Autonomous Parallel Parking of a Nonholonomic Vehicle

EE 4951 Class Project

Lisa Sullivan, Christopher Wiens, Mun Hoe Sze Tho,
Trenton Palm, and Joel A. Hesch

Submitted on May 15th, 2004

ABSTRACT

The purpose of this project is to design an autonomous parallel parking algorithm implemented on a Pioneer 1 series mobile robot. This algorithm employs the robot's laser scanner to obtain data from the environment and use this data to execute a parallel parking maneuver. This maneuver is constrained to the kinematics of an actual automobile and enables the robot to both park and unpark on its own. The design and strategy including a brief explanation of the mathematical equations used in the algorithm are discussed.

TABLE OF CONTENTS

INTRODUCTION

With the proper algorithm and equipment, many aspects of driving can be automated. On average, each parked car requires 300 square feet of space allowing for room to enter and exit. Because of this, between 25% to as much as 60% of the urban landscape is devoted to automobile use. Americans spend $68 billion for parking and storage each year (AAMA). Because of this, a parallel parking algorithm is worth pursuing. Not only would it save space, but would also save time, money, and headache caused by minor accidents between parked cars while attempting to park.

To employ our parallel parking algorithm on an actual automobile, we must constrain the Pioneer 1 robot's kinematics. The robot does not use Ackerman steering as an automobile does (does not turn with a fixed axle) but instead uses wheel differential. The difference in rotation between the front two wheels rotates the robot around its center of rotation. Our algorithm will emulate Ackerman steering and parallel park the tri-wheeled robot while only using the front two wheel differentials for steering.

A second task is to detect the dimensions of the surrounding environment using the Pioneer robot's laser scanning capabilities. Identification of the size and depth of the parking slot to determine if there is sufficient space for parking is paramount. Once a favorable space is found, the robot will move towards the slot on a backwards tightly-defined path. The desired path will take into consideration the parking area's dimension to create an efficient maneuver and reduce the time needed to park. If the robot cannot park in a single movement, corrections are then performed.

OVERALL STRATEGY

The task of programming the Pioneer 1 series robot to autonomously park has been broken down into several sections. The modularization of this goal allows the team to assess and complete smaller tasks individually. This is an aide not only during the design stage but also during the debug and post-completion stages. It provides an effective way to change or correct one portion of our design without the need to adjust other portions. The four tasks that need to be completed successfully in order to park are as follows.

Region Mapping. The Pioneer must scan the area, as it is moving along a preprogrammed trajectory. This will provide a mapping of the region, and the detection of any nearby obstacles. Given that no obstacles are encountered, the Pioneer will continue on its path looking for parking spots.

Parking Spot Detection, and Assessment. Once an opening in a row of parked robots is found, the Pioneer will determine if it is suitable by measuring its length and width. No height measurements will be taken into account. Measurements will be done by driving by the opening.

Parking. After driving past an open spot and measuring its dimensions, the Pioneer will be in position to park. The trajectory used to enter the spot will be modeled closely on the

arc that a full sized automobile follows as it parallel parks. The mechanics of turning will also be limited to those of an automobile. This is important because the Pioneer has much larger range of motion than a car does.

Exiting the Spot. When the Pioneer has fully parked, it can exit the parking spot. This will be done under the assumption that nothing out of the robots sensor range has moved. The exit trajectory will also be modeled from the movements of a car.

PROJECT SCHEDULE

Project tasks were broken down as follows:
Lisa Sullivan – Constructed map of parking area using laser data.
Christopher Wiens – Estimated position and orientation of robot using laser data.
Mun Hoe Sze Tho and Trenton Palm – Determined parameters of the trajectory the robot must follow and controlled robot to follow this trajectory while moving with the kinematic constrains of an actual automobile.
Joel Hesch – Integrated program modules and test algorithm.

Included in the appendix is a Gantt chart showing task completion.

