**ME 4773/5493 Fundamental of Robotics**

**Fall 2016**

**San Antonio, TX, USA**

Paralell Parking a Differential Drive Car

|  |  |
| --- | --- |
| Mario Chapa  Dept. of Mechanical Engineering  San Antonio, TX, USA 78249  rcv647@utsa.edu |  |

|  |  |
| --- | --- |
|  |  |

Abstract

A method of simulating the parallel parking maneuver for a differential drive car is proposed. The method provides good accuracy but relies on information that may not be known to a practical autonomous car. A method is discussed that will allow this simulation to be used such situations.

InTRODUCTION

Parallel parking is a maneuver that many new drivers are expected to learn before getting their license. It makes sense then that an autonomous car should also be required to perform this maneuver. In this paper, I go over setting up a simple simulation for using potential fields and controls to guide a differential drive car into a parking space without hitting the other cars.

METHODS

I began by considering the maneuvers involved in parallel parking [1]. Parallel parking can be split into three primary maneuvers. The starting point is parallel to the parked cars but on the adjacent lane. The first maneuver is driving straight until you are next to the car that will eventually be in front of you. The second is to drive in reverse into the parking slot. The third maneuver is to align yourself to be parallel with the parked cars. Of course, this is to be done without hitting any of the other cars. To start with the model, I planned out a two dimensional Cartesian coordinate system and placed the two parked cars and the initial position of the driving car. I centered a repulsive potential field on each of the two parked cars and I used squares to represent the cars to be compatible with the symmetry of the potential. The potential function [2] I used is

Where n is a user defined parameter, p is the distance from the center of the potential to the point of interest on the car, and p0 is the radius of the potential function. To get the car to drive to a point, I used a proportional controller [3]

Where kp is a user defined constant, Xref is the x, y, or theta coordinate that you want the car to drive to and X is the current x, y, or theta coordinate for the car. Figure 1 shows the layout of the simulation.

