Humanoid Robots - Turtle Maze

Cüneyt Erem[3277992], Kirill Surov[3287319], and Patrick Reitz[2906900]

Rheinische Friedrich-Wilhelms-Universität Bonn
Supervisor: Christopher Gebauer

Abstract. This Lab-report focuses on the task of a robot exploring and mapping an unknown environment in a simulation. The completed map is then used to calculate the optimal path from the starting position to a target position. The exploration is done with a wall-follower algorithm while simultaneously creating the map in log-odds notation from the robot position and camera data. After reaching the target, the robot is reset to its starting position and calculates the optimal path to the target with the A* algorithm. It executes this path until it reaches the target the second time.

1 Task

As a basis for the task, a simulation containing a robot equipped with multiple depth-cameras in a small maze was given. The robot can be controlled with the keyboard or with a robot-control function. The depth-cameras provide a constant stream of images and point clouds. First the robot is supposed to navigate the maze based on the camera data until it finds a target within the maze. During this exploration it should create a map of its environment. In the second step the map is used to calculate the optimal path between the starting position and the target, the robot is reset to its starting position and should execute the calculated path.

2 Approach

To complete the task, three main objectives have to be completed, their detailed description can be found in the next section 3. The basic idea is the following:

1. **Exploration:** The layout of the maze is simple and all walls are connected to the outer wall, which means no ring exists. This makes it possible to explore the maze by simply following the right hand wall until the target is found in the camera image. The work was split among all three people, Cüneyt and Patrick writing and testing the wall-follower and Kirill adding the target detection.
2. **Mapping:** The robot position is known exactly at all times, so the mapping algorithm can use the robot position and camera data to create an accurate map directly. No pose estimation like in SLAM is necessary. The final map is

a gridmap with a 5cm cell size, which is accurate enough while keeping the computational cost low. The mapping section was implemented and tested by Patrick.

3. **Path planing:** To plan a path based on the map, A* is a common and sufficient algorithm. It calculates the optimal path between the start and end point and is adjusted to keep the robots size in mind. This section was implemented by Cüneyt and Kirill, Patrick helped with testing.

3 Implementation

3.1 Initial Exploration

The initial task for the robot is to explore the maze on its own until it finds the target. This exploration step is executed by a simple wall follower algorithm, that always tries to follow the wall on the right hand side of the robot. To do so the algorithm processes the point clouds of the left, right and front camera to obtain the distance to the walls in multiple directions. The camera images are all rotated and translated into the robot coordinate system. As the walls are all flat and rectangular, it is sufficient to only check the center row of pixels, as processing the whole image does not add any further information but increases computational cost.
Four distance values are calculated by averaging over five neighbour points to cancel out noise. The right-side, right-front, front and left-front distance combined are a sufficient basis for the algorithm to decide where to go next. Figure 1 shows the four possible options that the robot has in a given situation.

1. **Turn right** As the robot is supposed to follow the right hand wall, it will always turn right when it is an option. It checks the right-front distance to see if there is an obstacle within a set distance, if not it will start turning to the right while moving forward .
2. **Go straight ahead:** If it is not possible to turn right, the robot will instead follow the wall to its right up to the next opening or obstacle ahead. This happens as long as the distance to the next obstacle ahead is below a set distance. While moving forward it also checks the distance to the left and right side to make sure it stays in the middle of the corridor.
3. **Turn left** If the right side and the front are blocked by an obstacle, the robot will check the left side. If there is an opening it will start turning left while moving ahead.
4. **Dead end - turn around** In the last case all options ahead of the robot are blocked by an obstacle and it reached a dead end. The only option is to turn around anti-clockwise for 180 degrees so it can start following the wall back.

The target inside the maze is a red pole, which is clearly distinguishable by its colour, as the rest of the maze is either white or grey (unlike shown in the simulation top-down view, where the walls are red as well). To find the target the front camera image is scanned for a red object, if such an object is found, its size determines when to stop the robot, as he is close enough to the target.

Difficulties : Adjusting the distance parameters has proven itself to be quite difficult. The robot would sometimes get stuck in a dead end oscillating between turning left and right, as the slight change in angle could put the right or left side distance over or under the threshold to turn. These issues were not always reproducible and would sometimes reappear after multiple successful tests, so it is never clear if the current parameters are optimal or not.

