EE 615 - Control & Computing Lab

Experiment - 1
Path Planning with Obstacle Avoidance

Submitted by

Mayank Soni & Ankit Singh
213070028 & 213070029

Contents

1 PROBLEM STATEMENT

The aim of the experiment is to make the robot move from the point (x,y) = (2,4) to the point (x,y) = (10,4). The robot has to pass through the point (x,y) = (2,10) on its way to the destination. Further, the robot has to avoid obstacles on its way as it moves through the target goals.

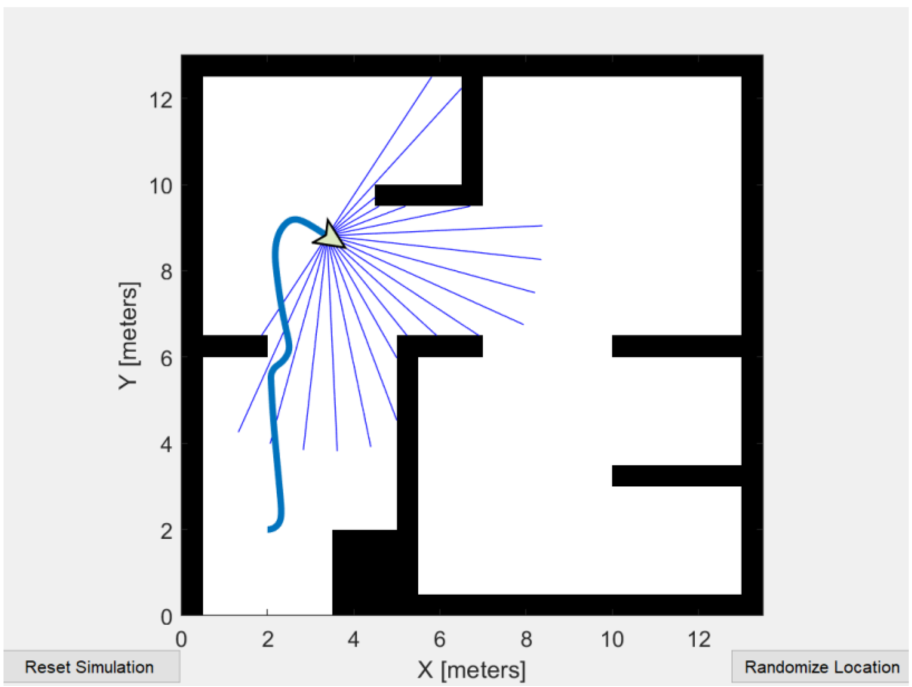

Figure 1: Proposed Outcome

2 PURE PURSUIT ALGORITHM

Pure pursuit is a tracking algorithm that works by calculating the curvature that will move a vehicle from its current position to some goal position. In this algorithm we choose some way-points that is ahead of our path. Our vehicle could be thought of as chasing these points. The name is basically derived from how humans drive the vehicle. We look ahead the road and decide accordingly to change the direction of the vehicle. This point (look-ahead) changes every time we move forward.

2.1 GEOMETRIC APPROACH OF PURE PURSUIT ALGORITHM

The pure pursuit approach is a method of geometrically determining the curvature that will drive the vehicle to a chosen way point.This goal point is a point on the path that is one look ahead distance from the current vehicle position. An arc that joins the current point and the goal point is constructed. The chord length of this arc is the look ahead distance, and acts as the third constraint in determining a unique arc that joins the two points.

Let point (x, y) be a way-point , which is a look ahead distance from the origin. The curvature of the arc that joins the origin to point (x, y) and whose chord length is l is calculated as

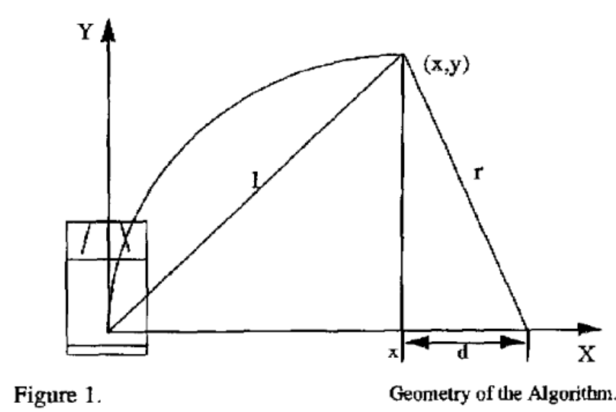

Figure 2: Geometric Derivation of Pure Pursuit

$$x^2 + y^2 = l^2$$

$$x + d = r$$
$$d = r - x$$
$$(r - x)^2 + y^2 = r^2$$
$$2rx = l^2$$
$$r = l^2/2x$$

2.2 INPUTS TO ALGORITHM

1. Linear velocity: This is specified as a scalar in meters per second.It is assumed that the vehicle drives at a constant linear velocity and that the computed angular velocity is independent of the linear velocity.

