University of California San Diego

CSE 276A: HW4 (Path Planning)

Varun Vupparige
November 24, 2022

1 Introduction

An autonomous system consists of various subsystems such as sensing, perception, path planning and control systems, and the study of path planning algorithms has been an important research topic. One of the major challenges on the way to obtain robot autonomy is path generation which is safe, reliable and dynamically feasible. The planned path of the agent can be evaluated based on metrics such as path length, movement time, energy consumption and risk levels.

In this assignment, we are required to design a minimum time and maximum safety path planning algorithm. The robot has access to the information of the landmarks. The obstacle of size 1x1 - 2x2 ft is placed in the middle of the workspace, and landmarks are placed around the obstacles to help localization.

2 Architecture

In this section, we describe the detailed architecture, different ROS nodes, interdependecies and their topic information.

- RB5 vision: the function of this node is to provide the images according to its sampling rate to the topic *camera*.
- April Tag: the function of this node is to provide the transformation of the perceived april tags with respect to the camera frame to tf topic. This information is used to localize the robot along with the PID controller.
- Static TF Broadcaster: these are set of 9 nodes which define the location of the placed april tag landmarks in the world frame. This information is used by the tf topic to compute transformations.
- TF topic: this ROS defined topic subscribes to the april tag and static tf broadcaster nodes to compute the transformation of the camera with respect to world frame.
- Offline Planner: this python script computes the waypoints from start to goal state based on the weighted a star algorithm, and these waypoints are saved in txt file to be read by the PID controller node.
- PID Controller: this node subscribes to the tf topic to receive the localization information of the RB5 robot with respect to world frame, and computed desired twist based on PID control parameters is published to the MPI control node which drives the motors of RB5

