CS 3630 Project 4 Report

Name: Moungsung Im
GT email: mim30@gatech.edu
GT username: mim30

1.1) What is the shape of the feature map after an image from the kMNIST dataset is passed through the first convolution layer of SimpleNet?

[5, 10,10]

1.2) Count the Number of Parameters:

- Relu Layer: 0
- AvgPool2D: 0
- The last Linear Layer (self.linear1): 490

1.3) What will be the size of the output when we apply the nn.AvgPool2d function on an input image of dimension [3x466x700]?
(kernel size: 4x4, stride: 1, padding: 0)

3 * 463 * 697

1.4) In one line, explain how average pooling is different from max pooling.

Average pooling averages the operation over an input window of a given size and max pooling maximizes the operation over an input window of a given size.

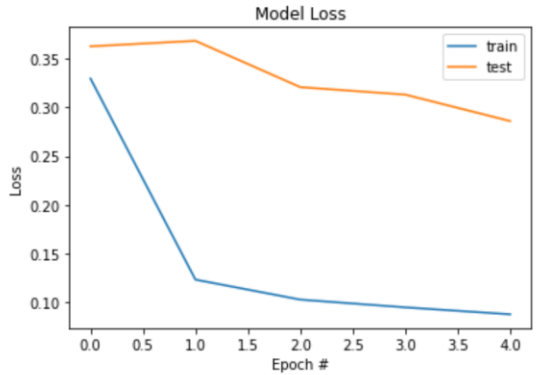
1.5) Try different combinations for the set of hyperparamters:
- batch size = {32, 64, 128}
- learning rate = {0.01, 0.05, 0.005, 0.001}
- You may also increase the number of epochs

Which set of parameters worked best for you? Explain your findings.

I had the best result when batch size is 32 and learning rate is 0.01.
The reason would be that as we increase the batch size, it increases the running time. In addition, as the learning rate decreases, the loss increases.

1.6) Paste a screenshot showing the loss vs epoch of your LeNet. State the train and test accuracies.

```
# Plot your train and test losses
plot_losses(train_losses, test_losses)
```



Print train/test accuracy using the accuracy function.

```
[78] accuracy(train_loader, model)
     accuracy(test_loader, model, train=False)
```

```
Train Model Predicted 58894 correctly out of 60000 from training dataset, Accuracy : 98.16
Test Model Predicted 9321 correctly out of 10000 from testing dataset, Accuracy : 93.21
```

2.1) Read through the code in the Setup section. Answer the questions below.

1. How many obstacles are in the environment? Exclude the walls along the corners of the environment.

   There are 10 obstacles.

1. What does the integer ‘0’ in the images_list (ex. Image([0,3], ‘e’, **0**) represent?

   There is an image, with **label number zero**, at x = 0, y = 3, and it can be seen from the east because the image is facing the east. Therefore 0 means the label number of image.

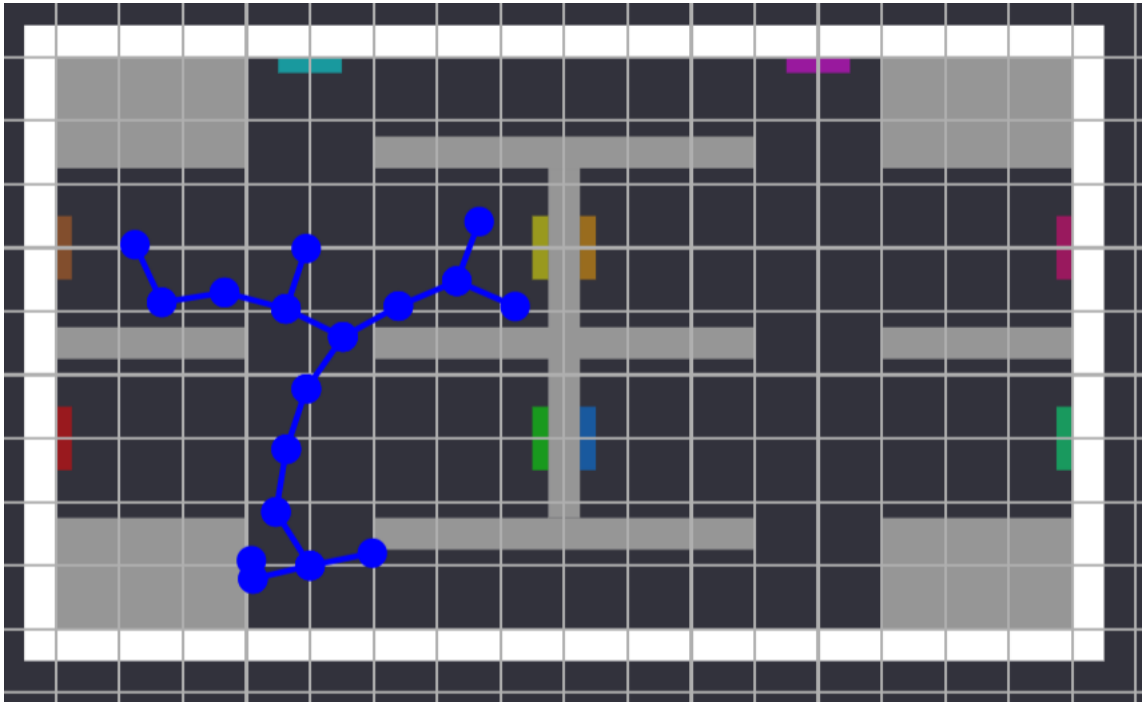
2.2) When randomly sampling points for RRT, what is the purpose of returning the goal position with some probability?

