

Anish Mohan
Thor Andreas Tangen
Xun Zhou
Amrudin Agovic

Autonomous Parallel Parking

1. Introduction

In highly populated areas it can be difficult to find available parking spots. Frequently parking spots are located on the side of the road, leaving the driver with no choice but to attempt parallel parking. In general it is considered to be a rather challenging maneuver. Since parallel parking requires driving backwards it becomes difficult to coordinate the correct motion of the car. Some drivers have to perform multiple corrections before they park the car properly. In the worst case an accident can occur. A car that can perform parallel parking by itself would save drivers time, especially those that are not very good with parallel parking. In addition cars that can parallel park autonomously in a reliable manner would most probably reduce the number of accidents related to parking. The objective of our work is to implement parallel parking using a car like robot. The robot that we used is of type pioneer 3. We restricted the motion of the robot to model the motion of a car. Using our model we present a solution to the autonomous parallel parking problem.

2. Literature Review

A broad range of literature on parallel parking procedures for Autonomous vehicles was studied. In this section we provide a short review of some papers that we found to be relevant to our project.

In [1], [2], and [3] three related approaches are presented. These papers are mainly concerned with practical aspects of motion generation and control for parking of non-holonomic vehicles. In all three papers the focus is on an iterative algorithm, where each iteration gives a full path to move inside the parking bay. The path that is followed is taken to be a sinusoid curve. Multiple steps are used to ensure that the vehicle has parked properly inside the parking bay. In order to achieve motion control during the parking procedure, feasible control values (such as steering angle and velocity) are computed iteratively. Motion control is supported by gathering and processing data about the objects around the vehicle. Range measurements are used for the localization of the

parking bay, computation of the start position for the parking procedure, and the evaluation of the existing longitudinal and lateral displacement of the vehicle within the bay.

Errors are bound to happen whenever large-scale motion is done in one step. It is claimed that the proposed methods would be somewhat self-correcting since in each iteration, the type of the motion is reversed and hence errors would cancel. However, this assumption does not always hold. There is a possibility of errors accumulating over a number of iterations.

The described papers that use the iterative approach do not deal with the initial problem of localizing the vehicle with respect to the parking spot or adjusting its position to initialize the parking procedure. In real life if a car is on the opposite side of the road it needs to approach the parking spot before the parking procedure can be initialized.

In [4] a new approach to trajectory generation for parallel parking is suggested. The main idea here is based on the fact that the parallel parking of a car is a reversible procedure, with the forward procedure being equated with a person retrieving the car from a parking spot. The trajectory is decomposed into arcs of two circles with different radii and different centers. This approach is practical if the vehicle is close to the parking lot. The paper does not discuss the potential cases when the vehicle is initially further away from the parking lot. Choosing circles with larger radii would be an obvious approach to solve the problem, however it would not be practical.

A possible difficulty while implementing the function is that it gives a continuously curving path. This function is computationally expensive because at various steps the wheel velocities have to be calculated. For accurate curve following, the resolution of steps should be small and more calculations would be required.

In [5] a method for calculation of time optimal movements for parking a non-holonomic vehicle is suggested. The parking problem is considered as specialization of the general problem of path planning for non-holonomic robots. This is formulated as a non-linear optimal control problem and solved using advanced numerical methods. The concept of a critical potential field is used for accounting for the obstacles in the environment. This method allows computation of the time-optimal control of all possible parking configurations (parallel, diagonal, row parking). In this paper a specific mathematical model, in terms of the linear, rotational velocities along with constraints for non-holonomic vehicles is used. The problem of time computation is solved by using state equations of the system (i.e. the mathematical model), the start state and the end state. A control vector is computed which takes the system from the start state to the end state while minimizing time.

The main advantage of the method is that it gives an elegant and general mathematical problem description. It tries to use all kinematics of the platform and it yields the geometric path, as well as the control values, which can be directly used for steering a platform. The limitation of the algorithm is its long execution time. The experiments performed showed an execution time as high as 15 seconds. A solution for all initial positions is not guaranteed. The existence of a solution depends on the initial

configurations and convergence of the method. Also the optimum values obtained for the parameters do not necessarily constitute a global optimum, it is generally a local optimum.