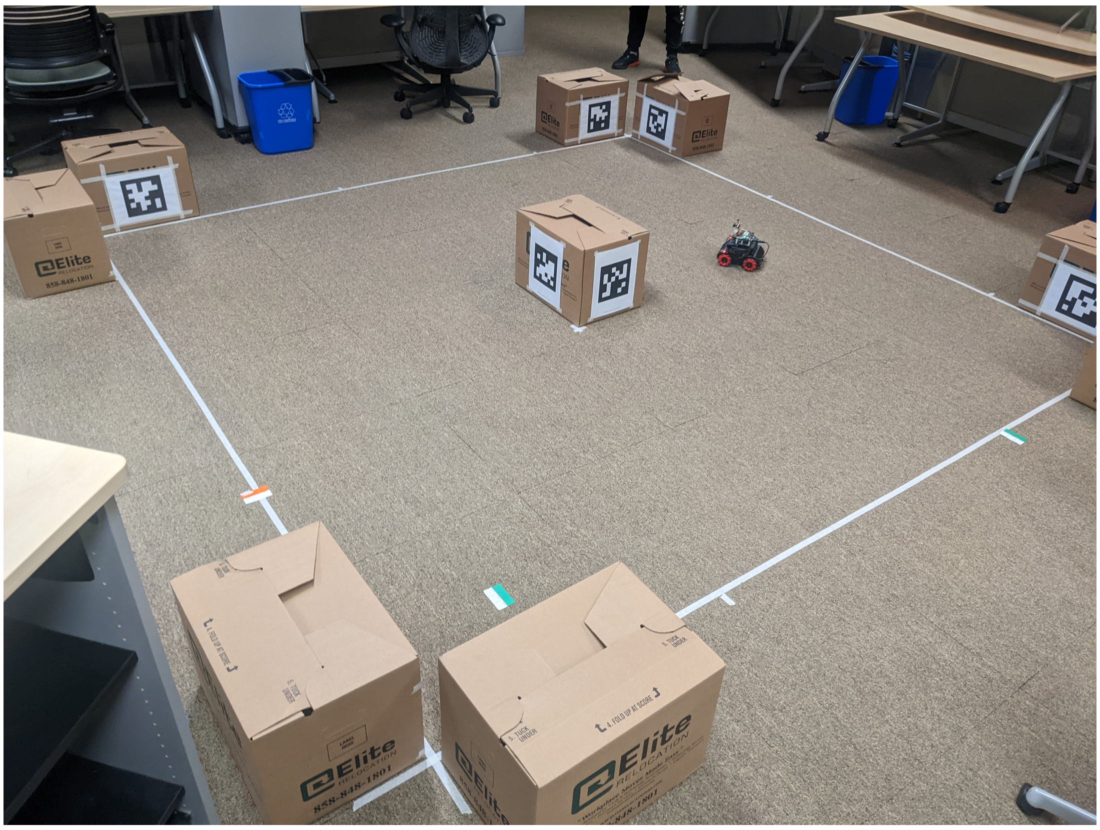

Figure 1.1: Constructed Environment with Landmarks

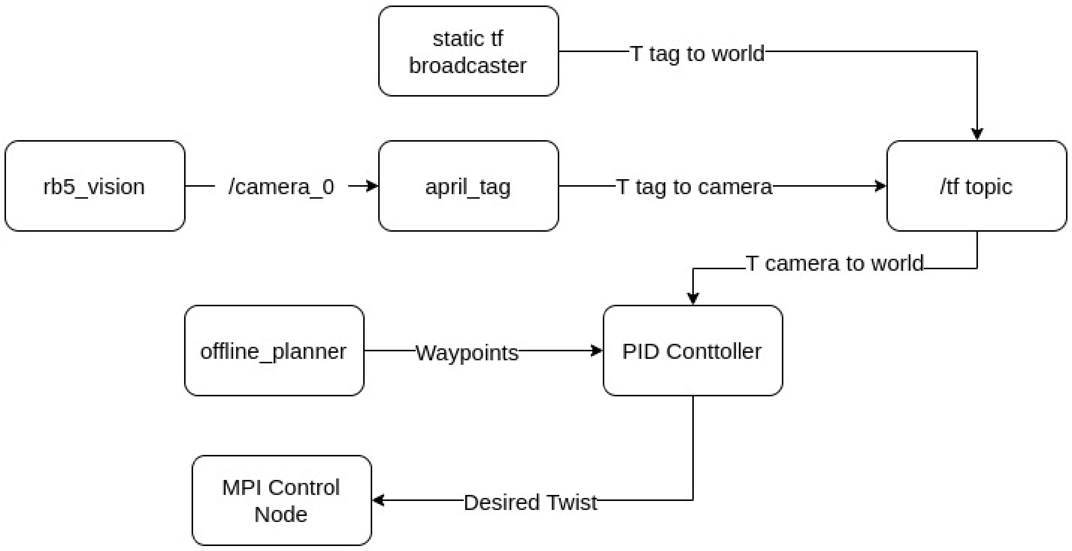

Figure 2.1: HW4 ROS Architecture

3 Representation, Planning Formulation and Algorithm

3.1 Representation

The map of the environment is represented as a binary occupancy grid. Occupancy grid is to represent a map of the environment as an evenly spaced field of binary random variables each representing the presence of an obstacle at that location in the environment. We basically construct a matrix of the size based on the discretization factor, and the location of obstacles corresponds to one entry in the matrix. The below figure corresponds to the ground truth binary occupancy grid map of the environment.

3.2 Formulation

In this section, the problem of path planning using search-based methods is formulated as Discrete Feasible Planning model by carefully explaining each of its parameter: A typical discrete planning model is composed of following parameters:

- X: Discrete/continuous set of states
- U: Discrete/continuous set of controls
- x_0: Initial robot/agent state
- $x_{t+1} = f(x_t, u_t)$: Motion Model/State Transition Function
- x_g: Goal state

Lets formulate the parameters of the Discrete Feasible Planning model in context of our project:

3.2.1 State Space

The state X is a vector consisting the location of the agent in terms of the x and y coordinates which specify the location of the agent with respect to the origin $(0, 0)$.

$$X = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \text{ where, } x, y, \theta \in \mathbb{R} \tag{3.1}$$

In this project, environments considered are two-dimensional grids consisting of static obstacles, moving goal and an agent. It is a occupancy grid map where the map of the environment is represented as evenly spaced binary variables. Therefore the state space is bounded by the size of the environment. The robot position is indicated by a blue square, while the target position is indicated by a red square and obstacles are denoted by black blocks.

