Optimal Control Theory

Parallel Parking with Collision Avoidance for Four-wheeled Vehicles

Honglu He

Yidong Fu

May. 4th

Content

Executive Summary

The purpose of this project is to explore approaches of optimal control taught in Optimal Control Theory, and their potential application on real-world control problem. The control problem we investigate is the parallel parking with collision avoidance for four-wheeled vehicles. The focus and concentration of the project is to apply collision avoidance capability on a relatively simple four-wheeled vehicle model and study its updated and advanced behavior. The optimal control problem is to minimize the cost defined by vehicle's position and pose, while avoiding any obstacles that is on the way to the desired position. The way how the problem is discretized and formulated into an optimization control problem will be discussed. The methods to approach and solve the optimization problem, such as gradient descent method, will be revisited. The main idea behind the scene is to solve a two-point-boundary-value-problem under some constraints. Finally, some future research that could further complement the problem will be discussed.

The result of the project is successfully reached that the vehicle is able to move to the desired position, in this case the parking slot, at minimum cost while avoiding collision with obstacles. This result only represents an idea to correctly approach and solve this kind of problem. It does not necessarily represent the best method comparing to other potential approaches.

Introduction

Automation has been a very popular topic for the past half century. More new advanced technology depends heavily on the development of automation. Areas like autopilot system and navigation system has many automation applications. The focus of the project will be only on vehicle automation, more specifically, parallel parking automation instead of other systems.

Vehicular automation is also important as daily transportation becomes quite pervasive. Lots of tech companies such as Tesla are developing relevant technology along with some motor corporations. Tesla has had multiple generations of self-driven vehicles that are in the market nowadays. Parallel parking is a very basic action that people will normally do every day. Also, this action takes in multiple driving factors, such as backing and steering. It is a good example to justify a vehicle's self-driving capability.

Technical Background

Although the technology is not a brand-new topic, the online resources that are open and available to the public are very limited. We gather some information online and find a MATLAB example about parallel parking. However, the source codes cannot run directly with up to date dependencies. The team adopts the dynamics for a typical four vehicles from the reference and develops our own optimization problem based on the behavior of parallel parking. Moreover, the simulation in the example does not identify the possibility of collision with obstacles. Usually parallel parking is not an optimal control problem that we usually saw, like minimizing costs or

maximizing profit. The project identifies the traditional trajectory planning problem of parallel parking as well as the optimal control problem of minimizing costs and combines them together to formulate this system.

Problem Breakdown

Model & Focus

Representation of a parking slot

The model of a parking slot represented in the Cartesian frame is adopted from the reference. As *Figure 1* shows, there are several parameters that define the size and position of an empty slot. Three parameters are used to model the size of the road and the size of the empty slot, including $l_{sl}, l_{cl}, and\ l_{sw}$ with each of them represents the length of the slot, the depth of the slot, and the width of the road respectively. These parameters are not very important in the scope of this project as long as the slots are large enough for a vehicle to fit in. Further research on this could be done in the future to determine whether or not there exists a possible solution for a vehicle to fit in a relatively smaller slot. The position of the slot is represented by the coordinates of the left top corner (denoted as O in the figure). The center, which is the desired position of the vehicle, can be determined by the coordinates and three size parameters together. The green shaded area and orange shaded area are the obstacles. In the scope of our project, the green

shaded area is the major issue that we are dealing with. The orange shaded area is on the other side of the "street" thus it does not affect the system.

Around the world, in about 70% of the countries, vehicles are driving on the right-hand side of the road. This model also takes it into consideration. In this project, the vehicle is driving in positive x direction and will back into the slot on the right side of the street. This will simulate the parallel parking behavior of a typical four-wheeled vehicle.