In [6] a method for steering a vehicle with non-holonomic constraints between arbitrary configurations is presented. The problem of steering a vehicle along a sinusoid curve is treated as a control systems problem. Issues with non-holonomic constraints are converted into steering problems in a control system.

The methods that we have described all propose rather interesting approaches. They are different from our method in the sense that most of these approaches were developed using a robot that resembles a car very closely. The robot that is available to us is of type pioneer 3. The pioneer three robot has three wheels. Differential velocities can be set on two of the wheels. The method that we develop needs to deal with setting the individual differential wheel velocities properly.

Another aspect in which our method differs from most of the described approaches is that it includes obstacle avoidance and localization of the parking spot. In order to perform autonomous parallel parking these issues are just as important as the parallel parking procedure itself. Extending our method in such a way introduced a somewhat higher level of complexity to the problem.

3. Problem Formulation

3.1 Problem Description

Given a pioneer 3 robot the task is to program the robot to localize a parking spot, to approach it and eventually to parallel park. The parking spot may not be longer than 1.5 times the length of the robot. The robot should be able to move on a straight line as it drives through the hallway. On the way to the parking spot the robot must be able to avoid obstacles. The robot needs to recognize a parking spot in advance. If the robot is driving on a line which is too far away from the parking spot the robot should turn towards the parking spot until it reaches a desirable position with respect to the parking lot. Once the parking spot is reached the robot should confirm it to make sure that the detection of the spot was really valid. When the parking spot is confirmed the robot should execute the parallel parking maneuver. The issues involved can be summarized as follows:

- Obstacle avoidance.
- Wall/Straight Line following.
- Detection of the parking spot.

- Approaching of the parking spot.
- Parking spot confirmation.
- Computation of a path to be followed to accomplish the parking maneuver.
- Computation of wheel velocities to follow a given path.

After stating our assumptions we will present our solution for each one of these issues.

3.2 Assumptions

We are assuming that our robot will be driving in a hall way with parallel walls. To simplify the kinematics derivations we will assume that our robot is always driving at constant velocity. A parking lot will be represented by two cube shaped objects placed some distance apart next to the wall. The minimum turning radius of the robot will be restricted to 0.5 meters to simulate the motion of a car.

All obstacles that are encountered are assumed to be static. Furthermore we will assume that there will be sufficient space next to the obstacle, meaning that the robot can drive around it. We will also assume that all obstacles are approximately convex.

3.3 Solution

3.3.1 Obstacle Avoidance

While driving down the hallway/road towards its goal, the robot might encounter obstacles. The robot can take two approaches; wait for the obstacle to move or go around it. If the world is said to be static, the second option is the only option. In a real world application, the car would go around or wait, depending on the obstacle encountered, size, space and if it seems to be static or not. We assume the world to be static, so the robot will try to go around the obstacle. The problem can be divided into three sub-problems:

1. Detecting the obstacle
2. Going around the obstacle
3. Getting back on the path towards the goal

When to continue towards the goal to obtain an optimal path is described in [7] and [9]. In our problem we are not concerned about finding an optimal path, we just want to get around the obstacle and continue along the path to the goal. The goal is in our problem some point along the initial heading of the robot. The goal point is not defined explicitly. For the obstacle avoidance we make the following assumptions:

- All obstacles are static
- There is sufficient space on the side of the obstacle we choose to go.
- The obstacles are approximately convex or at least not strongly concave.

Detecting an obstacle

In [8] an obstacle detection algorithm is described. Our approach is similar. The method in [8] tries to minimize the use of power and memory. In our method power and memory usage are not primary concerns. Another difference in [8] is that a camera is used to make a map of the environment. To scan the environment we use a SICK laser. It gives a 180-degree 2D representation of the world. The laser provides a range in 1-degree increments. The robot scans the data from the laser and classifies any point that is within some predefined distance from the robot to be a part of an obstacle. Everything that is further away is not considered an obstacle. The robot uses a look-ahead, i.e. it detects obstacles within some distance, but doesn't avoid the obstacle before the closest obstacle is closer than some shorter distance. This gives the robot a little bit more knowledge about the environment to base its decision on. The robot scans the data from the laser from the centerline and out to each side and records the angle at which the first obstacle is detected and where it ends:

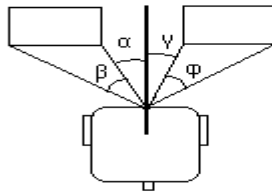


Figure 1

The angles α , β , γ and ϕ would be recorded in this case. The next step is to determine which side to choose to avoid the obstacle. Two values play an important role in this step. These values are the lengths from the outer obstacle boundaries to the centerline as illustrated below:

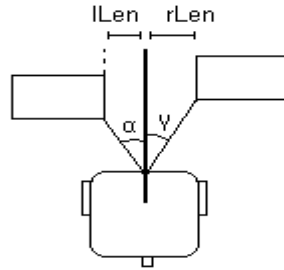


Figure 2

These lengths are found by use of simple trigonometry:

$$rLen = distL * \sin(\alpha)$$

$$lLen = distR * \sin(\gamma)$$

where $distR$ and $distL$ are the respective distances to the outer boundaries of the object. An important parameter is the radius/half-width of the robot, $halfwidth$. There are three cases:

Case 1: $rLen > halfwidth$ & $lLen > halfwidth$

This is the case where there is no obstacle in the way, so we can continue driving straight forward.

Case 2: $(rLen + lLen) \leq 2 * halfwidth$

In this case the robot is not able to pass between the obstacles, so it has to go around:

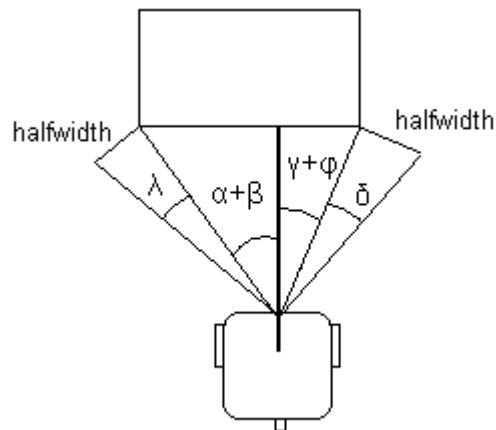


Figure 3

In this case the angles $\alpha + \beta$ and $\gamma + \phi$ is used to decide which way to turn. In consideration of which way to turn one must remember that the robot is not a point but has width, so

one cannot just take the minimum of $\alpha+\beta$ and $\gamma+\phi$ and choose side. U need some offset λ and δ . The values that is used to decide which way to turn is:

$$\begin{aligned} rDiff &= \alpha + \beta + \lambda \\ lDiff &= \gamma + \phi + \delta \end{aligned}$$

where λ and δ is computed as follows:

$$\begin{aligned} \lambda &= \sin^{-1}\left(\frac{\text{halfwidth}}{\text{distL}}\right) \\ \delta &= \sin^{-1}\left(\frac{\text{halfwidth}}{\text{distR}}\right) \end{aligned}$$

where distL and distR are the distances to the obstacle in the direction of $\alpha+\beta$ and $\gamma+\phi$, respectively. When $rDiff$ and $lDiff$ is computed, the robot turns left is $lDiff < rDiff$ and right otherwise. When side is chosen, the robot corrects its heading according to $lDiff$ or $rDiff$.

Case 3: $((rLen + lLen) > 2*\text{halfwidth})$ and $((rLen < \text{halfwidth})$ or $(lLen < \text{halfwidth}))$

In this case the robot can go between the two obstacles, but it must turn around some obstacle either on the right or left. For the left hand side:

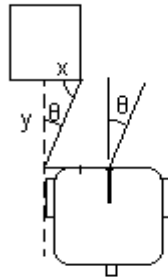


Figure 4

The value θ is the correction of heading that has to be made. The value can be found as:

$$\theta = 90 - \tan^{-1}(y, x)$$

where x and y :

$$\begin{aligned} x &= \text{halfwidth} - \text{lenL} \\ y &= \text{distL} * \sin(\alpha) \end{aligned}$$

in radians/sec. The angle α , is also known ($180 - \theta$). The length AB can be found easy and r can then be found by:

$$r = \left(\frac{AB}{2}\right) * \cos\left(\frac{\theta}{2}\right)$$

The rotational velocity needed to get back on the initial path is then given.