$$\begin{aligned} 0 \le x &\le map_{x,max} \\ 0 \le y &\le map_{y,max} \\ map_{black} &= obstacles \\ map_{blue} &= robot \\ map_{red} &= target \end{aligned} \tag{3.2}$$

3.2.2 Control Space

The control space $\mathcal{U}$ is the set of given actions taken by the agent to reach the final goal (τ). The control is u is taken to reach these neighboring states. The robot has eight control actions and the target has only four control actions since it has no diagonal movement. The cost of reaching any valid possible state is equal to unity. These action can be mathematically formulated as follows:

$$u_t = \begin{bmatrix} (1,0) & (1,1) & (1,-1) & (-1,1) \\ (-1,-1) & (0,1) & (0,-1) & (-1,0) \end{bmatrix} \tag{3.3}$$

Each element of the tuple denotes the translation in x axis and y axis. Using the definition of state space and control space, lets define the motion model.

3.2.3 Motion Model

Based on the definition of state space X and control space U, we can now define the motion model. Given a robot's state x_t, and control input u_t, the state x_{t+1} can be modelled as a probabilistic/deterministic function given by,

$$x_{t+1} = f(x_t, u_t) = p_f(.|x_t, u_t) \tag{3.4}$$

Since our project is deterministic in nature, we can discard the probability mass/density function. For a particular state, there can be eight possible next states based on the control actions. Few of the states which results in an obstacle state or out of bounds state are invalid.

$$x_{t+1} = x_t + u_t \tag{3.5}$$

3.2.4 Initial State and Goal State

The initial state x_0 represent the starting state of the robot/agent. The goal state x_g represent the target state of the robot/agent. Thus, the task of planning algorithm is to find a finite sequence of actions that when applied, transforms the initial state to some state in goal.

3.3 Algorithm

Now that we have formulated the deterministic shortest path problem using the discrete feasible planning model, lets discuss about one of popular search based planning algorithm to solve this problem. Search-based algorithm are an extension of the label correcting algorithm where we discover the nodes with lowest cost, and update the cost of their children nodes.

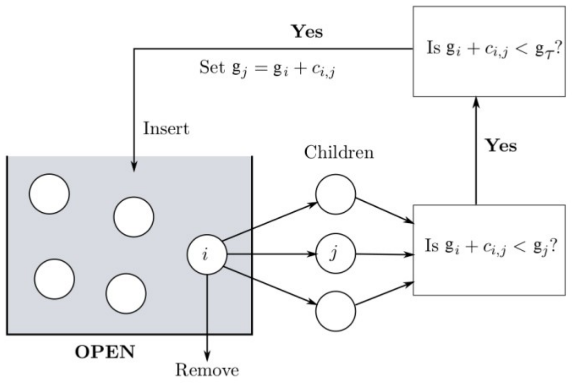
Yes
Set $g_j = g_i + c_{i,j}$
Is $g_i + c_{i,j} < g_T$?
Insert
Children
Yes
i
j
Is $g_i + c_{i,j} < g_j$?
OPEN
Remove

Figure 3.1: Label Correcting Algorithm

These set of nodes which can potentially be part of the shortest path are stored in OPEN list. The order in which we remove a node from the OPEN list defines different type of search algorithms such as Best-first search, Depth-first search and Dijkstra's algorithm.

The fig 3 depicts the flowchart explaining all the steps involved in the label correcting algorithm. Here, g_j is the present lowest cost of reach node j from the start node s, c_{ij} is the cost of travelling from node i to node j. Before diving into the A* algorithm, let's introduce a special type of function called as heuristic function. This function gives an estimate of the distance from the present node to the goal node. For A* algorithm to compute an optimal path, the heuristic function should be admissible, meaning that the function should output an underestimate of the actual distance between present node and goal node.