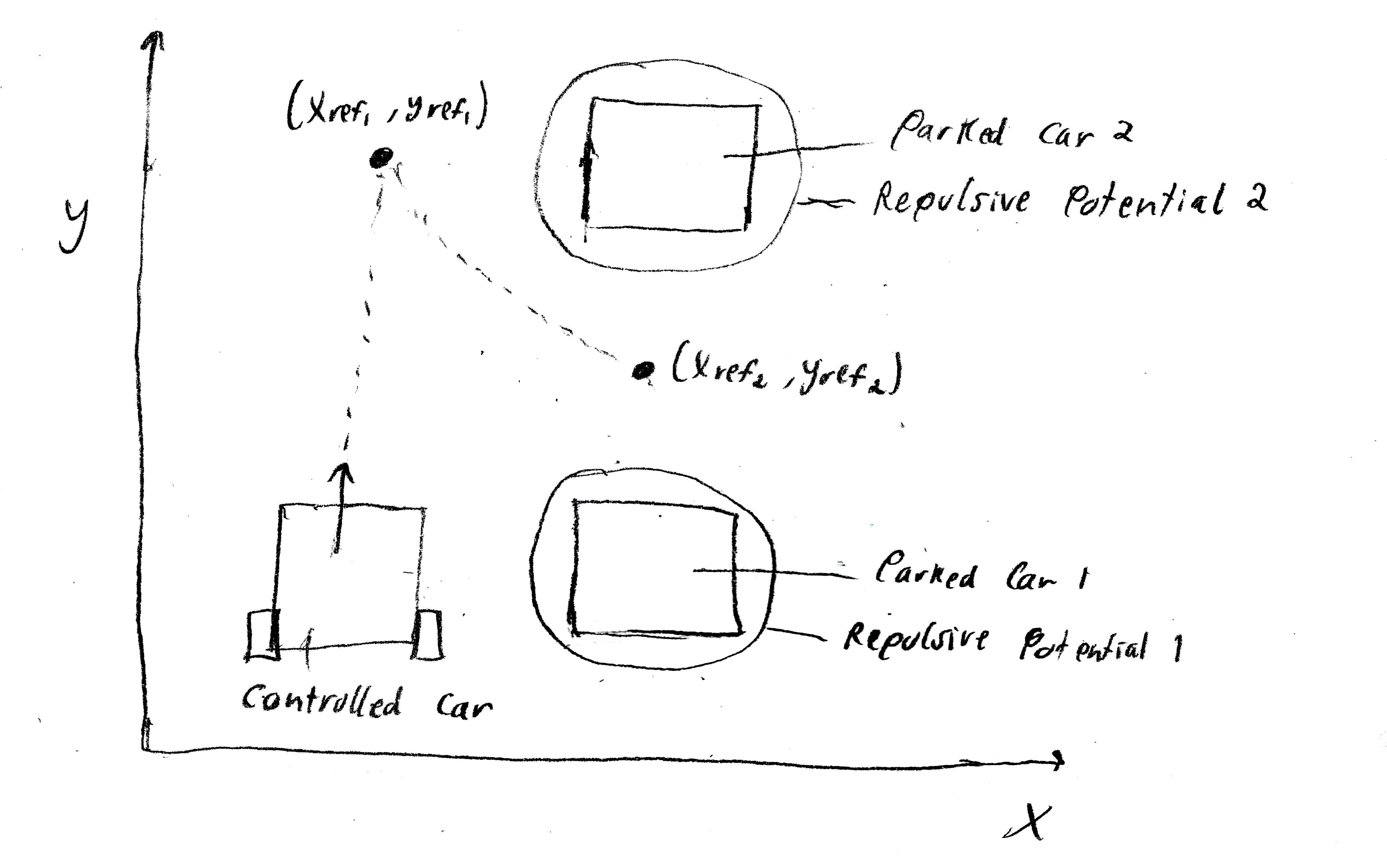


Figure (1)

For the first parking maneuver, I used the proportional controller and set the reference point (xref, yref) to be parallel with Parked Car 2 in Figure 1. For the second maneuver, I placed the reference point within the parking slot and subtracted the repulsive potentials of the parked cars from the proportional controller to ensure the car does not hit them. For the third maneuver I used a proportional controller for the angle to rotate it to be parallel with the parked cars. To update the position and angle of the car, I plugged the potentials and the proportional controller into an equation to find the desired linear and angular velocity of the car. Then I used the car velocities to find the desired wheel angular velocities [3]. Finally, I plugged the wheel velocities into the equations of motion for the car and integrated them using Euler’s method [4]. I then plotted the trajectory of the car. I also plotted an animation by keeping track of the points on the corners of the square representing the car. To translate the points, I found the displacement of the point of interest from its original position and added that to the original position of the other points. To rotate the points, I found the displaced angle and then I translated the points so that the center of the square was at the origin. I then applied the rotation matrix and translated back by the same amount.

RESULTS

The cars I used were 3 meters in width and 3 meters in length. The point of interest of the driving car is in the middle of the right edge (with respect the driver). For the controllers and potential fields, I used kp = 1, n = 50, and p0 = 2.5.

The first plot (Figure 2) is of the x position of a point on the car with respect to time. The parking space is between x = 4.5 and x = 7.5.

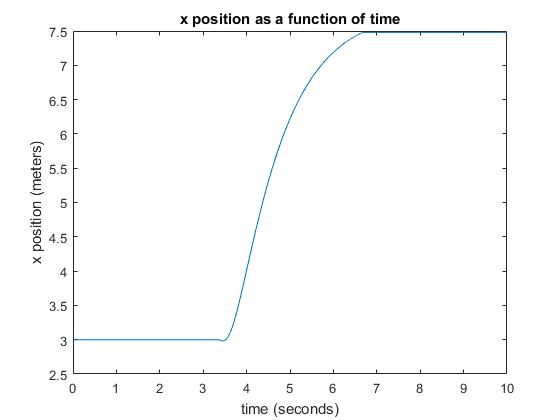


Figure (2)

This point is on the very right side of the car so it nicely matches with the edge of the parking space.

The second plot (Figure 3) is of the y position of the point of interest with respect to time. The parking spot is between y = 6.5 to y = 11.5.

