Norwegian University of Life Sciences
Faculty of Science and Technology

VT2022

Lab report / TEL200 – Introduction to Robotics

Motion & Path planning + Localization & Mapping

Author(s): Mohammed Idris Omar, Liiban Hassan Osman & Ammar Abou Madiara

Abstract

The design, development, and motion planning of a mobile robot are being explored as study topics in robotics these days. One type of mobile robot is a legged robot. Legged robots can be used for jobs that are too risky or difficult for humans to complete, such as planetary exploration, disaster relief, industrial application, and many others. As a result, the control algorithms for legged robots have become a significant study focused on robotics. This paper develops, and tests the motion, and path planning for A Simple Walking Robot using MATLAB. It covers theory and codes, what challenges were encountered, and their solutions.

An additional part localization and mapping is also covered. This involves modification of real sampled data from the MIT Killian court, path planning with the KillianMap, discussion of other path planning that would be suitable for this map. And lastly extracting and plotting the pose as a function of time using iterative closest point (ICP) function.

Contents

List of figures

1 Introduction

One of the most rapidly growing disciplines of scientific inquiry is mobile robots. Mobile robots can replace humans in a variety of industries due to their talents. Mobile robots can move without the help of human operators. A robot is autonomous when it can identify the steps to be taken to complete a job on its own, with the assistance of a perception system. To explain this more easily, a mobile robot is an automated machine that can move around in a certain area. Mobile robots can be classified according to their locomotion system and legs are a form of locomotion, which has led to the development of walking robots. Depending on the number of legs, there are many varieties of walking robots. One of the most important is a four-legged robot (Quadruped). There are several tools available for designing, testing, and simulating the mechanical systems for mobile robots. These tools systems aid in the design of mechanical parts by providing a real-time image of the equipment, which aids in the detection of any potential issues, these tools also assist to test the robots in three-dimensional environments. For this paper, we use Robotics Toolbox on MATLAB, this Toolbox provides many functions that are useful for the study and simulation of legged robotics, for example, such things as kinematics, dynamics, and trajectory generation.

1.1 Quadruped robot

Considering the model of a walking system (Figure 1) with one leg. The aerospace coordinate convention has the x-axis forward and the z-axis downward, constraining the y-axis to point to the right-hand side. The first joint will be hip motion, forward and backward, which is rotation about the z-axis or $R_z(q_1)$. The second joint is hip motion up and down, which s rotation about the x-axis $R_x(q_2)$. These form a spherical hip joint since the axes of rotation intersect. The knee is translated b L_1in the y-direion or $T_y(L_1)$. The third joint is knee motion, toward and away from the body, which is $R_x(q_3)$. The foot is translated by L_2in the zdirectin or $T_z(L_2)$. The transform sequence of this robot, from hip to toe, is, therefore, $R_z(q_1)\ R_x(q_2)\ T_y(L_1)\ R_x(q_3)T_z(L_2)$. [1]

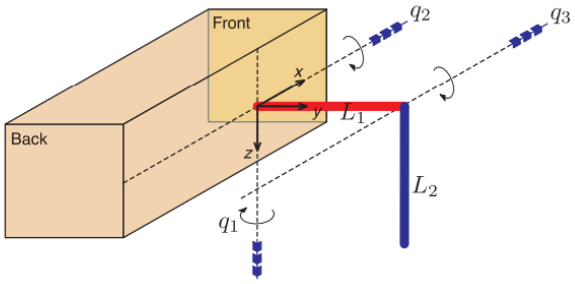

Figure 1.1 The coordinate frame and axis for the simple leg. The leg is shown in its zero-angle pose.

For the quadruped robot we consider the model with four legs, equally distributed along both sides of the robot body. The legs are connected to the robot body through the Hips joints. Each leg is made up of two parts: the Upper Leg and the Lower Leg. The upper leg is attached to the robot body through the hip joint and at the other end to the lower leg via a joint in the knee.

When taking motion into account the first consideration is that the end-effector of all feet moves backwards at the same speed as the ground plane propels the robot's body forward without its feet slipping. Each leg has a limited range of movement so it cannot move backwards for very long. At some point, we must reset the leg lift the foot, move it forward and place it on the ground again. The second consideration comes from static stability the robot must have always at least three feet on the

ground so each leg must take its turn to reset. This requires that any leg is in contact with the ground for ¾ of the cycle and is resetting for ¼ of the cycle. A consequence of this is that the leg must move much faster during reset since it has a longer path and less time to do it in. Since the trajectory is a cycle, we achieve this by having each leg run the trajectory with a phase shift equal to one-quarter of the total cycle time. [1]

