CS 460: INTRO TO COMPUTATIONAL ROBOTICS

Assignment 3

Lorenzo Nieto
len44@rutgers.edu
Tasha Pais
tdp74@rutgers.edu
Rutgers University, Department of Computer Science
December 13, 2023

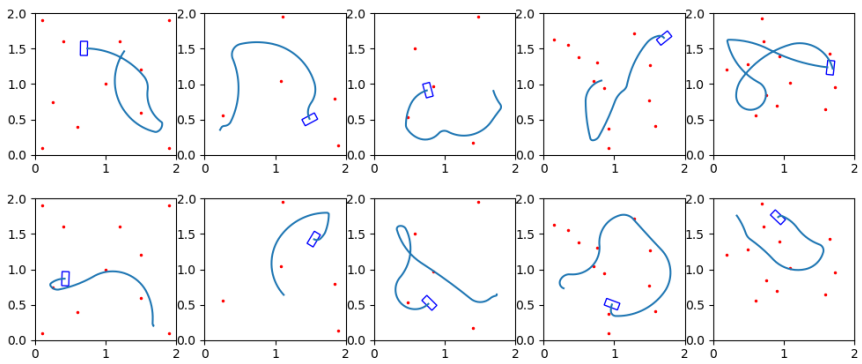
1 Setup maps and generate controls

1.1 Generate 4 maps with landmarks

The file "map_generator.py" handles map generation. It receives arguments –landmarks k and –name path. It generates a map with k landmarks with x and y coordinates each uniformly sampled in the range [0, 2] and saves the landmarks map in the file specified by path.

1.2 Generate robot control sequences

Running the file "controls.py" will populate the "controls" folder with two control files for each landmark map, making a total of ten control files. To generate a sequence of controls, first a start configuration was randomly generated. It was ensured that the initial configuration would stay 0.2m from the boundary. Then, a random control within the robot's limits was generated and integrated forward for two seconds. Only controls with positive velocity were generated to give the robot a smoother trajectory and make visualization easier. At each integration step, it was ensured that the robot stayed within 0.2m from the boundary and if it didn't, a new control was generated. Ten controls were generated in this way for each control file. Finally, a plot showing the trajectories of all the control files was generated.



Each column corresponds to the landmark maps 0-4 and each row corresponds to a different control sequence 0-1

2 Integrate Noisy Dynamics and Collect Sensor Readings

2.1 Simulate robot's motion and measurements

2.1.1 Actuation model

The ground truth files were generated by integrating forward the controls with added noise, as explained in the assignment 3 pdf. It was ensured that if the control value for either velocity or angular velocity was 0, then the noise added would also be 0. Also, the control values never exceeded the robot's limits, even after noise was added.

2.1.2 Odometry model

More Gaussian noise was then added to the executed controls from the previous step. Again, the values were clamped so that they would not exceed the limits of the robot's motor. Also, it was ensured that if the original controls had a value of 0, then the odometer's reading would also be 0.

2.1.3 Observation model

The landmark_sensor function in simulate.py calculates the distances and angles to each landmark. For distance, the Pythagorean theorem is used. To find the angle in the local robot's coordinates, first, an intermediate angle is found. This angle is equal to the arctan of the ratio between the landmark's rise and the landmark's run in relation to the ground truth configuration. This is done with the arctan2 function provided by numpy. However, we need to account for the robot's heading angle, which is not necessarily 0. So we have to subtract the ground truth angle from this intermediate angle and we get the angle that we want. Finally, the angle is adjusted to be in the range [-pi, pi]. Noise is added to these values from the landmark sensor in accordance with the assignment's instructions.

2.2 Visualize the dead reckoning solution

To visualize the dead reckoning solution, FuncAnimation from matplotlib was used. The landmark ground truth positions were drawn and these never change throughout the animation. Each frame, the robot's ground truth position (blue rectangle), odometry position (red rectangle), and landmark observations (red x's) are updated and redrawn. additional blue and red dots are drawn at the ground truth and odometry locations respectively to aid in visualization. Finally, red lines are drawn to connect the odometry motion's current point and each landmark. These lines are updated at each frame. See the dr folder for animations from five selected maps. It is clear that the odometry readings from the high noise files result in motion that strays further from the ground truth path, and as a result the landmark observation locations are more inaccurate too.

3 Implement Particle Filter for Localization

3.1 Given known initial robot pose

1. **Initialization:**
    - Purpose: To create an initial set of hypotheses (particles) about the state of the system (robot's pose).
    - Process: Each particle i is initialized with the same state $x_i^0 = \text{initial\_pose}$.
2. **Prediction:**
    - Purpose: To simulate the effect of the motion command on each particle.
    - Process: For each particle i, update its state x_i based on the control input u and motion model, with added noise.

