Efficiency Comparison of Path Planning Algorithms

Diego Colón

December 6, 2019

1 Introduction

The motivation behind this project is to compare how distance optimal different path planning algorithms are in an unstructured environment with obstacles. These environments are randomly generated obstacles in a 100m square grid in which a kinematically feasable trajectory must be generated. The selected kinematics for this project are:

$$\dot{x} = vcos(\theta) \tag{1}$$

$$\dot{y} = vsin(\theta) \tag{2}$$

$$\dot{\theta} = \omega \tag{3}$$

2 Requirements

In order for this program to work the system it is ran on must have:

- Python 3.7 or Anaconda
- Numpy
- Scipy
- Matplotlib
- Sympy
- PyQt5

If any of the modules are not installed, they can be installed through the command 'pip3 install [package name]'

3 Algorithm Explanation

3.1 Potential Field

The Potential Field algorithm consists of creating a field that quantifies errors in position and penalizes proximity to obstacles. The algorithm then procedes to minimize this value by traveling in the direction of the gradient of the field. The field is characterized by:

$$F(x,y) = C_{obstacle}\ (x,y)\ =\ (x-x_g)^2 + (y-y_g)^2\ +\ \sum_{n=1}^{N} e^{\frac{a}{((x-x_n)^2+(y-y_n)^2)}} \tag{4}$$

Where $F(x,y)$ is the field, (x_g, y_g) are the goal coordinates, and (x_n, y_n) are the obstacle coordinates, and a is a scalar that allows for control of how close the vehicle gets to a given obstacle.

This method has several downsides, the biggest one being that the path could get stuck at a local minimum of the field and never find a path. The search could also produce paths that are unfeasable given the kinematic contraints, but this can be mitigated by finding the input to the system that provides the closest output to what was intended.

3.2 Dynamic Window Approach

The Dynamic Window Approach was first introduced by D. Fox, W. Burgard and S. Thrun in an IEEE article. The basis of this algorithm is to create a window of possible velocities achievable from the current state and minimize a function over a the control inputs, assuming they are constant for a small dt. The original algorithm minimized a function that acounted for the heading angle of the vehicle, the distance to obstacles and the current velocity. This was adapted for this project into the follow equation.

$$J(x, y, \theta) = C_{heading} + C_{distance} + C_{obstacle} + C_{velocity} \tag{5}$$

Where:

$$C_{heading}(x, y, \theta) = (\theta - \arctan 2((x_g - x), (y_g - y))) \tag{6}$$

$$C_{distance}(x, y) = (x_g - x)^2 + (y_g - y)^2 \tag{7}$$

$$C_{obstacle}(x, y) = (x - x_g)^2 + (y - y_g)^2 + \sum_{n=1}^{N} e^{\frac{a}{((x-x_n)^2+(y-y_n)^2)}} \tag{8}$$

$$C_{velocity}(v) = (v - v_{limit})^2 \tag{9}$$

By minimizing this cost function, the algorithm provides a path that is feasable to the vehicle and continuos in velocity.

4 Usage

The following instructions will guide you through performing the path planning comparisons for a given obstacle field assuming all required packages are installed.

1. Run Main.py
2. Select Obstacle field from Maze section
3. Click 'Load Map' Button
4. Input inital and final conditions
5. Click start button

At this point the report will print out in a text field and a plot will appear showing the plotted trajectories. In the plot, the trajectories might appear to intersect with the obstcles but this is an artifact of the plotting software.