Norwegian University of Life Sciences
Faculty of Science and Technology
Department of Mechanical engineering and technology management

2022

Technical report

Walking Robot Project

Authors: Peder Ørmen Bukaasen, Bård Tollef Pedersen and Lavanyan Rathy – Group 8

Abstract

Peder Ørmen Bukaasen, Bård Tollef Pedersen and Lavanyan Rathy

Walking Robot project, 10 pages

Norwegian University of Life Science

Thesis 2022

Instructor: Alireza David Anisi, Norwegian University of Life Science

The purpose of the project was to develop a walking robot program to navigate in an indoor environment in MATLAB. This thesis was our final project in our spring course TEL200 – Introduction to robotics at Norwegian University of Life Science.

This thesis describes the entire process of how a walking robot simulation works in MATLAB. The results of this project can be a good steppingstone when working with applied walking robots.

Keywords: walking robot, MATLAB, PRM, motion planning, path planning.

Table of contents

1 Introduction

In this project we are going to be working with mobile robots. This class of robots can move through various kinds of environments, like over the ground and water. The most important task for a walking robot is to get to a destination, which is common for all kinds of walking robot. We are going to look into motion planning, path planning and localization & mapping. The use of a walking robot that can navigate in an indoor environment is especially useful. The most famous walking robot is Boston Dynamics' own Spot robot, which goes on four legs.

2 Theory

The main goal is to understand a four-legged walking robot. The use of robot legs is nothing more complicated than using a 3-axis robot arm as a leg. The kinematics for a robot leg and robot arm are much alike. Learning the movement of the robot, we can start researching path planning. Before starting this project, we needed to gather more information about the theory behind walking robots.

2.1 Motion planning

We need motion planning to move around, and not collide with obstacles when moving from point A to B. We can split motion planning in two parts: deliberate(map-based) and reactive(behaviour-based). Reactive planning does not need an explicit world representation. With this method there is a real-time planner, which makes it plan fast. Deliberate method needs a world-map as a basement. With the map-based method the robot can solve more complex problems. We can think of each method as a simple organism and a mammal. Where the deliberate method is more human-like, and the reactive is simple minded. (Anisi, 2022)

2.2 Path planning

We are using an algorithm named probabilistic roadmap, this is a type of map-based planning, and more specifically Roadmap Methods. This method creates a map by randomly placing points/nodes on the specified map and connects these with lines/edges. We also used the lattice planner; this method of mapping also takes into consideration the motion constraints of wheeled vehicles. This planner has a starting point and a grid, the starting point is often referred to as the origin and the grid is used to find

the next points. From the origin the robot either moves straight forward or in an arch as seen in the pictures below (Corke, 2017).

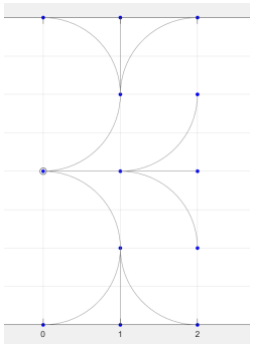


Figure 1 Shows the Lattice planner after 1 iteration

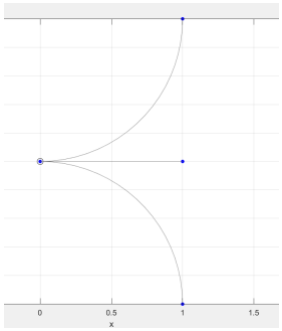


Figure 2 Shows The lattice planner after 2 iterations

2.3 Localization

A problem many faces when working with robot navigation is the current location of the robot after moving. When we gather information and process the data, we call it localization. We have several types of robot localization. GPS systems are useful when it is possible to append it to the system. In many cases we cannot use GPS since the robot may operate indoors or underwater. Therefore, we try different strategies like odometry where the robot is integrating the rotation speed of a wheel. (Anisi, 2022)

2.4 Laser-based map building

After learning about each part of a walking robot, we can put the knowledge together and make a laser-based map. This means that the robot can transfer the laser scans to a global coordinate frame and then build the map. This method uses the range of measurement to determine the coordinates of a cell that has an obstacle. (Corke, 2017)

2.4.1 Bresenham algorithm

Bresenham line algorithm takes two points and returns all the intermediate points. This algorithm is often used to draw lines on a computer screen of pixels, since all pixels have integer coordinates. In our case the algorithm is used to check if points are occupied or not and create paths.

