University of California San Diego

CSE276A: Introduction to Robotics

HW4: Path Planning

Novermber 28th, 2023

Author	Student ID
Mazeyu Ji	A59023027
Zejia Wu	A59026752

Link to the Video:

A* (Shoter Path):

https://drive.google.com/file/d/1NFzCmvf9zmcCO7j58Lb463Bd921EpvUH/view?usp=sharing

APF (Safer Path):

https://drive.google.com/file/d/1iRJFWni840nqhxZM-1LfAVGnAjSFIMbi/view?usp=sharing

1 World Representation

We set up a square area measuring 2.0 meters by 2.0 meters and use a two-dimensional grid as a simplified representation of the world, where each grid cell represents either open space (value 0) or an obstacle (value 1). Obstacles are created by placing a square area of size 0.3×0.3 at the center of the grid. Two AprilTags are placed on each side of the square and one on each side of the obstacle. Each AprilTag has unique Tag id. We set $(1.5, 0.5, \frac{\pi}{2})$ as the initial pose of the robot, and $(0.55, 1.55, \pi)$ as the goal of the robot. Our map is shown as in Figure 1.

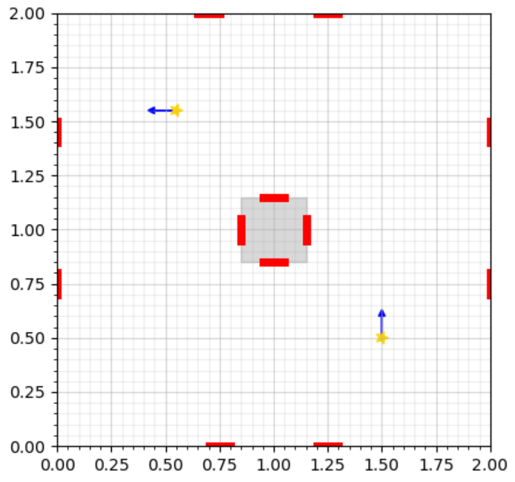


Figure 1: The sketch of our map in the top-down view layout, where the golden stars and blue arrows represent the initial pose and the goal of the robot, red lines represent AprilTags. The gray area indicates the obstacle, i.e. grids with value 1.

2 The Path Planning Algorithms

We consider two different algorithms, the A Star(A*) algorithm and the Artificial Potential Field (APF) algorithm. Each algrithm has its own advantages. A* has greater precision and robustness, often guaranteeing the identification of the lowest cost path from the starting point to the destination in many scenarios. The APF algorithm, on the other hand, is more suitable for dynamic environments. It guides the object towards the goal by simulating attractive and repulsive forces, while avoiding obstacles, offering ease of adjustment and flexibility.

In our work, we utilized the A* algorithm to generate the path with minimum distance/time from a designated starting point to a goal, simulating the real size of the robot through obstacle dilation to prevent collisions. Concurrently, we employed the APF algorithm to generate a longer but safer paths, ensuring safety by adjusting the parameters of the attractive and repulsive potential fields.

2.1 The A Star (A*) Algorithm

A* algorithm is an informed search algorithm, which uses a heuristics function to guide its search towards the goal more efficiently than an uninformed search algorithm like Dijkstra's algorithm. The heuristic function $h(n)$, for instance, Euclidean distance, is utilized to estimate the cost from any node n to the goal node. The algorithm selects nodes to expand based on the minimum $f(n)$ value, where

$$f(n) = g(n) + h(n), \tag{1}$$

where $g(n)$ is the cost of the path from the start node to node n, and $h(n)$ is the heuristic function described above. We used the Euclidean distance in this work:

$$h(n) = \sqrt{(x_{\text{goal}} - x_n)^2 + (y_{\text{goal}} - y_n)^2} \tag{2}$$

The pseudo-code of this algorithm are shown in the Appendix 4.

