CSC477 – Introduction To Mobile Robotics

assignment 4, 15 points
due: Dec 4, 2019, at 6pm

Course Page: http://www.cs.toronto.edu/~florian/courses/csc477_fall19

Overview: In this assignment you will implement a particle filter for Monte Carlo localization. You are also given the option of working on one of two possible bonus questions for an extra 3 points. The first bonus question is about computer vision, specifically about stereo depth estimation on a self-driving dataset. The second possible bonus question is about deploying your wall following code from assignment 1 on one of the self-driving racecars. You may only submit work for one bonus question.

1 Monte Carlo Localization (15 pts)

In this exercise you are going to implement Monte Carlo Localization (i.e. localization in a known occupancy grid map, using particle filters), as discussed in class. Your robot is going to start by being completely lost in the environment, so particles are going to be spread out uniformly at random in the known world. After many LiDAR measurements, the robot's pose is going to be constrained and the particles are going to converge to a small cluster. The main mechanism for survival of the fittest among particles will be: which particles are more likely to have generated the laser scans that the robot is observing?

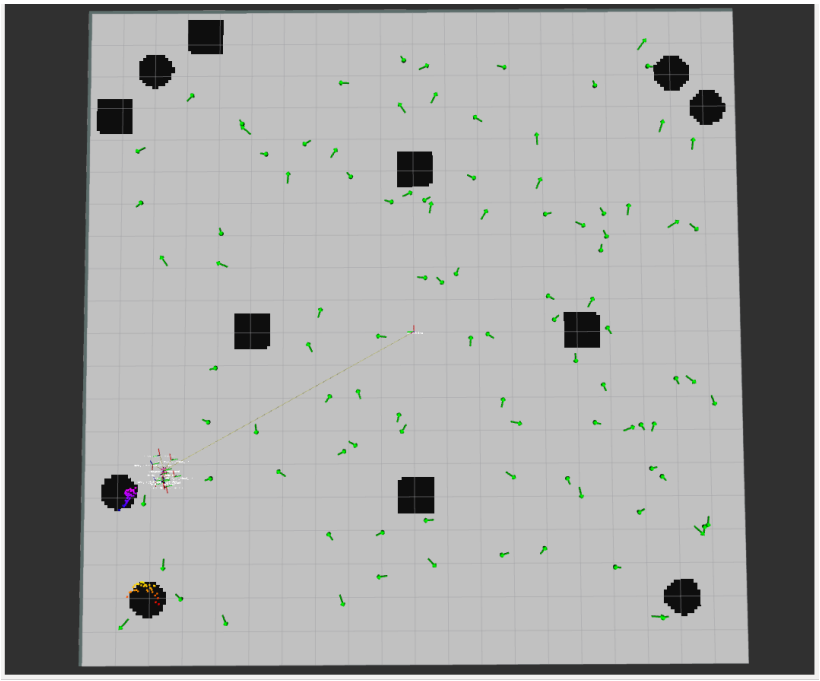


Figure 1: Uniform initialization of particles in Monte Carlo Localization, within the boundaries of a workspace. Your task is to make the particles cluster around the robot, as it makes LiDAR measurements of the environment. The environment consists of the black square and circular-shaped obstacles, shown above.

Starter code

Do `git pull` under your CSC477 repository to get the starter code. The functionality that you need to implement is marked using comments in the file `estimation_and_vision_assignment/python/monte_carlo_localization_v2.py`. To run your code, `cd path/to/csc477_fall19/estimation_and_vision_`

`assignment/` and execute the following commands on three different terminals:

```
rosbag play laser_controls_and_odometry.bag
roslaunch estimation_and_vision_assignment monte_carlo_localization_v2.launch
rosrun rviz rviz
```

When rviz initializes, go to `File` > `OpenConfig` and then load the configuration file in `estimation_and_vision_assignment/resources/comp417.rviz` which is going to start the visualization of laser scan messages, frames of reference, and particles. Save this configuration file as the default in your `/home/username/.rviz/default.rviz`, so you won't have to do this every time you restart rviz. What you will see initially, before the robot makes any measurements is Fig. 1: What you need to submit: in addition to your code, a video recording of the rviz visualization demonstrating your particles converging from beginning to end. Your video should be named `mcl_firstname_lastname.mp4/avi/ogg`.