3 Method

3.1 Motion planning – part 1

We started off with a MATLAB script, walking.m and tried to understand the code and its functionality. This script had a simulation of a robot that could lift one of the four legs up and down, but not move anywhere. Our first task was to make the robot move forward after each step, using the example code. This was done by moving many of the functions from the script under the For-loop and making the needed functions to loop so the movement can interact right with the script. We made a function that updates the position of the robot for each iteration. This function replaces the current x-values with the

new ones when moving forward. This function also subtracts or adds the size value of the robot, so each point keeps the same distance with respect to the local origin. Then we started to work on the rotation. The robot shall rotate around the local origin (Leg 1) we first calculated rotation by hand, using sin and cos with drawings to figure out a type of rotational function that would work on all legs and rotate anywhere in the xy-plane. The solution was a type of rotational matrix. Where each leg was represented by x-coordinates, y-coordinates, and rotation with respect to the first leg. Later we added the forward motion to the rotation matrix, so we had one complete matrix representing movement. This made the code easier to understand and with fewer functions and variables as we do not need one function for each leg.

After we had both rotating and walking in place, we worked on moving from one point to another. Where the points are represented by x, y, and rotation around the z-axis. Here we had two working solutions, we could make the robot first walk along the x-axis, then the y-axis or we could make the robot walk the shortest distance. On the xy-solution, we decomposed the movement into 5 steps: Rotation so it aligns with the x-axis. Walking along the x-axis. Rotating either plus or minus 90 degrees to align with the y-axis. Walking along the y-axis. Last rotation to match the last point. On the shortest solution, we decomposed the movement into only 3 steps, and we look at the points as corners of a right triangle. Rotating to align the hypotonus between the two points. Walking along the hypotonus. Final rotation to match the last point. So, when calling the function for part 1, walking_rotating_from_point(A, B, way). You need to specify two points, the start point A and the endpoint B. Where each point is described using x-coordinates as first value, y-coordinates as second, and rotation around the z-axis as final value. And then specify the way you want the robot to walk. Use ‘py’ to walk the shortest distance, which stands for Pythagoras because that is the method to calculate the distance to walk. Or use ‘xy’ to first walk the x-direction then the y-direction.

3.2 Path planning – part 2

To create the probabilistic roadmap in MATLAB the RVC toolbox was used. From this toolbox we used functions from the PRM.m file. The map we used was the house map included in the toolbox. To create the PRM object the PRM function was used with house as argument. The roadmap was then created with prm.plan function that takes npoints and distthresh as arguments, npoints is the number of random points to be added to the roadmap, and distthresh is the maximum distance where a line between two points can be added. To improve the roadmap additional points and edges were added to the map. This was done with the Add_node function that was created using code from the original PRM.m file. This function takes a point with x and y coordinate adds it to the map and connects it to the closest points. For making the paths a function PlanPrm was created, this function takes the prm object, and coordinates for the start and stop. The function then checks if the start and stop points are not inside a wall, then plans the path. To test the roadmap, we created 10 by randomly selecting start and stop points, this was

done using the randi function. To make the code run 10 times try and catch was used, since the probability of a point being inside a closed room. The VideoWriter object was used to create the video.

For the second part of this part, the same code was used to create the roadmap and Video, but the path was created with a known start and stop point. Then query function from PRM.m was used to create the path. The query functions take start and stop points as inputs and returns a matrix that contains the x and y coordinates for the nodes in the path. Then the walking_rotating_from_point_part2 function from part 1 was used. This function had to be modified a little to make the movie maker work, this was done by adding the animate object and adding a.add() after the plotting part of the code. The animate code in walking_rotating_from_point_part2 was also removed. To make the walking robot move along the path a for loop was with the walking_rotating_from_point_part2 iterating trough the path array from the third element to the last. It was running from the third since the two first elements of the array are at the starting point. The way input to the function is how the robot should move, in our case py was used, this makes the robot walk from a to b in a straight line. The number of iterations in the walking code was also reduced to make the code run faster.