To simulate the actual size of the robot and prevents collisions, we use obstacle dilation, i.e. dilate the obstacle size by the robot radius. We measured the radius from the center of the robot to its outermost edge, setting it at 0.3. In practice, we can enhance the safety of the trajectory by enlarging the robot radius inputted to the program, thereby expanding the obstacles further.

2.2 The Artificial Potential Field (APF) Algorithm

The Artificial Potential Fields (APF) algorithm is a widely used method in robotics for real-time path planning and obstacle avoidance, which plans paths by simulating physical fields. Assuming that each obstacle emits a repulsive force, while the goal location generates an attractive force, the total potential U is the summation of the attractive potential U_{att} and the repulsive potential U_{rep} :

$$U_{\text{att}}(i,j) = \frac{1}{2} k_{\text{att}} d_{\text{goal}}^2(i,j) \tag{3}$$

$$U_{\text{rep}}(i,j) = \begin{cases} \frac{1}{2} k_{\text{rep}} \left(\frac{1}{d_{\text{obs}}(i,j)} - \frac{1}{d_0} \right)^2, & \text{if } d_{\text{obs}}(i,j) \leq d_0 \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where $d_{\text{goal}}(i,j)$ is the distance from grid (i,j) to the goal, $d_{\text{obs}}(i,j)$ is the distance from grid (i,j) to the obstacle, and d_0 is the influence range. By tuning the scale parameters k_{rep} and k_{att}, we can enable the algorithm to generate a longer but safer path, such as by increasing the repulsive force generated by obstacles.

Note that the APF algorithm has local minima problems. The robot can get stuck in positions where the attractive and repulsive forces balance each other, but the goal is not reached (local minima).

Due to its high computational efficiency and ability to handle dynamic environmental changes by recalculating forces, the APF algorithm is often used for real-time, online path planning. In our work, we still adopt an offline path planning setting, merely adjusting parameters to make the paths generated by the algorithm safer, i.e., less prone to collisions.

2.3 Post-Processing: Path Smoothing

Path smoothing is applied to the paths generated by both the A* and APF algorithms to enhance the feasibility of the actual execution of the robot. The post-processed paths offer the robot a set of waypoints for its actual movement, ensuring path coherence and practicality.

2.4 Architecture

Our architecture is very similar with that in HW3, except that the landmark position is known. Therefore we do not need to build the map. We utilize the EKF to update the estimation of the pose of the robot with the information obtained by observing the AprilTags. Path planning process is offline, which outputs all the waypoints the robot should follow. The diagram is shown in Figure 2.

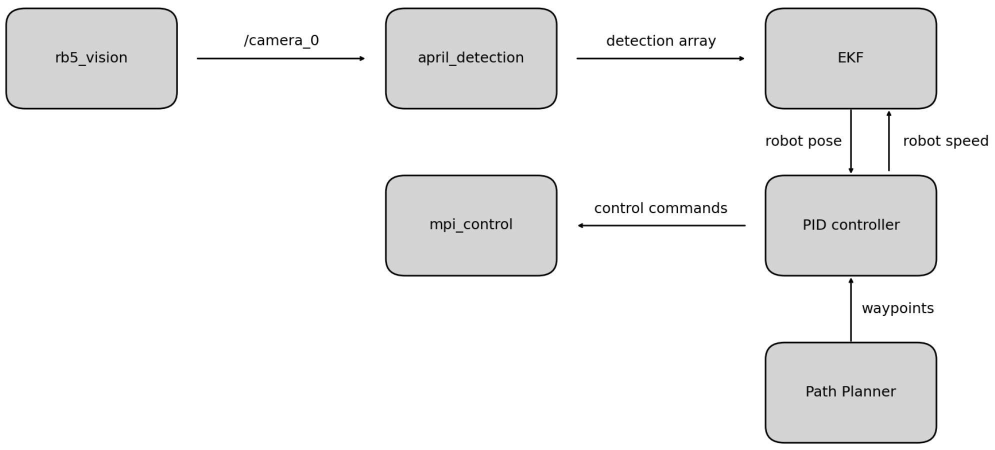


Figure 2: The architecture.

