Navigation: Mapping, Localization and Planning
Evan House
Alaina Kafkes, Wenbo Yu
11/11/16

ABSTRACT:

Navigation, traveling between one location and another, is a fundamental problem in robotics. Navigation involves multiple steps, such as planning, localization and mapping. In this lab we write behaviors and functions that allow the robot to generate a plan to travel between two points in a maze without running into an obstacle. We also write functions that allow the robot to explore and produce a map of the environment as it explores.

INTRODUCTION:

Representation and memory are important for robotics because it is helpful for the robot to remember what it has seen in the past. Representation of the world is called a *world model* [1]. A map is the most common usage. There are many ways a robot can remember a map. It can remember an exact path, a sequence of moves, landmarks in the world, or it can form a metric map making measurements of distances and lengths. The first is very specific, so the downside is that it is no longer useful if the world changes. The second does not require the same measurements, but still requires that the map not change. The fourth type is more difficult because it is not easy to make all the exact measurements of the world.

In robotics, planning is the process of deciding the best way to act by looking at the future possible outcomes. Deliberative control does this in three steps, the first of which is searching. In order to act, the robot must first look at the results of different outcomes. The robot then sees which path(s) would lead it to the goal then picks the optimal path. In more complex situations, it is important that the robot can act quickly and respond in real time. For this reason, the method of sensing, planning, then acting has many drawbacks. First, it may take a significant amount of time and memory for the robot to determine a plan. Secondly, if the environment ever changes, the robot's plan of action will fail.

Hybrid control uses both reactive and deliberative control. Hybrid control must have three levels: a reactive level, a planner, and a level that links the two together. The primary challenge is getting the reactive and deliberative control to work together. This can be done through dynamic replanning, where the reactive system signals the deliberative system if it runs into an obstacle, so that it may come up with a new plan. Another method involves having the robot moving based on its reactive system, while the deliberative control determines the optimal plan to reach its goal. Another technique is to keep a record of potentially useful actions for situations that may recur.

An alternative method of control is called behavior-based control (BBC). BBC is based on the use of "behaviors", which are more complex than actions and include things like following the light or finding an object as opposed to an action like turn left. There are a few advantages to using BBC in contrast to the methods discussed previously. For one, the robot is able to react in real time, and the behaviors are generated efficiently. BBC systems also tend to be less dependent on the specific environment. Another important part of robotics is determining the best behavior or action; this is called behavior coordination. There are a few ways of doing this. One is typically called arbitration which is the term used to describe selecting a behavior from a set of choices. Behavior fusion is another option, where different candidate behaviors are combined into one behavior which the robot carries out.

Navigation is defined as the way a robot travels through an environment as it goes from one place to another. This is a difficult problem for robots because there is a lot of uncertainty involved. There are many situations in which the robot simply lacks information, such as where it is, what the environment is like or what obstacles are in the way. Navigation is often broken down into a few main steps: planning, coverage, localization, and mapping. Planning, as we discussed, is the development of a path between two points; coverage is the process of searching the unknown environment; localization is keeping track of where the robot is as it moves along; mapping is

recording a map of the environment as the robot explores. When the process of localization and mapping are combined we have what is called simultaneous localization and mapping (SLAM).

One common method of localization is odometery or path integration; this is the method of measuring the robot's movements from its starting point. One of the downsides of this is as the robot moves more and more, the error in measurements begin to add up, and this can cause the robot to get lost. In path planning, the robot determines the best path between a given starting point and end goal. The primary problem with path planning is that it tends to be time intensive. One method of path planning is the wave propagation technique. This technique sets a cost for each location on the map. The end coordinate has a cost of zero, and the next available locations have an increased cost. The cost increases and propagates out as wave until it covers the map. The robot can then use this map to pick the optimal path by simply traveling to locations with a cost that is less than its current location. In this lab we will deal with a specific problem of navigation and its various components.

METHODS:

After we had read through the tasks to complete in this lab, we decided that the first thing we should do is build a new robot. We wanted to build a model which would allow us to control walking and turning more easily. We decided upon a simple setup with four wheels and three infrared (IR) sensors facing in the forward, left and right directions. Once our new robot was setup, our first task was to write new functions that would allow the robot to move forward, backward and turn left and right. We also implemented a new function that would allow us to stop the wheels from moving from the command line. Another important step for us at the beginning of this lap was to read through the map starter code to ensure we understood all the functions we might need for the planning and mapping portions of the lab. We used the map functions to print out an example obstacle map so we had an idea of the environment we would be working with.

After we had written these functions, the next step was to walk one cell in each of the cardinal directions. This task was pretty straightforward, as we simply had to combine the turning and moving functions that we had previously written. We had to adjust the time that our function ran so that it would move forward about one cell.

