EE615 Control Computing Lab

Experiment 2

Samay Pritam Singh | 20307R002
Harshit Garg | 213070025

Contents

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.

1 INTRODUCTION

The objective of this experiment is to understand the kinematics of differential drive robot and to implement the path following control algorithms viz. pure pursuit algorithm and vector field histogram.

2 Pure Pursuit Algorithm

Pure pursuit is a path tracking algorithm. It computes the angular velocity command that moves the robot from its current position to reach some look-ahead point in front of the robot.

The algorithm moves the look-ahead point on the path based on the current position of the robot until the last point of the path.

Communication and path management:

In real tracker implementation The tracker is driving the vehicle along the old path.When the planner completes its part of the cycle it must send the new path to the tracker. This new path probably partially overlaps the old path, so the planner must also tell the tracker what pat of the old path can be overwritten with new information.

2.1 Inputs to Algorithm:

1. Linear velocity: This isspecified 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. maximum 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.

2.2 Algorithm output:

1. linear velocity:scalar in meters per second

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

2.3 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.

$$r = \frac{l^2}{2x}$$

$$\gamma = 1/r = \frac{2x}{l^2}$$

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

Properties of Algorithm:

1. Effects of Changing the Look ahead distance :the longer the look ahead distance, the less "curvy" of a path that can be followed.

2. Non Unique Look ahead for a Given Path Curvature: Given a look ahead distance we can define a curvature, but given a curvature the look ahead distance is indeterminate.

3. The method does not model the capability of the vehicle or of its actuators, and so assumes perfect response to requested curvatures. This causes two problem.
1) A sharp change in curvature can be requested at a high speed, causing the vehicle's rear end to skid.
2) The vehicle will not close on the path as quickly as desired because of the first order lag in steering.

2.4 Implementation:

2.4.1 MATLAB CODE

```
function [V, W, TargDir, i] = purepursuit(current_pos, WP, i)
%WP = [2.00      4.00
 %       2.00    10.00;
 %      10.00    4.00;];

robotCurrentLocation = current_pos(1,:);
W=0
TargDir=0
V = 0.7;
x = current_pos(1);
y = current_pos(2);
theta = current_pos(3);
l=0

theta_new=atan2((WP(i,2)-y),(WP(i,1)-x))
TargDir =  theta_new - theta
l = ((WP(i,2)-y)^2+(WP(i,1)-x)^2)
r = l/(2*x)
W = V/r

if l<0.5
    i=i+1
end

end
```

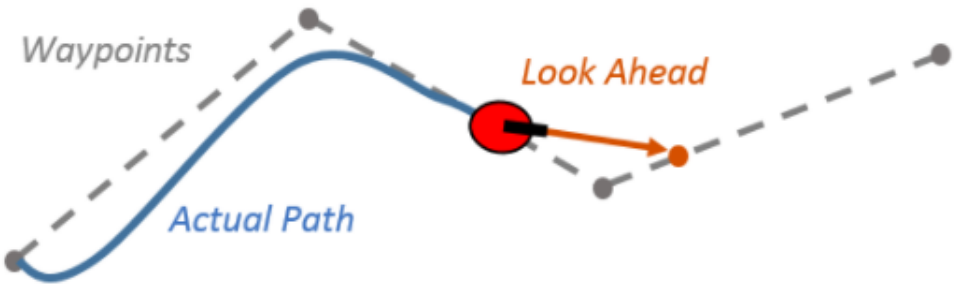

Figure 1: Pure pursuit algorithm

[1] Coulter, R. Implementation of the Pure Pursuit Path Tracking Algorithm. Carnegie Mellon University, Pittsburgh, Pennsylvania, Jan 1990

3 Vector field histogram

The vector field histogram (VFH) algorithm computes obstacle-free steering directions for a robot based on range sensor readings. Range sensor readings are used to compute polar density histograms to identify obstacle location and proximity. Based on the specified parameters and thresholds, these histograms are converted to binary histograms to indicate valid steering directions for the robot. The VFH algorithm factors in robot size and turning radius to output a steering direction for the robot to avoid obstacles and follow a target direction.