Introducing the heuristic function in the A* algorithm would result in a more stringent criterion such that it biases the search direction towards the goal node. Furthermore, the accuracy of the heuristic function is directly proportional to the efficiency of the algorithm. So, we change the condition for node expansion and g value update as follows:

$$g_i + c_{ij} + h_j < g_{tau} \tag{3.6}$$

Few commonly used heuristic functions are as follows:

- Euclidean Distance:$h_i = \|\mathbf{x}_\tau - \mathbf{x}_i\|_2$
- Manhattan Distance:$h_i = \|\mathbf{x}_\tau - \mathbf{x}_i\|_1 := \sum_k |x_{\tau,k} - x_{i,k}|$
- Diagonal Distance:$h_i = \|\mathbf{x}_\tau - \mathbf{x}_i\|_\infty := \max_k |x_{\tau,k} - x_{i,k}|$

$$g_i + h_i = f_i \tag{3.7}$$

$$g_i + \epsilon h_i = f_i \tag{3.8}$$

The eqn 3.6 depicts the condition for a* algorithm. Thus, the f value is defined as the sum of g value and the heuristic value as shown in eqn 3.6. In an *epsilon* consistent heuristic, we scale the heuristic by a parameter *epsilon* which trusts more on the heuristic value.

4 Implementation and Results

4.1 Implementation

4.1.1 Algorithm 1: Minimum Time

The algorithm 1 summarises the implementation of A* path planning algorithm for our project to compute a minimum time path. To implement this algorithm, we need to store nodes in two lists namely OPEN and CLOSED. The OPEN list is a priority queue. A priority queue is an abstract data-type similar to a regular queue or stack data structure in which each element additionally has a "priority" associated with it. We use pqdict() method in python where the lowest value has the highest priority, which is exactly what we need for implementing A* algorithm. We initialize the start state in OPEN list and set it's value to be zero and g value of all other nodes is set to be infinity. The CLOSED list is initialized as an empty list.

Under the while loop, we remove the node with smallest f value and insert it into the CLOSED list. Then we update the g value of its children nodes, and store the information of parent node. These promising nodes are added into the OPEN list and pqdict() method takes care of updating its priority. In order to recover the path with the lowest cost, we back track the parent node from goal until we reach the start. For this particular project, the robotplanner function computes only the immediate next step of the robot.

4.1.2 Algorithm2: Maximum Safety

The algorithm 2 has a similar implementation like the algorithm 1 with added safety parameters as explained in this section. The Minkowski sum is the set of all coordinates in which one polygon overlaps another. It is computed in four easy steps. In the following, we label the stationary polygon the obstacle and the moving polygon the robot. The Minkowski sum is useful for robot navigation because, once it is computed, the problem of finding a collision-free path for a polygon-shaped robot moving through a workspace filled with obstacles is reduced to finding a path for a point-shaped robot through a configuration space filled with Minkowski sums. We use this mathematical technique to augment our obstacles to find a max safety, collision free path.