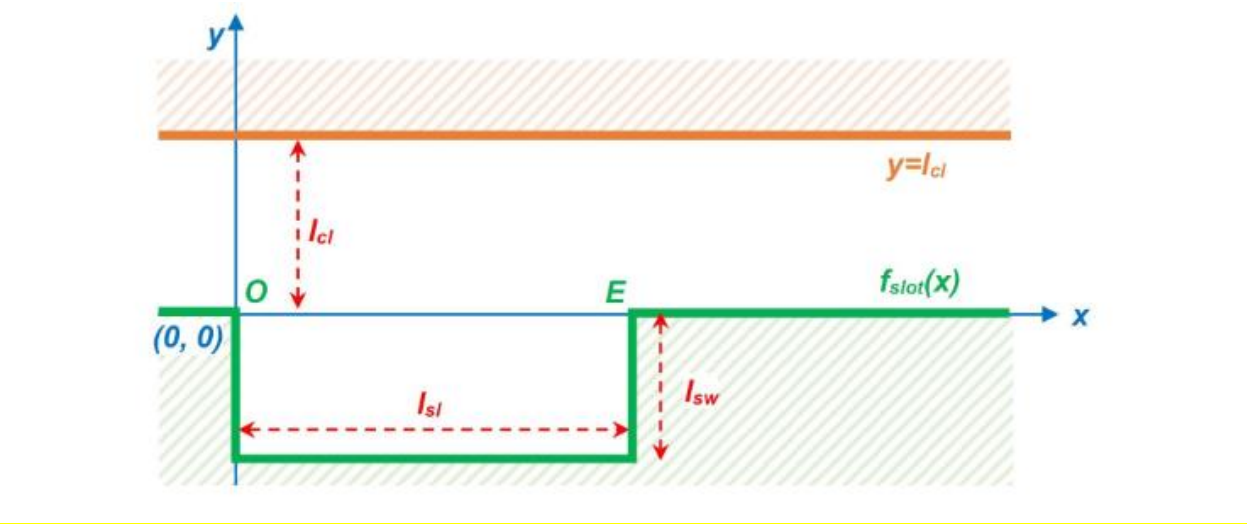

Figure 1: Parking Slot Model [1]

Representation of a vehicle

A typical four-wheeled vehicle model can be found below in *Figure 2*. This model is also adopted from the reference. The simulation happens on a two-dimensional scale where the angle of view is from the top of vehicle. The motion dynamics is based on this model. Multiple parameters are used to define the position and pose of the vehicle. Two angles represent the direction and the pose of the vehicle. ϕ is the angle formed by the wheels and vehicle, as known

as the steering angle. θ is the angle between the vehicle and positive x-axis. Another coordinate (x, y) represents the position of the vehicle. The space the vehicle will occupy can be then found by other size parameters such as $l_r, l_a, l_f, and\ b$. The idea of the parallel parking is to correct the pose of the vehicle such that θ is very close or equal to zero when the vehicle comes to a parking slot.

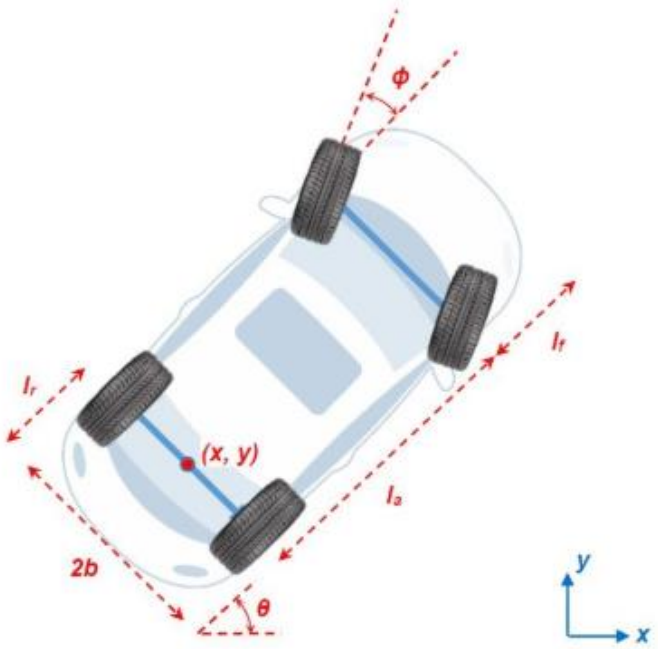

Figure 2: Vehicle Model [1]

Dynamics

The system of this model is nonholonomic. It is different from typical holonomic problem where all constraints are integrable into positional constraints. The states in dynamics of a nonholonomic system depend on the path it takes to achieve. In other words, there is a differential relationship between the states and the inputs. The system of this model has three domains of freedom at any given states in the Cartesian frame. They are x-coordinate, y-

coordinate, and direction. On the contrary, the instant number of domains of freedom is only two because the vehicle cannot slide sideways. The motion of vehicle is determined together by these three vital parameters. More specific numerical dynamics of this problem is as follows.

$$x_{next} = x + v * \cos(\theta),$$

$$y_{next} = y + v * \sin(\theta),$$

$$\theta_{next} = \theta + v * \frac{\tan(\phi)}{\mathrm{l}}.$$

The dynamics can be further interpreted by the matrix below.