1.2 Navigation

Robot navigation means the robot's ability to determine its position in its frame of reference and then plan a path toward some goal location. To navigate in its environment, the robot or any other mobility device requires representation, i.e., a map of the environment and the ability to interpret that representation. [2] The key to achieving the best path between two points is to use a map. There are many algorithms for robot path planning, such as Distance Transform, D* and Roadmap methods. In this paper, we work with the Probabilistic Roadmap Method (PRM). Which is a path planning method based on a random sampling strategy, which can solve the problem that effective paths are difficult to construct with most algorithms in high dimensional space. [1]

Probabilistic Roadmap Method (PRM): The mobileRobotPRM object randomly generates nodes and creates connections between these nodes based on the PRM algorithm parameters. Nodes are connected based on the obstacle locations specified in Map, and on the specified Connection Distance. we can customize the number of nodes, NumNodes, to fit the complexity of the map and the desire to find the most efficient path. [3]

2 Method

2.1 Motion planning

The original walking.m file, had a robot that stood still and moved its leg. To make the robot into a unicycle model the base of the robot as well as its legs were contentiously updated in a loop through the path. Two of the key motion primitives that the robot should have the ability to move forward, and the other is to stand still while rotating. For the robot to move forward its legs and body needs to be translated in the direction it is facing. This means that the x and y coordinates of the robot needs to change for each step that it takes as this robot only walks in the xy-plane. For the robot to orient itself in the xy-plane it needs to rotate. Rotation of the robot can be achieved with some simple trigonometry like finding the x and y values after a rotation and a suitable selection of a point on the body for the robot to rotate about. To rotate the robot about the center of its body all edge points must be translated to the center and then apply the rotation with trigonometry.

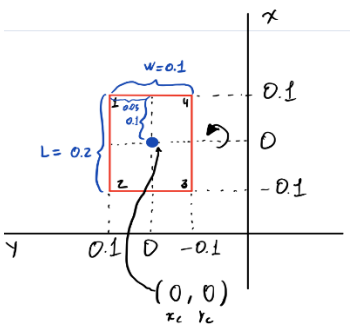

Figure 2.1 Rotation about the center point of body.

2.2 Path planning

`prm_planner(n, n_points)` generates n random paths between start and goal points in the loaded house map. The function has two input arguments, n is the number of maps that should be produced and `n_points`, which are the nodes. The function uses `PRM` object from the robotics toolbox and plans a path between two random start and goal points. If the random points generated are occupied, then it should skip the path planning process and rather generate other random start and goal points again and repeat the process of checking if it is occupied. If the points are not occupied then it should try to plan a path from start to goal, but if an error of no close by vertices should occur then the vertices should be increased with +500 `'npoints'`. This will take a longer time to generate the path, but this results in a function which does not crash that often due to empty vertices.

The waypoints generated by the function prm_planner was used to construct the path for the robot to follow. For this, the function `prmpath` was developed. `prmpath` takes in two arguments, the first is the waypoints and the second is the intervals. The function computes the difference between the points and computes the angle with the built-in `atan2` function in matlab in order to find the angles in all four quadrants needed to reach the next via point in the way points generated by `prm_planner`. `prmpath` works a lot like `createpath` except that the angles are calculated from x, and y coordinates form the `prm_planner` instead of being passed in as an argument.

2.3 Localization & Mapping

Robot localization is the process of determining where a mobile robot is located with respect to its environment. Localization is one of the most fundamental competencies required by an autonomous robot as the knowledge of the robot's own location is an essential precursor to making decisions about future actions. In a typical robot localization scenario, a map of the environment is available, and the robot is equipped with sensors that observe the environment as well as monitor its own motion. [4]

Robot localization is informed by measurements of range and bearing to landmarks. Sensors that measure range can be based on many principles such as laser range finding. A laser range finder emits short pulses of infra-red laser light and measures how long it takes for the reflected pulse to return. A scanning laser rangefinder contains a rotating laser rangefinder and typically emits a pulse every quarter, half or one degree over an angular range of 180 or 270 degrees and returns a planar cross-section of the world in polar coordinate form. If the robot pose is sufficiently well known, through some localization process, then we can transform all the laser scans to a global coordinate frame and build a map. [1]

2.3.1 Bresenham algorithm

For a robot to navigate through any area, it must localize and map the area by using a sensor. Lidar is a common sensor used in self-driving cars to collect data, so it can map an area with high accuracy. The lidar sensor calculates how long it takes for beams of light to hit an object or surface and reflect the laser scanner. The distance is calculated using the velocity of light. After we have collected data by using a laser, we can create a global coordinate frame and build a map. Since we know the range measurement, we can decide the coordinates of a cell that contains an obstacle, but it is uncertain to know about cells further along the ray. By making a binary occupancy grid as a matrix, it is possible to

use the Bresenham algorithm to find all cells along the ray-based on the robot's pose and the laser range. 0 indicates that cells are free and safe, while 1 indicates that it may hold an obstacle.

