# ALGORITHM :

#define max_movable_distance 500 // distance to move if there is
. //no obstacle in a particular direction

#define min_allowed_obstacle_dist 100

Set<Obstacle> Obstacle_Array;
// Obstacle structure has 3 data members : the x,y coordinates of obstacle and angle at which vehicle is approaching it wr.t the line drawn from obstacle to goal. It is a tuple ( x , y , $\theta$ )

init() {
    Set destination and other route specific variables;
    Set initialization values for IPS ( Indoor positioning system);
    Obstacle_Array.clear();
    Check LiDAR health;
}

scandata() {
    Give scan command to LiDAR

Process the scan data and obtain an array of 360 values where each has the distance of obstacle at the angle range of 1 degrees.

If no obstacle array has -1

Return Array

}

```
stop()
{
   stop_scanning();
}
```

```
main() {
        init();
        Curr_destination = start;
        While ( curr_desination != Goal) {
                Data D = scandata();
                Angle G,Distance S;
                {G,S} = check_direction_to_move(D);
                curr_destination = move_accordingly(G,S);
        }
}
```

```
check_direction_to_move(Data D)
{
	Angle Init=get_direction_towards_goal();// Uses IPS
	Angle curr = get_curr_direction(); // Uses IPS
	Distance S;
	{ Init,S } = Find_best_direction( Init,curr ,D);
	If (S < max_movable_distance ) {
		{ x,y } = getxy ( curr_position , Init , S );
		Angle A= get_angle ( {x,y} , goal ,Init );
		Obstacle curr_obstacle( x,y,A);
		Obstacle_Array.insert ( curr_obstacle );
	}
	Return { Init,S };
}
```

```
Find_best_direction(Angle towards_goal,Angle curr,Data D)
{
	Int optimal = ((toward_goal-curr)+360)%(360);
	Int left = optimal,right = optimal;
	Bool L=false,R=False;
	Distance D1=-1,D2=-1;
	while( !(left == 0 && right == 359))
	{
```

```
        {L,D1} = check_obstacle(left,D);
        {R,D2} = check_obstacle(right,D);
        if(L==True or R==True) break;
        if(left>0) left--;
        if(right<358)right++;
    }
     if(L==True or R==True)
     {
            if( L==true && R==true ) {
                  if(D1>D2) return {left,D1};
                  return {right,D2};
            }
            else if( L==true) {
                  return {left,D1};
            }
            else return {right,D2};
     }
    // L or R will always be true
    stop();
}
```

```
check_obstacle(Angle angle, Data D)
{
      // from scandata() check whether we have obstacle in this
direction or not
      {x,y} = get_cordinates(curr_destination);
      If ( D[ angle ] == -1 ) {
            Return { True , max_movable_distance};
      }
      Else If( D[angle] > min_allowed_obstacle_dist ) {
            int  r = D[angle];
            x1 = x + rcos(angle);
            y1 = y + rsin(angle);
            θ  =  get_angle({x,y},goal,angle);
            if(Obstacle_Array.find({x1,y1, θ }) == Obstacle_Array.end())
                return { true , min(D[angle],max_movable_distance) }
      }
      return  {false,-1}
}


Move_accordingly( Angle G, Distance S) {
      rotate_by_angle( G );
      move_distance(S);
      curr= get_current_position() // IPS
      return curr;
}
```

rotate_by_angle(angle $\theta$)

{

// we will maintain some speed profile (gradual increase in w)

// and then decrease w and rotate to particular angle

// let's say the distance between left wheels & right wheels is L

,if left wheels are moving forward with speed v & right wheels are moving backward with speed v then w is (2*v)/L.

// so we will gradually increase w and move at max w and then decrease such that

$$\int_0^t w\, dt = \theta$$

}

move_distance(angle $\theta$, distance S) {

//similar to rotate, here too we shall have a speed profile according to which we can move.

//speed profile will be : increase speed upto maxspeed and then move at maxspeed then gradually decrease.

// $\int_0^t v\, dt = S$ can be used to calculate for what time the algorithm shall run

}