

Parallel Parking Simulation Using MATLAB

Jagadeesh Gowda

KTwo Technology Solutions

North Block, 1st Floor, IT-BT PARK,

Rajajinagar Industrial Estate, Bangalore-560 044, Ph: +91 80 2314 4422,

Fax: +91 80 2314 4411, e-mail: jagadeesh.gowda@ktwo.co.in

Abstract

Parallel parking in narrow spaces is often considered a tedious and annoying task by many drivers. The situation becomes even harder when visibility behind the vehicle has decreased because of aerodynamic design or other environmental conditions. Thus, there is a demand for systems that perform the parking maneuver automatically. Development of such systems on a real platform is very expensive, so the simulation tool helps the developer to understand the parking system, behavior of the sensor and enables comparison of simulation results with actual data. In this paper we focus on simulation of a parallel parking system of a scaled model car.

As we know, the commonly encountered problem of finding the number of sensors required to detect objects in a parking maneuver is very critical and difficult to determine. For the parking system, sensors play a very important role in terms of cost and coverage area. Deciding how many sensors are required is very crucial and depends on many factors, some of which maybe personal choices; this paper provides one solution for it. Also this paper provides a solution for positioning sensors on the car for an automatic parallel parking system based on the simulation results.

Key words: parking simulation, parking guidance, sensor behavior in automotive industry, automatic parking.

1. Introduction

Intelligent Transportation Systems and supporting technologies have been an active area of research for some time. Human drivers exhibit slower response times and errors in judgment that can have serious adverse affects on a parking maneuver. These types of errors can be reduced or eliminated from the driving experience by introducing computer control systems into the automotive arena. It has been shown in other similar studies that such work is often directly applicable to full scale transportation systems. People now take for granted automotive systems like emission control and fuel injection. In fact, many people do not realize how many systems inside their automobile are already monitored and controlled by computers [2]. Fuel delivery, ignition, emissions, air-conditioning, automatic car parking and automatic transmission systems are examples of the systems used daily by a car that are computer controlled or assisted.

The parking simulation system was applied to a scaled model vehicle in MATLAB to assess the practical use of the system on a parking maneuver. Satisfactory results were obtained after testing was completed, and it was found that the application of such systems to scaled model platforms is feasible.

2. Working Principle of Ultrasonic Sensors

Ultrasonic signals are like sound waves, except the frequencies are much higher. Ultrasonic transducers have piezoelectric crystals which resonate to a desired frequency and convert electric energy into acoustic energy and vice versa. Fig. 1 shows how sound waves transmitted in the shape of a cone are reflected from a target.

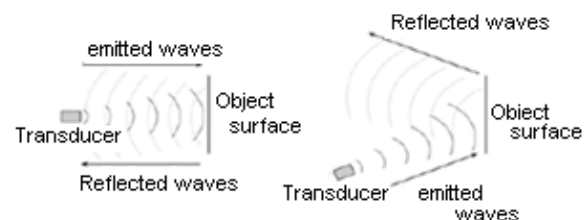


Figure 1: Operation of Ultrasonic Transducer

These sensors have some limitations. They do not detect objects which are above the maximum range of the sensor and if any object lies within the minimum range, the sensor returns the minimum range itself as the distance of the object from the sensor. Depending on the sensor we choose, we will have variations in terms of minimum and maximum range, and cone angle. Fig. 2 shows (a) b) is the

minimum range and (b c) is the maximum range of the sensor.

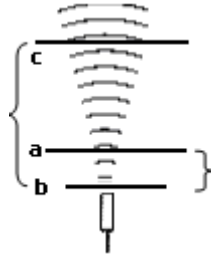


Figure 2: Transducer Limitations

3. Environmental Model for Parking Assistance System

Fig. 3 shows the parking environment model considered in the simulation. It has two stationary cars (Object 1 and Object 2) and a curb (Object 3) which is adjacent to the stationary cars.

For simplicity, let us name the front, back, left and right sides of an object as S1, S2, S3, and S4 respectively, as shown in Fig. 3. Since the side S4 of Object 1 and Object 2 is facing the curb (Object 3), we may not have a situation wherein this surface has any bearing on the parking of the car, so it has been ignored in this simulation. P1 to Pn are positions of sensors on the model car. A detailed explanation of sensor placement has been given in section 5.