3 Results

3.1 Test of motion primitives

To create the path that the robot would follow the function `createpath(x_steps, phi, intervals)` was coded and used in order to make the robot move `x_steps` forward i.e. a list of steps and a list of rotation angles phi. The function returns a compounded matrix with x, y coordinates, and phi angles values all in chosen intervals. The function creates the intervals for the first element in the list of `x_steps` and then the first element in phi which is the rotation angle while holding the element of `x_steps` constant at last value in the interval.

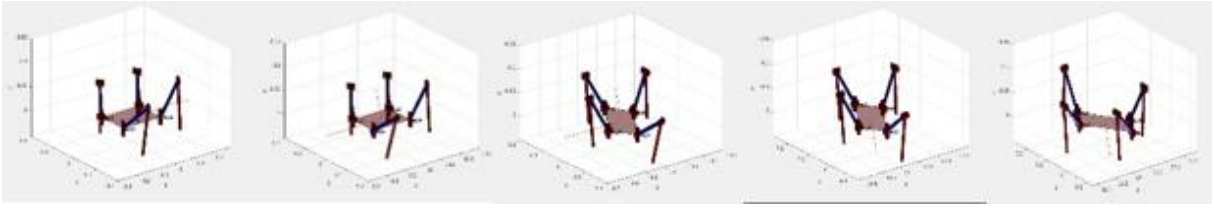

Figure 3.1 Here the robot walks 10 cm forward, then rotates pi/4 radians and walks 10 cm more and lastly rotates pi/2 this is all done in 400 intervals each. The red line indicates the path it should follow.

3.2 Planning paths in house map

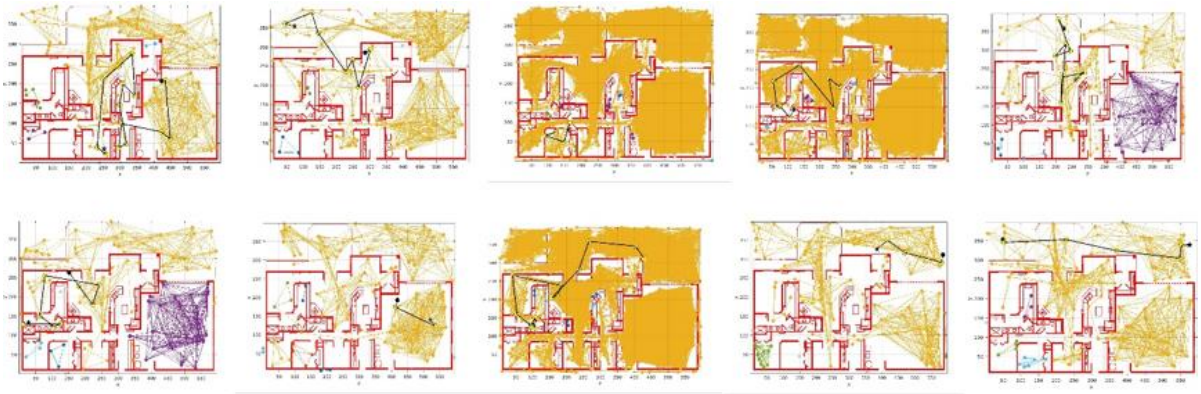

Figure 3.2 10 random generated paths between start and goal in house map using PRM algorithm.

To test the path planning algorithm, 10 random generated start and goal points and their paths were generated in the house map. In three of the paths generated the `'npoints'` were increased by +500 due to empty vertices this can be seen by the count result provided by the function `prm_planner` in MATLAB console as well as clearly in the maps (see Fig 3.2), where the yellow color of the points is a lot denser.

This function does not absolutely ensure 10 generated maps with path planning as empty vertices can still possibly occur and result in a crash and failing to query a path for the start and end goal. It is possible however to make the function only absolutely generate `n` paths with the given input of `'n_points'` however this is to the loss of having start and end goals that are not occupied, but their path will not be generated due to empty vertices.

3.3 Planned path for robot

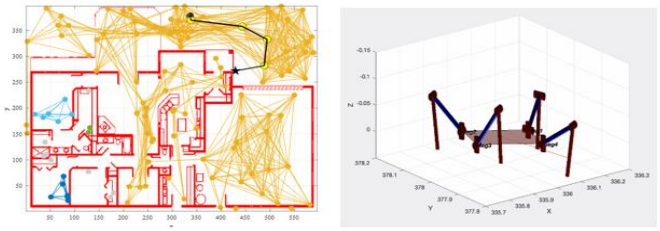

Figure 3.3 The robot following the path generated from prm_planner function

The robot starts by rotating towards the way point, then it walks straight forward towards the point and does continues to execute these primitive motions of rotating and walking until it has reached the end goal of the path generated from `prm_planner`.