3.3.2 Wall following

The robots wheel encoders have errors. The error is quite significant. If you don't apply some error correction scheme the robot will drift and get off track. By only relying on the wheel encoders the robot will get lost and the heading and position of the robot will be wrong. We are not so concern about the accuracy of the position of the robot because we don't use the absolute position. We are more concerned about the heading of the robot. Since we assume that we have a straight wall on the left hand side we can use this it to compute our heading. To compute the heading we fit straight-line segments to the points that we get from the laser. These readings have some noise, so we don't use them as they are. To minimize the error we merge these line segments. Line segments are merged if their dot product is close to 1. We then use the longest line segment (AB in figure 6).

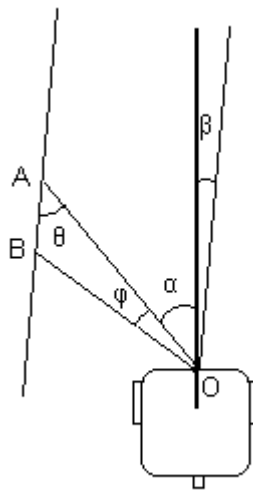


Figure 6

In figure 6 the following relationship holds:

$$\theta = \alpha + \beta$$

So the angle β that the heading is wrong can easily be found. The angles α and φ is known. From the sine relation, θ can be found by the following:

$$\theta = \sin^{-1}(OB * (\sin(\varphi)/AB))$$

AB can be found from the cosine rule:

$$AB^2 = OA^2 + OB^2 - 2 * OA * OB * \cos(\phi)$$

3.3.3 Detection of the Parking spot

In real life it is unrealistic for a car to initialize a parallel parking procedure if the car is on the opposite side of the road. Usually a driver will recognize a parking lot in advance. As the car is approaching the driver will make sure to get to the correct distance to the parking spot. It was our intent to enable the pioneer robot to behave in the same way. Using the laser scanner on the robot we collected several readings while the parking spot was in front of the robot. We used the data to develop an algorithm, which would allow us to detect the boundary points of a parking spot.

The algorithm finds connected line segments in the laser data by comparing slopes of a moving set of three points. The set is moved one point at a time. After the line segments are determined, consecutive line segments whose dot product is bigger than .95 and smaller than 1.05 are merged. The slope as well as dot product threshold values were determined experimentally.

The resulting line segments are searched for angles close to ninety degrees. (where close is defined by a dot product of less than 0.3, this threshold value was chosen experimentally) If the positions of such angles indicate a potential parking lot, the points are marked as boundary points. Which sets of close to ninety degree angles constitute a potential parking lot was determined by computing the relative distances and alignments between the points at which these angles occur. The two figures below illustrate the patterns that our algorithm tries to find. The black dots represent points at which close to ninety degree angles occur. The red points are obtained by taking the end of the line that starts at a previous black point. For instance the point C is the end of the line starting at point B in the figures below.

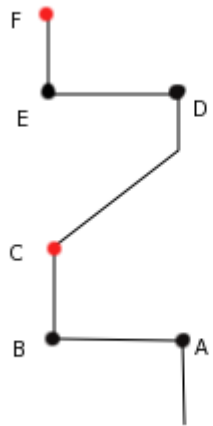


Figure 7

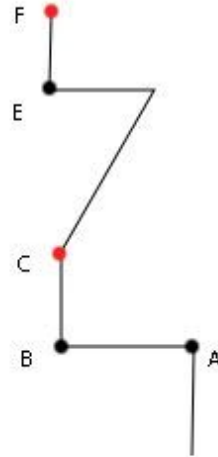


Figure 8

We keep track of the red colored points C and F because they provide us with information about a potential parking lot. The distance between point C and E gives us the length of the parking lot. As illustrated in the figures above our algorithm will return either six or five points. If the relative distances and alignments between the points do not match one of the patterns above the algorithm indicates that no parking lot was detected, otherwise the potential boundary points are returned by the algorithm.

The following figure illustrates the result of our algorithm on real laser data, detected boundary points are circled.