2 BONUS QUESTION: Depth from stereo disparity (3 pts)

In this exercise you are going to solve the depth from stereo disparity problem that we discussed in class. You will be given pairs of stereo-rectified images from the KITTI and Middlebury datasets, such as the ones outlined in Fig. 2. The KITTI dataset is one of the main ways to benchmark visual tasks that are of relevance to self-driving cars. Note, in the last two years there have been much larger, and more diverse datasets for self-driving, so KITTI is slowly becoming obsolete for some tasks. The Middlebury dataset

is one of the oldest computer vision benchmarking services out there, and has become very popular for its stereo evaluation suite. You are encouraged to explore these datasets to become aware of what they offer. In the meantime, we are going to use the following rectified stereo image pairs for this assignment:

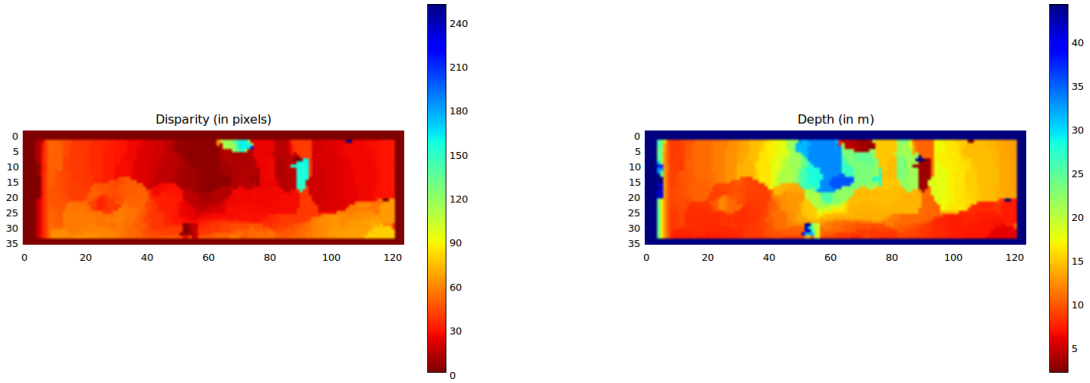
For each of these image pairs you will compute the disparity map in pixels, using block matching along horizontal epipolar lines, as we saw in class. Note that this is definitely not the state-of-the-art algorithm for stereo disparity, and your implementation will most likely not run in real-time, but it is enough to get you thinking about the problem.

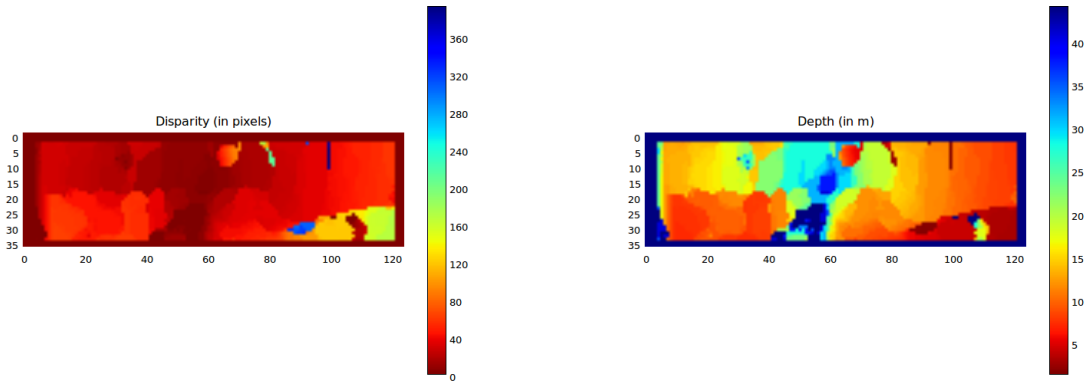
Figure 2: Stereo image pairs 1 to 6 are from the KITTI dataset. The baseline between the cameras is about 0.5 meters. The 7th image pair is from the Middlebury stereo dataset. The baseline is about 0.2 meters.