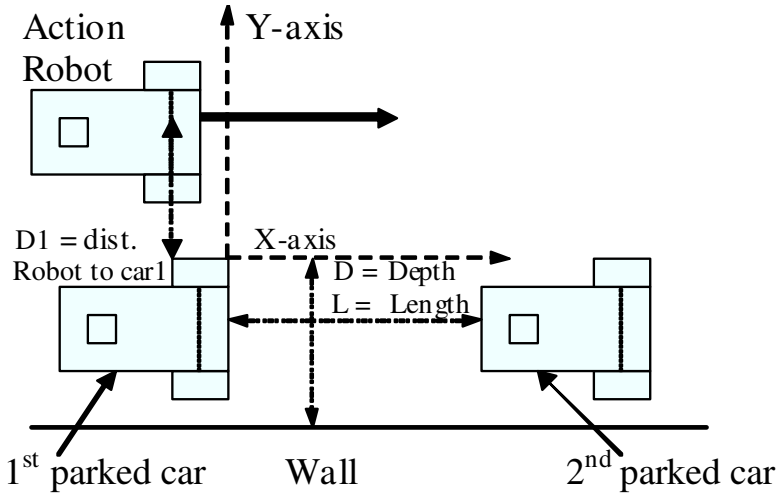
DESIGN APPROACH

Region Mapping and Parking Spot Detection

Prior to parking, the robot drives and aligns itself to the middle of the lane. This alignment is done using laser data from the scanner. If the robot is outside the center zone adjustments are made to the heading of the robot based on the difference between the closest point on the right, and the closest point on the left hand side of the robot. If the robot is within 30 cm of the center of the room, then its heading is set to 0 degrees (straight ahead), and no further adjustments take place until it exits the center zone. This is illustrated in the following figure. The equations which govern this heading adjustment are:
Outside center zone: deltaHeading = 35 * ((distanceDelta) / (distLeft + distRight))
Inside center zone: deltaHeading = angleDelta

As the robot travels down a lane, it avoids obstacles in real time. When the robot detects a row of parked cars, it goes from the avoidance state to the align state. The align state merely involves having the robot align itself parallel to the parked cars with a suitable gap between it and the other cars to ensure accurate laser data.

The key point of the transition of state zero to state one is that the robot must recognize the difference between objects it must avoid and the row of parked cars. A counter was created determining if a vertical line had been detected. The first time a vertical line was detected, was an indicator to the robot to start align itself to the cars.

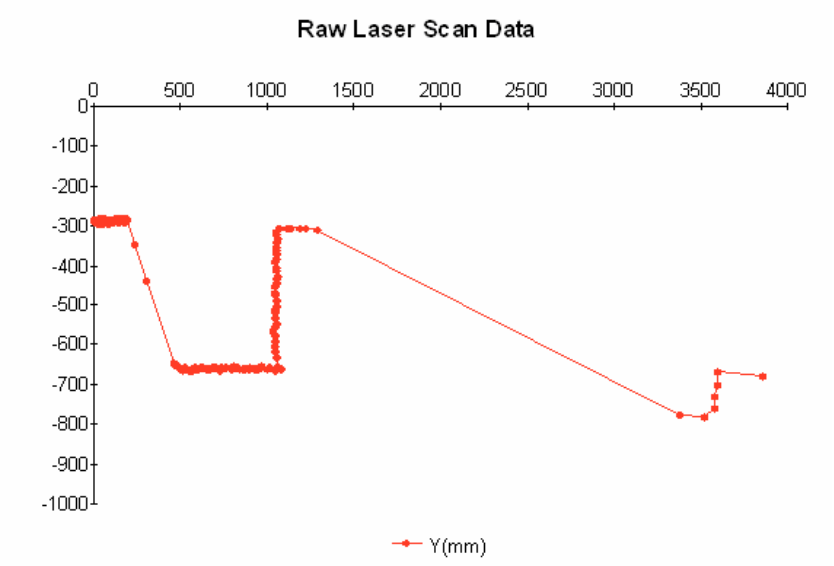


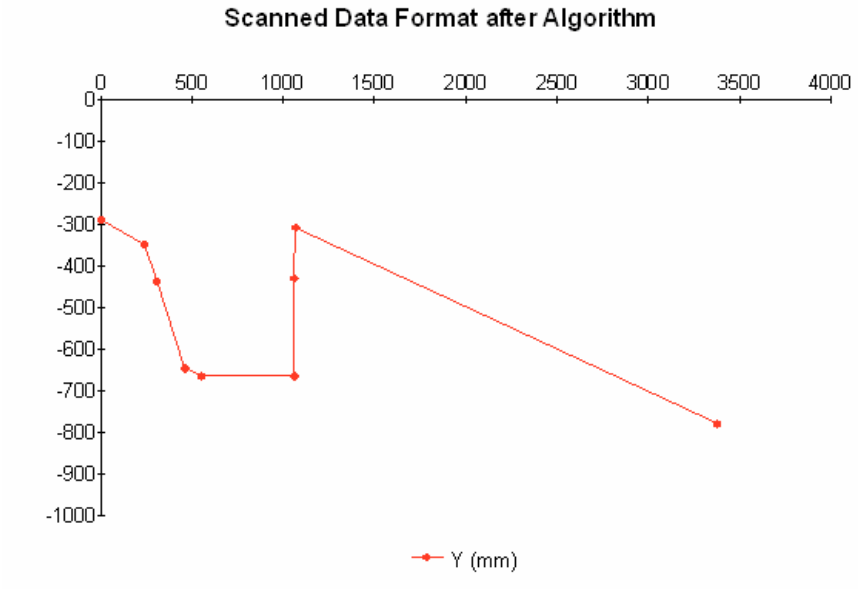
For convenience the laser scanner data was converted from cylindrical coordinates into rectangular coordinates. Then, by using line regression techniques (Cornell), the slope of a line was found between each point. Since sigma was necessary for all the line regression equations, sigma values had to be assigned to each point. Since the line regressions were inversely proportional, the smaller the sigma, the more important the data. Thus the data was divided up into three regions: within four meters, within eight meters and greater than eight meters. Any data determined to be greater than eight meters away was thrown due to the great deal of inaccuracies associated with it. Also, any data that would create a divide by zero error in the determination of the slope between two points was also discarded.