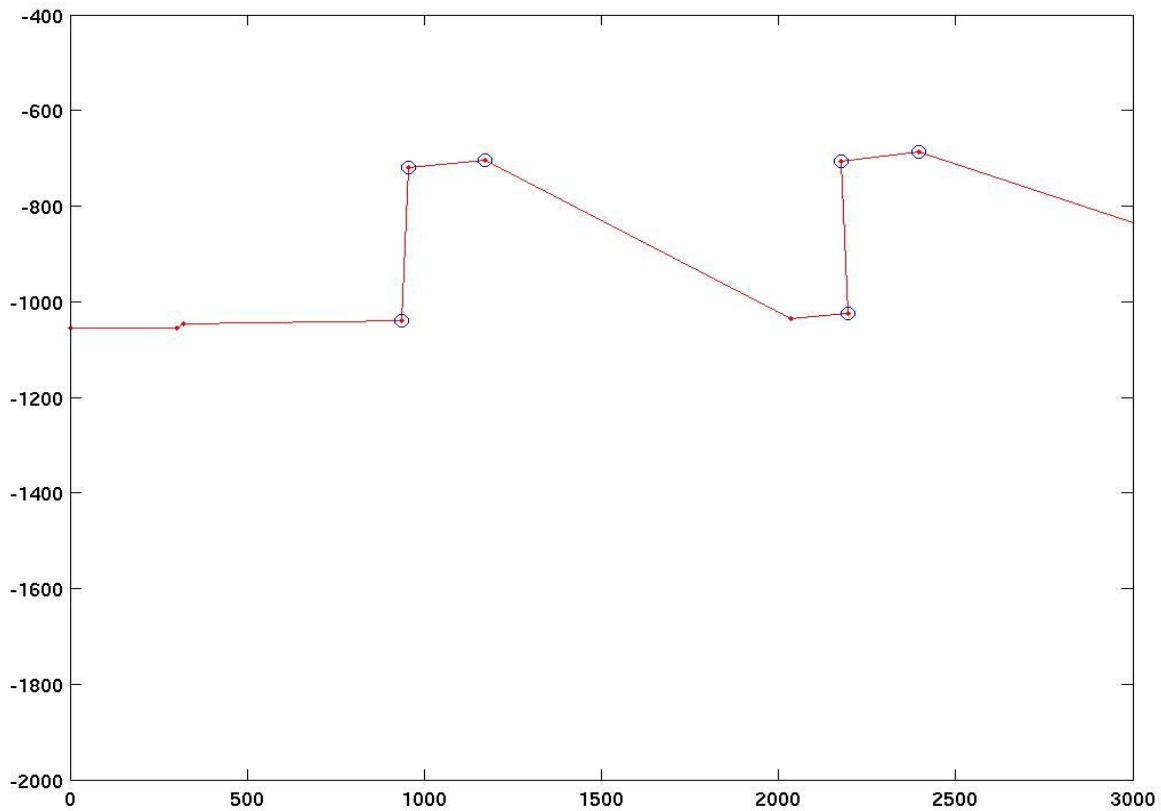


Figure 9

Once the robot detects a set of boundary points it uses curve following to come to a desirable distance to the parking lot. The goal of the robot is to align itself with a line, which is approximately 15 cm to the left of point C. How we achieved curve following is described in detail in section 3.3.5.

3.3.4 Confirmation of the parking spot

While the robot is passing the points which are labeled as B, C and E in Figure 7 it looks at a set of five laser readings starting at 0 degrees. The zero degrees angle is set at -90 degrees from the line the robot is currently following. The robot takes readings from the right wall. For each set of readings the minimum distance value is remembered as the current distance to the right wall. The set of five points was chosen to

The displacement of the left wheel and the right wheel is denoted by D_l and D_r respectively.

$$D_l = b\theta \quad (1)$$

$$D_r = (d+b)\theta \quad (2)$$

The center displacement is denoted by D .

$$D = (b+d/2)\theta = ((d+b)\theta + b\theta)/2 = (D_r + D_l)/2 \quad (3)$$

The linear velocity of the robot is given by:

$$V = \frac{V_r + V_l}{d} \quad (4)$$

From (1) and (2) we obtain

$$\theta = \frac{D_r - D_l}{d} \quad (5)$$

The rotational velocity of the robot is the derivative of θ

$$\omega = \dot{\theta} = \frac{\dot{D}_r - \dot{D}_l}{d} = \frac{V_r - V_l}{d} \quad (6)$$

Relevant frame transformation:

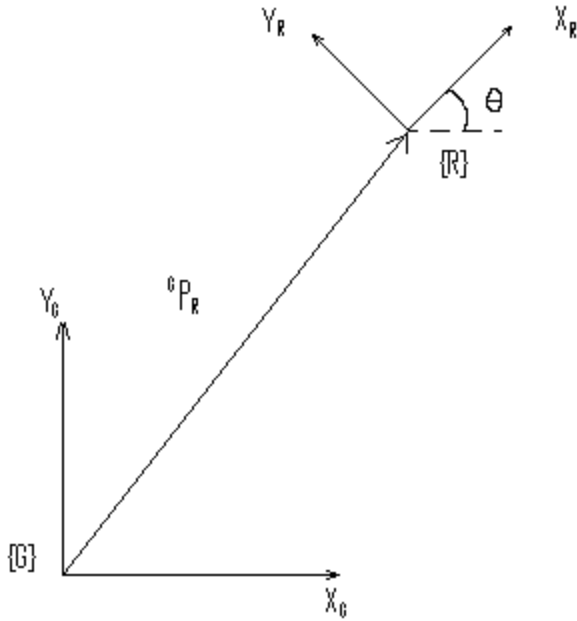


Figure 11

Frame G is the global frame. {R} is the frame attached to the robot.

We have,

$${}^G R = \begin{bmatrix} C\theta & -S\theta \\ S\theta & C\theta \end{bmatrix} \quad (7)$$

$${}^R V_R = \begin{bmatrix} V \\ 0 \end{bmatrix} \quad (8)$$

$${}^G V_R = {}^G R \cdot {}^R V_R = \begin{bmatrix} C\theta & -S\theta \\ S\theta & C\theta \end{bmatrix} \begin{bmatrix} V \\ 0 \end{bmatrix} = \begin{bmatrix} VC\theta \\ VS\theta \end{bmatrix} = \begin{bmatrix} \dot{{}^G X} \\ \dot{{}^G Y} \end{bmatrix} \quad (9)$$

$$\begin{cases} {}^G X = \int VC\theta \, dt \\ {}^G Y = \int VS\theta \, dt \end{cases} \quad (10)$$

The above equation can be approximated to

$$\begin{cases} X(k+1) = X(k) + V\Delta t C\theta(k) \\ Y(k+1) = Y(k) + V\Delta t S\theta(k) \end{cases} \quad (11)$$

If we set the linear velocity to a constant V , the problem becomes to find the rotational velocity ω

$$\omega = \frac{V_r - V_l}{d} = \frac{\theta_{k+1} - \theta_k}{\Delta t} \quad (12)$$

Given the curve:

$$Y = f(x)$$

The heading of the robot at x will be:

$$\theta = a \tan(Y'(x)) \quad (13)$$

From (X_k, Y_k) we increase x by a small distance ΔX to arrive at the point (X_{k+1}, Y_{k+1})
We know θ_{k+1} and θ_k by using (13)

Calculating ΔT :

From (11) we know

$$\Delta X = V \Delta T C\theta(k)$$

$$\Delta Y = V \Delta T S\theta(k)$$

We take the sum of the squares of the above equations and then take the square root:

$$\Delta t = \frac{\sqrt{(X_{k+1} - X_k)^2 + (Y_{k+1} - Y_k)^2}}{|V|} \quad (14)$$

By substituting (13) into (12), we can compute ω

From (4) and (12) we can obtain expressions for the velocities of each wheel.

$$V_r = (2V + d\omega)/2 \quad (15)$$

$$V_l = 2V - V_r \quad (16)$$

To follow a curve given by $Y = f(x)$, we execute a loop in which we compute the two wheel velocities, set the wheel velocities and wait for ΔT .

Using the described method the robot can now perform the parallel parking procedure by following a sine curve.

4.Results

Our robot was robot tested thoroughly in a hallway at the University of Minnesota. Nine out of ten times the robot was able to avoid a static obstacle, detect the parking spot, approach it, confirm it and eventually parallel park. When a failure occurred it usually happened either in the obstacle avoidance phase or the approach of the parking spot. In the obstacle avoidance part the robot very rarely touched the corner of the obstacle. Since the robot did not have a lot of space to move around the obstacle a tight curve around the obstacle was generated. Slight errors in the laser readings as to where exactly the obstacle is located caused the robot to touch a corner of the obstacle. But as was pointed out, this happened one out of ten times. The second source of failure was the approach of the parking lot. Errors in the detection of the parking lot together with accumulated error in the approach of the parking lot caused the robot to be too close or too far from the parking lot.