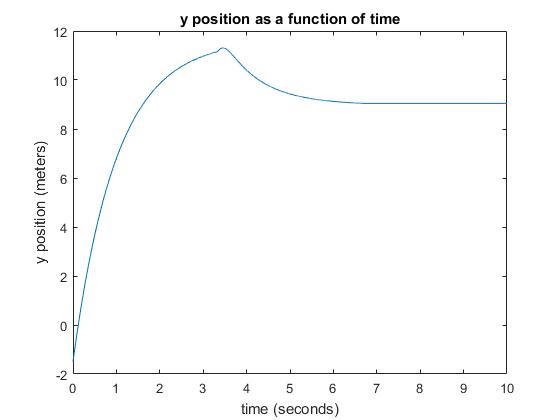


Figure (3)

The car stops at about y = 9 and the car extends +- 1.5 meters in the y direction from the point of interest so this places the car neatly in the parking space.

The next plot (Figure 4) is the angle of the car with respect to time.

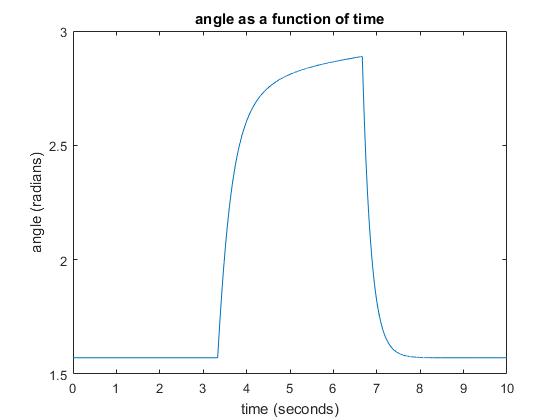


Figure (4)

The car starts at a 90 degree angle. The angle then increases during the second maneuver while the car is backing into the space. The angle then goes back to 90 degrees during the third maneuver as it reorients itself.

The fourth plot (Figure 5) is of the x and y trajectory of the car.

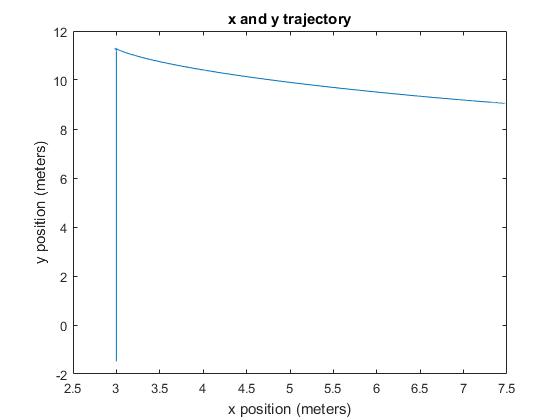


Figure (5)

The first maneuver is the vertical straight line and the second maneuver is the curve to the right of it. The third maneuver isn’t very clear in this plot but can be understood by looking at the angle vs time plot.

DISCUSSION

In my simulation, the car places itself neatly into the parking space but it requires knowledge of the locations where you want to car to go (which gets put in the proportional controller). In the real world, an autonomous car has to use sensors to make sense of the world around it. What it can do is use its sensors to get an estimate of the dimensions of the parked cars and the parking space (it will already have its own dimensions). Then it can run an optimization algorithm to figure out what points to use in the proportional controller. The optimization algorithm would use the simulation I have outlined here to find multiple different trajectories. Heuristics will be assigned to each trajectory as a measure of how close it is to what is desired and as a measure of how to proceed with the next iteration of the optimization.

References

[1] "Parallel Parking Animation." Wikipedia. <https://en.wikipedia.org/wiki/File:ParallelParkingAnimation.gif> Web. 8 Dec. 2016.

