EE4308 Project 1
Part 1

Advances in Intelligent Systems and Robotics
Academic Year 2016-2017

Preben Jensen Hoel - Paul-Edouard Sarlin
A0158996B - A0153124U

1 Overview

The path planning system developed for the first part of the project is inspired by the ROS navigation stack. It is composed of a global and a local planner. The first uses a stored map to find the shortest path from a start point to a goal point. The latter uses this path to control the robot movement by outputting the correct velocity commands.

2 Map topology

From the project sheet, we notice that the map is composed of square cells. We denote as integer coordinates the centre of the cells, e.g. the start cell is $(0, 0)$, while the one above is $(0, 1)$. The map is stored as a one-dimensional array containing the coordinate points of the centre of the walls. As such, horizontal walls coordinates are of form $(n, m + 0.5)$ $(n, m \in \mathbb{N})$, while vertical ones are expressed as $(n + 0.5, m)$. This makes it easy to manually change the map and to process it.

3 Global planning

3.1 A-Star

We start by applying the famous algorithm A-Star to the data. It first labels the cells (x, y) as a single index $i = y \cdot \text{width}_{\text{map}} + x$, and then propagates a move cost from the start point. The algorithm always expands first the path that minimizes the total semi-estimated cost f from start to goal. For each cell i, $f(i) = g(i) + h(i)$, where g is the movement cost from start to i, and h is the estimated cost from i to the goal, called the heuristic function and chosen to be the Manhattan distance.

The Python implementation is largely inspired by [1], but adapted to the above-mentioned custom map topology. Additionally, a new parameter allows to encourage straight paths over successive sharp turns, or conversely. This is done by dynamically changing the move cost from one cell to the other by looking at the current virtual orientation of the robot.

3.2 Global Smoothing

The path outputted by A-Star contains sharp turns that force the robot to stop and turn at each corner (red in Figure 1). In order to maximize the speed of the robot along the path, we apply a least-squares smoothing regularization that takes into account the whole path, hence called global. Let (x_i, y_i) the n points of the A-Star path, and $(x_i^{'}, y_i^{'})$ the ones of the smoothed path, computed such that they minimize the cost function:

$$J = \frac{1}{2}\sum_{i=1}^{n} \alpha \underbrace{\left(\left(x_i - x_i^{'}\right)^2 + \left(y_i - y_i^{'}\right)^2\right)}_{\text{original path}} + (1-\alpha)\underbrace{\left(\left(x_i^{'} - x_{i+1}^{'}\right)^2 + \left(y_i^{'} - y_{i+1}^{'}\right)^2\right)}_{\text{shortened smooth path}} \quad (1)$$

The parameter $\alpha \in [0, 1]$ expresses the trade-off between closeness to the original path and smoothness. We use gradient descent to minimize J, with the constraint that the start and goal points remain unchanged, i.e. $(x_1, y_1) = (x_1^{'}, y_1^{'})$ and $(x_n, y_n) = (x_n^{'}, y_n^{'})$.

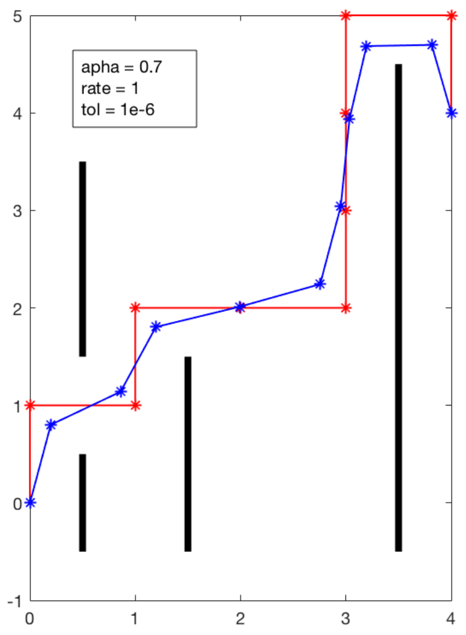

Figure 1: Outputs paths of A-Star (red) and simple global smoothing (blue) for a test scenario.

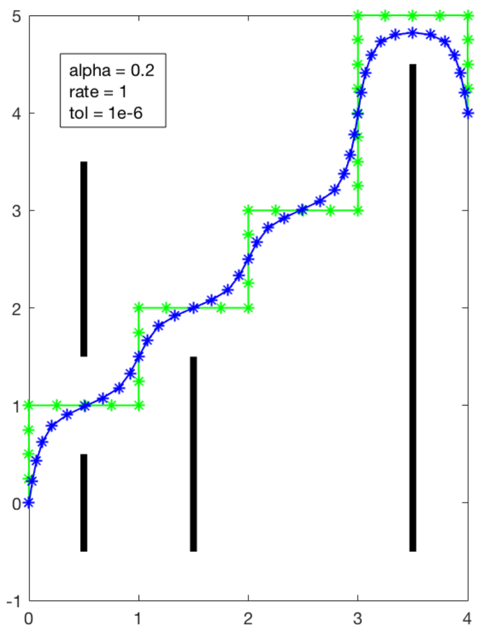

Figure 2: A-Star with maximization of turns (green) and dense global smoothing (blue), same scenario.

The blue path of Figure 1 corresponds to the result. It is indeed smoother than the red one, but still contains sharp turns. We decide to tune the A-Star path by maximizing the number of turns, and to make it denser by adding three new points on each segment, resulting in the green path of Figure 2. Applying the previous global smoothing results in the blue path, which seems much nicer than the previous one.

4 Local Planning

4.1 Low level control

For each new Odometry message, the linear and angular velocity commands are computed and sent to the robot. We use a PI controller on the distance and the orientation to the next point of the path. If the orientation error is too high, the robot might significantly diverge from the path. In that particular case, the linear velocity is set to zero, so that the robot can first turn towards the next point, and then move to it. Transition from one point to the other happens when the distance error is lower than a predefined tolerance.

4.2 Local smoothing

Although the path outputted by the global planner module is already smooth, the above-described control process produces a jerky movement. We thus apply a second smoothing layer, using only the next few points of the path, hence called local. Instead of using the orientation and distance error of the next point only, we decide to take into account k points by computing the weighted average of their errors. Let θ_{PI} and d_{PI} be the orientation and distance errors to be fed into the PI controller, θ_i and d_i the errors from the current position to the point i, $W = (w_i)$ a vector of k weights, and j the index of the current point:

$$\theta_{PI} = \frac{\sum_{i=1}^{k} \theta_{j+i}\, d_{j+i}\, w_i}{\sum_{i=1}^{k} d_{j+i}\, w_i} \quad \text{and} \quad d_{PI} = \frac{\sum_{i=1}^{k} d_{j+i}\, w_i}{\sum_{i=1}^{k} w_i} \tag{2}$$

As the robot gets closer to the next point, the corresponding distance error decreases, giving less importance to this point in the orientation correction: the robot smoothly turns towards the next one. After experimental trial, we find that $k = 4$ and $W = \begin{bmatrix} 5 & 3 & 2 & 1 \end{bmatrix}$ give good results. The first two points have thus more influence on the controller than the others, although the following two help to smooth the transition from one current point to the other.

The overall behaviour from start to goal is thus really smooth and time-efficient.

[1] `http://www.redblobgames.com/pathfinding/a-star/implementation.html`. [Python implementation of the A-Star algorithm].