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 Diirection: 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 Algorithm output:

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 parameters:

1. No. of Angular Sectors:This property defines the number of bins used to create the histograms.

2. Range distance limit(m):Limits for range readings, specified as a 2-element vector with elements measured in meters. The range readings specified when calling the object are considered only if they fall within the distance limits.

3. Histogram thresholds:User defined threshold to determine candidate valley in histogram.

3.3.1 Vehicle parameters:

1. Vehicle Radius: vehicle Radius specifies the radius of the smallest circle that can encircle all parts of the robot. This radius ensures that the robot avoids obstacles based on its size.

2. Safety distance(m): this property add a factor of safety when navigating an environment.

3. Min turning radius: specifies the minimum turning radius for the robot traveling at the desired velocity.

3.4 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: $m_{ij} = (c_{ij})^2(a - b\sqrt{(x_i - x_0)^2 + (y_i - y_0)^2})$
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 alligned 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 $q = (k_n + k_f)/2$.

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.

Properties of Algorithm:

1. The Threshold: maladjustment of the threshold have the following effect on the system performance:
1)If the threshold is much too large the robot approaches on a collision course.However, during the approach sensor readings further increase the CV s representing that obstacle, in this case robot may approach the obstacle too closely (especially when traveling at high speed) and collide with the object.
2)on the other hand, the threshold is much too low, some potential candidate valleys will be precluded and the robot will not pass through narrow passages.

2. Speed Control: The robot tries to maintain maximum speed(given as parameter) during the run unless forced by the VFH algorithm,requires in many cases like when obstacle is far or close or to reduce speeed to change direction.

3.Locality of algorithm:The VFH method is a local path planner, i.e., it does not attempt to find an optimal path (an optimal path can only be found if complete environmental information is given).

3.5 Implementation:

3.5.1 MATLAB CODE

```
1 function [y,w_ij]= VFH(Ranges, Angels, TargDir,RobotPos)
2 a = 100 * ones(21,1);
3 b = 20.0;
4 targetSector = 0;
```

```
range = Ranges(1:21);

angle = Angels(1:21);
w_ij = 0.81 * (a - b * (range));
for i=1:21
    if isnan(w_ij(i))
        w_ij(i)=0
    end
end
y=0
thetaRobot = RobotPos(3);
A = [8 9 10 11 12];
theta_threshold = find(w_ij>70);
theta_valley = find(w_ij<30);
% targetSector = uint32( 21 * (TargDir + thetaRobot - pi/2) / pi);
% targetSector = uint32((TargDir)*57.3248/(9));
ang_index = find(abs(Angels(theta_threshold)));

x = any(A(:))==theta_threshold;
% if x==1
if w_ij(ang_index)>70
    y =ang_index(1);
else
    y = TargDir;
end
```

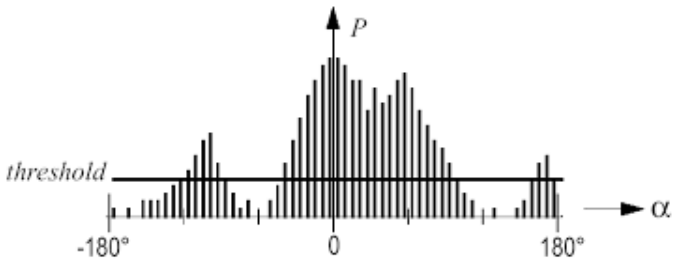

Figure 2: Vector field histogram

[2] Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," in IEEE Transactions on Robotics and Automation, vol. 7, no. 3, pp. 278-288, June 1991, doi: 10.1109/70.88137.