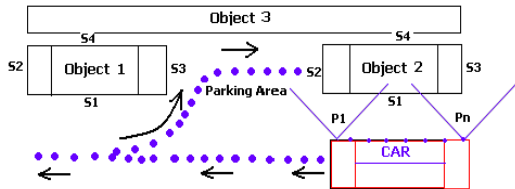


Figure 3: Environmental Model Used for Parking Simulation

4. Vehicle Model

The model we are considering in this work is designed for a car-like vehicle. This means we consider a vehicle with the following properties [1]:

- It moves on a 2D surface.
- Its configuration is represented by a position (x, y) and an orientation θ . This position corresponds to the middle of the rear axle (see Fig. 4).
- Its controls are its speed S and the steering angle of the front axle

- Its dynamics are described by the following equations:

$$\dot{x} = S \cos(\theta) \quad (1)$$

$$\dot{y} = S \sin(\theta) \quad (2)$$

$$\dot{\theta} = S \tan(\Phi)/L \quad (3)$$

where L is the distance between the front and rear axles and Φ is the steering angle.

Fig. 4 summarizes these variables and their relations.

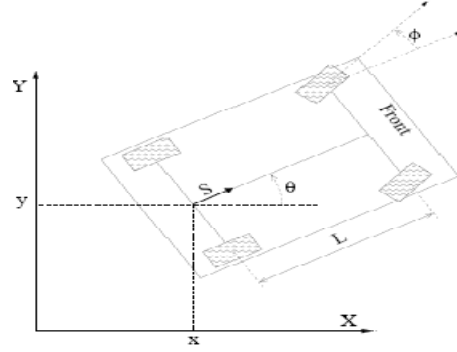


Figure 4: Model of a Car-like Vehicle

The model can be formalized using equations 4 and 5. It is parameterized by six variables:

- λ : the duration of the movement;
- γ : the time when the steering angle changes sign;
- S_{max} : the maximum speed used in the simulation;
- Φ_{max} : the maximum absolute steering angle used in the simulation;
- β : time to reach S_{max} from 0;
- α : time to reach Φ_{max} from 0.

The formal expression of our control profile is:

$$S(t) = S_{max} X$$

$$\begin{cases} t/\beta & 0 \leq t < \beta \\ +1 & \beta \leq t < \lambda - \beta \\ (\lambda - t)/\beta & \lambda - \beta \leq t < \lambda \end{cases} \dots (4)$$

$$\Phi(t) = \Phi_{max} X$$

$$\begin{cases} t/\alpha & 0 \leq t < \alpha \\ +1 & \alpha \leq t < \gamma - \alpha \\ (\gamma - t)/\alpha & \gamma - \alpha \leq t < \gamma + \alpha \\ -1 & \gamma + \alpha \leq t < \lambda - \alpha \\ (t - \lambda)/\alpha & \lambda - \alpha \leq t < \lambda \end{cases} \dots (5)$$

We have implemented the algorithm to generate values corresponding to the 2D space of the model car from equations 4 and 5 using Microsoft Excel.

These generated values are taken as inputs for simulation for drawing the model and typical movement of the car with different orientations.

It has been proved in [4] and [7] that using $\gamma = \lambda/2$ generates a “parallel parking” movement. i.e. orientation is the same at the beginning and the end of the motion. Using an arbitrary γ , [5] and [6] demonstrated that it is possible to reach an arbitrary orientation. Fig. 5 illustrates typical movements of all possible reverse parking steps with varying γ values.

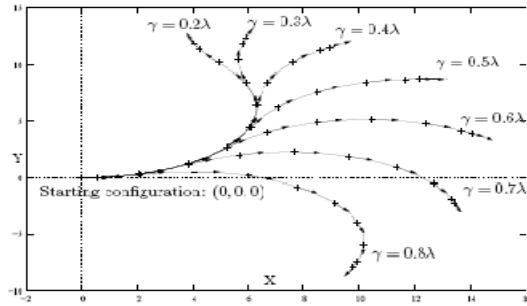


Figure 5: Resulting Elementary Movement: $\lambda = 10s$, $V_{max} = 2m/s$, $\Phi_{max} = 0.45rad$, varying γ

The Fig. 6 shows the conventions used for drawing a 2D car in MATLAB.

