CSE 571 - Robotics
Guided Project 2 - 2D Navigation

1 Description

In this project, you will select a topic related to localization, planning and control for 2D navigation. We will provide you with a set of occupancy maps and a laser scanner simulator for you to generate training data and run simulations. You may borrow code from the three homework assignments as you wish.

You will work in the same team as your first project. In addition to restricting scope and environment, teams MUST use PyTorch (as opposed to other deep learning frameworks) in order to implement their networks.

2 Timeline

- Mid-Progress Report: due on Tuesday 05/26/20.
- Poster Presentation: 10:30am - 12:30pm on Monday 06/08/20.
- Final Report: due on Tuesday 06/09/20.

3 Deliverables

To give you more time for this project, you **are not** required to submit a proposal if you choose one of the suggested project ideas (see next page). If you decide to work on your own idea, please submit a proposal (no need to be formal) to us similar to the first guided project. Please submit the proposal early so that you can work on the project as soon as possible.

The mid-progress report will be hosted on the same Google Doc used by your first project. For the final report, you will submit a PDF so that you can use Latex.

Mid-Progress Report [1-page] The mid-progress report should report on successes and unforeseen problems. If the timeline/project outcome needs to be updated, please make those changes in the web-based blog, and note this in the mid-progress report. We highly encourage visualizations, e.g. images and/or short videos.

Final Report [2 pages] The final report will summarize the guided project, not including references.

Poster Presentation There will be a poster session during the final exam week so that students can check out and discuss each other's project in breakout rooms on Zoom. Each team will prepare slides and a short 3-minute presentation video by speaking to the slides. Finally, teams must submit executable code (with a README) so the TAs can replicate the results. More details will come later.

Dataset

We provide you with around 100 occupancy maps from real world environments for you to generate training data and run simulations. You should perhaps use 90 maps for training and 10 maps for testing. A link to the dataset will be posted on the Ed discussion board. Please do not distribute the dataset outside the class!

Utilities

We provide you with a set of utilities for you to read maps, generate laser scans and visualize results. You may freely modify the code and adapt it to suit your needs. Check out `README.md` in the code directory for more information.

Suggested Project Ideas

Localization

This task is to predict robot state $[x, y, \theta]$ given an occupancy map of the environment M and laser scan measurements $d_1, d_2, ..., d_N$. Figure 1 illustrates the input/output of the task.

Example ideas are:

- Predict a heatmap of robot's location and heading given robot's current laser scan measurement. You may treat this as a classification problem, where each class is one of the candidate robot states.
- Similar to above, but instead learn the distribution of robot's location and heading directly using a conditional variational autoencoder (cVAE) [1]. You can then sample robot's state $[x, y, \theta]$ from the model.
- (Advanced) Predict a heatmap of robot's location given robot's observation history (multiple laser scans). Intuitively, as robot traverses inside an environment, it will get increasingly confident about its current location. To generate a trajectory, you may use A* or RRT. You may get some useful ideas from this paper [2].

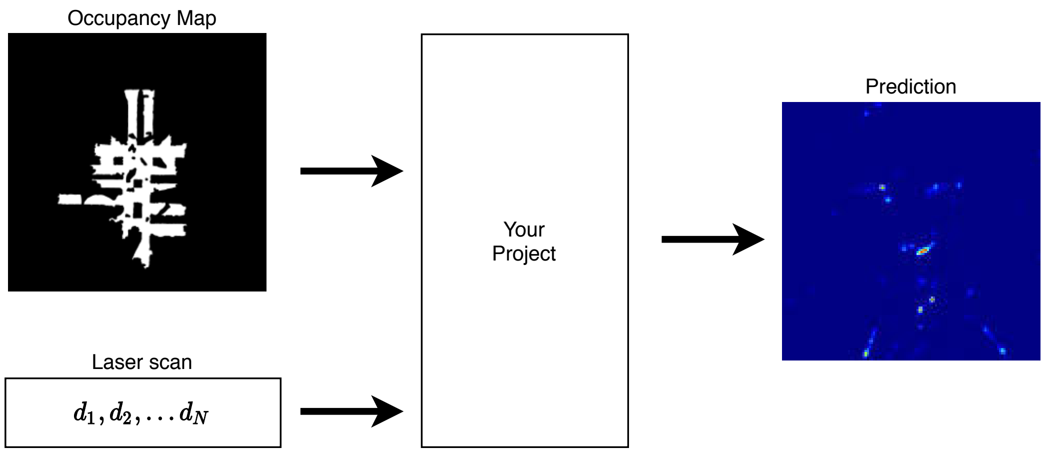

Figure 1: Localization

- (Advanced) Implement Differentiable Particle Filter for localization [3].

