A2: Robot Mapping and Exploration

Benjamin Kuipers and Tiancheng Ge

Assigned: Sept. 23, 2019
Challenge: October 11, 2019, 12 - 2 pm, in lab
Report Due: October 11, 2019, 11:59 pm

Purpose

The goal of this assignment is for you to master robot mapping and localization from sensor information and to use the constructed map to navigate through the environment.

Your work in A1: Robot Control showed that odometry has cumulative drifting errors, making localization and mapping increasingly unreliable. However, if we have a reasonably accurate map of the environment at one step, we can use that map to improve the robot's localization, and then update the map to continue to be reasonably accurate at the next step.

We are going to build up to an important problem called Simultaneous Localization and Mapping (SLAM), and a particular solution to a variant of the problem called online SLAM.

For this assignment, you will be working in randomly assigned groups. Each group will have access to the code repository on Gitlab. The code is available on Canvas as well, at `Files/A2/a2_code.tar.gz`. In this assignment, you will mainly modify the code in the `botlab-f19` folder inside the repository (or if you download the code from Canvas, you will see this folder after extracting `a2_code.tar.gz`).

Task I: Occupancy Grid Mapping

An occupancy grid describes the space around the robot in terms of the probability that each small cell of that grid is occupied (or if not, then free). For this assignment, we will define the grid to have 5 cm cells. You will be given sensor logs, which contain `lidar_t`, along with a `pose_xyt_t` for each laser scan. For use in Task I only, you can assume the actual (ground truth) pose (position and orientation) of the robot is published on the `SLAM_POSE` channel, for each laser scan.

Since the laser range-finder's single beam rotates at about 5 Hz, the robot will move a non-negligible distance while the laser rotates. Therefore, you will need to estimate the origin pose of each beam to determine the cell where the beam terminates. This will require you to interpolate between the estimated poses for each laser scan. For this task, these poses are the provided ground-truth poses. For the remaining tasks, they will be poses estimated by your localization algorithm. We have provided code to handle this for you, which is located in `src/slam/moving_laser_scan.hpp`.

You will be provided with three sensor logs:

1) `data/convex_10mx10m_5cm.log`: convex environment, where all walls are always visible and the robot remains stationary (use for initial testing of algorithms).
2) `data/drive_square_10mx10m_5cm.log`: a convex environment while driving a square
3) `data/obstacle_10mx10m_5cm.log`: driving a circuit in an environment with several obstacles

You can use `lcm-logplayer-gui` to playback the logs. Only enable the channel of `LIDAR` and `SLAM_POSE` when doing the playback. Using the provided sensor logs, your task is to create an occupancy grid map of the environment, incrementing or decrementing the log-odds-occupancy value in each cell of the grid, for each beam that terminates in, or passes through, that cell.

An `occupancy_grid_t` data type is in the file `src/slam/occupancy_grid.cpp` and must be used for this assignment. Your occupancy grid must be published as an `occupancy_grid_t` on the channel `SLAM_MAP`.

Create a Vx visualization of the occupancy grid. Updating this display every scan (or few scans) will be a useful debugging tool, and the final version will show your constructed map. It is natural to display occupancy as a range from black (high-confidence occupied) to white (high-confidence free). Include the ground-truth trajectory of the robot in red, on this black-white display of the environment.

Submit: Write a couple paragraphs describing any difficulties or innovations you came up with while implementing the mapping algorithm. Include a figure showing the final map for `data/obstacle_10mx10m_5cm.log`.

Hint: The skeleton code of the occupancy grid mapping algorithm can be found in `src/slam/mapping.cpp`.

log-odds values are integers in the range [-127,127].

To perform the ray casting on your occupancy grid you might want to implement Breshenham's line algorithm [1].

When you run `src/slam/slam_main.cpp` (the main program for this task), use `--mapping-only` command-line option. That will disable the localization.

Task II: Monte Carlo Localization

Using the sensor and motion models taught in class, use a particle filter to estimate the most probable poses of the robot. Particle filter-based localization is typically called Monte Carlo localization or MCL.

For this task, when you run `src/slam/slam_main.cpp` (the main program for this task), use `--localization-only <map_filename>` command-line option. The `.map` files are under the `data` folder.

Like the previous task, you can use `lcm-logplayer-gui` to playback the logs. For this task, only enable the channel of LIDAR, ODOMETRY and TRUE_POSE when doing the playback. The ground truth pose of the robot is published on the channel of TRUE_POSE, and you will compare your inferred SLAM_POSE against the ground truth.

Here are suggestions for how to implement your Monte Carlo localization algorithm.