After the slope between all points was found, best-fit techniques needed to be added to decrease the number of total lines. Assuming that a row of cars would be on the right hand side of the robot, only the data involving the right hand side was used. Three separate chunks of code were used to determine three different lines, which were a vertical line, a horizontal line or a diagonal line. If a set of four pairs or more fell into one of these categories, the first point of the first set and the second point of the last set were determined to be the respective beginning and end of a line. The overall slope of the line was thus equal to the quantity of the last y coordinate minus the first y coordinate divided by the quantity of the last x coordinate minus the first x coordinate.

The charts on the next page show the comparison of the end result of line fitting techniques (the bottom one) to the raw laser scan data (the top one). This laser scan was taken just before the robot began the parking maneuvers. All distances are in relation to the robot sitting at the origin about to move in the positive x direction. The data from both charts were outputted into a text file, before being translated into an excel spreadsheet. Thus the charts are merely a graphical representation of what the robots sees.

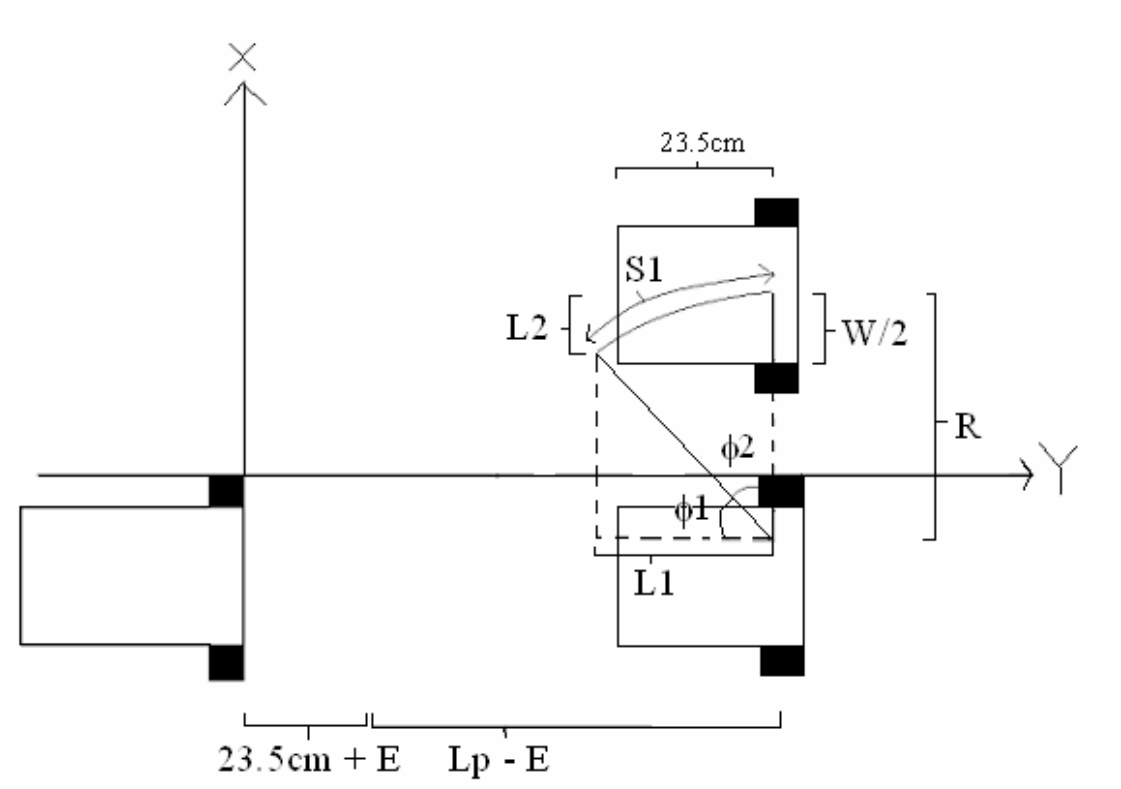
The other function of this code was that since the laser data was inaccurate for very close distances, the robot needed a key landmark to have a sense of where it was within the space. Thus the corners of the line representing the back bumper of the second car became the key landmarks.