4 Result

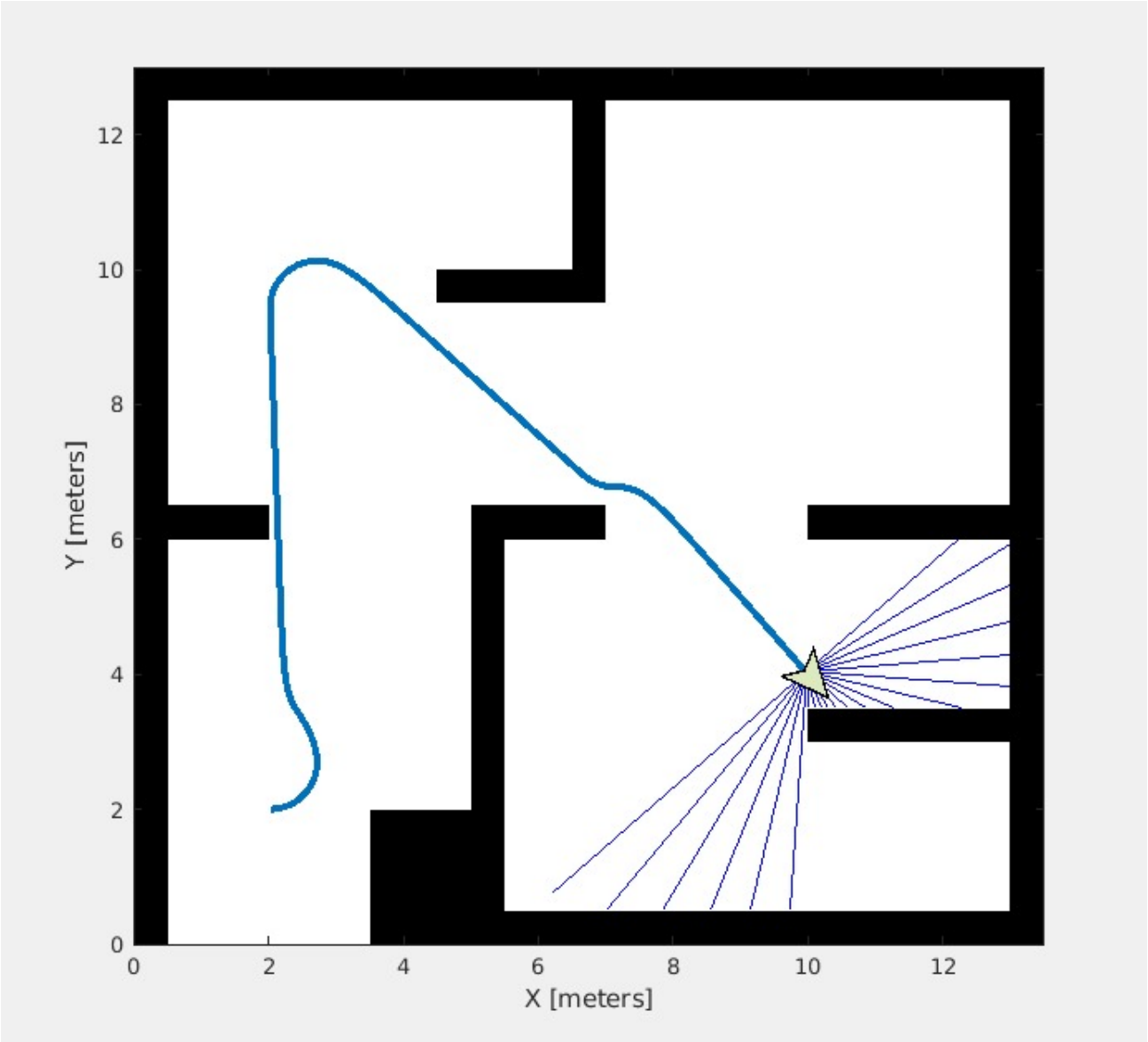

Figure 3: Path followed by robot

The path following control algorithms viz. pure pursuit algorithm and vector field histogram are successfully implemented. Objective to avoid the obstacles and move the robot from given starting point to given destination point is successfully completed.The robustness of designed control strategy is verified by exposing robot to different obstacle conditions.