- Update your particle filter each time a lidar_t message (contains a full 360 degree scan) arrives.
- From the previous state of the particle filter, use odometry and the action model to predict the distribution of particles at the time corresponding to the last measurement time in the laser scan. The skeleton of the action model can be found in src/slam/action_model.cpp.
- Each particle in the distribution represents a specific pose at that point in time. Treat that hypothesized pose the same way you used "ground truth" in Task I.
- Use the sensor model and the map to compute the likelihood of the measurements, which is the particle's weight. You will need to account for the motion of the lidar using the particle's pose estimate. The skeleton of the sensor model can be found in src/slam/sensor_model.cpp.
- Resample to create a new distribution of particles and use the maximum likelihood particle as the "true" location. You might also try a weighted mean of the particles' pose estimates for the "true" location. The skeleton of the particle filter can be found in src/slam/particle_filter.cpp.

After each localization step, compare the maximum likelihood pose with the ground-truth pose, and provide a graph of errors as a function of time. Note that the ground-truth poses were estimated using an implementation of this assignment, which means they aren't perfect and also used randomized sampling. Thus, your error will never be exactly 0. However, the mean error should be small.

This part of the assignment is primarily for your use in developing and debugging the localization code, in preparation for the next task, so the errors should be consistently small.

Submit: Write several paragraphs describing your MCL implementation. Include information on your action and sensor models. Be sure to include information on what values you used for your parameters, including how many particles you used and what constants were used for your action and sensor models. Include a description of the structure of your code. Include a figure showing the final trajectory for data/obstacle_10mx10m_5cm.log. On the map, show your estimated trajectory in blue and the ground-truth trajectory in red.

Provide a separate graph of the error between the two trajectories, as a function of time. Since you will update your localization for each laser scan, you should have one error measurement for each laser scan. This graph does not need to be created using Vx. We recommend Matlab or Python.

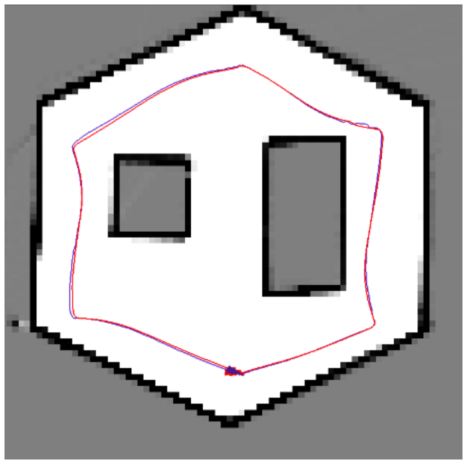
Fig. 1. Example Vx visualization.

Task III: Simultaneous Localization and Mapping (SLAM)

Simultaneous Localization and Mapping (SLAM) is the problem of exploring an unknown environment and building a map. You will implement and evaluate your occupancy-grid-based SLAM algorithm by combining the mapping algorithm you wrote for Task I with the localization algorithm you wrote for Task II.

For this task, you will once again use the sensor logs we provided you. To initialize your map, define the robot's starting pose as (0,0,0). Use the first laser scan to create an initial map. For subsequent scans, you'll first localize the robot in the map using MCL and then update the map using your occupancy grid mapping algorithm.

If the localization algorithm fails, your evolving map will deteriorate dramatically into incomprehensible chaos, making localization impossible. This failure is a valuable experience on the way to debugging your SLAM algorithm.

Once your algorithm is working well, your map should look similar to the map you built in Task I and your trajectory should be similar to the trajectory estimate for Task II.

Submit: Write a couple of paragraphs describing changes or updates you needed to make to your implementations from the previous tasks to get your SLAM algorithm working. Include a figure showing the final map for each provided sensor log. Also, provide a link to video showing the evolution of the map for data/obstacle_10mx10m_5cm.log.

On the map, show your estimated trajectory in blue and the ground-truth trajectory in red. Provide a separate graph of the error between the two trajectories, as a function of time. If your SLAM algorithm works well, this error should be small.

Task IV: A* Path Planning

Given a map of the environment, an initial pose of the robot and a goal pose, a path planner will compute a feasible path connecting the initial and goal pose, or tell you that there is no feasible path.

For this task, you will implement a search algorithm, A* search, as the path planner. A description of it is available on Wikipedia [2].

The skeleton for the A* planner is located in `src/planning/astar.cpp` and `src/planning/motion_planner.cpp`.

`src/planning/astar_test.cpp` can be used to test the performance of your A* planner.

You can simply run `astar_test` after implementing your algorithm. This program provides some test cases, and checks

- whether your implementation of A* search can find a safe path (no collision) if there exists one.
- whether the found path can avoid the obstacles correctly.
- when there doesn't exist a safe path, whether the implementation decides that correctly.

The path found by A* search may contain some unnecessary waypoints. If the path following scheme of the robot is turn-travel-turn, the robot may constantly stop before performing a turn. To speed up the robot, eliminate those unnecessary waypoints, while ensuring the path is still safe and collision-free.

Submit: Write several paragraphs describing your path planner implementation, and include the parameters used in A* search. Describe the meaning of each parameters.