Trajectory Calculation/Parking

When the robot is moving parallel to the wall, the front left edge of the first parked robot is fixed as the origin of the Cartesian coordinates which we will use. As the robot moves along, it will measure the depth of the parking spot which has to be at least W + ϵ (W is width of robot and $\epsilon \approx$ 5cm). The length of the parking spot is L_p + 10cm and L_p has to be at least one and a half times the length L of the robot (1.5 x 48.26cm = 72.39cm). When the robot has moved the entire length of the parking spot, it will aligned itself to the second parked robot and initiate its parallel parking trajectory using the parameters obtained from the time it initially detects the first parked robot until it gets ready to park. We use two semicircles to model the robot's parallel parking trajectory (see figures on next page). From Ackerman's Steering Theorem, the distance between the point located at half width of the robot aligned with the two front wheels to the center of radius is R. Also, the maximum turning angle of the inner wheel is 37º and given by the equation θ_i = atan(L/R) (L is length of robot) (Modelica). The turning angle of the outer wheel is given by θ_o = atan(L/(W + R)) and $\theta_i > \theta_o$, with θ_i given as 37º:



First half of trajectory

R = L/tan θ_i
= 48.26cm / tan 37º
= 64.04cm = L2 + h where h = $[(L_2)^2 - (L_1)^2] / 2L_2$
For the first half of the trajectory, the robot will turn with turning radius of R where the center of the circle is located a distance R – L2 from the x-axis.

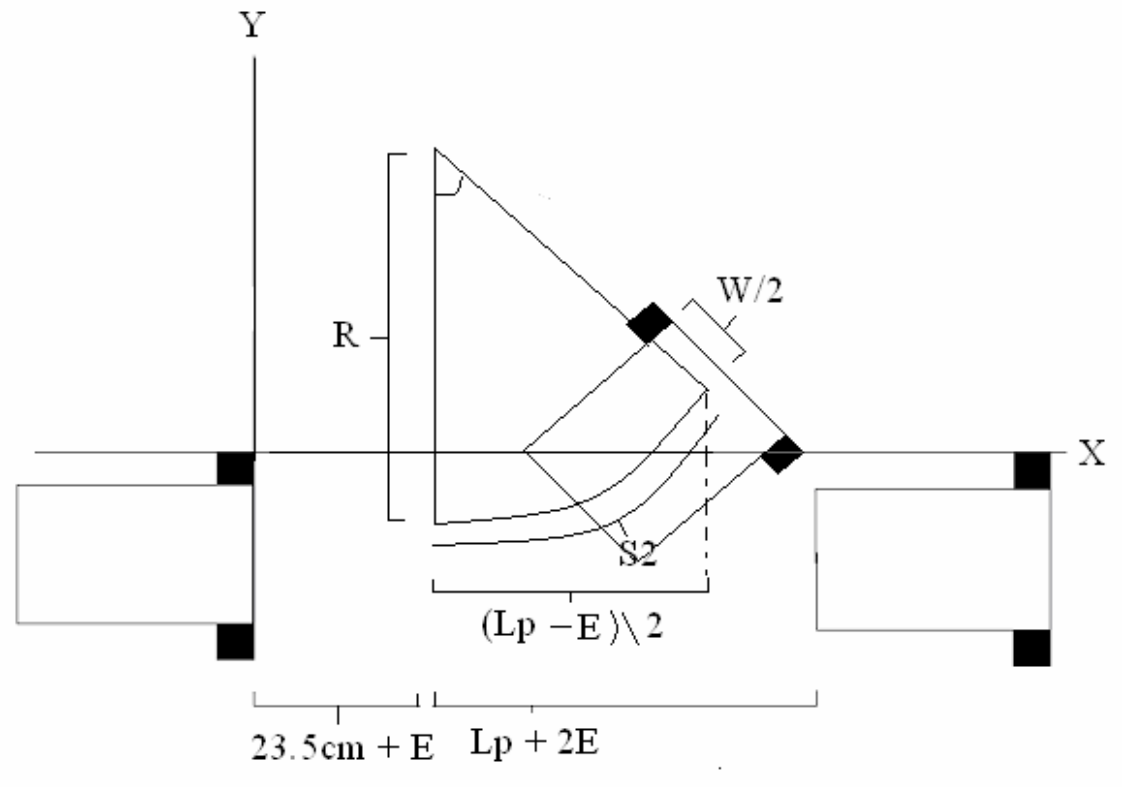
This trajectory will be terminated once the robot has traveled a horizontal distance of L1. The distance S1 which it has traveled corresponds to the angle Φ_2. This angle can be calculated by finding Φ_1:
$\Phi_2 = \cos^{-1}(R - L_2) / R$

Then, $S1 = R\ \Phi_2$.