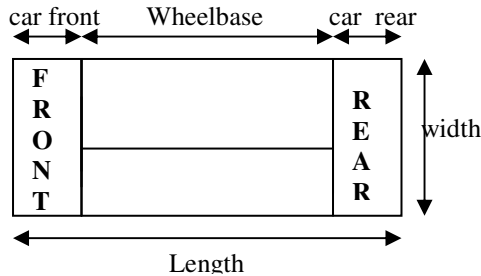


Figure 6: Conventions Used for Drawing 2D Car

The Fig. 7 shows the 2D dimensions of the car, where S is the origin and ' θ ' is the orientation. With S as origin, we are able to calculate the (x, y) positions of all corners of a car.

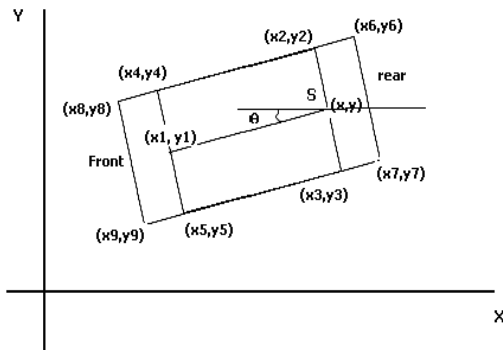


Figure 7: 2D Dimensions of a Car

By knowing the Origin (x, y) and ' θ ', we can calculate other points of the model car such as (x1, y1) to (x9, y9).

Given below are the co-ordinates of the model car.

$$\begin{aligned} x1 &= x - \text{Wheelbase} * \cos(\theta), \\ y1 &= y - \text{Wheelbase} * \sin(\theta), \\ x2 &= x + (\text{width}/2) * \cos(90+\theta), \\ y2 &= y + (\text{width}/2) * \sin(90+\theta), \\ x3 &= x - (\text{width}/2) * \cos(90+\theta), \\ y3 &= y - (\text{width}/2) * \sin(90+\theta), \\ x4 &= x1 + (\text{width}/2) * \cos(90+\theta), \\ y4 &= y1 + (\text{width}/2) * \sin(90+\theta), \\ x5 &= x1 - (\text{width}/2) * \cos(90+\theta), \\ y5 &= y1 - (\text{width}/2) * \sin(90+\theta), \\ x6 &= x2 + \text{car rear} * \cos(\theta), \\ y6 &= y2 + \text{car rear} * \sin(\theta), \\ x7 &= x3 + \text{car rear} * \cos(\theta), \\ y7 &= y3 + \text{car rear} * \sin(\theta), \\ x8 &= x4 - \text{car front} * \cos(\theta), \\ y8 &= y4 - \text{car front} * \sin(\theta), \\ x9 &= x5 - \text{car front} * \cos(\theta), \\ y9 &= y5 - \text{car front} * \sin(\theta). \end{aligned}$$

5. Determining Number of Sensors Required

Objective:

Finding the minimum number of sensors required for a given car with the constraint of detecting anything between the min and max range of the sensors.

Assumptions:

We assume that the sensors have been placed on the right side of the given model car. The first and last sensors have to be placed at a distance of 5% of the length of the car at either end. The sensor's beam of angle (Ψ), minimum distance (m) and maximum distance (Mx) it can cover are provided. Using simple trigonometric principles, we can calculate the minimum number of sensors required as shown below.

Solution:

In Fig.8, let 'L' be the length of the car, 'm' be the minimum range of the sensor, ' Ψ ' be the cone angle of the sensor, 'w' be the width of the sensor beam, and 'n' be the minimum number of sensors.

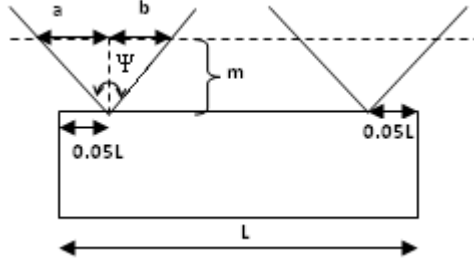


Figure 8: Placement of Sensors

Let $\delta = \Psi / 2$ and 'a' be the opposite side of the angle δ .