Advice on the design/implementation of this task:

- The original occupancy maps are fairly large. You may resize them to a resolution suitable for a convolutional neural network. A size between 128×128 to 256×256 is a good starting point. If you decide to use networks pretrained on ImageNet (e.g., ResNet-18), they usually take images of size 224×224.
- One trick to improve network generalization is data augmentation. Since we only have 100 maps available, feeding an entire map into the model may not make the network generalize to unseen maps. You may randomly crop a map to use only certain part of it during training. When testing, you can do the same thing: extract patches from the map, run your model on each patch and stitch the results together.
- You can represent laser scans as either depth values or $[x, y]$ coordinates. You may experiment with both to see which one works best for you. Note that when you use the $[x, y]$ representation, the coordinate system is w.r.t the robot's laser scanner.
- For networks that produce heatmap-like output, you may get useful ideas from papers on keypoint estimation [4, 5].

Planning

This task is to learn a neural network model that imitates a classical planner such as A* or RRT. Your model takes an occupancy map of the environment M, robot's current state $s_0 = [x_0, y_0, \theta_0]$, and the goal location $s_g = [x_g, y_g]$. Your model then generates a collision-free trajectory that connects s_0 and s_g. Example ideas are:

- Learn a planner that takes M, s_0 and s_g, outputs action a robot should execute. Robot executes a for a small time step (e.g., 0.1) to get an updated s. By repeating this, you will get a sequence of robot states connecting s_0 and s_g. Note that action should respect robot's motion constraint for a non-holonomic robot (e.g., the one in homework 3). You may get useful information from [6, 7].
- (Advanced) Implement Value Iteration Networks [8].
- (Advanced) Implement Latent-Sampling Based Motion Planning [9]. This approach can learn to plan from pixels. If implementing this, we would require a more complex system such as the non-holonomic car from HW3, a point robot with single integrator dynamics (as mentioned in the paper), or a point robot with double integrator dynamics.

Advice on the design/implementation of this task:

- If the original maps are too big for you to get any meaningful results, try randomly cropping the map to reduce map complexity. This may ease learning.
- s_0 and g are global coordinates. It might not be a good idea to feed these directly to the network. You may try other representations such as normalizing x, y into the range of $[0, 1]$, or using two separate image channels to graphically represent s_0 and g.
- You may train your network with A* or RRT as supervision. For initial experiments, you may assume the robot to be a point and can move in any direction (i.e., ignore heading). In that case A* is sufficient.
- Once you get the point robot working, try the non-holonomic robot model in homework 3. Note that s_g does not contain heading. This could speed up data generation when using RRT.
- For an ultimate challenge, train the network with Reinforcement Learning with partial observation similar to [10] (not recommended unless you are familiar with RL).

Control

As you may have noticed in homework 3, RRT can be slow for non-holonomic robots. Can you learn a controller for a non-holonomic vehicle (e.g., the one in homework 3) that can get to a nearby location, while avoiding obstacles along the way? This is useful for avoiding dynamic obstacles like humans.

The input to the model is current laser scan $o = [d_1, d_2, ..., d_N]$ and waypoint $g = [x, y]$ (relative to robot's current location and heading). You may convert the laser scan depths into local $[x, y]$ coordinates by transforming the coordinates using robot's current state $[x, y, \theta]$, but the model should not use any global information. The output of the model is the action $a = [v, \omega]$ which consists of velocity and steering angle. Robot executes a for a small time step (e.g., 0.1) to reach a new state, after which it gets an updated laser scan measurement and relative waypoint location. Robot repeats this process until it reaches the waypoint.