3.3 Localization & Mapping – part 3

The numbers in each cell of scanMap are being calculated by the number of times a line runs through the cell. The line goes from one wall to another and adds a negative integer number on the open spaces and a positive number on the obstacles.

The code we added inside the scanmap function is on the right. Here you can see that we iterate through each cell and check if it has a value lower than negative M, in this case M= 10. If it does have a value less than -M it returns as a 0, or free. If the point is unknown meaning it original has a value of 0, or if it is an obstacle meaning having a positive value it will returns as a 1. In the Part3 code we also check if the Map we made matches the KillianMap from canvas. And our map is completely alike.

```
[numRows,numCols] = size(world);
 M = 10;
 for i = 1:numRows
     for j = 1:numCols
         if world(i,j) < -M
             world(i,j) = 0;
         else
             world(i,j) = 1;
         end
     end
 end
```

PRM was to be used in path planning, this was done with the RVC toolbox. In the code a is the KillianMap created with part3. Then a prm object is created with PRM. The planning is then done with prm.plan. We used 1000 points and distthresh at 100. Lastly the map is plotted with prm.plot.

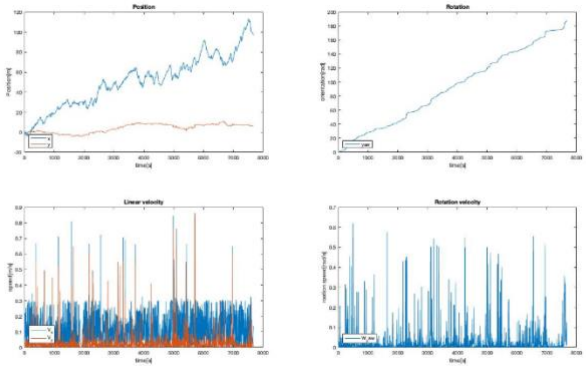
```
%part3 prm
a = part3;
prm = PRM(a);
prm.plan('npoints',1000,'distthresh',100)
prm.plot()
```

We also did map-based planning with Lattice planning. A is the same as in the PRM planning. The lattice object is created by Lattice, where a is the map, grid is the size the planner uses, root is the starting point of the planner. The plan is then created by lp.plan and plotted by lp.plot.

```
a = part3;

lp = Lattice(a,'grid',14,'root',[950 950]);
lp.plan()
lp.plot()
```

In the final part of the Localization & Mapping, we were tasked to plot the pose of the robot as a function of time. To get this plot we needed to extract a matrix pose from the icp function. To get the icp function to work we had to delete closest.m in the RVC package. This is because MATLAB has a built-in function that shall be run, but if this function exists MATLAB automatically chooses this in favour of the other. Then we iterated through each node and got one transfer matrix for each step between the nodes. From the matrix which contains both the rotation and the translation. Where the 2x2 upper left represents the rotation, and the right represents the translation. The last row is to complete and get a homogeneous matrix. And because the pose consists of position and rotation this is what we choose to plot. As a result, we see that in both the position and rotation speed there are some steep spikes, this is not the optimal as a constant velocity is gentler on the motors and these high velocity spikes will do so we need larger more powerful motors.



4 Results

4.1 Motion planning -part 1

When testing the last motion primitives, we typed in the following commands in the command window:
>>A=[0,0,0];B=[100,0,10];way='py'; walking_rotating_from_point(A, B, way);

This worked well. The robot and the coordinate system get updated for each iteration and for every opt.itteration all the legs have completed one movement. The only downside is that the robot keeps moving its legs even when it is done. This is because the number of iterations is compatible with both the 3 motion xy-walking and the 5-motion Pythagoras-walking.