4.2 Results

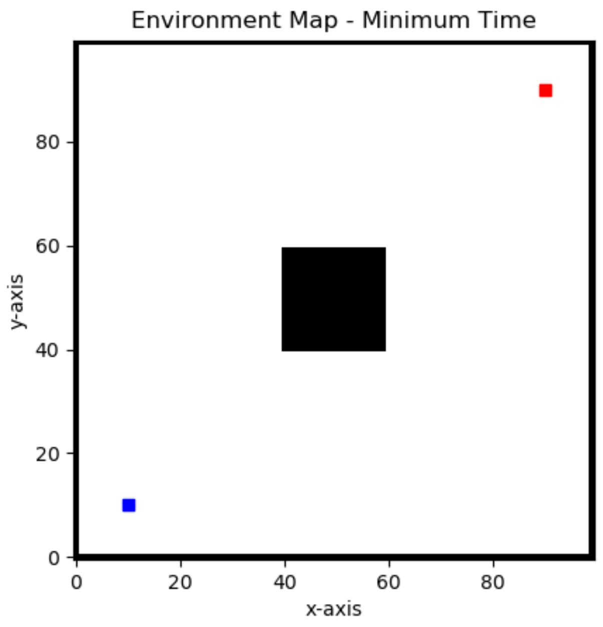

Figure 4.1: Binary Grid Map - Minimum Time

4.2.1 Video Links

link 1: Max Safety
link 2: Min Time

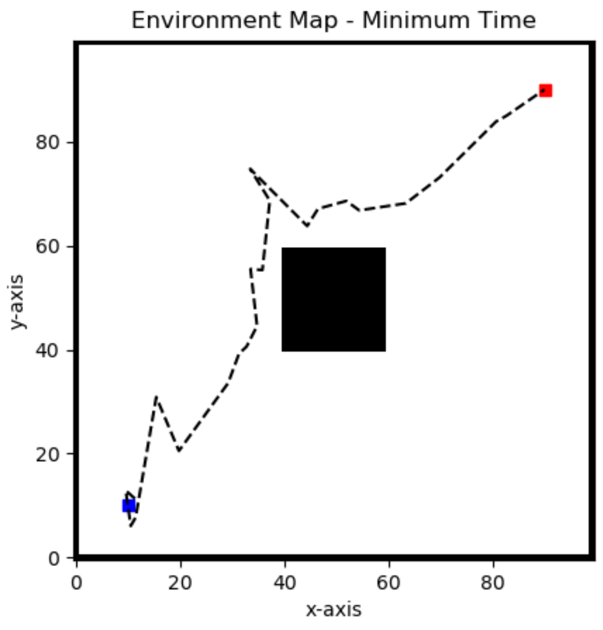

Figure 4.2: Executed Path - Minimum Time

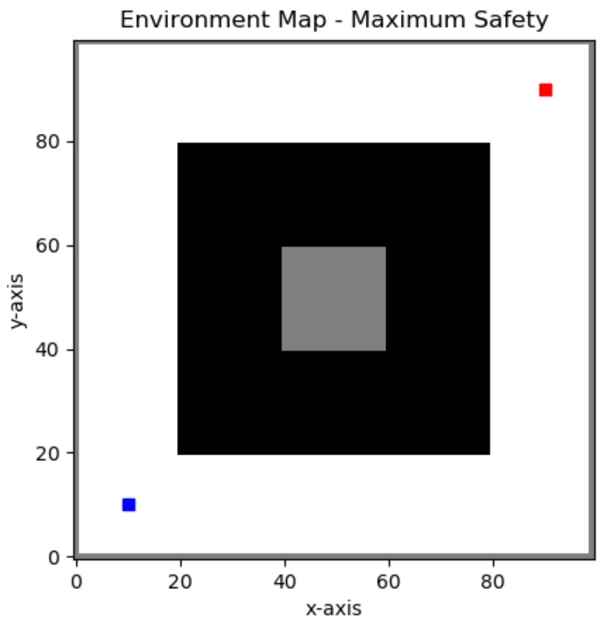

Figure 4.3: Binary Grid Map - Maximum Safety

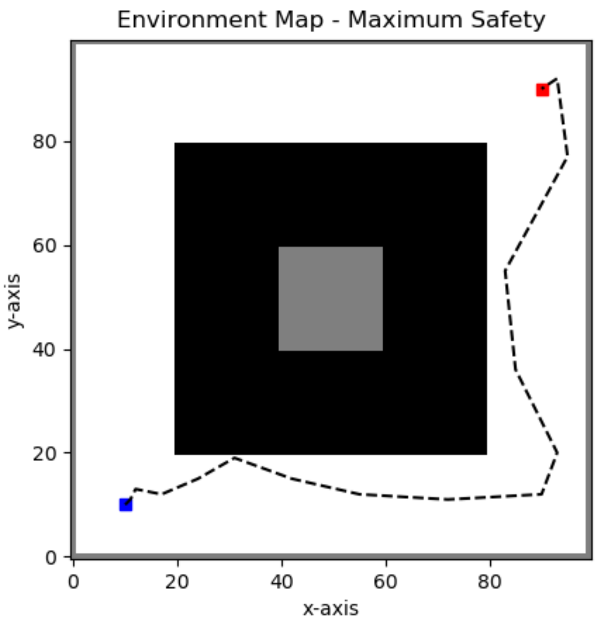

Figure 4.4: Executed Path - Minimum Time