CSE 276A: Intro to Robotics
Homework 4

Orish Jindal (A59010554)

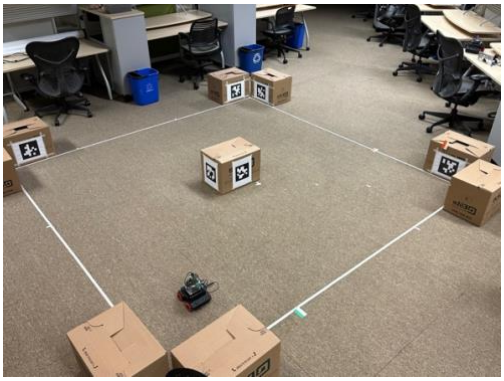
Introduction:

This report focuses on the importance of path planning and elaborates the popular methods to address two specific requirements about the nature of the path traced by the robot (Distance optimality and maximum safety).

Environment and Problem Statement:

The environment that needs to be mapped is in the form of a 10ft ×10ft area with 8 landmarks (AprilTags) placed two on each side of the square environment facing inwards and four April tags placed covering sides of obstacle placed in the middle. The obstacle size was under 1.5ft ×1.5ft.

The environment:

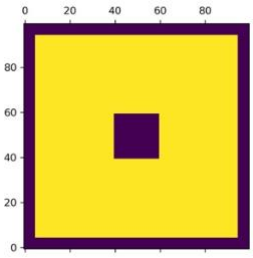


The provided robot needs to be placed at one corner and it needs to reach the other corner while avoiding the obstacle. The AprilTags does not need to be unique, and the robot should be able to work around that. There are two pathways requires:

1) The robot needs to trace the shortest path possible.
2) The robot needs to trace the safest path possible.

Environment Representation:

A grid map was prepared for the environment with 1:10 scale i.e., a matrix with dimensions 100×100 was initiated. Each 1ft ×1ft sub-block of the real environment is accounted by the 10×10 grid squares on the environment matrix. The obstacle being of 1.5ft ×1.5ft is represented as 15 ×15 set of grids. Extra padding of 0.5 ft (nearly 15 cm) was used to cover the outside of the obstacle and the inside of the boundary to account for the robot's size and potential errors. The top view of the matrix is represented as shown in the below mentioned picture where the yellow part represents the free field for movement and the other part represents the obstacle and boundary space along with the mentioned padding.



Planning Algorithms:

There are several available search-based and sampling-based planning algorithms to choose from. The particularly famous ones are Djkstra, A* and other versions of A*.

For the **Safest Path**:
I chose A* algorithm to obtain the waypoints for tracing the safest path in the environment as it serves as a perfect balance between the implementation complexity and speed of obtaining results. The basic structure for the algorithm is as follows:

```
1:  OPEN ←{s}, CLOSED ←{}, ε = 1
2:  g_s = 0, g_i = ∞ for all i ∈ v \ {s}
3:  while τ not in CLOSED:
4:      Remove i with smallest f_i := g_i + εh_i from OPEN
5:      Insert i into CLOSED
6:      for j ∈ Children(i) and j not in CLOSED:
7:          if g_j > (g_i + c_ij + k_ij) then
8:              g_j ←(g_i + c_ij + k_ij )
9:              Parent(j) ← i
10:             if j ∈ OPEN:
11:                 Update priority of j
12:             else:
13:                 OPEN ← OPEN ∪{j}
```

- *s* is the start position and *τ* is the target position for the robot.
- h_i in the above pseudo code represents the heuristic function's value when on grid *i* is considered which is the Euclidian Distance between the grid *i* and the target grid.
- c_{ij} is the cost of moving from the grid *i* to the grid *j*. It is kept as **1** for if the robot from any grid to one the 8 grids that touches our grid (i.e., **children** of our focused grid) in the map representation. It accumulates with each step.
- An additional cost k_{ij} accounts for the safety of a particular grid. It proportional to the sum of inverses of the shortest distance of a grid from the obstacle space and the boundary space.

With these factors, the robot would tend to prioritize the grids with lower $c_{ij} + k_{ij}$ values and hence would choose the grids with higher distance from the obstacle and map boundary. The results are shown in the results section.

For the **Shortest Path**:
Since the A* algorithm implemented via the grid method will have a limitation of robot movement in only vertical, horizontal and diagonal (±45 degrees), I implemented a much simpler method for this problem.

Two lined were extended from each corner of the padded obstacle: one to the start position and other to the target position of the robot. The shortest addition of these two lines (for each of the four corners) that does not go through the obstacle (belongs to the rectangular family) is selected as the shortest path. It can also be obtained via implementing the A* mentioned above but without the additional k_{ij} cost and then smoothening the thus obtained lines to account for angles other than ±45 degrees. The results thus obtained are shown in the results section.

Architecture of the project:

The architecture of the code implemented to solve the problem statement includes the following four nodes. The function of each of the four nodes is described below:

Node 1: Camera node initialized using rb5_camera_main_ocv.launch executable
This node initializes the main camera node, published as /camera_0. It enables the camera to capture image frames of the environment basis its field of view.