4.2 Path planning – part 2

The creation of ten paths was successful, our roadmap covered most of the map. In the figure below one can see one of the ten paths created.

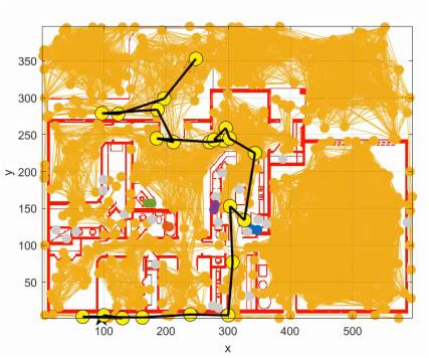


Figure 3 shows one of the ten paths created

We also managed to make the walking robot follow the path, but we did not manage to plot the walls in when the robot walked.

4.3 Localization & Mapping – part 3

The first part which was to make the killian.g2o map into the KillianMap, this took some time in the beginning to understand the code, mostly because the lack of comments and documentation in the tool pack. But after understanding the code, it was manageable. And the map we got was the exact same as the one we could get from canvas.

We had success with PRM planning and Lattice planner, the PRM run a little bit quicker than the Lattice, but the lattice planner has 4 times more points than the PRM planner.

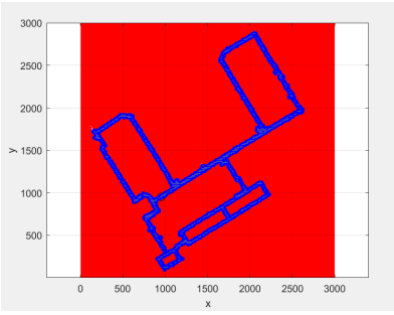


Figure 4 Shows the Lattice plan of the killian map

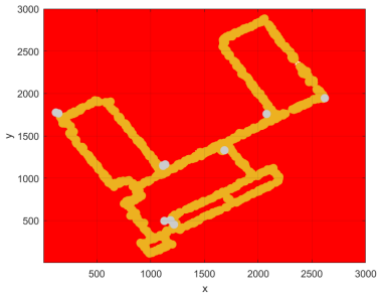


Figure 5 Shows the prm plan of the killian map

On the plotting of pose task, we did not know what to expect, or how to plot the pose. But when decomposing the pose matrix into rotation and movement it became a little clearer. The plots show us how the robot moves with each step from one node to another. In other words, the plots show how the pose changes from one node to another, as a function of time. From the plots we can easily see some challenges the robot faces. One for instance is the uneven speed, which create a need for large powerful motors that are rarely used in their full potential.

5 Discussion

5.1 Motion planning – part 1

The walking robot works fine, one of the main improvements is to make the iteration change depending on which route it takes and when it has completed the walk from point A to B.

5.2 Path planning – part 2

The path planning of the task worked fine, could have used less points on the PRM-roadmap and gotten satisfactory results. When the walker followed the path, we could have plotted the walls of the house together with the walker, but we did not know how or had time to do this. Improvements could also be made with calculations of the kinematics of the robot since this was calculated for each point in the path. Since the number of iterations in the walking animation was turned down to make the code more efficient, the movements do not look as good as they would with more iterations.

5.3 Localization & Mapping – part 3

Our initial thought when planning the paths using a method, we selected were distance transforms. The problem with these methods was that it did not run with our map. So therefore, we used Lattice, which is great for wheeled robots, but creates unnecessary complicated paths for walking robots that does not have the same limitations as a wheeled robot.

6 Conclusion

The project was completed with help from Peter Corke's Robotics Toolbox in MATLAB. The main goal of this project was to understand the concept of a walking robot. The project successfully fulfilled its goal to enable a walking robot to move between any two points given in a given map using "motion primitives" and PRM, and lastly navigation on real robot data and map.

References

Anisi, A. D. (2022). TEL200 ch 5 - Navigation I.

Anisi, A. D. (2022). TEL200 ch 6 - Localization.

Corke, P. (2017). *Robotics, Vision and Control Fundamental Algorithms In MATLAB.* Springer.