While our parking spot localization and obstacle avoidance failed in some rare situations, the confirmation of the parking spot and the actual parking procedure were highly reliable. Once a parking spot was detected, approached and confirmed we never observed a failure of the parking procedure. The curve following turned out to be very flexible. Even if our robot did not start the parking procedure from the same distance to the parking spot (due to errors in detecting and approaching the spot) the robot was able to adapt and park properly.

The wall following algorithm that we implemented was quite reliable as well. However, it only worked when there is a straight wall to follow. When we tried to demonstrate our implementation at the University of Minnesota, the robot failed three times. The reason was because there was a door on the wall which the robot used for straight line following. As soon as the robot was moved to a location where the same wall was straight it worked. While our line following is very reliable in the assumed environment, it does not work very well when the assumptions about the environment are changed.

5. Discussion and Future work

We have successfully implemented a reliable parallel parking procedure. The procedure was tested numerous times to ensure that it truly works in practice. While our robot is only a model, it would not be difficult to implement the same algorithm in a real car. The obstacle avoidance and parking spot localization worked with a success rate of approximately 90%. It is a success rate, which is acceptable in an experimental setup, however in the real world it would not be acceptable. If the car crashes one out of ten times no one would use the method. At the same time it needs to be emphasized that obstacle avoidance and localization are separate from the actual parallel parking procedure. So we could implement a low risk real car solution by only including the parallel parking procedure.

In the future we intend to improve our obstacle avoidance and parking spot localization to work reliably. The parking spot detection algorithm needs to be modified to better deal with the noise in the data. The same is true for the obstacle avoidance algorithm. Currently our implementation can only park on the right side of a hallway. We intend to extend the algorithms to allow the robot to park on either side. Because of the symmetry that should not be difficult. We also would like to improve our actual parallel parking procedure, by allowing the robot make adjustments once it is parked. Making adjustments would allow the robot to park in spots that smaller in length.

Another problem with our current method is the restricted environment. Our algorithms don't perform well if the environment even slightly deviates from the assumed environment. In real world situations unexpected situations may arise. We would like to extend our algorithms to cope with such situations. For instance if there is a door on one of the walls the wall following algorithm should still be able to work. If the parking lot is denoted by markings on the ground (such as it is usually in real life), rather than by boxes, the robot should be able park. To accomplish this we might use a computer vision system.

References

- [1] Igor E. Paromtchik, Christian Laugier, "Autonomous Parallel Parking of a Nonholonomic Vehicle", Proc. of the IEEE Intelligent Vehicle Symposium. pp. 13-18 (1996)
- [2] Igor E. Paromtchik, Christian Laugier. "Automatic Parallel Parking and Returning to Traffic Maneuvers", Miscellaneous, Ch. licra98 inria - ucis (1998)
- [3] Igor E. Paromtchik, Christian Laugier. "Motion Generation and Control for Parking an Autonomous Vehicle", Proc. of the IEEE Robotics and Automation. pp 3117-3122 (1996)
- [4] Y. K. Lo, A. B. Rad, C. W. Wong and M. L. Ho "Autonomic parallel parking"
- [5] Konstantin Kondak and Gunter Hommel. "computation of time optimal movements for autonomous non holonomic mobile platforms".
- [6] S. Sastry, R. Murray Walsh. "Non Holonomic Motion planning: Steering Using Sinusoids, IEEE Transactions on Automatic Control (1994)
- [7] V. Lumelsky and A. Stepanov, Path-Planning Strategies for a Point Mobile Automaton Moving Amidst Unknown Obstacles of Arbitrary Shape, Algorithmica (1987) 2: 403-430.
- [8] S. Laubach and J.W. Burdick, "An Autonomous Sensor-Based Path-Planner for Planetary Microrovers", In Proc. IEEE Int. Conf. on Robotics and Automation, 1999
- [9] I. Kamon, E. Rivlin, and E. Rimon, "A New Range-Sensor Based Globally Convergent Navigation Algorithm for Mobile Robots", In Proc. IEEE Conf. Robotics Automation, 1996.