Final Project - SuperTuxKart ice hockey

Group I: Jennifer Hsu, Henry Zhong, Siddhartha Edara, Andre Duong

1 Abstract

In this project, we built an image-based agent that learns a SuperTuxKart hockey game environment and runs a controller to navigate and score points against opposing teams.

2 Data Collection and Model Training

Data is collected through recording matches between different agents. Data includes images taken from the player point-of-view and the projection of the puck on the player image.

2.1 Training agent selection

We started collecting data from AI against AI matches first. but the agents were playing the game without crashing, making the model unable to accurately predict puck locations when the kart is stuck on the playing field. To provide a more diverse set of training data, we coded up a random agent that steers in a random angles under a random probability. Agents that use the ground truth and current best model were also used to to generate training data.

2.2 Roll out of multiple matches

Even with running matches between the random agent vs AI, we were still not getting a diverse set of training data. To resolve this, we loaded multiple match rollouts, and modified the code to take multiple recording data files (.pkl).

2.3 Data augmentation

To increase training data variety, we implemented random horizontal flip and random initial puck location in our training

2.4 Continuous training

To increase the robustness of our models further, we continued to fine-tune well-performing models with data from games played by the latest controller running. We used this iterative approach to slowly improve the accuracy of the model for a specific controller strategy.

3 Model Design - environment ground truth

3.1 Is the puck in frame?

For training, we tried different approaches to handle cases where the projection of the puck location from _to_image() is negative meaning the puck is out of frame.

Homogeneous coordinates. We tried to find the pattern between the homogeneous projection coordinates and whether the puck is on-screen. For most cases, when the *z* and *w* coordinates in *(x, y, z, w)* are both positive, the puck is on-screen. Yet, there are edge cases when the puck is too close or too far from the kart.

Dot product. We can determine whether the puck is in front of the kart through taking the dot product between the vector pointing from the kart to the puck and the vector pointing to the front of the kart.

```
func is_puck_visible():
    v1 = sub(puck, player)
    v2 = sub(player_front, player)
    return dot(v1, v2) > 0
```

But this approach also returns *True* for pucks 180 degrees from the kart's view, which is smaller than the angle provided in the image taken by the kart

Detect object type from pytsk. From pytsk documentation, there is an object_type property which we can determine if the puck is on-screen. This proves to be a significant improvement from the dot product approach, with the MSE validation loss dropping from 0.040 to 0.003.

4 Model Design - ConvNet

4.1 One ConvNet model for puck position on an image

Our final model is structured as a 4-layer convolutional neural network model. Given an image take from the view of the player kart, we predict a heatmap of the object's location and obtain the spatial argmax for the position of the puck. Previous iteration of our model included a FCN model and, a CNN model that predicts both the puck coordinate as well as whether it is on screen.

Hard coding ground truth labels when puck is not in view.

Bottom left and bottom right. Based on the image projection, when the puck is out of view, we can determine from the world coordinates and assign bottom left or bottom right image coordinates.

It turned out that the model was having a hard time predicting whether the puck is on the left or right when the puck is off-screen.

Center bottom. To resolve the issue above, we tried hard coding the ground truth to be the center bottom [0, 1] (normalized) if the puck is not in view and resulted in a better model.

4.2 Experimenting with two models

We experimented with splitting the original ConvNet model into two different models, one for predicting the puck camera coordinate when it is on-screen and another predicted whether the puck was on screen or not. Both used the same 4 layer CovNet model, but the puck camera coordinate model used the MSELoss function, while the puck is on screen model used the BCEwithLogitLoss function. Ultimately, 2 models weren't necessary as when the puck is not on-screen, the puck camera coordinate is hard-coded as the bottom center point.

5 Controller Design

5.1 Approach

The controller at a baseline uses the model's predicted puck coordinates, the goal location, and proximity information to determine an aim target and then moves towards the target at a maximum determined velocity. In most cases we steer with acceleration 1, unless the puck is close to the player, in which case we slow the kart down to improve puck control for the player.

Rescue In addition we also implemented a rescue function in case the kart got stuck. We determined stuck as the current kart velocity being less than 1. For rescue, we either use the kart's rescue action or backup at a random direction. We found alternating between the two actions helped decrease situations where a player is stuck for multiple frames at a time.

Dribble We tried a dribble strategy which the player would aim for a point slightly to the side of where it should hit the puck. This works when the kart is navigating at a lower speed.

Player 2 controller strategy Since the objective is to score as many goals as possible, both players follow the same logic to push the puck into the opposing goal.

5.2 Alternative Approaches

There were many modifications to the baseline controller that we tried which didn't provide much help. To list a few we tried building rescue based on the kart's location which didn't help as in certain situations the kart would be stuck but slightly move, we tried smoothing out the visibility of the puck on screen by taking an average over multiple frames instead of deciding an action per frame but the result increased erratic behavior.

6 Additional approaches / utilities

6.1 Plot ground truth and prediction labels during game play

To better view the predicted puck positions, we implemented a utility that plots both the ground truth and prediction labels during match roll out. This was a big breakthrough on our progress and led us toward **2.2 Random agent vs AI**, **2.4 Roll out of multiple matches**, and **2.4 Continuous training** approaches.

6.2 Change kart types

The karts in Pystk can be classified into 3 different weight categories: light, medium, and heavy. Research shows that light- weighted karts are the easiest to drive because they turn easily. Light weighted karts also have a fast boost start that is extremely important for reaching the puck before opponents.