduction in training loss value wasn't nearly as large. As a result, we scrapped this approach and didn't think of doing even smaller networks. We also considered the segmentation model, but felt it wasn't necessary for us as we were only focused on predicting the puck location and nothing else (location of opponents, items, etc).

5 Training and Data Collection

We gathered images from recorded videos of our agent playing against the AI and the TA's agents. At first, we gathered all the images from both teams, but then figured that it would be most effective to only gather the ones from our point of view. We compiled over 36 matches worth of data (40 GB of images) with 6 matches against the AI and each of the TA agents. For the labeling of the images, we just used the actual puck location and the overall structure of labeling the images and training was similar to the past homeworks. The training code was similar to that of homework 5 except we realized that even at 50 epochs, it seemed to us that there was no convergence and so we increased to 100 epochs and even 200 to train it over night. The save model function code was modified such that we saved model checkpoint files at each epoch, so we could test out the ones where training loss values seemed to converge. This was beneficial as it turned out that the best model appeared to occur at epoch 40 for us.

6 Projecting Screen to World Coordinates

As mentioned earlier, we figured that the use of world coordinates would help in giving our attacker a more streamlined approach to scoring as we would now be able to compute the vectors between kart, goal, and puck. Even though we technically don't have access to the depth information needed to do the inverse of the to_image function, we can take advantage of the fact that the puck is pretty much always on the ground, so its height is pretty constant and this allows for the projection onto world coordinates. This makes sense as the puck lies on a flat plane, so we can essentially find the intersection of the plane and the line. Using the projection formula and inverting it as necessary, we essentially get the formula we need to get the world coordinates. You can see the formula and the overall inverse operation in our *screen_to_world_* function (approximately lines 101-113 in player.py).

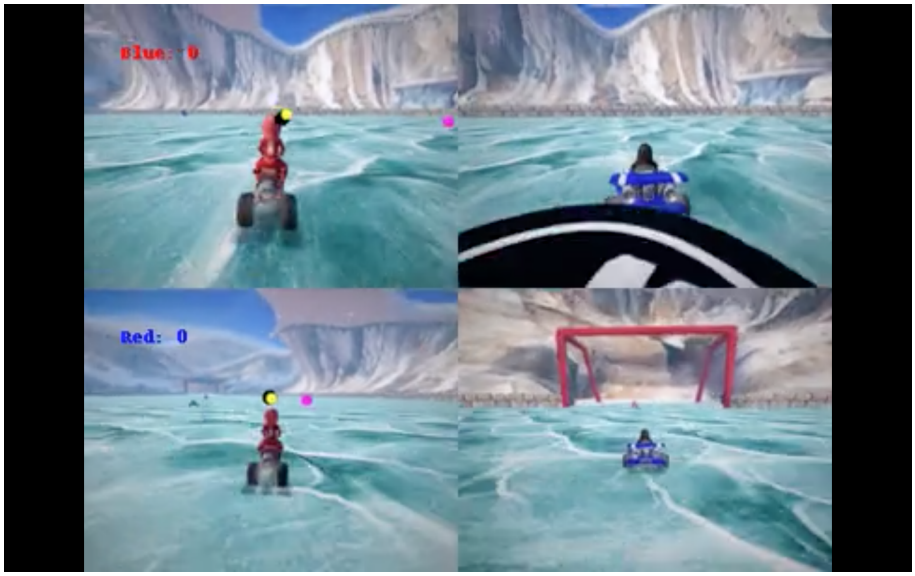

Figure 2: Sample image of one of our games. We are red team. Pink dot represents the aim point of the direction it should be turning in, yellow dot represents our model's prediction of puck location, and black is obviously the puck.

7 Conclusion

In general, our model performs pretty well. We definitely do not beat all of the TA agents every time but our model is able to score at least one goal against the TA agents pretty consistently. In the worst case, out of 8 games, our model usually ends up scoring 5-6 goals overall and go all the way up to 12. Our model is also relatively fast in terms of producing actions for its two player karts. By saving the timestamp when the method is called to generate actions and subtracting from the timestamp right before the actions are returned, we are able to measure on average how long it takes to generate actions. This number comes out to around 10-15 milliseconds per call which is significantly less than the required threshold of 50 milliseconds. You can check out our model in action here against one of the TA agents.

There are, of course, natural limitations that come with using a vision-based agent with a hand-made controller. Obviously, the Jurgen agent works very well when given the state of the game but by using this through a vision-based approach, it becomes much worse. There are many inaccuracies which get compounded by using a vision-based agent. By only having an image to view, predicting the puck's location on the image, and using that to convert to world coordinates, the world coordinates will tend to be at least slightly different compared to the true location. Furthermore, when the puck is not seen by either kart, then the Jurgen agent is not even applicable anymore since it has no idea what to do in those scenarios (since it was trained to always know where the puck is at all times).

In terms of what we could have done differently, it was recommended to use the model from homework 3 with segmentation to detect the puck so that might have been a better

approach. Also, it could have much better to use imitation learning to try and learn how to best play the game perhaps by watching the AI and TA agents play against each other. However, this wasn't really feasible in our case and we wanted to play it safe by following an approach that we were already more familiar with from homework 5. By doing this project, we have become much more interested in how vision-based agents are able to operate for use cases such as driving (self driving cars are the perfect case study)!