Node 2: April tag detection node initialized using apriltag_detection_array.py executable
April tag detection node subscribes to the camera node to identify AprilTags visible in the camera frame. Each AprilTag has been assigned a unique id. If the tags are present, this node calculates the position and

orientation of robot with respect to tag in terms of (x, y, z) coordinates and orientation quaternion. This information is published to the /apriltag_detection_array rostopic.

Node 3: MPI control node initialized using hw2_mpi_control_node.py executable
MPI control node subscribes to /twist rostopic to obtain twist vector calculated by the PID control loop. It is then scaled using a calibration factor. The Jacobian matrix transforms the twist vector into PWM signals for individual motors. The PWM signals for each of the four motors are sent to MegaPiController firmware to control the motors individually.

Node 4: HW solution node initialized using hw4_sol.py executable
This is the main node which executes the following functions required to solve the problem statement:

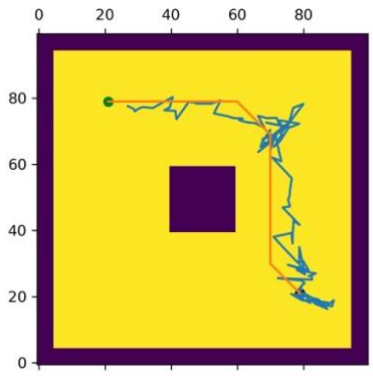
1. Search based motion planning algorithm – A* is implemented to obtain the waypoints for trajectory with minimum distance as well as trajectory with maximum safety.

2. PID control loop is implemented which takes the waypoints calculated by A* algorithm and enables the robot to follow a desired trajectory using position feedback from the AprilTags.
    a. This node subscribes to the /apriltag_detection_array rostopic to obtain position and orientation of robot with respect to tag.
    b. As the world frame coordinates of the AprilTags are known, the world frame coordinates of the robot are calculated using the feedback from tags. This process is executed continuously to update the current position of the robot at each step.
        i. World frame coordinates of the robot are calculated by taking into account all the visible tags in the camera frame.
    c. Once the current position is updated, PID control loop calculates the error w.r.t target position and transforms it into a twist vector. This vector is published to the /twist rostopic.
    d. These steps are executed continuously in a loop until all the waypoints required to follow the given trajectory are covered.

    The algorithm is designed to consider visibility of the repeated tags in the environment. While the case of unique tags is simple, the case of repeated tags is dealt by including each set of world frame coordinates of the repeated tag. If the id of the tag visible in camera frame corresponds to that of a repeated tag, the world frame coordinates of the robot are calculated w.r.t each instance of repeated tag. For each set of robot coordinates, the error is calculated w.r.t its previous position. The set of coordinates which give the least error is considered as the new current position of the robot.

Results and Discussion:

For the **Safest Path**:
The below mentioned figure represents the results obtained for the safest path traced by the robot.

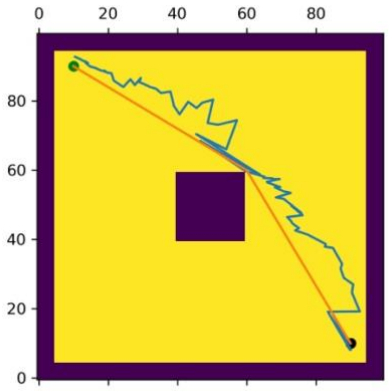


The overall figure represents a 100x100 grid map as mentioned in the representation. The orange line represents the path calculated by the A* algorithm which is fed to the PID controller in form of discretized waypoints. The blue line represents the actual path traced by the robot on the field as recorded by storing the current positions of the robot. Black dot is the starting point and green dot represents the end point of the motion.

The video thus obtained is as follows: https://youtu.be/kmp3vgNX7hQ

The P, I, D values that were kept are 0.08, 0.008, 0.35 respectively. The relatively higher value od D explains the sharp edges and sudden trends in the actual path as shown. This is because with a higher D value, the robot will tend to correct its position quickly to support the higher speeds (because of P value and calibration) and minimize overshoot.

For the **Shortest/optimal Path**:
The below mentioned figure represents the results obtained for the shortest/optimal path traced by the robot.



The overall figure represents a 100x100 grid map (along with padding) as mentioned in the representation. The orange line represents the path calculated by the algorithm mentioned in the algorithm section which is fed to the PID controller in form of discretized waypoints. The blue line represents the actual path traced by the robot on the field. The P, I, D values are same as the previous case. Black dot is the starting point and green dot represents the end point of the motion.

The video thus obtained is as follows: https://youtu.be/oM9HmLbQD8w

Following commands need to be executed in the rosws in order to mimic the results shown in the videos.

1. source devel/setup.bash
2. roslaunch rb5_vision rb_camera_main_ocv.launch
3. rosrun april_detection april_detection_node
4. rosrun rb5_control hw2_mpi_control_node.py
5. rosrun rb5_control hw4_sol.py