Using tangent formula, we can calculate the opposite side 'a'

$$\tan \delta = a / m$$

$$a = m * \tan \delta$$

$$\text{Since } a = b, w = 2 * a.$$

$$\text{Sensors required } (n) = (L - 0.05L - 0.05L) / w.$$

6. Method to Find the Position of Sensors

We compute all sensor positions based on the origin of a model car. By knowing the origin and orientation (θ) of the model car, we calculate the sensor positions of a car is as follows.

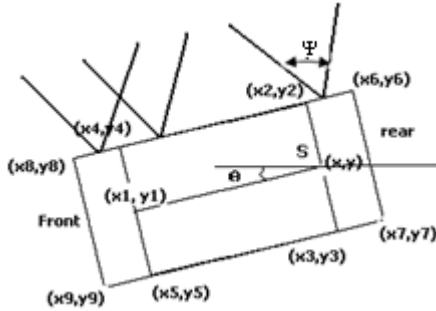


Figure 9: Sensor Positions

Considering Fig. 9, let $(x8, y8)$ and $(x6, y6)$ be the endpoints of the car, i.e. right side. It is assume that the first and last sensor have to be placed at 5% of the length on the side of the car. We have calculated the position difference between sensors is as follows.

$$SPx = x8 + (\text{Length} * 0.05) * \cos(\theta)$$

$$SPy = y8 + (\text{Length} * 0.05) * \sin(\theta), \text{ where } SPx, SPy \text{ are start positions.}$$

$$SPd = (0.9 * \text{Length}) / (n-1),$$

Where SPd is distance between the sensors at the ends and 'n' is the number of sensors.

$$Ai(x) = SPx + (i * SPd) * \cos(\theta)$$

$$Ai(y) = SPy + (i * SPd) * \sin(\theta)$$

Where $Ai(x,y)$ is the sensor position and 'i' ranges from 0 to (n-1)

Once the sensor positions are computed, we construct the sensor cone with the given maximum range 'max_length' and the sensor angle ' Ψ ', as shown in Fig. 10.

Let P, P1 and P2 be the origin of the sensor cone and θ be the car orientation. Then

$$P1(x) = Ai(x) + \text{max_length} * \cos(90+\theta - (\Psi / 2))$$

$$P1(y) = Ai(y) + \text{max_length} * \sin(90+\theta - (\Psi / 2))$$

$$P2(x) = Ai(x) + \text{max_length} * \cos(90+\theta + (\Psi / 2))$$

$$P2(y) = Ai(y) + \text{max_length} * \sin(90+\theta + (\Psi / 2))$$

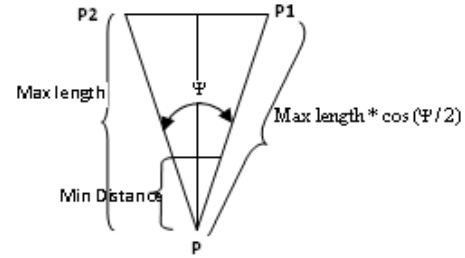


Figure 10: Pictorial Representation of Sensor Cone

7. Intersection of an Object and Car

To find the shortest distance between the car and Object 2 (see Fig. 3), we check the intersection of the sensor cone with the side of Object 2 as shown in Fig.11. Here we were interested in calculating the area of intersection of the sensor cone with side 1 of Object 2.

Let P1 ($x1, y1$), P2 ($x2, y2$) be the endpoints of side 1 and P4 (Sx, Sy) be the sensor position on the model car, P3 (Lx, Ly), P5 (Rx, Ry) be the left and right end points of the sensor cone respectively.

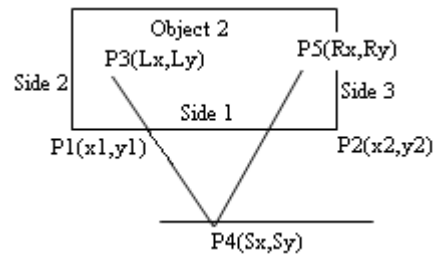


Figure 11: Sensor Cone Intersection for Object 2

By considering P3 (Lx, Ly) and P4 (Sx, Sy) as endpoints of a line and side 1 as another line, we now

find the intersection of the two lines using the algorithm explained below.