- Formula:

$$x_i^{'} = x_i + \begin{bmatrix} v \cdot \cos(\theta) \\ v \cdot \sin(\theta) \\ \omega \end{bmatrix} \cdot dt + \text{noise}$$

  where v and ω are the linear and angular components of the control input u, and dt is the time step.

3. **Update (Weighting):**

   - Purpose: To evaluate each particle's likelihood based on the observed sensor data.
   - Process: For each particle i, compare its predicted state with the actual sensor measurements and update its weight.
   - Formula:
     - For each landmark j, calculate the expected measurement z_{ij}^{exp} based on the particle's state.
     - Compare z_{ij}^{exp} with the actual measurement z_{ij}^{obs}.
     - The weight w_i for each particle is updated based on the likelihood of observing z_{ij}^{obs} given z_{ij}^{exp}, which is modeled as a Gaussian distribution:

$$w_i = w_i \cdot \prod_j \mathcal{N}(z_{ij}^{obs}; z_{ij}^{exp}, \sigma^2)$$

     - The likelihood is computed for both distance and angle to each landmark.
     - Here, σ represents the standard deviation of the sensor noise.

4. **Normalization:**

   - Purpose: To ensure the weights sum up to 1.
   - Formula:

$$w_i = \frac{w_i}{\sum_k w_k}$$

5. **Resampling:**

   - Purpose: To focus on particles that are more likely, based on their weights.
   - Process: Randomly select a new set of particles from the current set, where particles with higher weights are more likely to be selected.
   - Method: Stochastic universal sampling or similar.
   - Result: A new set of particles, which will undergo the prediction step in the next iteration.

6. **Repeat Steps 2-5:**

   - For each time step or control input, repeat the prediction, update, normalization, and resampling steps.

In summary, the particle filter iteratively predicts the effect of actions, updates particle weights based on sensor observations, normalizes these weights, and resamples the particles to focus on the most likely states. This process enables the filter to approximate the posterior probability distribution of the system's state.

4 Evaluation and Experiments

The file evaluation.py generates animations for the estimations gathered from the particle filter. In the assignment instructions, the code is not run with the readings files as a parameter, so in order to include the landmarks observation locations in the animations, the readings file is used corresponding to the x, y, z from the other parameters. Similar to the dead reckoning animations, the robot's locations are drawn with rectangles and the noisy landmark locations are drawn with lines connecting them to the current configuration. The only difference is that the estimated locations from the particle filter are drawn in black and so are the noisy landmarks.

To the right of the robot animation, plots with animations displaying the error are shown. In the middle plot, the euclidean distance between the ground truth and estimate is plotted on the y axis, and the time on the x axis. In the right plot, the difference between the ground truth angle and the estimated angle is plotted (in radians) on the y axis, and the corresponding time is plotted on the x axis. It was determined that when there was a high amount of noise in readings, there was more error. However, choosing a higher amount of particles decreased this error. However, choosing a high amount of particles severely lowered the performance of the program. See video1 and video2 folders for visualizations.