[2] “Class 28b” Bhounsule, Pranav

[3] “Class30\_diffdrive\_ik” Bhounsule, Pranav

[4] “class7b\_DifferentialDrive” Bhounsule, Pranav

**Appendix**

Below is a section of the Matlab code. This section is the numerical simulation of the car’s location and angle.

for i = 1:n

if(i<n/3)

p1 = sqrt(((x-b1x)^2) + ((y-b1y)^2));

p2 = sqrt(((x-b2x)^2) + ((y-b2y)^2));

V = [cos(theta)-((ly/lx)\*sin(theta)), sin(theta)+((ly/lx)\*cos(theta)); -(1/lx)\*sin(theta), (1/lx)\*cos(theta)]\*[kp\*(cE(1)-x); kp\*(11.625-y)];

v = V(1);

w = V(2);

phiR = (v+(w\*W))/r;

phiL = (v-(w\*W))/r;

x = x + (h\*(r/2)\*(phiR+phiL)\*cos(theta));

y = y + (h\*(r/2)\*(phiR+phiL)\*sin(theta));

theta = theta + (h\*(r/(2\*W))\*(phiR-phiL));

X(i) = x;

Y(i) = y;

t(i) = i\*h;

th(i) = theta;

end

if(i>n/3 && i<(2\*n/3))

p1 = sqrt(((x-b1x)^2) + ((y-b1y)^2));

p2 = sqrt(((x-b2x)^2) + ((y-b2y)^2));

V = [cos(theta)-((ly/lx)\*sin(theta)), sin(theta)+((ly/lx)\*cos(theta)); -(1/lx)\*sin(theta), (1/lx)\*cos(theta)]\*[kp\*(xref-x)-(1/2)\*n1\*(((1/p1)-(1/p01))^2)\*(p1<p01) - (1/2)\*n2\*(((1/p2)-(1/p02))^2)\*(p2<p02); kp\*(yref-y)-(1/2)\*n1\*(((1/p1)-(1/p01))^2)\*(p1<p01) - (1/2)\*n2\*(((1/p2)-(1/p02))^2)\*(p2<p02)];

v = V(1);

w = -V(2);

phiR = (v+(w\*W))/r;

phiL = (v-(w\*W))/r;

x = x + (h\*(r/2)\*(phiR+phiL)\*cos(theta));

y = y + (h\*(r/2)\*(phiR+phiL)\*sin(theta));

theta = theta + (h\*(r/(2\*W))\*(phiR-phiL));

X(i) = x;

Y(i) = y;

t(i) = i\*h;

th(i) = theta;

end

if(i>=(2\*n/3))

w = 5\*((pi/2)-theta);

phiR = ((w\*W))/r;

phiL = (-(w\*W))/r;

x = x + (h\*(r/2)\*(phiR+phiL)\*cos(theta));

y = y + (h\*(r/2)\*(phiR+phiL)\*sin(theta));

theta = theta + (h\*(r/(2\*W))\*(phiR-phiL));

X(i) = x;

Y(i) = y;

t(i) = i\*h;

th(i) = theta;

end

end

Below is another section of the Matlab code. This section translates and rotates the animated car (which is represented by a square).

while i<n

c1 = c10 + [-cE0(1)+X(i);-cE0(2)+Y(i)];

c2 = c20 + [-cE0(1)+X(i);-cE0(2)+Y(i)];

c3 = c30 + [-cE0(1)+X(i);-cE0(2)+Y(i)];

c4 = c40 + [-cE0(1)+X(i);-cE0(2)+Y(i)];

cc = cc0 + [-cE0(1)+X(i);-cE0(2)+Y(i)];

c1 = c1 - cc;

c2 = c2 - cc;

c3 = c3 - cc;

c4 = c4 - cc;

R = [cos(th(i)-th0), -sin(th(i)-th0); sin(th(i)-th0), cos(th(i)-th0)];

c1 = R\*c1;

c2 = R\*c2;

c3 = R\*c3;

c4 = R\*c4;

c1 = c1 + cc;

c2 = c2 + cc;

c3 = c3 + cc;

c4 = c4 + cc;

line([c1(1) c2(1)],[c1(2) c2(2)]);

line([c2(1) c4(1)],[c2(2) c4(2)]);

line([c4(1) c3(1)],[c4(2) c3(2)]);

line([c3(1) c1(1)],[c3(2) c1(2)]);

%F(count) = getframe(gcf);

%axis([0 15 0 15])

axis([-3 15 -3 15])

i = i + 100;

pause(0.1);

clf('reset');

rectangle('Position',[b1x-1.5,b1y-1.5,3,3]); %13.

rectangle('Position',[b2x-1.5,b2y-1.5,3,3]);

title('Animation of Car Path');

xlabel('x position (meters)');

ylabel('y position (meters)');

end