For the second half of the trajectory, the robot will turn in a semicircle again but this time in the opposite way but with the same turning radius, R. This is shown on the following page. The trajectory will be terminated once the robot has traveled a horizontal distance of L_1 as described in the first half of the trajectory. This means that the vertical line along the center of both front wheels should be at a horizontal distance of 23.5cm + ε from the front of the left robot.

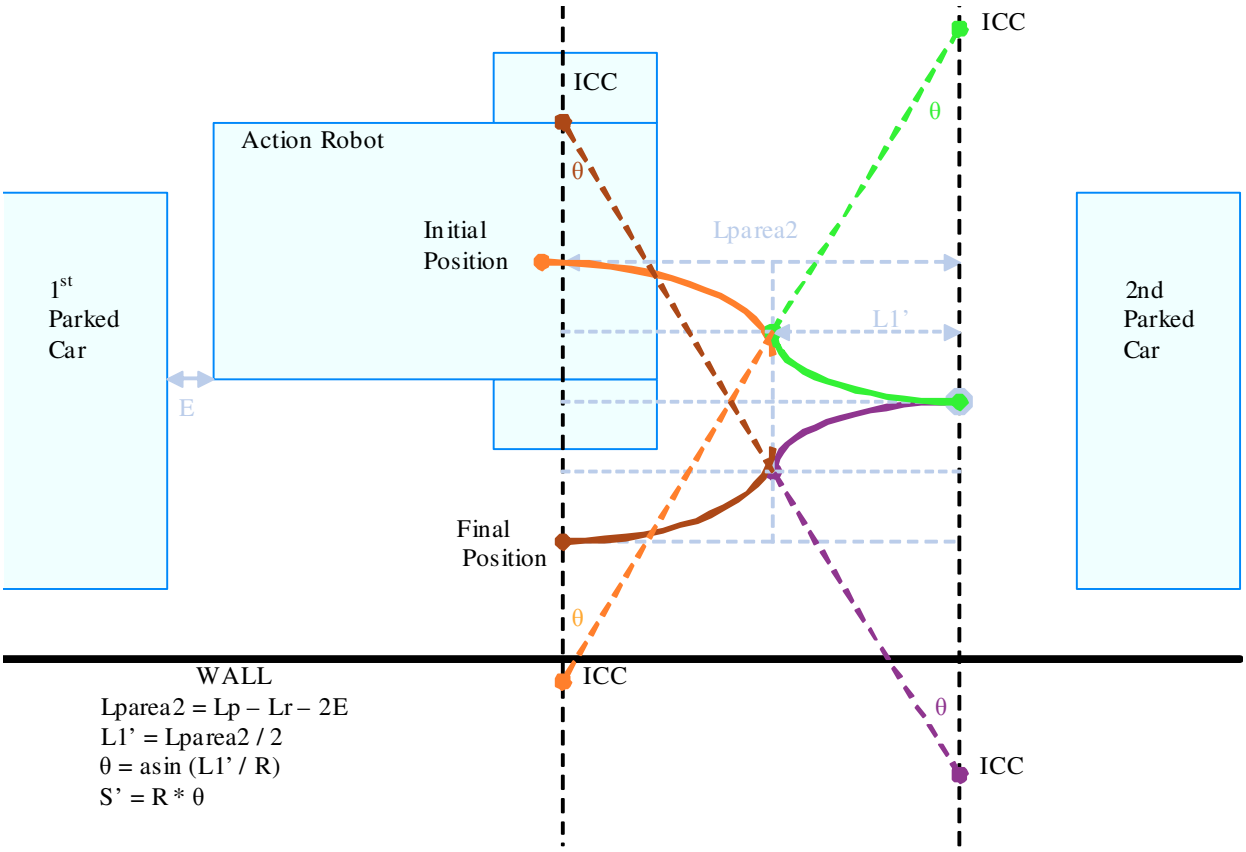
Also, the distance S2 which the robot has traveled corresponds to the angle Φ_3:
$\Phi_3 = \cos^{-1}(R - L_2) / R$