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \\ \theta_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \\ \theta_n \end{pmatrix} + u_1 \begin{pmatrix} \cos(\theta_n) \\ \sin(\theta_n) \\ \frac{\tan(u_2)}{l} \end{pmatrix}.$$

In the dynamics, u_1 and u_2 are the inputs to the system. In physical example, u_1 is the velocity of the vehicle and u_2 is the steering angle of the vehicle. l is the distance between the front wheels and rear wheels of a vehicle. There are constraints applied to the inputs and they are as follows.

$$-10 < u_1 < 10,$$

$$-0.8 < u_2 < 0.8.$$

Approach

According to our experience, most robotic vehicle has wheel encoder so that the lower level controller can control the velocity with feedback. Also, the most popular robotic control

interface is Robot Operating System, and the higher-level controller publishes velocity command (Twist Message) at each timestep, while the lower-level controller executes the command with velocity control. The problem is discretized by a certain amount of steps. In the simulation, the team is dealing with the motion per unit time, so that the output trajectory of this project contains given number of waypoints. If the scale is applied to the position and it is in the International System of Units, the unit is $meter$. Under such assumption, the unit of u_1 is $meter/s$, and the unit of u_2 is rad/s. The goal for this project is to output a trajectory formed with waypoints, so that the vehicle can follow the trajectory through traversing each waypoint. We will discuss how the vehicle can traverse all of them later.

Algorithm 1: without Collision Avoidance

Without any obstacles, this problem can be formed as an optimal control problem objective as follows.

$$J = \frac{1}{2}((x(N) - x_d)^2 + (y(N) - y_d)^2 + w(\theta(N) - \theta_d)^2), w = 50,$$

subject to the state dynamics as well as the control input constraints. Notice that the angle θ may go beyond $\pm 2\pi$. However, it is not possible in our case, as parallel parking is a relatively small movement for a vehicle comparing to the size of it. This objective only involves the boundary cost, and this problem can be solved with what we've done before as follow:

$$\mathrm{L} = \mathrm{J} + \sum_{k=0}^{N-1} \lambda_1(k+1)(x(k) + u_1(k)\cos(\theta(k)) - x(k+1)) +$$

$$\sum_{k=0}^{N-1} \lambda_2(k+1)(y(k) + u_1(k)\sin(\theta(k)) - y(k+1)) +$$

$$\sum_{k=0}^{N-1} \lambda_3(k+1)(\,\theta(k) + \frac{u_1(k)\tan\big(u_2(k)\big)}{l} - \theta(k+1)).$$

Forward state dynamics:

$$x(k+1) = x(k) + u_1(k)\cos\big(\theta(k)\big),$$

$$y(k+1) = y(k) + u_1(k)\sin\big(\theta(k)\big),$$

$$\theta(k+1) = \theta(k) + \frac{u_1(k)\tan\big(u_2(k)\big)}{l}.$$

Backward costate dynamics:

$$\lambda_1^*(k) = \lambda_1^*(k+1),$$

$$\lambda_2^*(k) = \lambda_2^*(k+1),$$

$$\lambda_3^*(k) = -\lambda_1^*(k+1) * u_1(k) * \sin\big(\theta(k+1)\big) + \lambda_2^*(k+1) * u_1(k) * \cos\big(\theta(k+1)\big) + \lambda_3^*(k+1).$$

Terminal conditions:

$$\lambda_1^*(N) = x(N) - x_d,$$

$$\lambda_2^*(N) = y(N) - y_d,$$

$$\lambda_3^*(N) = w * (\theta(N) - \theta_d).$$

Gradient of control inputs:

$$\nabla \mathrm{u}_1(k) = \lambda_1^*(k+1)cos\theta(k) + \lambda_2^*(k+1)sin\theta(k) + \frac{\lambda_3^*(k+1)\tan\big(u_2(k)\big)}{l}$$

$$\nabla \mathrm{u}_2(k) = \frac{u_1(k)\sec^2\big(u_2(k)\big)}{l}\lambda_3^*(k+1)$$