Line Intersect Algorithm

This section describes the technique and algorithm for determining the intersection point of two lines in 2 dimensions [3].

Let P1 (x1, y1), P2 (x2, y2) be the endpoints of one line and P3 (Lx, Ly), P4 (Sx, Sy) be the endpoints of another line as shown in Fig.11 above. The equations of the lines are

$$P_a = P1 + U_a (P2 - P1)$$

$$P_b = P3 + U_b (P4 - P3)$$

Solving for the point where $P_a = P_b$ gives the following two equations in two unknowns (U_a and U_b)

$$x1 + U_a (x2 - x1) = Lx + U_b (Sx - Lx) \text{ and}$$

$$y1 + U_a (y2 - y1) = Ly + U_b (Sy - Ly)$$

Solving gives the following expressions for u_a and u_b

$$U_a = \frac{(Sx - Lx)(y1 - Ly) - (Sy - Ly)(x1 - Lx)}{(Sy - Ly)(x2 - x1) - (Sx - Lx)(y2 - y1)}$$

$$U_b = \frac{(x2 - x1)(y1 - Ly) - (y2 - y1)(x1 - Lx)}{(Sy - Ly)(x2 - x1) - (Sx - Lx)(y2 - y1)}$$

Substituting either of these into the corresponding equation for the line gives the intersection point. For example the intersection point (x, y) is

$$x = x1 + U_a (x2 - x1)$$

$$y = y1 + U_b (y2 - y1)$$

Notes:

- The denominators for the equations for U_a and U_b are the same.
- If the denominator for the equations for U_a and U_b is 0 then the two lines are parallel.
- If the denominator and numerator for the equations for U_a and U_b are 0 then the two lines are coincident.

Once we get the intersection point i.e. (x, y), we need to check whether the point falls within the two line segments, P1 (x1,y1), P2 (x2,y2) and P3 (Lx,Ly), P4 (Sx,Sy). i.e. whether x lies between (x1, x2), (Lx, Sx) and y lies between (y1, y2), (Ly, Sy).

It can be expressed in mathematically with reference to the Fig. 11 as follows.

$$x1 \geq x \text{ and } x \leq x2, y1 \geq y \text{ and } y \leq y2$$

This will check for the line 1 boundary.

$$Lx \geq x \text{ and } x \leq Sx, Ly \geq y \text{ and } y \leq Sy,$$

This will check for the line 2 (i.e P3,P4 endpoints) boundary.

The equations above works only when $P1 < P2$ and $P3 < P4$, otherwise we have to interchange the values of (x, y) coordinates of the respective points.

Similarly we can also find out the intersection points for the line points P4 (Sx,Sy), P5 (Rx,Ry) and P1 (x1,y1), P2 (x2,y2).

The same procedure is applicable to get the intersection points for side 2 and side 3 of object 2 and also for the Object 1 and Object 3.

8. Possible Scenarios of Intersection

We have studied the possible intersection of the objects (i.e Object 1, Object 2, Object 3) in a parking maneuver. There are six possible cases we can encounter and they are subject to finding the intersection of the two lines as explained in section 7. Listed below are the possible scenarios.

- In the Fig.12, the left and right sides of the sensor cone are only intersecting side 1 of the object, so there is no need to find intersection point with respect to side 2 and side 3 of the object because the sensor cone is not intersecting these sides. iL, iR and P4 makes a triangle. Using this triangle we can find the minimum distance from point P4 to the side of the object, as shown in section 9. Using the method of intersection given in section 7, we can find iL, iR as two intersection points.