Write a paragraph that describes your strategy to eliminate unnecessary waypoints.

Task V: Exploration of an Environment

For this task, you'll be combining your closed-loop odometry control from A1, your freshly-implemented SLAM algorithm from Task III, and a path planner from IV to explore an unknown environment. To explore an environment, you'll follow this basic outline:

- Build a non-convex environment to explore. (shaped like an O, or 8, or E, etc.)
- Take a picture of the environment to include in your report.
- Define the starting position of the robot as (0,0,0).
- Don't move for at least the first laser scan to build an initial map of the environment.
- After each SLAM update, evaluate the map to find frontiers – the boundary between mapped and unmapped territory. Frontiers are defined in `src/planning/frontiers.cpp`.
- Start moving toward some frontier by sending a command to the path planner.
- Continue exploring and mapping until there are no more frontiers in the map.
- Produce a final map with a robot trajectory, and a video showing its development. Show the photo of the environment for comparison.

Once your exploration algorithm works, test its limits. With a small, simple environment, cumulative errors will be small enough that you should be able to create quite a good complete map. By creating a maze, requiring a long trajectory to close the loop, can you get the robot to confidently create an incorrect map? When the algorithm tries to incorrectly close a large loop, chaos will probably result. Larger maps require more memory, and long loops can accumulate more error.

As your robot explores the environment, it should publish a `exploration_status_t` message on the `EXPLORATION_STATUS` channel. While exploring, the status should be set to `EXPLORING` and the frontier target should specify the position of the current frontier being explored. Once exploration is complete, i.e. no frontiers exist, the robot should return to (0,0,0). Once the initial pose is reached, the status should be set to `COMPLETE`.

To implement this task, you will be using LCM and a utility program, `bot-lcm-tunnel`, to communicate between your Mbot and your laptop. Running the SLAM and path planning algorithm on your laptop allows you to focus on getting the system running properly, rather than hand-optimizing the algorithms for the more computationally limited Mbot. However, the delay in the communication may cause the map to be inaccurate. If you want to get good performance in Challenge, you will have to optimize the algorithms and run them on Mbot directly.

Submit: Write a couple paragraphs describing the overall system use for exploring the map. Include your algorithm for finding frontiers and your decision process for selecting the next frontier to visit. Include a figure showing the final map along with a photo of the actual environment, and a link to the video showing the evolution of the map. On the map, show your estimated trajectory in blue.

Challenge: SLAM and Exploration

We will be holding a map exploration challenge in CSRB 429 on October 11 during the lab session.

Each group will have three attempts to explore an environment constructed by the course staff. Your exploration will be evaluated using the following criteria:

- Exploration speed – the total time your robot takes to explore the environment.
- Map completeness – how much of the map did you explore?
- Map accuracy – how accurate is the constructed map?

The exploration time starts when you send the first `exploration_status_t` message and ends when you send a message with a status of `COMPLETE` or if your robot gets stuck against an obstacle.

Map completeness will be determined by the staff's implementation of occupancy grid SLAM. We'll be building a map using the sensor data from your Mbot to run our own SLAM. We'll compare this map against a ground-truth map to determine how much of the environment your robot mapped.

Map accuracy will compare your constructed map with our constructed map using the correlation-based map matching algorithm described in Chapter 6.5 of Probabilistic Robotics by Thrun, et al. The higher the correlation, the better.

Report: Robot Mapping and Exploration

Each group should produce a report on your project, formatted using LATEXin IEEE two-column conference format. This style can be found in the ieeeconf package.

Use Vx to produce the needed figures, and then include them into the report, which is submitted as a PDF file. Each figure should include a detailed caption, explaining the method, conventions, and significance of the information in the figure. The main body of text should refer to each figure, but should not duplicate the caption explanation. (Think of the figure and its caption as a module being used by the main body to communicate the message of the paper.)

Compress all the code into a tar.gz file, `team0X_a2.tar.gz`, where `0X` is your team number, and submit it together with the report. For each team, one submission is sufficient.

Certification [required for credit, 0 points]

Print or write the following on a sheet of paper:

”I participated and contributed to team discussions, and I attest to the integrity of the report.”

Each team member should sign the paper (physically, not digitally), and a photo of the paper should be included with your report. By doing so, you are asserting that you were materially involved in the team work, and that you certify that every solution complies with the collaboration policy of this course. In the event of a violation of the course’s guidelines, your certification will be forwarded to the honor council.

If the statement requires qualification (i.e., it’s not quite true), please describe the issue and the steps you took to mitigate it. The staff will use their discretion based on the specific circumstances you identify, and we are happy to discuss potential problems prior to the report due date.

If your team works together, as required, this will be painless! :)

References

[1] Breshenham’s line algorithm.
`https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm`

[2] A* search.
`https://en.wikipedia.org/wiki/A*_search_algorithm`.