$S2 = R\ \Phi_3$



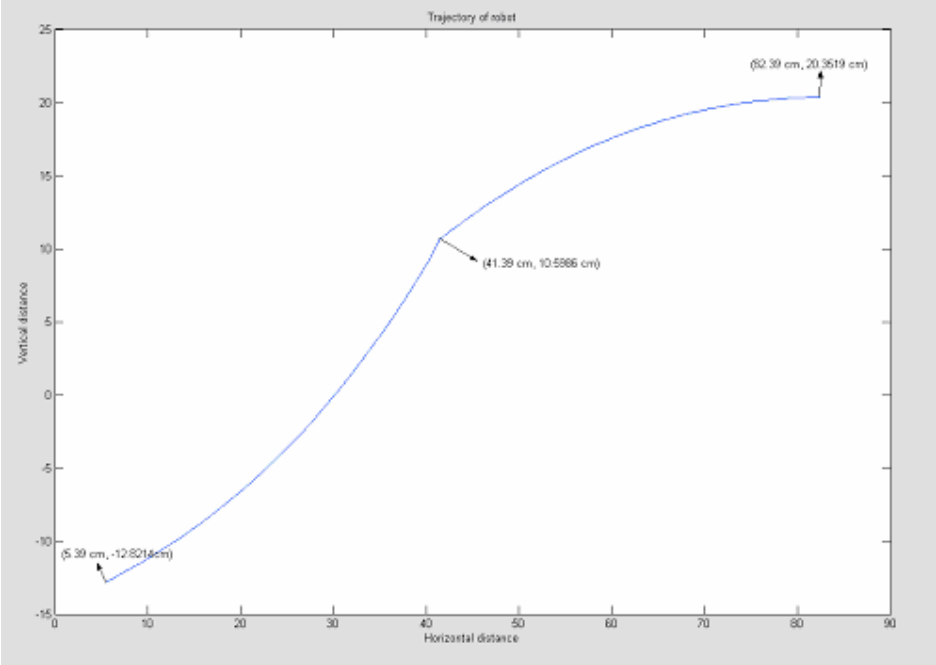
Second half of trajectory

After completing the whole trajectory, the robot is not yet centered in the parking spot due to minor errors in the laser scanners of the robot. Therefore, several corrections have to be made and these corrections are the exact same trajectory as described in this figure that they are carried out in the opposite direction until the robot is centered within the parking spot.



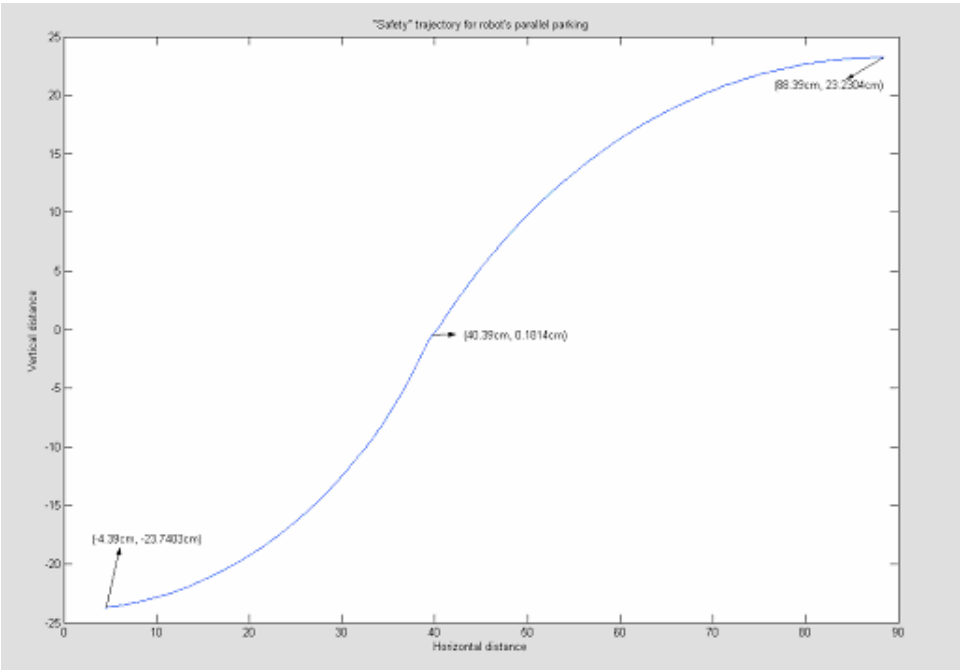
In the beginning of our design process, we proposed to use two semicircles for our trajectory. However, both these semicircles are not exactly half circles due to the dimensions of our robot as well as the constraints imposed by the Ackerman's Steering Theorem. This trajectory is measured relative to the rear of the robot. The first half of the trajectory is smaller than the second half as we can see from the figure on the following page. The first "semicircle" is terminated once it has traveled a horizontal distance of one-half of the length of the parking spot. The second "semicircle" is terminated when it is a distance of 5cm from the parked robot on the left.

Referring to the figure on the following page, the trajectory has a sharp discontinuity at the point when the first half of the trajectory is terminated. In real life, this can not be achieved by the robot because its orientation is parallel to the tangent of the first curve. In order to be oriented parallel to the tangent of the second curve, it has to rotate on its own axis which is not within the constraints of a real car. Therefore, this trajectory is ruled out of our design. Included in the appendix is Matlab code to simulate this curve.