So that the robot can eventually reach to the goal node, but since it randomly returns random node, it can expand the tree in the given map.

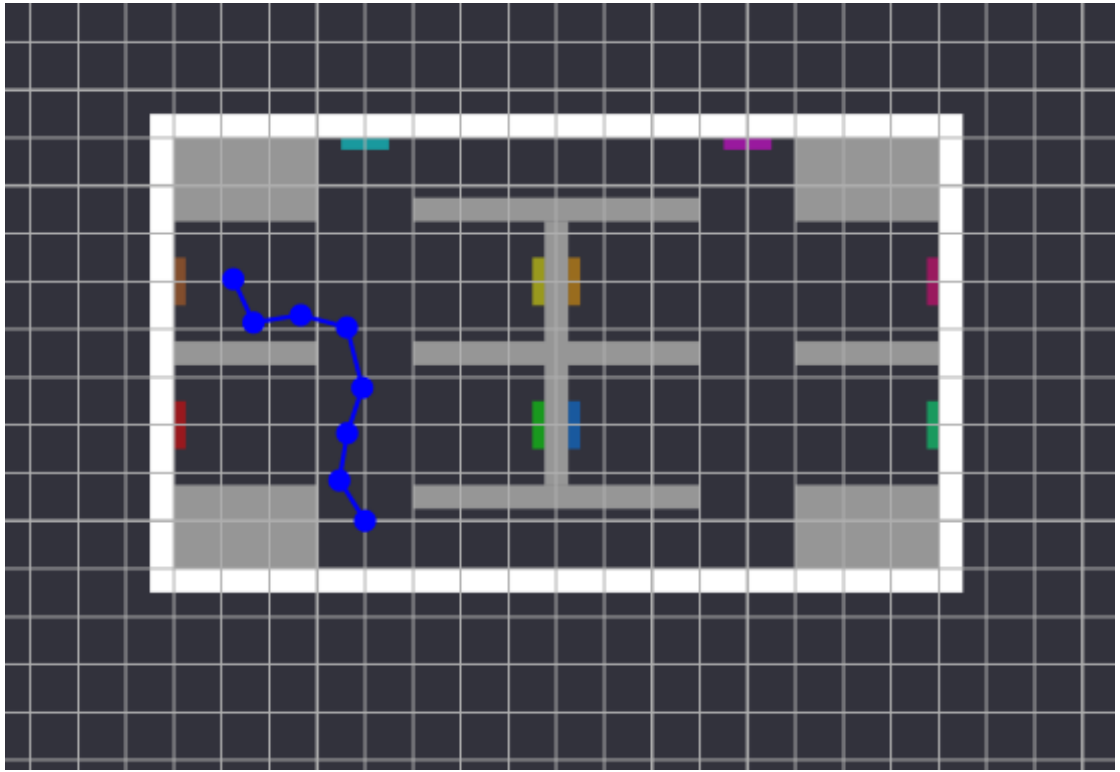
2.3) Write the high-level pseudocode for the RRT algorithm here. Add an explanation of each step beside the pseudocode.

```
While True: //loop until we reach to the goal
        random_node = ramdomSample from env with goal //get random node from env which targets the goal point.
        near = getNearest(rrt, random_node) //get nearest point to random node from rrt
        new_node = stepTo(near, random_node) //step toward random_node
        if inObstacle(new_node, env) or isCollosion(near, new_node) //if the robot get to the target point, continue.
        new_node = near //update new_node
        rrt_tree.append(new_node) //add current location to RRT
        if withInTolearnace(new_node, goal) //if the robot finally reaches to the goal,
                path.append(new_node) //add the final point to the path
                 rrtcopy = rrt_tree[:] //copy rrt
                curr = new_node //set curr new_node to traverse the rrtcopy backward
                while curr!= root: //while the current point is not root node
                        if curr in rrtcopy //remove current node from rrt copy
                                rrtcopy.remove(curr)
                        goback = stepTo(curr, root) //goback is the direction next node from current node
                        nearest_node = getNearest(rrtcopy, goback) //nearest node is the next(previous in the perspective of path) node
                        if isCollision((curr, nearest), env): //however if it collides to the wall, remove this scenario
                                rrtcopy.remove(nearest_node)
                                rrtcopy.append(curr)
                                continue
                        curr = nearest_node //if it doesn't collide, add the node to the path.
                        path.append(nearest_node)
                if withInTolerance(curr, root): //if the robot eventually reaches to the start point, reverse the path and return with the original RRT.
                        path.append(root)
                        return path.reverse(), rrt_Tree
```

2.4) Attach a screenshot of the rapidly-exploring random tree (RRT) generated to find the path from [4,1] to goal position [0,6].