The next task in our lab was to write planning functions that would allow our robot to generate and follow plans that would allow it to travel from a given starting point and heading to a given end point and heading. This was split up into four main tasks. The first of which was to create a map and set the cost of traveling to each cell. We used the wave propagation technique to determine the cost of each cell. Our methods would take the starting coordinates and the end coordinates as inputs. First, the cost of every cell would be set to a large number. From there, the algorithm would set the cost of the goal to zero. It would propagate out from there checking for obstacles and setting adjacent cells to be one larger than the current cell. A separate function then used the cost map this function had generated to create a path between the end point and the starting point. The algorithm would begin at the start point then choose a neighboring cell with a lower cost. In this way, the function created a list of the path coordinates. The final function that we wrote for this section took the results of the previous function and use the walking and turning commands we had written to make the robot follow the generated path.

One of the important steps in the lab was taking time to write out the best practices we had come up with in our time in the robotics lab. After coming up with these best practices our job was to improve our planning functions with these best practices in mind. One of the first best practices we came up with was to test out our functions more thoroughly. There are often errors in the code that only arise on certain occasions. Another best practice we found was to debug our code more rigorously using print statements to understand what exactly the function is doing and how it responds to control flow. On the first demo day, our planning function did not work as planned. When we went back and tested again on our own, we found that our planning functions only worked for a certain subset of start and end coordinates. Here we implemented the two best practices mentioned above. First, we tested out more start and end conditions. We were able to find situations where the functions did not run properly. Once we had a test case where an error arose, we then took a closer look at our three functions to see what may have caused the error. Unfortunately, the first time through we did not put in extensive comments or print statements so the code was quite hard to understand. We actually found it better to simply rewrite our function that builds the

cost map while using print statements as we built it, in order to ensure that the function was working as we had imagined it. After this, we decided to test our new planning function more rigorously. Instead of just trying a few coordinates, we used a larger number of test cases and made sure that the test cases covered different scenarios. Another difficulty we had was keeping track of the coordinates properly. We would often find our robot's map or path not matching up with the coordinate system of the physical maze. The more rigorous method of debugging also helped us to make sure the coordinate system was working properly.

The next task in this lab was to write a mapping function that would allow our robot to explore its surrounding and build a map based on its exploration. We were working under the assumption that the robot starts at the origin and is facing south. The first step in writing these mapping functions was writing a wondering algorithm. The idea behind the wandering algorithm is that the robot would walk around in the map so that it visits unknown regions of the map while at the same time not hitting any walls. It is also important that the robot keep track of where it was going as it wandered. We first wrote the skeleton of this function by having the robot randomly choose from the four cardinal directions. Having randomly chose a direction, the robot would check if there was a wall blocking the path in the given direction. If so, the robot would randomly choose a direction again until it chose a path that was not blocked. While this was not the most efficient method, it did work. We also found it helpful to create dictionaries with keys corresponding to a heading, and values corresponding to the new heading if the robot turned right or left. We later wrote a separate function that would choose a direction in a more efficient manner. We would then call this function inside of our wandering algorithm, then the robot would execute the proper movements given said direction. While this was more efficient, we eventually found our random wandering algorithm to work more reliably, as the updated method did not turn as planned during the demo.

The next function we wrote was one that would allow the robot to record walls as it walked. The first step in writing this function was to simply check the sensor values returned at a distance from the robot to a wall while it is in a given cell. We used these values to have the robot check for walls to its front, right and left. If there was a wall, the function would update the current map with a wall in the given direction. The last part of this task was to have the robot save the results of the map it recorded. Then, we would use the previously written planning function to have the robot travel between two points on the map it had recorded itself. To do this, we used a Python package called pickle. We had some trouble at first and did not get to test it out as thoroughly as we would have liked, but in the end, the planning function was able to successfully read the stored map file.

RESULTS:

Assessment 1:

(0, 0) --> (3, 0) Trials	Distance Error (cm)	Orientation Error (degrees)	(0, 0) --> (0, 3) Trials	Distance Error (cm)	Orientation Error (degrees)
1	10.6	14	1	15.6	21
2	15.2	8	2	7.9	14
3	1.3	9	3	17	16
4	10.7	10	4	15.6	18
5	9	9	5	10.1	7