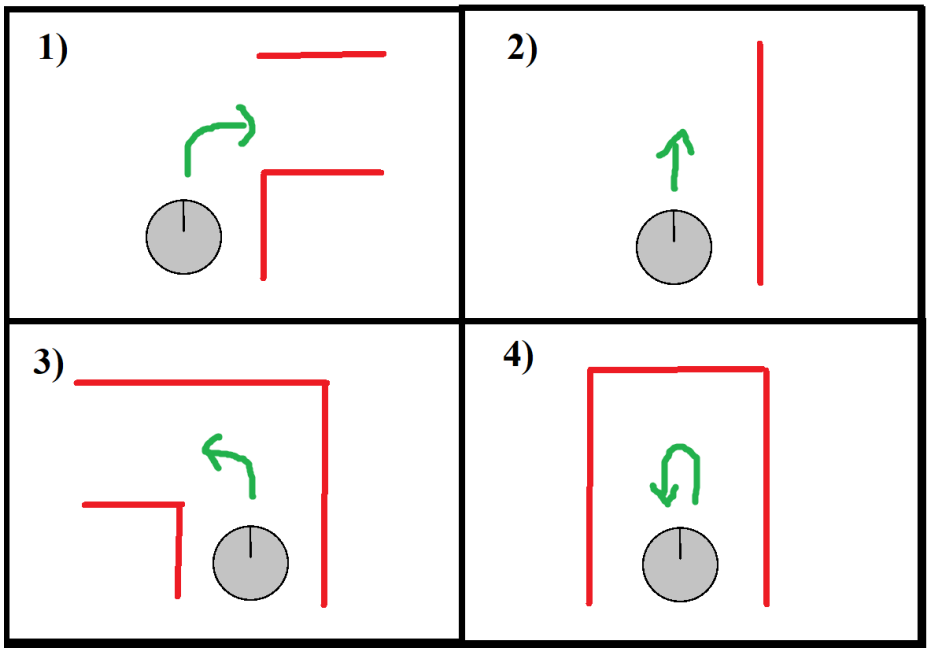

Fig. 1. Graphical representation of the four possible movement choices for the robot when following a wall

3.2 Mapping

The mapping algorithm is based on the Lecture *Cognitive Robotics* [1] Chapter 6 *Mapping with known poses*. The underlying grid has a cell-size of 5cm, which results in a 100 x 100 cells map for the given maze size. This map is saved as a two dimensional array in the RobotControl-class and gets updated for every camera frame in the simulation. Each cell contains a probability for the cell being occupied in log-odds notation.
To update the map, three camera images are processed every frame, the left, right and front camera. Once again only the center line of each image is used to safe computational cost. Each pixel in the image has a corresponding point, showing the wall in camera coordinates. This point is first translated and rotated

into the robots coordinate system with the given external camera parameters. Afterwards the point gets translated and rotated again based on the robots position and orientation, now it represents the wall in world coordinates. These coordinates get transformed into the corresponding grid cells, which are then updated to contain a higher probability of being occupied. By doing this for every pixel in all three camera images, the walls currently seen by the robot get added to the map.
To obtain the information about the open space between the walls each point gets passed on to the Bresenham [2] algorithm as well. With the camera position as the start-point and the image point as the end-point, this algorithm returns a list of all free grid cells between these two points. Their probability of being occupied is then lowered accordingly. It is important to note, that each grid cell only gets updated once per camera frame, so if multiple pixels point to the same grid cell, they can be skipped to safe computational cost.
The final result of the mapping algorithm is shown in figure 2. On the right side is a photo of the maze in the simulation, serving as the ground truth. It shows the robot in front of the found target (red half circle). On the left side is the final map printed to a text file, where a "1" corresponds to an occupied cell, "0" to a free cell and "X" an unknown cell that either was not scanned or was sometimes detected as occupied and sometimes unoccupied. To make it easier to see, the walls are painted red. The green circles show areas that were initially not mapped well, as they were in a dead angle of the front camera, to fix this issue the two side cameras were later added. Now there are no more dead angles and the map is fully known. The quality of the map decreased slightly when using all three cameras, but the result is still more than sufficient for the following path planning step.

Difficulties At first the mapping had a high computational cost, resulting in a strong lag in the simulation, dropping to one frame per second. This was fixed by only updating each grid cell ones and skipping duplicates. There are appear some slight artifacts on the wall sometimes, a few pixel outside the wall being detected as occupied. This is due to the algorithm giving a higher weight to occupied detections compared to free detections. In the end its better to have a few unnecessary wall pixels resulting in a slight detour later on instead of actual edges being marked as unoccupied which could result in a crash during path-planing and execution.
Its important to note that the simulation can start lagging quite a lot if the PC running it is not good enough. At a low frame rate the mapping becomes a lot worse, as less data is generated.