4.1 Given known initial robot pose

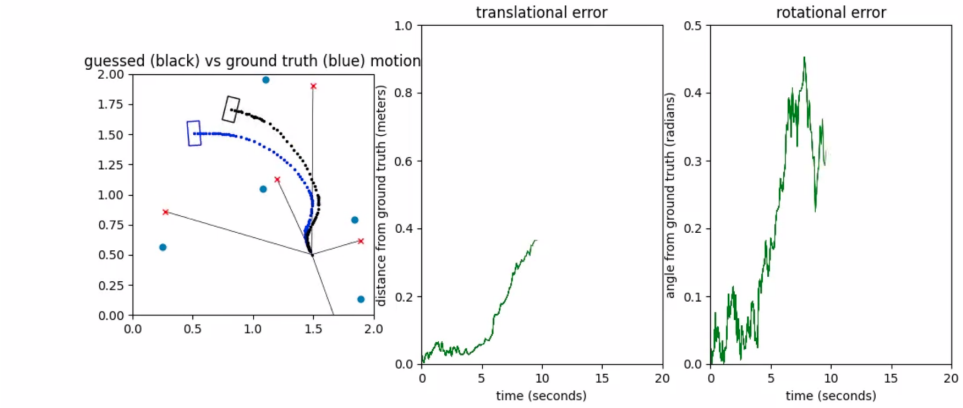


Figure 1: 0, 0, L, 200

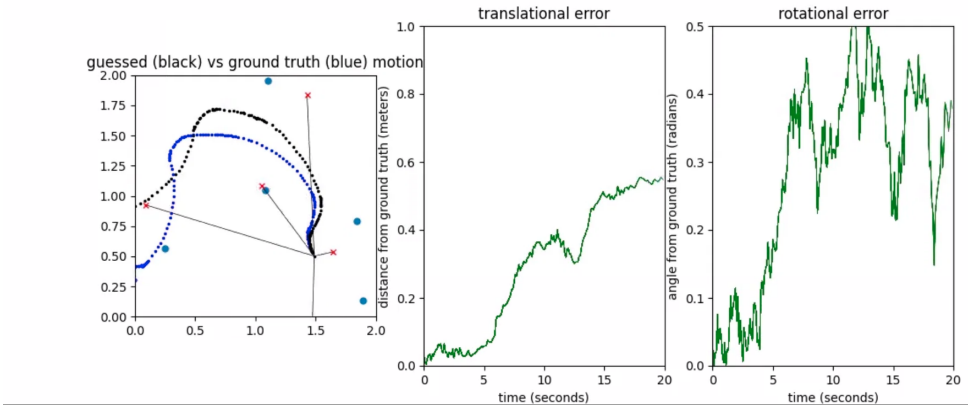


Figure 2: 1, 0, H, 200

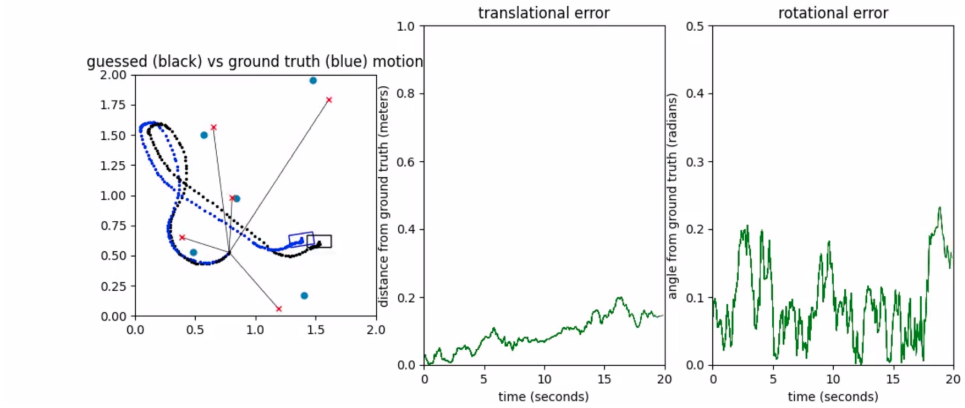


Figure 3: 2, 1, L, 1000

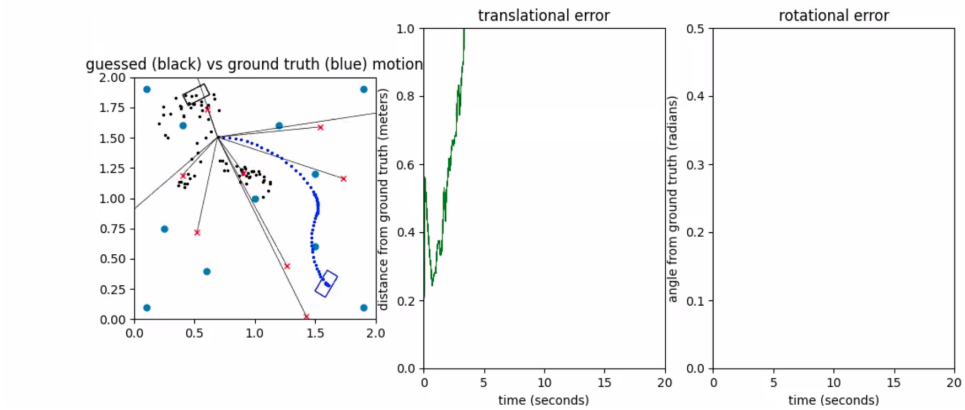


Figure 4: 3, 0, H, 500

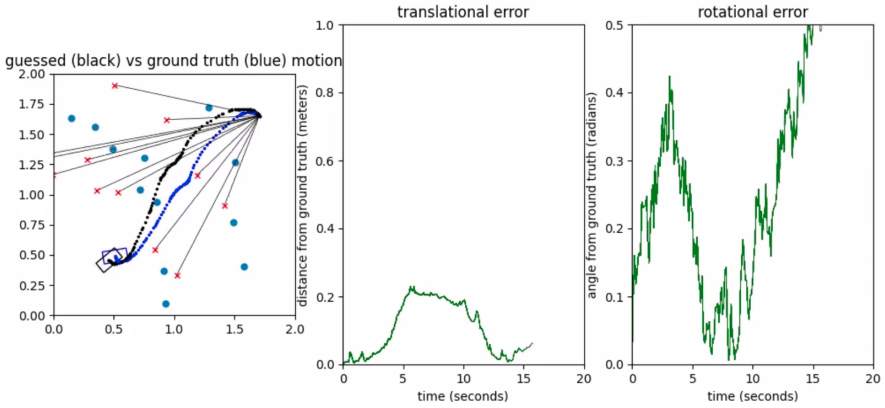


Figure 5: 4, 0, L, 200