Initial trajectory

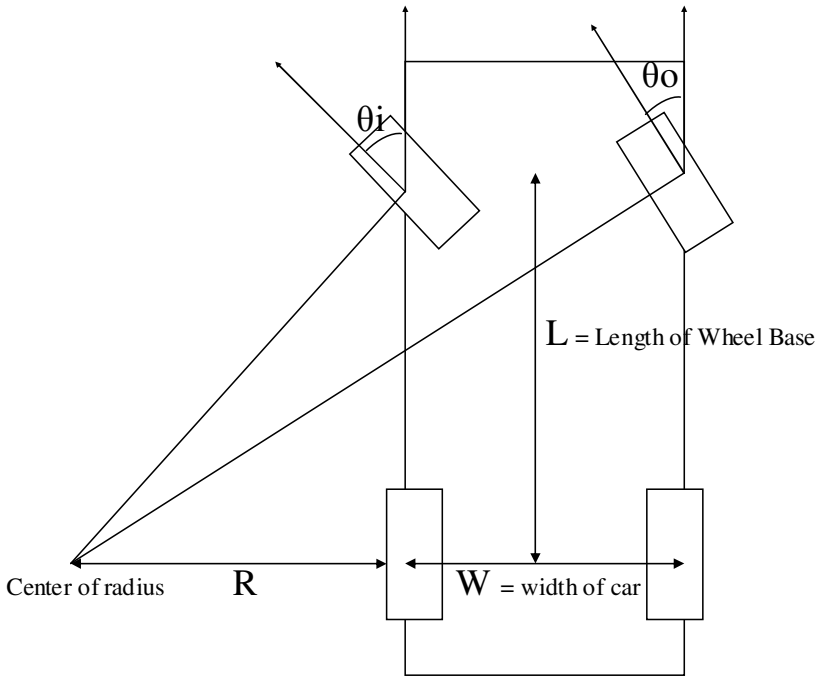
The "safety" trajectory is used as a backup just in case our main trajectory fails. Referring to Figure 4, the discontinuity which exists in our initial trajectory has been eliminated and the point where the first half of the trajectory is terminated is now smooth. However, this trajectory uses a turning angle of 50 degrees, which is not within the constraints of a real car. Included in the appendix is the Matlab code used for the "safety" trajectory.



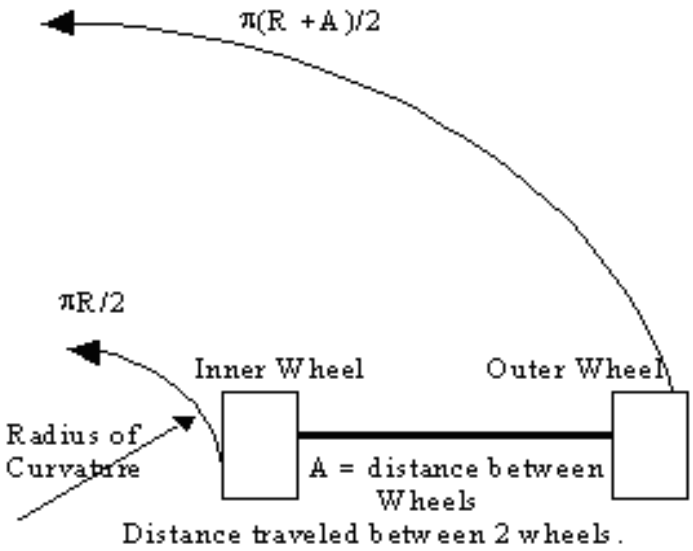
"Safety" trajectory

Wheel Velocity

The next thing that our project has to take into consideration is the velocity on each wheel while following the trajectory curve. One of the objectives of the project is to get the robot to model an actual car. On an actual car both wheels turn with the same velocity. When a car is turning a corner, geometry suggests that the outer wheel has to travel a greater distance than the inner wheel. Because of this, automobiles use the Ackermann steering system. This system makes the wheels turn at different angles to account for the larger distance of the outer wheel. The angle on the inner wheel is defined as atan(L/R) and the angle of the outer wheel is defined as atan(L/(W+R) where R is the turning radius of the inner wheel, b is the track width, and L is the wheel base[2].



Because of the fact that the two motorized wheels on the robot used in this project operate at different velocities we simply have to make the outer wheel go at a greater velocity to account for the greater distance that is has to travel. The figure shows a simple diagram showing that the outer wheel has to travel a greater distance for a designated curve. As you can see the inner tire has to travel a distance $\pi R/2$ and the outer tire must travel $\pi(R + A)/2$.