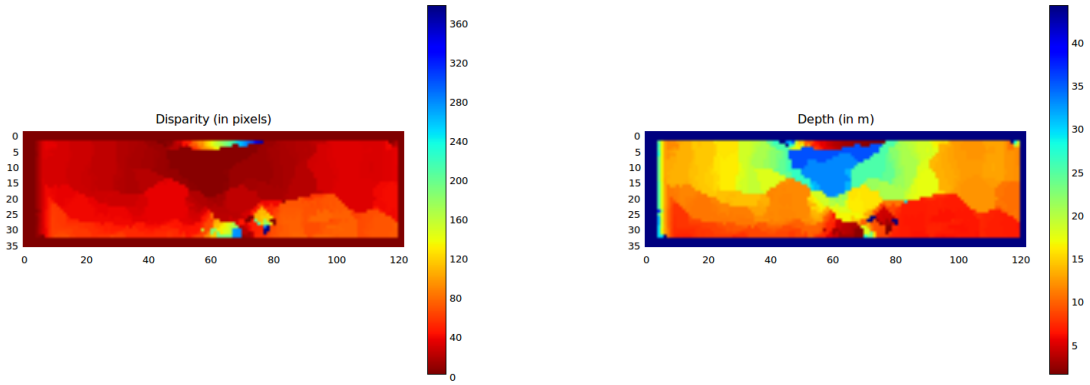
Given the stereo camera calibration and rectification parameters you will compute the depth map in metric coordinates from the pixel disparity map. For example, these disparity maps have been obtained with patches of width 64 pixels, and the sum of squared differences as the distance metric for matching patches along the horizontal epipolar lines. The most common options for a distance metric for comparing similarity between image patches are:

- sum of absolute differences (SAD), $\sum_{i,j} |L_{ij} - R_{ij}|$, where L,R are the left and right patches being examined.
- sum of squared differences (SSD), $\sum_{i,j} (L_{ij} - R_{ij})^2$
- normalized cross-correlation (NCC), $\sum_{i,j} \frac{(L_{ij} - \mu_L)}{\sqrt{\sum_{ij} L_{ij} - \mu_L}} \frac{(R_{ij} - \mu_R)}{\sqrt{\sum_{ij} R_{ij} - \mu_R}}$

The expected results you should get look approximately like this:



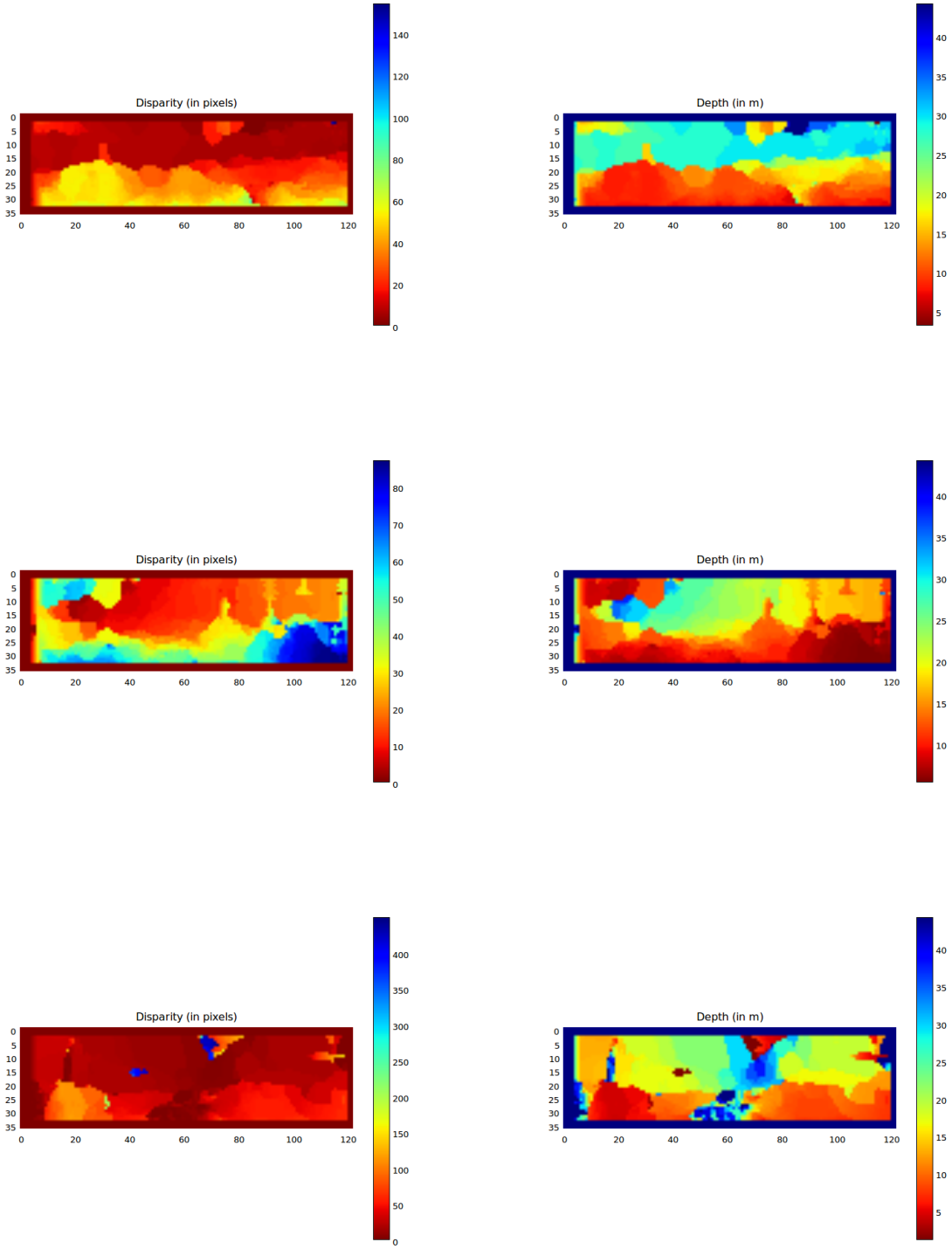




Starter code

You are given starter code in your CSC477 repository under `estimation_and_vision_assignment/python/stereo_disparity_map.py`. Do `git pull` from your CSC477 repository in order to receive it. The way to run this code is:

```
./stereo_disparity_map.py ../latex/stereo_images/images0/kitti_1_left.png ../latex/stereo_images/images1/kitti_1_right.png ../resources/kitti.yaml
```



and similarly for the motorbike photo: `./stereo_disparity_map.py ../latex/stereo_images/images0/middlebury_1_left.png ../latex/stereo_images/images1/middlebury_1_right.png ../resources/middlebury.yaml`

The `.yaml` files contain rectification parameters and other parameters for the two datasets.

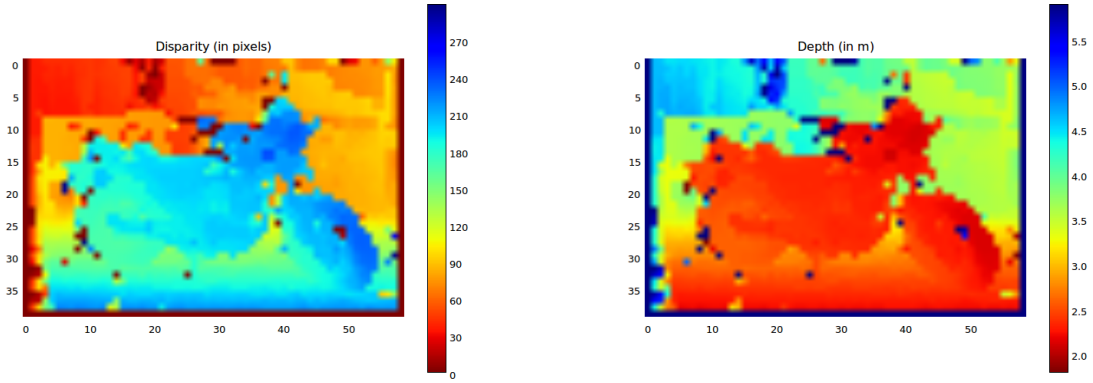


Figure 3: Example disparity and depth maps for the image pairs presented above. No post-processing has been applied to the disparity maps to encourage smoothness.

What to submit

Your implementation of disparity and depth estimation in `estimation_and_vision_assignment/python/stereo_disparity_map.py` using sum of squared differences for matching. Also, the following 14 images: 7 disparity maps and 7 depth maps for each of the image pairs mentioned above. Your files should be named `kitti_[1-6]_disparity.png`, `kitti_[1-6]_depth.png`, `middlebury_1_disparity.png`, `middlebury_1_depth.png`

3 BONUS QUESTION: Deploying wall-following code from A1 on the racecars (3 pts)

You may work in groups of up to 3 people, each group sharing a single racecar. You need to redirect the Twist ROS messages that your assignment 1 code computed and published, to the appropriate `cmd_vel` topic on the ROS system on the racecar's computer. You also need to tweak the PID parameters so that they make the racecar follow the wall smoothly. If you choose to work on this problem your group will present the solution to the instructor and the TA in the hallways of Deerfield Hall or the North Building during a mutually agreed upon time on Dec 2,3, or 4. You will also need to submit a video showing your car smoothly following a wall for about 10-20 seconds. The name of the file should be `racecar_firstname1_lastname1_firstname2_lastname2_firstname3_lastname3.mp4/avi/ogg`

4 How to submit

Submit all your work in a file called `estimation_and_vision_assignment.zip` that contains your extensions to the provided starter code, as well as the required files, as explained above. Submissions will be done on Quercus.