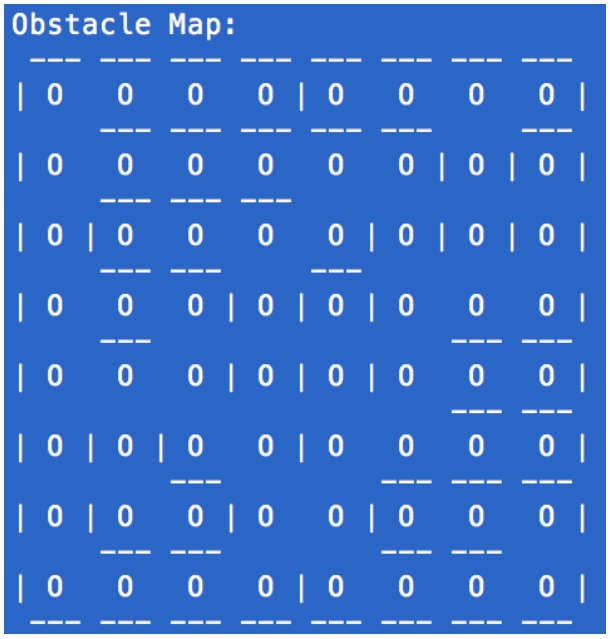
The first measurement that our group took in this lab involved testing out our localization functions. We used our moving and turning functions to have the robot travel between a starting point and an endpoint. We did five trials of this for three different endpoints. Each time, we measured the distance error, how the difference between the endpoint and where the robot actually stopped, and the orientation error, the angle difference between the end heading and the actual heading. We started the robot in the right corner of a cell with its front wheels at the line and its right wheels along the line. This way we knew exactly where to measure from when measuring the error in distance. For example, when the robot traveled from (0, 0) to (3, 0), the end goal was the line three cells away. We then took the measurement from that line to where the robot ended up. All measurements were taken to the front of the robot's wheels because it was the front wheels that were lined up at the front of the cell.

(0, 0) --> (2, 2) Trials	Distance Error in X-Direction (cm)	Distance Error in Y-Direction (cm)	Orientation Error (degrees)
1	13.3	0.1	9
2	13.6	2.4	5
3	12.7	20.1	13
4	11.6	-3.1	4
5	11.3	14.7	11

For the trials where the robot moved from (0, 0) to (2, 2) we took to distance measurements. One distance measurement was in the x-direction (horizontal error) and the other measurement was in the y-direction (vertical error). We found that the vertical error seemed to vary more widely for whatever reason, while the horizontal errors were more similar to each other. Below are the mean and standard deviation values for the three different setups. It is surprising that the mean error in orientation when the robot traveled from (0, 0) to (2, 2) is the lowest even though this was the most involved of the three setups.

Travel Path	Distance Error Mean (cm)	Distance Error Standard Deviation (cm)	Travel Path	Orientation Error Mean (cm)	Orientation Error Standard Deviation (cm)
(0, 0) --> (3, 0)	9.16	4.562	(0, 0) --> (3, 0)	10	2.098
(0, 0) --> (0, 3)	13.24	3.568	(0, 0) --> (0, 3)	15.2	4.707
(0, 0) --> (2, 2) X-Direction	12.5	0.91	(0, 0) --> (2, 2)	8.4	3.441
(0, 0) --> (2, 2) Y-Direction	8.08	7.862			

Assessment 2 and Comparative Assessment:

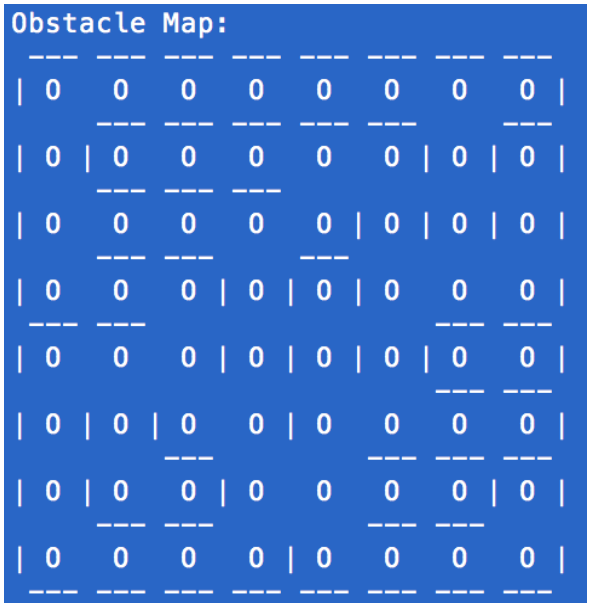


The next measurements that were to be taken as a part of this lab involved the planning functions. The idea was to run the planning functions so that the robot generates and executes a travel plan between two coordinates. After running the function, we measured the total execution time, as well as the heading difference. We were also supposed to repeat the same measurements again, after applying the best practices. In our case, the function was not working after the first demo day, and it was not until we considered and applied the best practices that our function was able to start working. Below we have the data collected after applying best practices. For comparison, keep in mind that the first time through we let the function run for 5 minutes (300 seconds), and 0% of the tiles were explored. We use a typical coordinate system (i, j), where the upper left corner is the origin with i running vertically from 0 to 7 and j running horizontally from 0 to 7.