Updated inputs:

$$\mathrm{u}_{1_{new}}(k) = \mathrm{u}_1(k) - \alpha\nabla \mathrm{u}_1(k)$$

$$\mathrm{u}_{2_{new}}(k) = \mathrm{u}_2(k) - \alpha\nabla \mathrm{u}_2(k)$$

The trajectory generated is shown below in *Figure 3* and *Figure 4.*

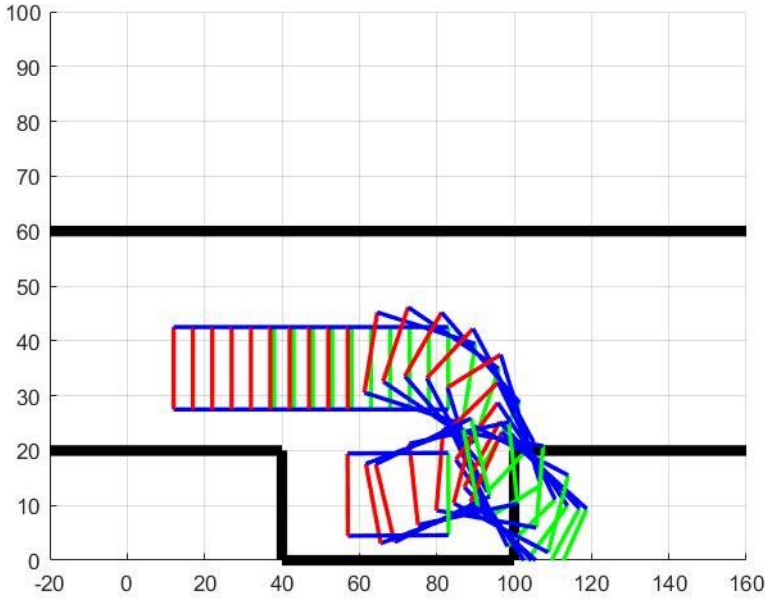

Figure 3: Results without Collision Avoidance 1

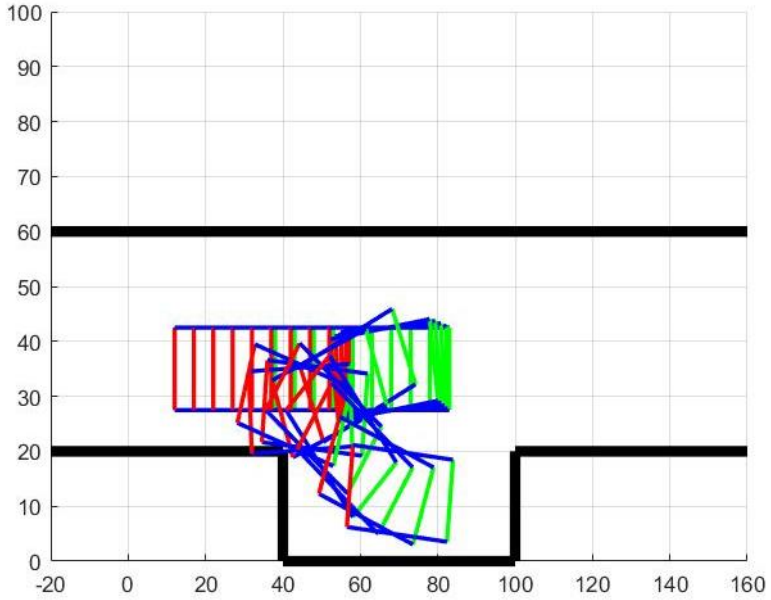

Figure 4: Results without Collision Avoidance 2

As one can see, the vehicle does achieve parallel parking goal at the final step, but it collides with the wall during the trajectory. The second figure above shows another possibility for parallel parking, but it's not what we are expecting, so it's possible to adjust the initial guess of the control input u such that the output trajectory always looks like Figure 3

However, we can see from above results that each step size is about equal, so if we want the vehicle to get to the slot as quick as possible, we can include the running cost in the objective as well as follow:

$$J = \frac{1}{2}\Sigma_{k=1}^{N}\gamma^{k}((x(k)-x_d)^2 + (y(k)-y_d)^2 + \mathrm{w}(\theta(k)-\theta_d)^2), \text{where } \gamma = 1.1.$$