3.4 Occupancy grid KillianMap

In order to modify the `scanMap` based on the `PoseGraph` of the raw data collected from MIT Killian Court, all cells in the scanned that have radar scans less than or equal to `M=-10` must be converted to the value of 1 which represents an occupied space. All other values are to be converted into 0. The function `MakeKillianMap` converts the `scanMap` into binary occupancy grid, its input arguments are the scanned map and the value of M. With this a binary occupancy grid has been achieved and it can be confirmed by checking if the `KillianMap` created by the function is equal to the original with built-in Matlab's function `isequal(MyKillianMap, KillianMap)`.

Using the `PRM` function and creating a path plan for the modified `scanMap` it is possible to cover the entire free space. This can be done with the selection 1000 `'npoints'` when using the planning method of `PRM` and the result can be seen in Fig. Alternative and efficient path planning algorithms are discussed in section 4.1.

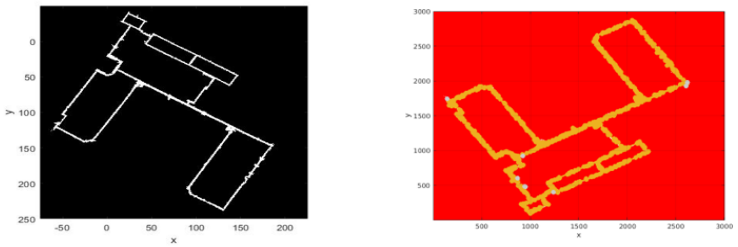

Figure 3.4 The plot to the left is the modified scanned map. Applying PRM and covering the entire free space.

3.5 Extraction of pose as a function of time with ICP

In order to extract the pose as function it necessary to extract an initial pose. This can be done scanning two closely or far apart nodes using the `scanxy` method. The scanned nodes are then passed into the `icp` function calculates the initial pose. The model `M` of the data is the `vertexlist`. `D` is the initial pose times the model of the data which is regarded as some noisy observed data. Now lastly

passing `M` and `D` to `icp` and plotting, will show the model and data points at each step as seen in Fig 3.5.

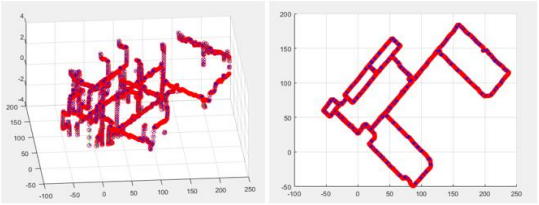

Figure 3.5 Matching of two-point clouds using ICP. The graph to the right is the image to the left from above.

The `icp` function does is that it matches two-point clouds: the model data (red) and data (blue). [1]

4 Discussion

4.1 Alternative path planning algorithms

For path planning, we used a probabilistic roadmap, but another popular path-planning algorithm is the rapidly exploring random tree. It starts with picking a random point within the environment, then we can create a line between a starting point and that point. For each iteration of the algorithm, we do the same process to add a new point and expand the tree. The tree continues to expand into the environment following this pattern. Once the tree reaches the ending point, we can return the path from the start node to the end node.

Dstar is seen as the best path planning algorithm for robots. It is the most efficient and suitable for path planning. It will find the optimal path between a starting point and ending point by using a graph structure consisting of nodes connected by edges. It tracks the cost of arriving which is the shortest path that connects the starting point and ending point. If the next position is an obstacle, the cost will set to infinity, because the pathfinder can never reach that point.

The reason why dstar is the most efficient path planning for robot is because it has other useful functionality such as replanning and path correction. These functions are very important if the robot discovers that the world is different to our map. The robot will correct itself to make sure it arrives to the end point. And if the robot discovers that the route is very costly, it can replan to find the shortest path to the endpoint. [1]

4.2 Further work

It was taken into consideration that the robot's physical size would crash into the walls of the house map generated by the `prm_planner`. To further improve on this and create a more realistic path planner when considering the size of the robot would be to increase the size of the occupied area i.e., the walls so that the true walls of the house map would not be reached.

For the last decade or two, robotics has been taking big steps forward due to human curiosity and problem-solving. Robotics has moved from imagination to our homes, offices and factories. They will become our partners that help us do more than we can do alone. Robots will save people time so they can focus on things they find interesting and important. We should see the next centuries with excitement as robots will get even better and improve our quality of life.

References

[1] P. Corke, Robotics, Vision and Control FUNDEMENTAL ALGORITHMS IN MATLAB®, Springer, 2017.

[2] WikipediA, 31 Deceber 2021. [Internett]. Available: https://en.wikipedia.org/wiki/Robot_navigation.

[3] MathWorks®. [Internett]. Available: https://ch.mathworks.com/help/robotics/ug/probabilistic-roadmaps-prm.html.

[4] S. a. G. D. Huang, «Robot Localization: An Introduction,» 1999.