3.3 Path-Planning

To execute the path planning phase, we used the A-star algorithm [3] that evaluates each grid cells positions and calculates the optimal path from the start point to goal point. We have implemented the algorithm by defining open and

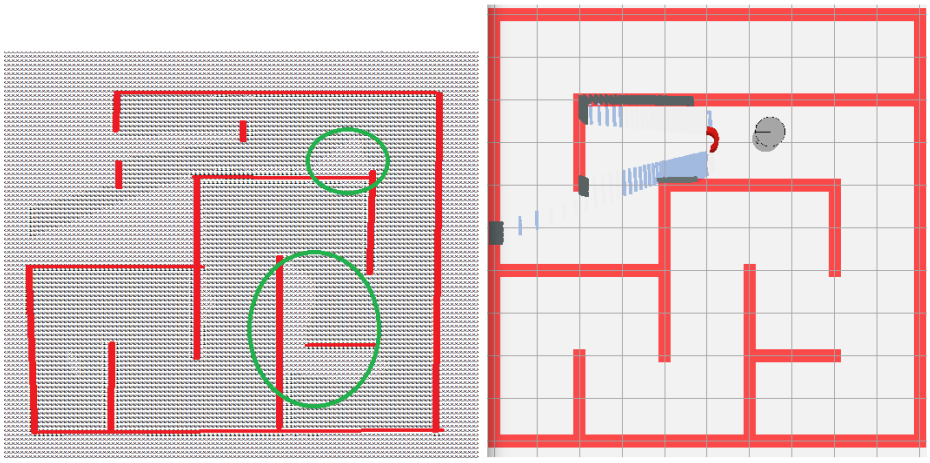

Fig. 2. Comparison between the map calculated by the algorithm on the left side and a photo of the real maze on the right side

closed states.
Until lowest rank in the open state is reached, we remove the minimum node from the open state and add the current node to closed state. For each neighbor of the current state, we calculate the values as follows; when neighbor is in open state and cost is less than g value (actual cost from start to n state) of neighbor, then we remove the neighbor from the open state, thus better path is found. When neighbor is in closed state and cost is less than g value of neighbor, then we remove that neighbor from the closed state. When the neighbor is not in both the open state and closed state, then we set g value of neighbor to the cost and add neighbor to the open state. Then we calculate f value (it is summation of the g and h) of successor as such prior rank to g value of neighbor and h value (heuristic that is estimated cost from state n to goal state) of neighbor summation and set parents of the neighbor to the current state.
Until we have done the same procedure for each values of the neighbor and reach the open state as empty state, we have become calculated the most efficient way from start state to final state. We reconstruct path by returning from end point to start point as considering their rank.
Here, we stated with the initial point 0, 0 and end point fx, fy. Then path returns each x and y positions of the map for current states. Then, after iterating the robot for the second movement, it follows these coordinates of the nodes, and apply the same following path algorithm as implemented in the exploration phase, and execute the plan to apply the A-star efficient path.

Difficulties When the result returns the final path, there are many states that have the same positions and only evaluates the small steps of the coordinates for each pixel. Because the maze that each pixel contains many grid cells of only 5cm

per frame, executing the path directly for each grid makes the movement slow and hard to execute the movement for the robot speed and velocity. Therefore, we subtracted the unnecessary points that contains almost having same positions repeatedly that we slow down the robot movement and execute necessary points only.

4 Open Issues

One issue remaining is the robot sometimes getting stuck during the exploration phase when confronted with a unusual situation. This can either occur when starting the simulation at an intersection with no wall on the right side or when the simulation lags and fails to update the movement commands. In that case the robot may get stuck either going in a circle never seeing a wall close enough to follow it. This could be fixed by setting a higher threshold value for the distances, but that resulted in other issues appearing. The second outcome is the robot standing directly in front of a wall switching between turning left and right. In A-star algorithm, it returns the planning path correctly but whenever we run the second termination, there is small bug for the second travelsal so that it should be fixed properly.

5 Conclusion and Future Work

In conclusion, we examined whether the Turtle bot mobile robot examine three main steps correctly step by step and gives expected results. It navigates the maze environment by using its three camera systems.
Firstly, It explores the walls inside of the maze and chooses a direction according to different situations. When the target is found, it detects the object and finishes the travel. Secondly, the grid map is created by using the robot positions during the entire exploration. It keeps the calculation cost low as much as possible. Lastly, when the robot reaches the final destination, it computes the efficient shortest path by using A-star algorithm node by node and creates a path. When we try to execute from zero, it follows this path and reaches endpoint efficiently.Therefore, the robot executes most of the steps correctly although there can be seen some minor bugs.
For the future work, dead end problem in some points and executing A-star algorithm more accurately can be fixed and apply all the steps much accurately. Also, other approaches like exploration can be done more efficient way instead of searching right walls all the time. Even if A-star algorithm is efficient, some other algorithms like ARA-algorithm or etc can be efficient a bit.

References

1. Homepage Insitute Computer Science IV, https://www.ais.uni-bonn.de/WS2021/L_Cognitive_Robotics.html, Lecture Cognitive Robotics WS20/21, Chapter 6 "Mapping with known poses"

2. Christopher Gebauer, Exercise 8 for the lecture "Humanoid Robotics" SS21 https://www.hrl.uni-bonn.de/teaching/ss21/lecture-humanoid-robotics/lecture-humanoid-robotics
3. Russell, Stuart J. (2018). Artificial intelligence a modern approach. Norvig, Peter (4th ed.). Boston: Pearson