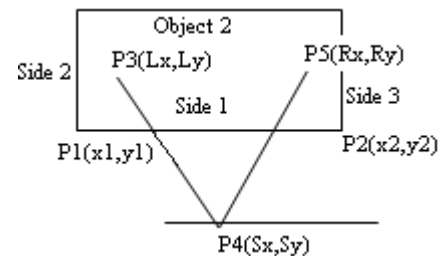


Figure 12: Side 1 Intersection

- In the Fig.13, only the right side of the sensor cone is intersecting side 1 of the object, so we should check whether the left side of the sensor cone is intersecting side 2 of the object. If it is not intersecting, we have to consider P1 as the point of intersection. i.e. P1, iR, P4 makes a triangle, we find the minimum distance from point P4

as explained in section 9. Using the method of intersection, we will get P1, iR as two intersection points.

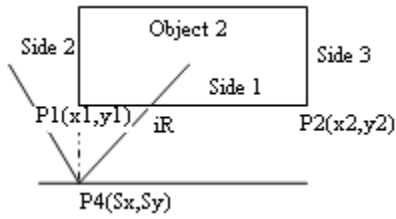


Figure 13: Sensor Cone Intersection with Side 1
(Right Side of the Cone)

- c) In the Fig. 14, only the left side of the sensor cone is intersecting side 1 of the object, so we should check whether the right side of the sensor cone is intersecting side 3 of the object. If it is not intersecting, we have to consider P2 as the point of intersection. i.e. iL, P2, P4 makes a triangle, we find the minimum distance from point P4 as explained in the next section. Using the method of intersection, we will get iL, P2 as two intersection points.

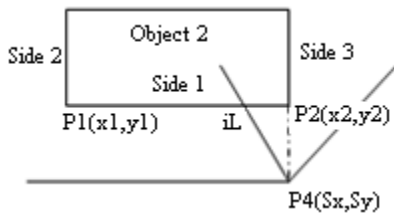


Figure 14: Sensor Cone Intersection with Side 1
(Left Side of the Cone)

- d) In the Figure 15, only left side of the sensor cone is intersecting side 1 of the object, so we should check whether the right side of the sensor cone is intersecting side 3 of the object. If it is intersecting, we have to consider P2 as point of intersection. Using the method of intersection, we will get iL, P2 as one pair of intersection points.

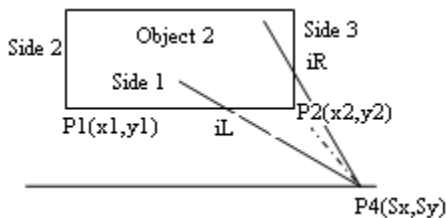


Figure 15: Side 1 & Side 3 Intersection

In this case the right side of the sensor cone is intersecting side 3 of the object, so we need to find out one more intersection point with respect to the left side of the cone. Using the method of intersection we will get P2, iR as another pair of intersection points.

The points iL, P2, P4 makes one triangle and points P2, iR, P4 makes another triangle. Using triangulation method we find the shortest distance of both triangles from point P4. Minimum among the two will be the shortest distance.

- e) In the Fig. 16, left side of the sensor cone is intersecting with both side 1 and side 3 of the object. Since both left and right sides of the sensor cone are intersecting side 3, we are ignoring all other intersection points. Considering the points iL, iR and P4, we find the minimum distance from point P4 as explained in the next section. Using the method of intersection we will get iL, iR as points of intersection.

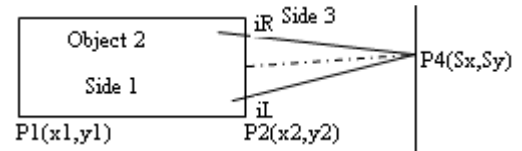


Figure 16: Side 3 Intersection

9. Algorithm for Finding Shortest Distance Between a Line and a Point

To get the shortest distance between a line (L1) and a point, we consider P1, P2 as end points of a line and (Sx, Sy) as a point.

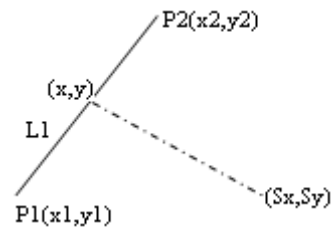


Figure 17: Shortest Distance Between a Line and a Point

As we know, the shortest distance between a point and a line should be the perpendicular line to the given line that passes through the point. If P1 (x1, y1), P2 (x2, y2) are the coordinates of the endpoints

of the line, the dotted line from (S_x, S_y) to a point (x, y) on line is the shortest distance since this line is perpendicular to L_1 and passes through (S_x, S_y) .