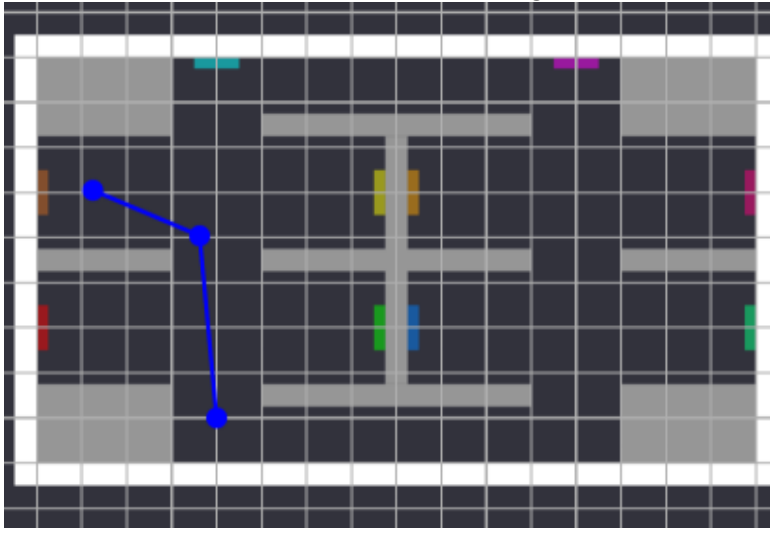


2.5) Attach a screenshot of the path found from the tree in 2.4 without path smoothing.

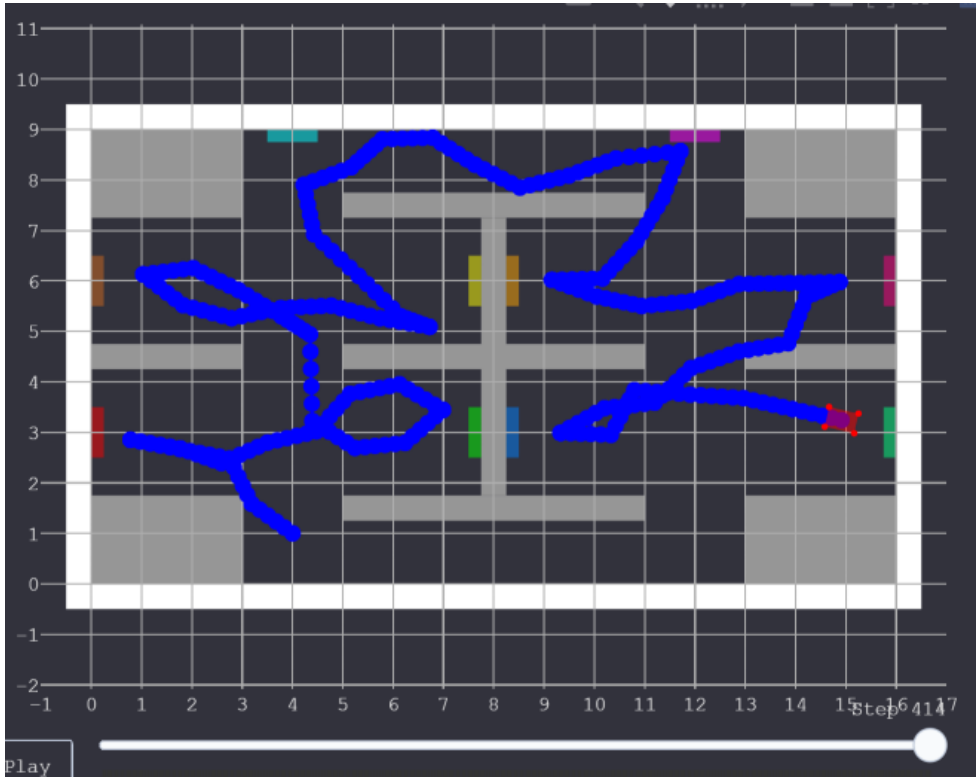


2.6) Attach a screenshot of the path found from the tree in 2.4 with path smoothing. Explain the differences with and without smoothing. If you haven't implemented smoothing, just explain the expected differences in the path when smoothing is applied.

The difference between paths with smoothing is that the robot doesn't have to visit a point if its next point isn't blocked by the wall. Then the robot can efficiently skip unnecessary points while it is reaching to the goal.



3.1) Attach a screenshot of the paths generated by the robot to look at all ten images in the museum.



3.2) The RRT algorithm implemented in this lab plans paths composed of straight-line segments. How would you edit the differential drive section to plan smooth curved paths?

In order to curve the path, the robot must rotate when it move forward at the same time.

3. Feedback

Please provide feedback on the coding portion of the project. How did it help your understanding of the material? Is there anything that you think could have been made more clear?

This is very interesting and fun assignment while the material is hard. Wish the speed of robot gets faster for testing.