2. Angular velocity: Maximum angular velocity, specified a scalar in radians per second. The controller saturates the absolute angular velocity output at the given value.

3. Look-ahead distance: The look-ahead distance is how far along the path the robot should look from the current location to compute the angular velocity commands.

4. Vector of ordered waypoints(path):Waypoints, specified as an n-by-2 array of [x y] pairs, where n is the number of waypoints.

2.3 OUTPUTS OF ALGORITHM

1. Linear velocity: scalar in meters per second

2. Angular velocity: specified as a scalar in radians per second

2.4 ALGORITHM

1. Determine current location

2. Find path points closest to vehicle

3. Find the goal point (x, y) on the path at the distance $'l'$ from the current location

4. Transform the goal point to robot frame: The geometric derivation for the curvature was done in vehicle coordinates and curvature commands to the vehicle make sense in vehicle coordinates

5. Find the curvature of such a curve C such that the $'l'$ is the chord length of C

6. Set steering to the differential drive robot along a given curvature

2.5 USING SIMULINK PURE PURSUIT BLOCK

In order to understand the basic working of pure pursuit algorithm, we first used the pure pursuit block available in simulink for path planning as shown in below figure

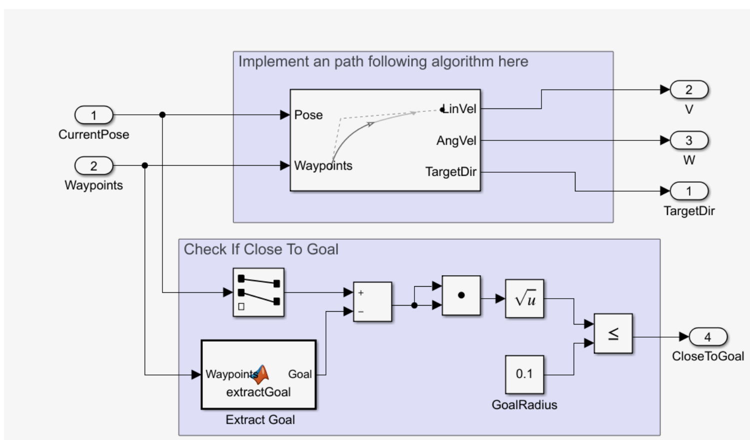

Figure 3: Simulink Block of Pure Pursuit

2.6 IMPLEMENTATION OF PURE PURSUIT ALGORITHM

In this section, we implement the pure pursuit algorithm from scratch using the user defined function block in simulink

```
function [linvel,angvel,theta,i] = fcn(cpos,waypt,i)
linvel = 0.5;
```

```
angvel = 0;
theta = 0;
theta2 = 0;
c1 = 0;
c2 = 0;
xt = cpos(1,1);
yt = cpos(2,1);
theta1 = cpos(3,1);
if i < 4
    c1 = waypt(i,2) - yt;
    c2 = waypt(i,1) - xt;
    l = sqrt(c2^2 + c1^2);
    r = l^2/(2*xt);
    angvel = linvel/r;
    theta2 = atan2(c1,c2);
    theta = theta2 - theta1;
    if l < 0.5
        i = i+1;
    end
end
```

3 OBSTACLE AVOIDANCE USING VECTOR FIELD HISTOGRAM

This method basically detects the unknown obstacles and avoids collisions while simultaneously steering the mobile robot toward the target. It uses a two-dimensional Cartesian histogram grid as a world model. The VFH method employs a two-stage data reduction technique and three levels of data representation exist:

1. The highest level holds the description of the robot's environment.The two-dimensional Cartesian histogram grid C is continuously updated in real-time with range data sampled by the on-board range sensors.
2. At the intermediate level, a $1 - D$ polar histogram H is constructed around the robot's momentary location. H comprises n angular sectors of width a. This maps the

active region $C*$ onto H, resulting in each sector k holding a value h_k that represents the polar obstacle density in the direction that corresponds to sector k.

3. The lowest level of data representation is the output of the VFH algorithm: the reference values for the drive and steer controllers of the vehicle.