Step 1

Find the equation of line L_1

We know the equation of a line

$$y = mx + b \quad (6)$$

Slope of line L_1 is $m = (y_2 - y_1) / (x_2 - x_1)$

Using equation of line $y = mx + b$, find the value of slope intercept 'b'.

By knowing (x_1, y_1) , we can find the y-intercept 'b'

$$y_1 = mx_1 + b$$

$$b = y_1 - mx_1$$

Then substitute the value of 'b' and 'm' in equation (1), you will get the equation in the form

$$a_1x + b_1y + c_1 = 0 \quad (7)$$

where a_1 and b_1 are the coefficients of x and y and c_1 is the constant.

Step 2

Draw a perpendicular line to L_1 , which passes through the point (S_x, S_y) .

The slope of the line perpendicular to another line (L_1) with slope m will be $-1/m$.

Substituting the slope and point (S_x, S_y) in line equation $y = mx + b$,

$$S_y = (-1/m) * S_x + b$$

$$b = S_y + (1/m) * S_x$$

On substituting slope (m) and y-intercept (b), we get

$$y = (-1/m) * x + (S_y + 1/m * S_x) \quad (8)$$

On simplifying the equation (8), we will get the equation of a line in the form

$$a_2x + b_2y + c_2 = 0 \quad (9)$$

where a_2 and b_2 are the coefficients of x and y and c_2 is the constant.

Using cross product of equations (7) and (9),

i	j	k
a_1	b_1	c_1
a_2	b_2	c_2

We find the determinants as follows:

$$i(b_1 * c_2 - c_1 * b_2) - j(a_1 * c_2 - c_1 * a_2) + k(a_1 * b_2 - b_1 * a_2) = 0.$$

By dividing the coefficient of k , we will get the point of intersection (x, y) .

We check whether the intersection point lies within the line segments, if so; we proceed with Step 3 of the algorithm, otherwise Step 4.

Step 3

Finding distance between two points i.e. intersection point (x, y) and point (S_x, S_y)

We know the distance formula

$$D = \sqrt{(S_x - x)^2 + (S_y - y)^2}$$

Here d is the shortest distance between two points.

Step 4

The algorithm above works fine with all the cases to get the shortest distance between a line and point, except when the point (S_x, S_y) lies outside the line segment as shown in the below Fig. 18

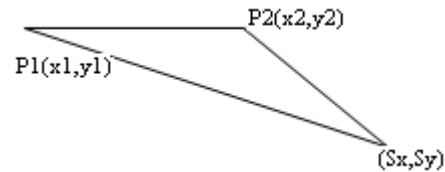


Figure 18: Point Lies Outside the Line Segment

Calculate the distance between Point (S_x, S_y) and $P_1(x_1, y_1)$

$$D_1 = \sqrt{(S_x - x_1)^2 + (S_y - y_1)^2}$$

Similarly distance between Point (S_x, S_y) and $P_2(x_2, y_2)$

$$D_2 = \sqrt{(S_x - x_2)^2 + (S_y - y_2)^2}$$

Between two distances D_1, D_2 , the minimum will be the shortest distance between the line and the point.

10. Automatic Parking Algorithm

The automatic parking process consists of several steps as shown in the Fig. 19. The simulation assumes that alignment of the car is correct and starts moving forward to reach the ready-to-park position with vehicle orientation parallel to the parking space using simulated sensors. Later, values are scanned for identifying the required parking space. The algorithm verifies the length and depth of the parking space. After detecting and checking the size of the parking bay, the simulation makes a decision on the possible parking method – 'possible', 'impossible'. If parking is possible, the vehicle then stops at the recommended ready-to-reverse position from which the parking maneuver starts. Finally, the automatic parking simulation algorithm is executed and the car is parked in the parking space.