And the optimal control problem can be solved in the similar way:

$$\mathrm{L} = \mathrm{J} + \sum_{k=0}^{N-1}\lambda_1(k+1)(x(k) + u_1(k)\cos(\theta(k)) - x(k+1)) +$$

$$\sum_{k=0}^{N-1} \lambda_2(k+1)(\, y(k) + u_1(k)\sin\,(\theta(k)) - y(k+1)) +$$

$$\sum_{k=0}^{N-1} \lambda_3(k+1)(\, \theta(k) + \frac{u_1(k)\tan\big(u_2(k)\big)}{l} - \theta(k+1)).$$

Forward state dynamics:

$$x(k+1) = x(k) + u_1(k)\cos\big(\theta(k)\big),$$

$$y(k+1) = y(k) + u_1(k)\sin\big(\theta(k)\big),$$

$$\theta(k+1) = \theta(k) + \frac{u_1(k)\tan\big(u_2(k)\big)}{l}.$$

Backward costate dynamics:

$$\lambda_1^*(k) = \gamma^k * (x(k) - x_d) + \lambda_1^*(k+1),$$

$$\lambda_2^*(k) = \gamma^k * (y(k) - y_d) + \lambda_2^*(k+1),$$

$$\lambda_3^*(k) = w * \gamma^k * (\theta(k) - \theta_d) - \lambda_1^*(k+1) * u_1(k) * \sin\big(\theta(k+1)\big)$$
$$+ \lambda_2^*(k+1) * u_1(k) * \cos\big(\theta(k+1)\big) + \lambda_3^*(k+1)$$

Terminal conditions:

$$\lambda_1^*(N) = \gamma^N(x(N) - x_d),$$

$$\lambda_2^*(N) = \gamma^N(y(N) - y_d),$$

$$\lambda_3^*(N) = w * \gamma^N(\theta(N) - \theta_d).$$

Gradient of control inputs:

$$\nabla \mathrm{u}_1(k) = \lambda_1^*(k+1)cos\theta(k) + \lambda_2^*(k+1)sin\theta(k) + \frac{\lambda_3^*(k+1)\tan\big(u_2(k)\big)}{l}$$

$$\nabla \mathrm{u}_2(k) = \frac{u_1(k)\sec^2\big(u_2(k)\big)}{l}\lambda_3^*(k+1)$$

Updated inputs:

$$\mathrm{u}_{1_{new}}(k) = \mathrm{u}_1(k) - \alpha\nabla \mathrm{u}_1(k)$$

$$\mathrm{u}_{2_{new}}(k) = \mathrm{u}_2(k) - \alpha\nabla \mathrm{u}_2(k)$$

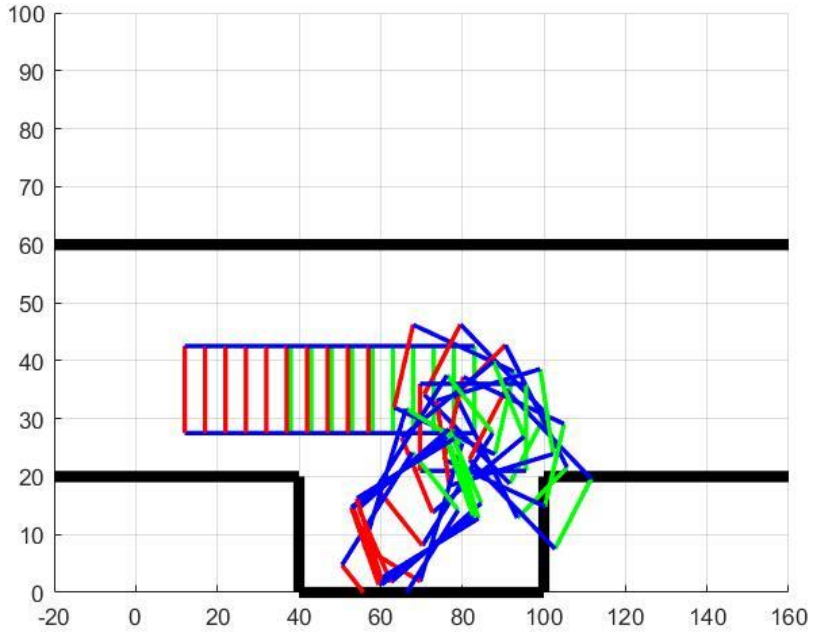

Figure 5: Results without Collision Avoidance 3

As above result shows, the vehicle tends to make larger movement at start, and gradually smaller movement until reaches the destination. It's possible to increase the weight of θ to make sure the vehicle parks in the desired orientation.

