Comp Aspects of Robotics HW4

ms5510

December 2020

1 Problem 2

Algorithm:

I followed the algorithm given in the HW4 handout, but then found that the robot would get stuck in LEFT-RIGHT-LEFT or RIGHT-LEFT-RIGHT continuous loops, as the turning angle was not small enough such that the calculated "middle point" of the observed line would coincide exactly with the middle point of the image, at which point the robot should go forward.

I devised the following algorithm to be followed in that case:

1. A boolean variable flag is set to False outside the function (as an instance variable), which signifies whether the robot has been caught up in a LEFT-RIGHT loop yet.

2. Previous actions and the values of the midpoints of the threshold intensity plots of the lines are stored outside the function as instance variables, with initial values as FORWARD and 0 resp. They are updated at the end of the function.

3. If the $current\_action = LEFT$ and $previous\_action = RIGHT$ or $previous\_action = LEFT$ and $current\_action = RIGHT$ (essentially the robot is caught in a loop), then calculate the difference between the midpoint of the image window (=112) and the midpoints of the threshold intensity plots for both the current and previous actions.

For whichever one is smaller (meaning the line is closer to being aligned with the midpoint for that direction), the robot is assigned that corresponding action.

4. The action is implemented for that step of the iteration (calling of the function).

5. In the next calling of the $predict\_action()function$, flag is set to True.

6. There is a conditional statement at the end of the function that states that if the flag variable is set to True, then it means that the robot just got

out of the LEFT-RIGHT loop. Therefore, the next action must be FORWARD to move the robot forward, so inside the conditional, the action is assigned to FORWARD, and the flag is updated to be False, since the robot has moved on from the LEFT-RIGHT loop.

I also added one more condition to the original algorithm that the robot should continue to turn RIGHT if it encounters a blank, black square in its robot view window.

Performance on test maps:
The algorithm passes all 8/8 landmarks in maps/test/map1 and all 6/6 landmarks in maps/test/map2.

2 Extra credit

Algorithm:
Steps or important features of the model to make it work:
1. Parameters– I utilized an 85% train-test split, batch size = 5.
2. Model-specific parameters– I chose to use a Stochastic Gradient Descent optimizer with learning rate= 1e-3, a Cross Entropy Loss loss function, and to run the model for 5 epochs.
3. I chose to unfreeze all the layers of the network, as that improved the accuracy of the model by almost 10% (even though it took a while to train). I believe this is because the earlier layers in the network were trained on a very different kind of data (ImageNet), so allowing them to be re-trained accounts more for the new dataset.
4. Also, instead of 1 fully-connected layer, from 512 to 4 feature units, I decided to have 3 fully-connected layers, from $512 - - > 256$, $256 - - > 128$, $128 - - > 4$, which I believe helps to progressively down-sample and thus capture smaller features in the data.
5. Also, I increased the training set's size (and thus increased robustness, especially for the LEFT move) by going through each map in both the RIGHT and LEFT directions, while also going around certain maps twice.

The model performed very well and passed all of the landmarks in all 3 test maps.