CNN for Robot Localization

Paul Yoo

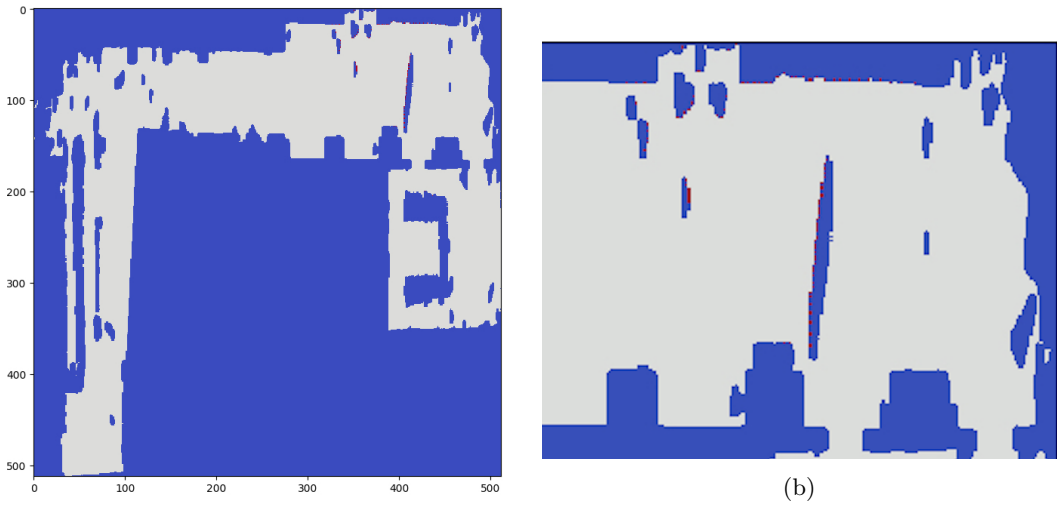
June 2020

1 Summary

We tried to solve the problem of localizing a robot given an occupancy map and laser scan measurements using neural networks. There are many different ways to design a neural network to fit our problem. However, we ended up using a CNN since localizing a robot can be viewed as classifying which part of the image the robot is likely to be in. In order to make CNN compatible with our problem, we had to separate the map into small grids and let the CNN output which one of the grids the robot is most likely located in given a map of the environment with laser markings.

2 Dataset

The input to our model is a map of the environment that encodes which parts are occupied and which parts are not. Additionally, the map contains the positions of the laser scans. We collected 100 different robot configurations from each occupancy map we were provided and generated laser measurements for each of those 100 configurations. The position of the robot was sampled randomly from the unoccupied space and the heading angle of the robot was also sampled from a list of possible angles. During training, the model learned from various maps and configurations of the robot. Since the maps are of different sizes, we resized all maps to 512 by 512 in order to fit the network and produce consistent output. Although the map was resized, we sampled positions and headings and collected laser measurements from the resized map in order to address any inconsistencies with scales. In terms of training and testing sets, we allocated 60 maps for our training set and 40 maps for the testing sets.
Figure 1 shows the one of the dataset image we gained, B is a larger and part version of A, and we can see the red dots are the obstacles detected by laser scan in image B.



(a) (b)

Figure 1

3 CNN structure

Since the output space will be super big if we put the heading in the CNN model, so we put it out, we flatten out the two dimensional label [x position, y position] and train the dataset only one time in the model, CrossEntropyLoss is our loss function and Adam as our optimizer with the learning rate is 0.001. Figure 2 shows our basic cnn model.
Architecture:
Conv(Relu) → MaxPool → Conv(Relu) → MaxPool → Linear(Relu) → Linear(Relu) → Linear

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 125 * 125, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, GRIDS)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x.float())))
        x = self.pool(F.relu(self.conv2(x)))
        #print(x.shape)
        x = x.view(-1, 16 * 125 * 125)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Figure 2: CNN model

4 Input / Output

The output of the model is a list of grids on the map where the robot can possibly be located in. The output contains different probability values for these grids and the most likely location of the robot can be obtained from grids with highest probability.

5 Result

We have taken the top predictions output by the CNN model and visualized the points. In most cases, the ground truth position of the robot falls under one of the top predictions. In cases where the terrains look similar, the model's uncertainty can be seen through dots scattered over the map that would result in similar laser measurements. figure 3 shows multiple results we tested, the left image is the heatmap that indicates the top likely position where the robot in the map. And the right image is the occupancy map and the robot's true position (red dot).

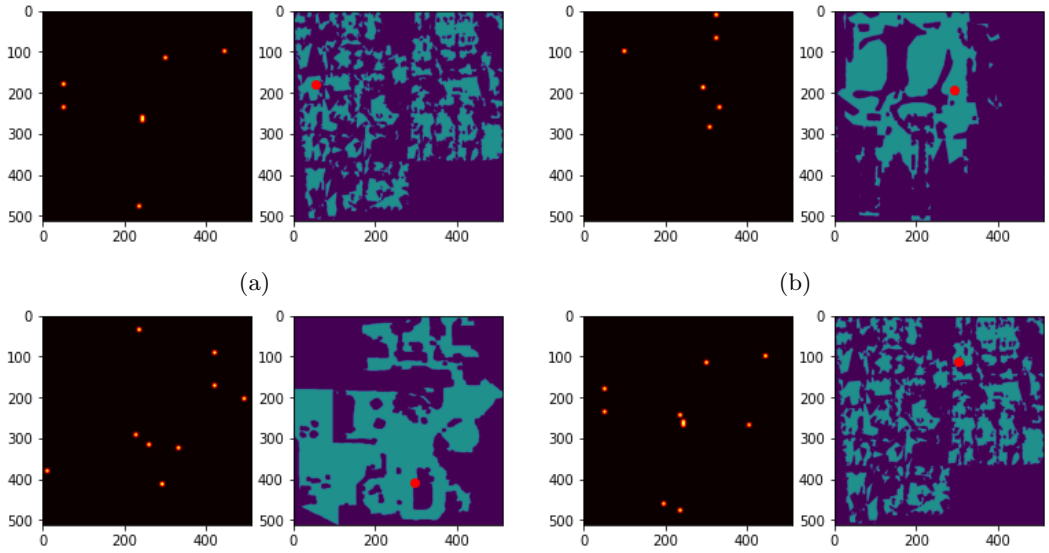


Figure 3