Collision Detection

One reason for Algorithm 1 is to identify the potential collision with the wall boundary, and as a result we can see the possible collision point can be identified.

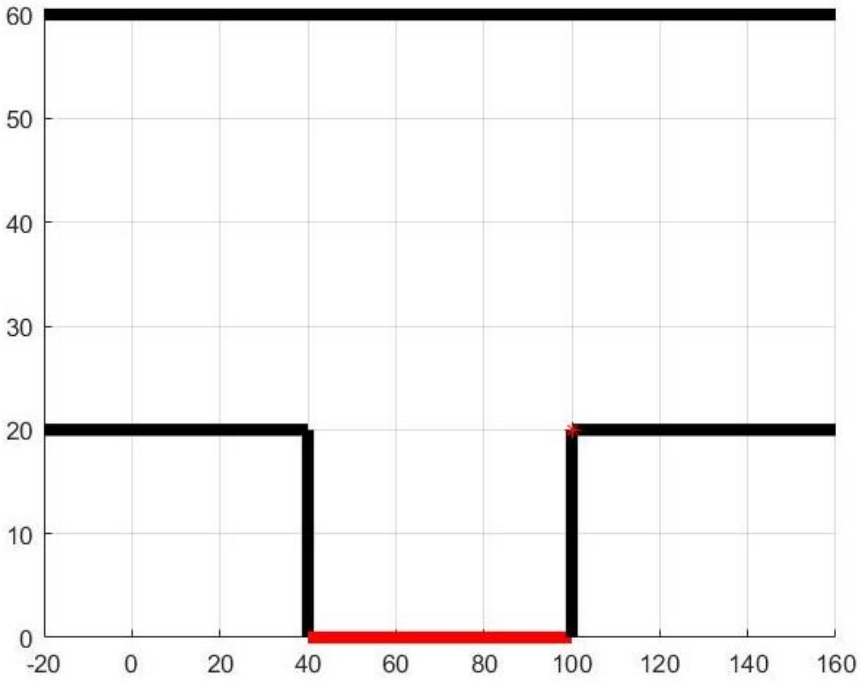

Figure 6: Parking Slot Model with Collision Points

The possible collision points include the upper right corner at location (100, 20) and the bottom boundary wall marked red.

First, in order to identify the collision with the upper right corner, from the reference website, we can label the corners as below:

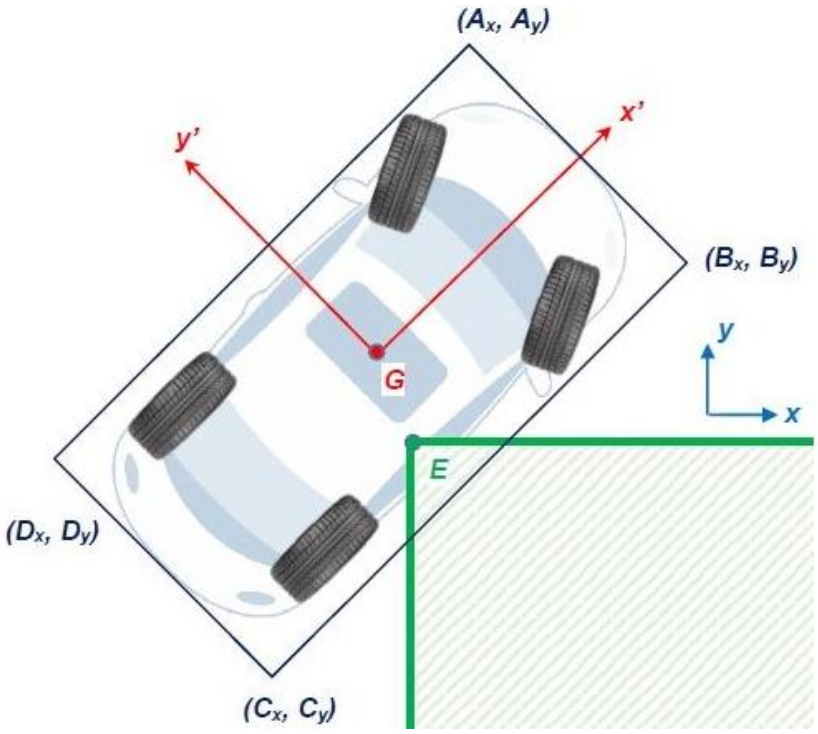

Figure 7: Collision Model [1]