3 Visualization and Results

3.1 Path Visualization

The paths calculated by the algorithms are visualized on the grid map, as shown in Figure 3. The A* algorithm path is shown in the left figure, while the APF algorithm path is shown in the right. Both are represented by blue dashed lines. The post-processed paths are depicted by red solid lines, which are smoother.

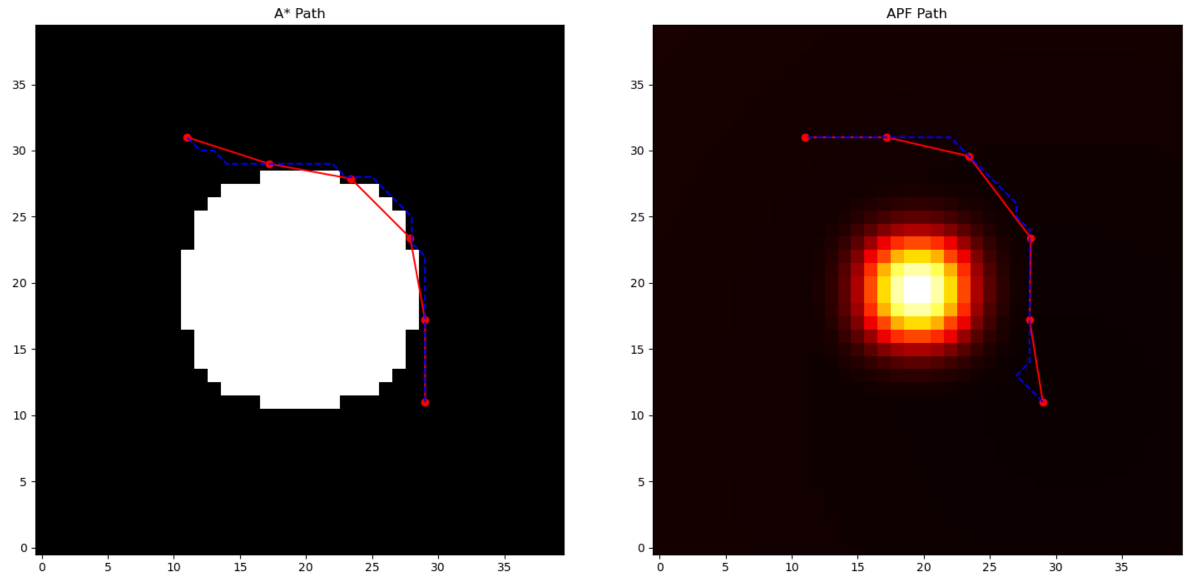
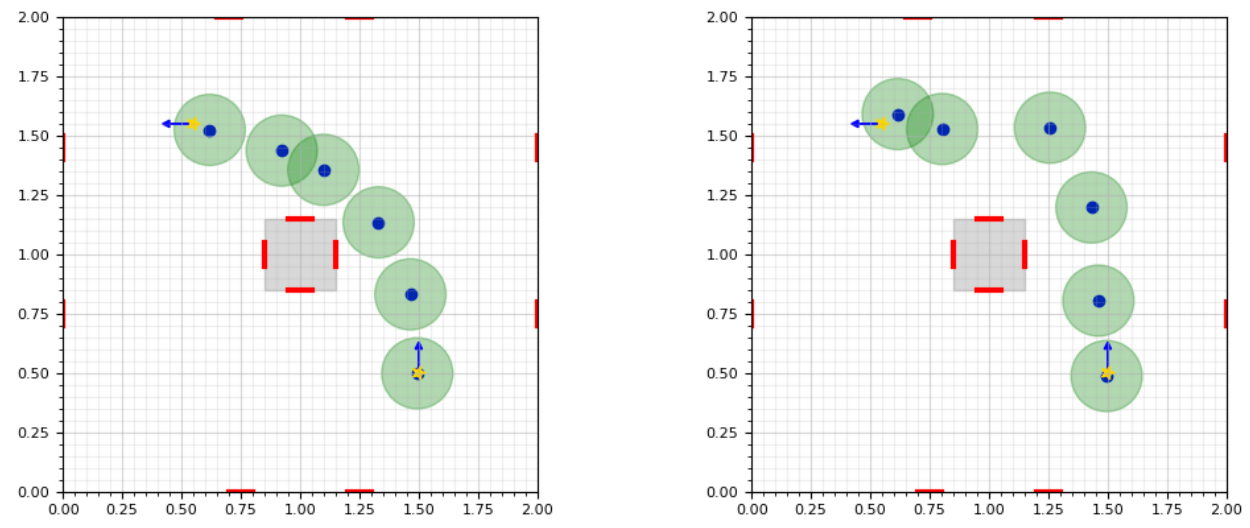