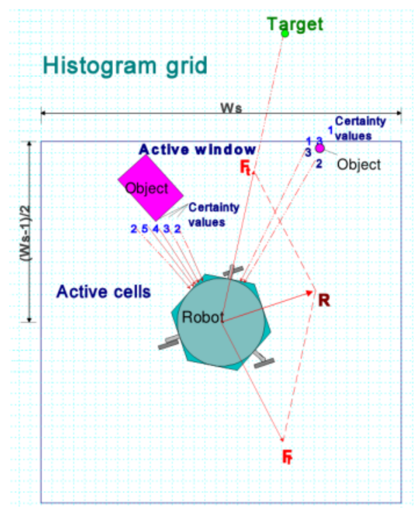

Figure 4: Vector Field Histogram

3.1 INPUTS TO ALGORITHM

1. Ranges: Range values from scan data, specified as a vector in meters. These range values are distances from a sensor at given angles. The vector must be the same length as the corresponding angles vector.

2. Angles: Angle values from scan data, specified as a vector in radians. These angle values are the specific angles of the given ranges. The vector must be the same length as the corresponding ranges vector.

3. Target Direction: Target direction for the vehicle, specified as a scalar in radians. The forward direction of the vehicle is considered zero radians, with positive angles measured counterclockwise.

3.2 OUTPUT OF ALGORITHM

1. Steer Direction: Steering direction for the vehicle, specified as a scalar in radians.The forward direction of the vehicle is considered zero radians, with positive angles measured counterclockwise.

3.3 ALGORITHM

1. Direction of cell is calculated

$$\beta_{ij} = \tan^{-1}\frac{y_j - y_0}{x_i - x_0}$$

   where (x_0, y_0) is the vehicle centre and (x_j, y_j) is the active cell coordinate.

2. Cost function: $m_{ij} = (cij)^2 \left(a - b\sqrt{(x_i - x_0)^2 + (y_i - y_0)^2}\right)$ where a and b parametrs need to be tuned. m_{ij} is proportional to $-d$. Therefore, occupied cells produce large vector magnitudes when they are in the immediate vicinity of the robot, and smaller ones when they are further away. Specifically, a and b are chosen such that $a - bd_{max} = 0$, where $d_{max} = \sqrt{2}(w-1)/2$ is the distance between the farthest active cell and the Vehicle Center Point(VCP). This way $m = 0$ for the farthest i, j active cell and increases linearly for closer cells. C^* is squared. This expresses the confidence that recurring range readings represent actual i, j obstacles, as opposed to single occurrences of range readings, which may be caused by noise.

3. Create angular histogram H with resolution such that $n = \frac{180}{\alpha}$.

4. A threshold (user specified) is used to detect the candidate valley: Any valley comprised of sectors with smoothed polar obstacle density (POD)s below a certain threshold is called a candidate valley.

5. Choose the candidate valley aligned most with target.

6. Steers through the middle of the chosen valley: First, the algorithm measures the size of the selected valley (i.e., the number of consecutive sectors with PODs below the threshold). Here, two types of valleys are distinguished, namely,wide and

narrow ones. A valley is considered wide if more than s_{max} consecutive sectors fall below the threshold. Wide valleys result from wide gaps between obstacles or from situations where only one obstacle is near the vehicle. The sector that is nearest to k_{targ} and below the threshold is denoted k_n and represents the near border of the valley. The far border is denoted as k_f and is defined as $k_f = k_n + s_{max}$. The desired steering direction q is then defined as $q = \frac{k_n + k_f}{2}$. where,

a, b Positive constants

$C^*_{i,j}$ Certainty value of active cell (i, j)

$d_{i,j}$ Distance between active cell (i, j) and the VCP

$m_{i,j}$ Magnitude of the obstacle vector at cell (i, j)

x_0, y_0 Present coordinates of the VCP

x_i, y_j Coordinates of active cell (i, j)

$\beta_{i,j}$ Direction from active cell (i, j) to the VCP

3.4 USING SIMULINK VECTOR FIELD HISTOGRAM BLOCK

In order to understand the basic working of vector field histogram, we first used the VHF block available in simulink for obstacle avoidance as shown in below figure