When there's no collision, following statement holds true:

$$Area(AEB) + Area(BEC) + Area(CED) + Area(DEA) > Area(ABCD).$$

It would also be true if the vehicle is completely inside the wall, but in our case, the vehicle can't teleport inside the wall instantly, as it will go through this configuration first.

As for the bottom wall boundary, we can simply detect if any corner of the vehicle falls below the $y = 0$ axis, as the vehicle is a convex object.

Algorithm 2: Collision Avoidance

Given Algorithm 1 and collision detection algorithm, we can perform the trajectory generation with collision avoidance. At each update loop, the state will propagate forward and the costate will propagate backward. While propagating forward, the collision detection is

performed at each state. Once a collision is identified, we're going to use Algorithm 1 again, but with totally different initial state, end state and number of steps.

Suppose the vehicle is at the configuration shown in Figure 7 when propagating forward, the goal at current state is to move the vehicle in the y' direction (normal direction from collision configuration). Therefore, instead of keep updating current control inputs with gradient descent, inside current loop Algorithm 1 is being called again, with 1 steps back as initial state, total of 3 steps available, and the goal pose is 0.2 units away from the configuration in Figure 7 in y' direction.

The idea applies to the bottom boundary wall as well. If a collision with bottom boundary wall is detected during propagating forward, the Algorithm 1 will serve as a collision avoidance to push the vehicle upward directly in 3 steps. By adding this function inside the gradient descent loop of Algorithm 1, the output trajectory is collision free in *Figure 8*.

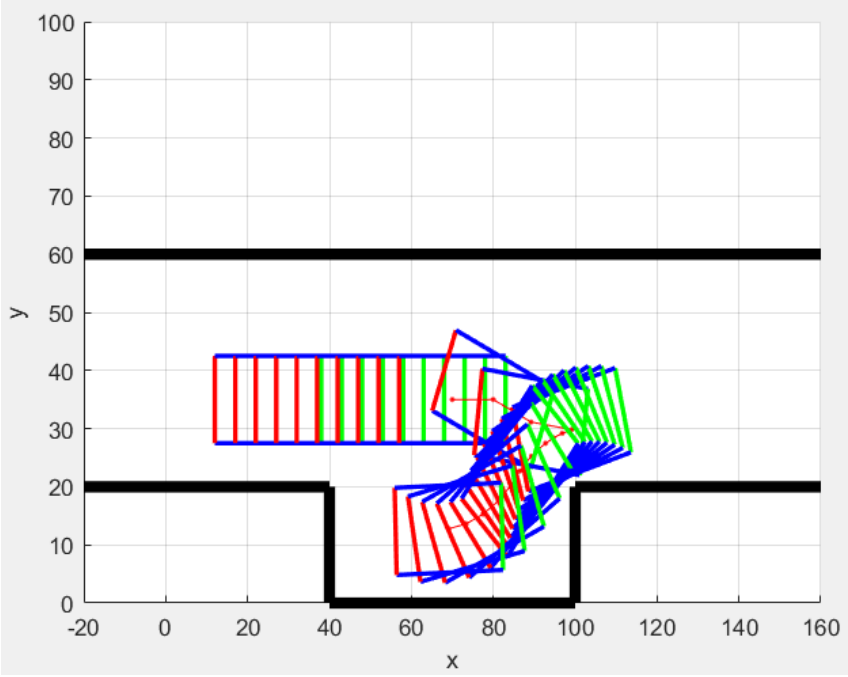

Figure 8: Result Trajectory from Algorithm 2

Algorithm 3, Trajectory Optimization

The motivation for trajectory optimization is to save computational power. Taken the input trajectory from Algorithm 1 like shown in Figure 3, we optimize the trajectory in a while loop until collision free. The collision avoidance idea is the same, but this time it applies to the trajectory directly instead of gradient descent loop. As part of the trajectory is modified, the following trajectory is also re-computed with the remaining amount of steps. The trajectory optimization loop ends until the final trajectory is collision free.