Trials	Start Coordinate	End Coordinate	Execution Time (s)	Orientation Error (degrees)
1	3, 4, S(3)	6, 2, E(2)	180.3	-43
2	7, 6, N(1)	4, 7, E(2)	113.29	3
3	4, 5, N(1)	0, 3, W(9)	143.04	23

In each of these three trials, after applying best practices, we were able to successfully reach the end coordinate and heading. To make the measurements, we simply started the timer as soon as the code began executing. Once the robot reached the final position and had stopped moving, we stopped the timer. The orientation measurement was similarly straight forward; we simply used a protractor to measure the difference between the directing the front of the robot was facing and the end heading goal.

Assessment 3:



After this, we made measurements in the lab which were related to our mapping functions. The measurements we made here were similar to those for the planning functions. This time, we ran our functions that would cause the robot to wander around and record its surrounding as it traveled. Once again, we measured the total execution time, in addition to the amount of tiles the robot traveled and the number of times the robot hit a wall. After running the mapping functions, we then used our planning functions to generate a path between two coordinates given a saved version of the map that the robot had just recorded. The map that the robot traveled on in this portion of the lab is displayed to the left. We use a typical coordinate system (*i, j*), where the upper left corner is the origin with *i* running vertically from 0 to 7 and *j* running horizontally from 0 to 7.

Right when the code began executing, we started a timer, and we ran the timer until the robot was no longer exploring. We repeated this two times to get a more desirable result. The count of tiles explored *do* include the tile on which the robot starts because the robot does check for walls and add them to the map while at the starting tile. After running the planning functions on our own map, we repeated the same measurements as we did in assessment two.

Trials	Execution Time (s)	Walls Touched	Tiles Explored	Percent Traveled
1	74.7	1	9	14.06%
2	106.24	0	13	20.31%

From looking at the data above, it is obvious that when the robot ran for longer, it explored more tiles. The robot was stopped in trial 2 after it had gotten stuck oscillating between two tiles. We then used the map that the robot had created and ran our planning functions on this map. The data for this is below.

Start Coordinate	End Coordinate	Tiles Traveled	Percent Traveled
1, 0, S(3)	4, 2, E(2)	3	42.86%

Our program was able to successfully read the saved version of the map; however, it seems the generated path did not work as it was planned. After the robot had traveled three tiles, the robot did not turn properly but instead traveled into a wall. The robot completed about 43% of the path before hitting the wall.

DISCUSSION:

We were able to successfully write localization functions that allowed the robot to keep track of where it was at all times. We then went on to incorporate these localization functions into our planning and mapping functions. The first time through, our planning functions turned out to be unsuccessful. We found that we did not properly test and debug our code the first time through. For this reason, the function worked in the specific scenarios we had tested. However, when it came time for the demo, we found that our function did not account for many scenarios. In order to fix this, we went back and started from the ground up on some of our functions, more rigorously debugging so that we knew exactly what the function was doing, and we could see where it did not line up with our expectations. After we *thought* we had a successful planning function again, we were much more careful to test out different scenarios that utilized all the different control flow statements.

We were able to successfully write a mapping function, however there was definitely room for improvement in this area. From the results sections it is easy to see we only traveled 20% of the map before running into problems. This is definitely something that should be improved upon. Our algorithm could be improved in terms of both readability and efficiency so that we could improve upon our map coverage. When we ran our planning function for the map which the robot had recorded, we ran into additional problems. It is not clear whether the problem was in the plan which the robot had come up with, the map that the robot had recorded, or in the communication between the two. Taking an extended look at this would also be helpful.

Something that we have noticed for a while now in robotics, is that the actuators are never exact. For this reason, it is not uncommon for the robot's error to add up over time and run into a wall while it is traveling in the map. Another potential best practice we identified was implementing reactive control behaviors where the robot would identify when it was veering towards a wall and slightly correct its path. This is definitely an important practice that should be implemented in the future.

In this lab we began to really confront some more fundamental and challenging problems in the robotics field. Because the problems were more challenging our algorithms and behaviors were necessarily involved. Unfortunately, this often led to some messy code that made it difficult to identify problems. It is important that as we are writing the algorithms and behaviors, we test and debug *as we write them*. If we do this, it will become much easier to identify the specific problems that arise.

CONCLUSION:

In this lap we developed a navigation system for our robot. This involved localization, planning and mapping functions. We collected data on each of these behaviors individually. We found many areas for improvement. Our measurements showed that the map coverage in our wandering algorithm can be improved significantly. The communication between our mapping and planning function also led to problems that we did not foresee. It is important in the future that we are more rigorous in testing and debugging our algorithms and functions.