Here is a video showing how the obstacle-avoidance behavior looks like for a car-like vehicle: `https://courses.cs.washington.edu/courses/cse571/20sp/projects/sim_lidar3.mp4`. Note that this is just an illustration (the car in the video is different from the one in homework 3). You are not required to implement exactly what is shown in the video.

Example ideas are:

- Train a MLP network that takes o and g as input, and predicts action a with RRT as supervision. Feel free to use other controllers (e.g., Model Predictive Control (https://engineering.purdue.edu/~zak/ECE680/MPC_handout.pdf), RMP[11], etc.) as supervision.
  - To generate training data, you could randomly choose an occupancy map, then randomly select a start and goal point, and run RRT to generate a trajectory. Use the laser scan measurement at the starting point as o and use the goal point (relative to the start point) as g. The first action in the trajectory can serve as ground truth for your network.
  - You may experiment with recurrent networks (RNN, LSTM) to see if they improve the performance.
- Implement Motion Planning Networks [6].
- (Advanced) Learn obstacle avoidance with Reinforcement Learning [12].

Advice on the design/implementation of this task:

- Your model should not take any global state as information. However, when you generate training data and run simulations, you could still maintain the global state of robot $[x, y, \theta]$.

Resources

- If you don't have access to a machine with a GPU, you can use Google Colab (access to a free GPU for up to 12 hours per session). You only need a Google account. Colab operates as notebooks, very similarly to Jupyter Notebooks if you have used them. PyTorch/TensorFlow is pre-installed in the Colab notebooks, so no need for manual installation. You can find more information here: https://colab.research.google.com/notebooks/intro.ipynb
- PyTorch: A framework that allows you to do Numpy-esque computations on GPU. It also has a lot of support for autodifferentation and neural networks, making it one of the most popular deep learning frameworks. You are required to use this framework for the guided projects. See tutorials here: https://pytorch.org/tutorials/

References

[1] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[2] Devendra Singh Chaplot, Emilio Parisotto, and Ruslan Salakhutdinov. Active neural localization. *arXiv preprint arXiv:1801.08214*, 2018.

[3] Rico Jonschkowski, Divyam Rastogi, and Oliver Brock. Differentiable particle filters: End-to-end learning with algorithmic priors. *RSS*, 2018.

[4] Jiajun Wu, Tianfan Xue, Joseph J Lim, Yuandong Tian, Joshua B Tenenbaum, Antonio Torralba, and William T Freeman. Single image 3d interpreter network. In *European Conference on Computer Vision*, pages 365–382. Springer, 2016.

[5] Chen-Yu Lee, Vijay Badrinarayanan, Tomasz Malisiewicz, and Andrew Rabinovich. Roomnet: End-to-end room layout estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4865–4874, 2017.

[6] Ahmed H. Qureshi, Anthony Simeonov, Mayur J. Bency, and Michael C. Yip. Motion planning networks, 2018.

[7] Ashwini Pokle, Roberto Martín-Martín, Patrick Goebel, Vincent Chow, Hans M Ewald, Junwei Yang, Zhenkai Wang, Amir Sadeghian, Dorsa Sadigh, Silvio Savarese, et al. Deep local trajectory replanning and control for robot navigation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5815–5822. IEEE, 2019.

[8] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162, 2016.

[9] Brian Ichter and Marco Pavone. Robot motion planning in learned latent spaces. *IEEE Robotics and Automation Letters*, 4(3):2407–2414, 2019.

[10] Arbaaz Khan, Clark Zhang, Nikolay Atanasov, Konstantinos Karydis, Vijay Kumar, and Daniel D Lee. Memory augmented control networks. *ICLR*, 2018.

[11] Xiangyun Meng, Nathan Ratliff, Yu Xiang, and Dieter Fox. Neural autonomous navigation with riemannian motion policy. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8860–8866. IEEE, 2019.

[12] Guangda Chen, Lifan Pan, Yu'an Chen, Pei Xu, Zhiqiang Wang, Peichen Wu, Jianmin Ji, and Xiaoping Chen. Robot navigation with map-based deep reinforcement learning. *arXiv preprint arXiv:2002.04349*, 2020.