Figure 3: The paths given by the algorithms. The left is the A* algorithm path, while the right is the APF algorithm path. Both are represented by blue dashed lines. The post-processed paths are depicted by red solid lines.

From the Figure 3, it is evident that the A* algorithm generated the shortest path closely tangent to the obstacles, while the path generated by the APF, through parameter adjustments, circumvents the obstacles at a greater distance. The latter has a longer overall distance and time, yet it is safer. We can also calculate the overall distance theoretically, which are 1.582 and 1.740 respectively.

3.2 Trajectory Visualization

We recorded every waypoint the robot passed through during its actual operation, as shown in the Figure 4. It can be observed that the actual performance of the robot largely aligns with the simulated results in Figure 3. The A* algorithm found the shortest trajectory, while the trajectory generated by the APF algorithm kept a greater distance from obstacles, owing to our setting of a larger repulsive force.



(a) A* Trajectory (b) APF Trajectory

Figure 4: The actual trajectory of the robot movement.

The path lengths and actual running times generated by the two algorithms are as follows, as shown in the Table 1 below.

Algorithm	Path Length	Time
A Star	158.2cm	57s
AFP	174.0cm	65s

Table 1: Results of Alignment and Average Errors.

The A* algorithm typically finds shorter paths, whereas the APF algorithm exhibits better adaptability in dynamic environments. The paths, after smoothing, are visually coherent and suitable for the practical movement requirements of the robot.

4 Conclusion

The capability of the A* and APF algorithms in planning effective paths has been showcased in this report. Future work may focus on the optimization of these algorithms and their application in more complex environments.

Appendix

Pseudo-Code of the A* algorithm

```
function A_Star(start, goal):
    // Initialize the open and closed lists
    openList = set(start)
    closedList = set()

    // Costs from start to a node
    g = map with default value of Infinity
    g[start] = 0

    // Estimated costs of the cheapest paths from start to goal through a node
    f = map with default value of Infinity
    f[start] = heuristic(start, goal)

    while openList is not empty:
        current = node in openList with the lowest f[node] value
        if current == goal:
            return reconstruct_path(cameFrom, current)

        openList.remove(current)
        closedList.add(current)

        for neighbor in neighbors(current):
            if neighbor in closedList:
                continue

            tentative_gScore = g[current] + dist_between(current, neighbor)

            if neighbor not in openList:
                openList.add(neighbor)
            elif tentative_gScore >= g[neighbor]:
                continue

            // This path is the best so far, record it
            cameFrom[neighbor] = current
            g[neighbor] = tentative_gScore
            f[neighbor] = g[neighbor] + heuristic(neighbor, goal)

    return failure

function reconstruct_path(cameFrom, current):
    total_path = [current]
    while current in cameFrom.keys():
        current = cameFrom[current]
        total_path.prepend(current)
    return total_path
```