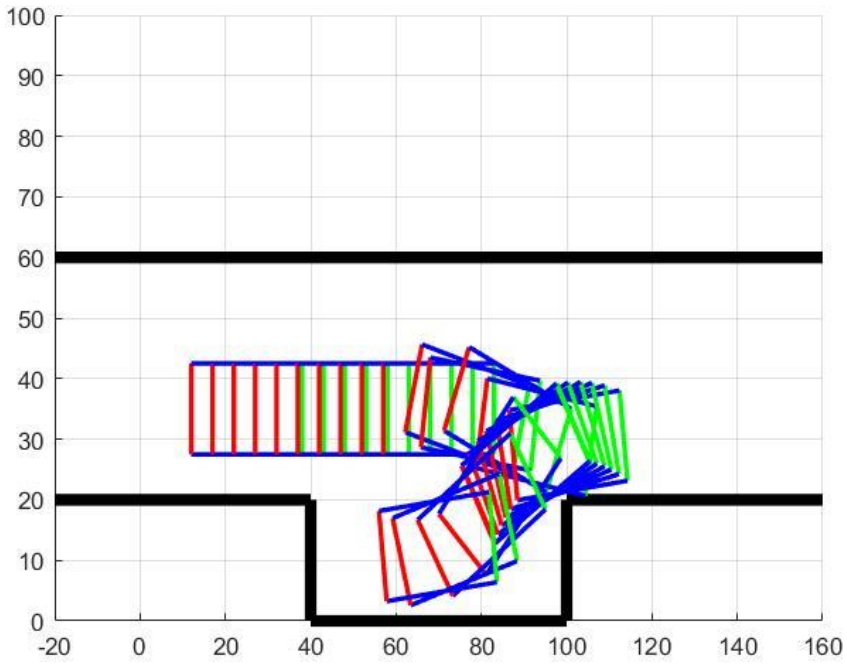

Figure 9: Result from Trajectory Optimization

As one can see, the output of this algorithm is also collision free. Both this algorithm and Algorithm 2 can output feasible trajectories for parallel parking.

Future Work

As mentioned earlier, this project outputs the trajectory of given number of waypoints. While the vehicle can definitely follow it with output control input, the frequency wouldn't be ideal. Usually for a robotic vehicle control, the velocity command is published around 10Hz to 50Hz. In that case it would take hundreds of small steps, and that would take the higher-level controller much longer to compute a collision free trajectory. Therefore, with the trajectory computed from this project, we can fill in steps between the waypoints with Algorithm 1 again (known start and end state, fed with known initial guess at that start state to converge quickly), as it would be much faster since we won't consider collision any more. After that, a complete feasible trajectory with control input with hundreds of time step can be directly passed to lower level controller.

Conclusions

The project successfully explores the idea of optimal control behind a daily activity, parallel parking. The system properly deals with the possible collision and correctly finds the optimal solution of the vehicle to approach a desired position. Gradient Descent method is revisited to solve a two-point-boundary-value-problem in this system. This project is an open-ended project, which has great potential of research extensions. As mentioned in previous section, to find out whether a solution is possible given different constraints can be another small focus of the project. Other vehicle behavior, such as reverse parking, can also be modeled and simulated using similar method to solve the optimal solution. Under real-world circumstances, any complex parking situation would further complicate the control problem. For example, parking on a nonlevel surface would require more base level control knowledge. Although the

inputs in this system can be determined by lower level control, it is still worth researching on a more comprehensive scale. Finally, auto driving vehicles are a closed feedback system involved with many sensors and controls that acquire tons of various environment factors. It could be further developed into a multi-disciplinary research for many fields.

Reference

[1] "Example: Parallel Parking," *ICLOCS2: A MATLAB Toolbox for Optimization Based Control - Example: Parallel Parking*. [Online]. Available: http://www.ee.ic.ac.uk/ICLOCS/ExampleParallelParking.html. [Accessed: 22-Apr-2020].

Appendix

Source Code: https://github.com/hehonglu123/optimal_control