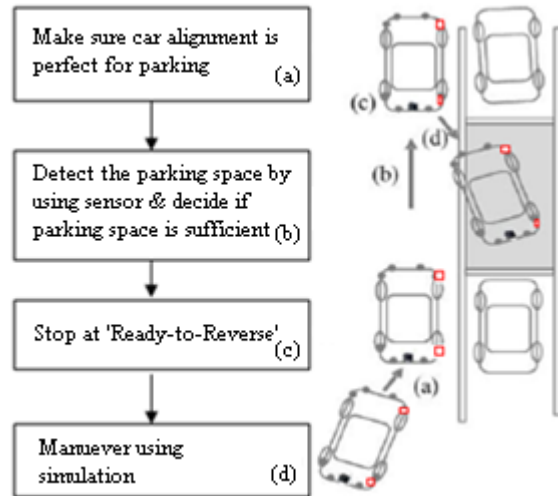


Figure 19: Process of the Automatic Parking System [8]

11. Simulation Results

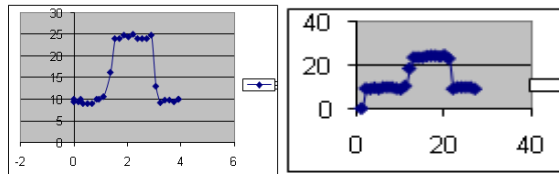


Figure 20: Sensor Values for Different Positions

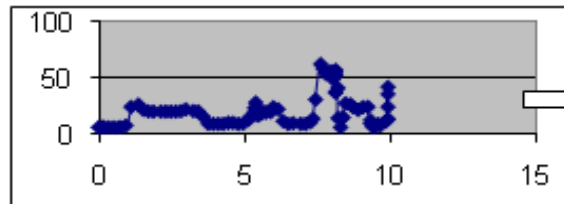


Figure 21: Sensor Values for Irregular Path

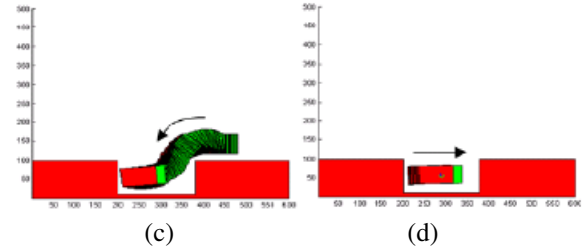
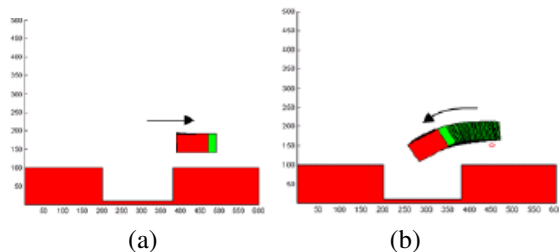


Figure 22: Parking Algorithm (a) Initial Position (b) 'Ready-to-Reverse' Position (c) Reversing into Position (d) Final Position.

12. Conclusion

In this paper we presented a simulation technique that helps to take a decision for the car to park. Our implementation relies on the simulation results of MATLAB to compute the elementary motions of the Objects in a maneuver.

Ultrasonic sensors on the car scan the surroundings until they find a space that is large enough to park in. The simulation then notifies the system of the availability of area. If it is determined that the space is sufficient, then the car is automatically parked in the space.

13. References

- [1]. Path planning for a Parking Assistance System: Implementation and Experimentation by C. Pradalier, S. Vaussier and P. Corke
- [2]. Bosh. *Automotive Handbook*. Third Ed., Robert Bosh BmgH, 1993.
- [3]. Intersection point of two lines (2 dimensions) by Pael Bourke
- [4]. Igor E. Paromtchik and C. Laugier. Autonomous Parallel parking of a nonholonomic vehicle. In Proc.of the IEEE Intelligent Vehicles Symp., 1996.
- [5]. Igor E. Paromtchik. Planning Control commands to assist in car maneuvers. In Proc. Of the 11th IEEE Int. Conf. on Advanced Robotics, 2003.
- [6]. Igor E. Paromtchik. 2004
- [7]. Igor E. Paromtchik and C. Laugier. Motion generation and control for parking an autonomous vehicle. In Proc. of the IEEE Int. Conf. on Robotics and Automation, 1996.
- [8]. Robust Automatic Parking without Odometry Using an Evolutionary Fuzzy Logic Controller Young-Woo Ryu, Se-Young Oh, and Sam-Yong Kim