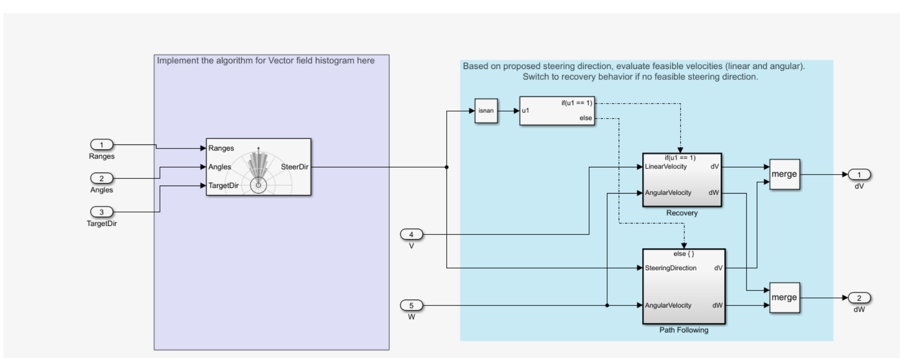

Figure 5: Simulink block of Vector Field Histogram

3.5 IMPLEMENTATION OF VECTOR FIELD HISTOGRAM

In this section, we implement the vector field histogram from scratch using the user defined function block in simulink

```
function SD = fcn(Ranges,Angles,TD)
    a = 100 * ones(21,1);
    b = 20 ;
    distance = Ranges(1:21);
    Angles = Angles(1:21);
    m = ((0.89)^2) * ( a - b*distance) ;
    m_th = find(m>55) ;
    ang = find(abs(Angles(m_th)));
    if m > 55
        SD = ang(1) ;
    else
        SD = TD ;
    end
end
```

4 RESULTS & OBSERVATIONS

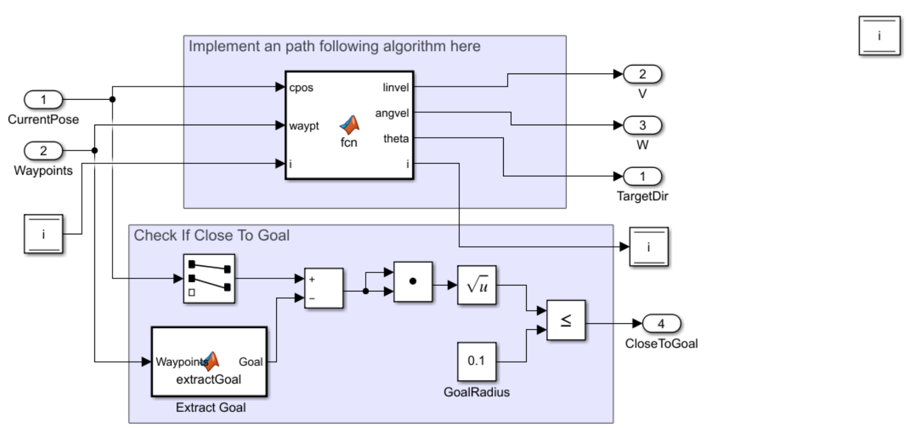

Figure 6: Implementation of Path Following Algorithm

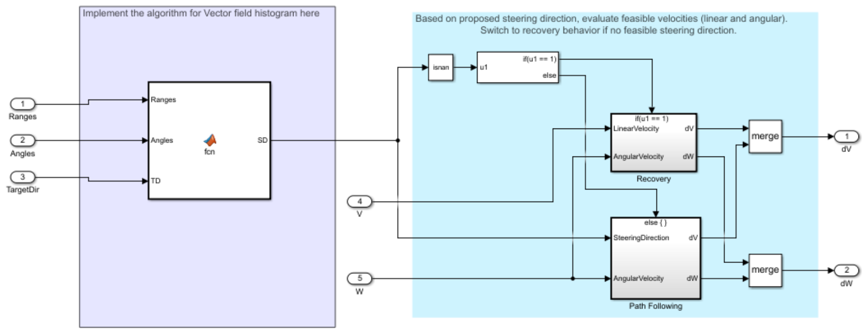

Figure 7: Implementation of Vector Field Histogram

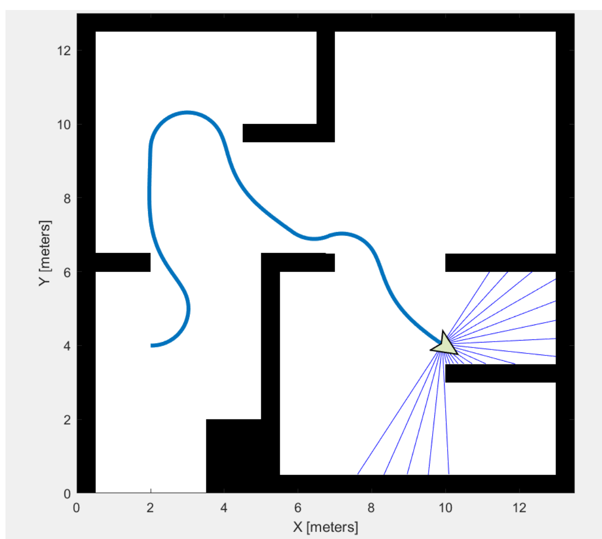

Figure 8: Output