Exiting the Parking Spot

In order to exit the parking spot, our robot is going to follow the same trajectory that it used to enter the spot. It will simply follow the curve in the opposite direction that it uses to enter the spot including the correction curves.

POTENTIAL RISKS AND SOLUTIONS

Our team was fortunate to not undergo a robot malfunction. If the robot did malfunction, a graduate student is assigned to repair or replace the offending components. A second Pioneer 1 series robot was available in case the components could not be repaired in time for a deadline. If the laptop computer controlling the robot should have failed, a backup algorithm on a different computer would have been employed. In order to correct and refine our code, team members cooperated to provide fresh ideas and advice.

CONCLUSION

To create an algorithm to parallel park a Pioneer 1 series mobile robot, we employed the robot's laser scanner to define the environment. After finding the optimum path for parking, the robot parks, emulating an automobile. We park between objects without hitting them or the surrounding environment. To accomplish this, we broke the robot's path into two semicircles that robot was constrained to follow. These semicircles could be followed by an actual automobile. Corrections were then performed as needed until the robot was correctly parked.

This resulted in an algorithm that could be employed on an actual vehicle with the proper environment mapping equipment allowing for reduced urban congestion and savings in time and money. Future work includes integrating the algorithm into an actual vehicle.

Another algorithm could further save the user's time by automatically paying parking meters when needed through a wireless connection or, when driving is completely automated, to unpark and find a different parking area when a parking meter has expired. This will necessarily require notifying the user that the automobile has moved.

REFERENCES

- Statistics from American Automobile Manufacturers Association and http://www.nevco.com/stats.html. February, 2004.

- Line regression techniques from http://www.library.cornell.edu/nr/bookcpdf/c15-2.pdf.

- Ackerman equations from http://www.modelica.org/Conference2003/VehicleDynamics/help/VehicleDynamics_Chassis_Components.html. 2003.

APPENDIX

```
%Complete parking trajectory of robot for MATLAB

L = 48.26; W = 33.02; Lp = 72.39;

theta_i = 37*pi/180;
R = L/tan(theta_i);
R_prime = R + (W/2 + 10.5);
theta_o = atan(L/(W+R));
Lp_prime = Lp + 10;
d = sqrt(R_prime^2 - (Lp_prime/2)^2) - 10.5;
i = ceil(Lp_prime/2);

X = zeros(1,i);
Y = zeros(1,i);

X(1,1) = Lp_prime;
Y(1,1) = R_prime  - d;

for n=1:1:i-1
    x = X(1,1) - n;
    X(1,n+1) = x;
    y = sqrt(R_prime^2 - (Lp_prime - x)^2);
    Y(1,n+1) = y - d;
end

plot(X,Y);
hold on

%part 2
j = ceil(Lp_prime/2 - 5);

X2 = zeros(1,j);
Y2 = zeros(1,j);

X2(1,1) = X(1,i);
Y2(1,1) = Y(1,i);


for n=1:1:j-1
    x = X2(1,1) - n;
    X2(1,n+1) = x;
    y = sqrt((R_prime - 10.5)^2 - (x + 18.685)^2);
    Y2(1,n+1) = 64.05 - y;
end

plot(X2,Y2);
```

%complete "safety" trajectory for MATLAB

```
L = 48.26; W = 33.02; Lp = 72.39; Wp = 41.02;%width of parking spot, 7
cm wider than robot's width

theta_i = 50*pi/180;%turning angle increased to 50 degrees
R = L/tan(theta_i);
R_prime = R + (W/2 + 10.5 - 6); %whole trajectory moved down by 6cm
theta_o = atan(L/(W+R));
Lp_prime = Lp + 16;%parking spot error increased to 8cm instead of 5cm
d = sqrt(R_prime^2 - (Lp_prime/2)^2) - 10.5 + 6; %whole trajectory
moved down by 6cm
i = ceil(Lp_prime/2 + 4);

X = zeros(1,i);
Y = zeros(1,i);

X(1,1) = Lp_prime;
Y(1,1) = R_prime - d;

for n=1:1:i-1
   x = X(1,1) - n;
   X(1,n+1) = x;
   y = sqrt(R_prime^2 - (Lp_prime - x)^2);
   Y(1,n+1) = y - d;
end
plot(X,Y);
hold on

%part 2

j = ceil(Lp_prime/2 - 8);
X2 = zeros(1,j);
Y2 = zeros(1,j);

X2(1,1) = X(1,i);
Y2(1,1) = Y(1,i);

for n=1:1:j-1
   x = X2(1,1) - n;
   X2(1,n+1) = x;
   y = sqrt((R_prime - 10.5 - 6)^2 - x^2);
   Y2(1,n+1) = 64.05 - y - 43;
end
plot(X2,Y2);
```