Obstacle Avoidance Trajectory Optimization

Chen Hu, Haosen Xing, Yuyang Wang, Yanjun Pan, Jundong Yi
Department of Mechanical Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
{chu3,haosenx,yuyangw,yanjunp,jundongy}@andrew.cmu.edu

Abstract

Motion planning is an essential part for autonomous driving vehicles. One interesting subject within this area is planning a trajectory that safely overtakes an obstacle in the front. Figure 1 gives a graphical illustration of this problem. This project solved this trajectory planning problem with various optimization based methods and presented comparison among results (i.e. final optimized cost, computational speed). This project included three different optimization methods: iterative linear quadratic regulator (iLQR), genetic algorithm (GA), and interior point methods (IPMs). iLQR was sensitive to initial guesses but had the lowest computational time with a smooth trajectory. GA had a high computational time with a non-optimal and non-smooth trajectory. IPM achieved a smooth and optimal trajectory with relatively lower computational time.

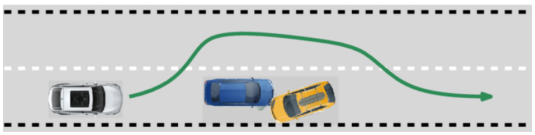

Figure 1: Trajectory Planning Avoiding Unanticipated Traffic Ahead

1 Introduction

Autonomous vehicles have been a trend in recent years, and one of the essential problems in this field is how to avoid obstacles while going back to previous lane afterwards. Our project aims at generating a optimal trajectory for an autonomous vehicle to avoid a single static obstacle and return to the lane as shown in Figure 1.

There are two major categories of constraints: vehicle dynamics and collision check. Our trajectory must be feasible in a way that vehicle dynamics is carefully considered. Another constraint is that the trajectory should be collision-free with the obstacle. For simplicity, circles are used to model the vehicle and the obstacle is modeled with an ellipse as shown in Figure 2.

The key trade-off in the problem is to use the least control input: acceleration and yawn rate in our case to avoid the obstacle while going back the lane as soon as possible. Detailed formulations of the constraints and objective function are introduced in the next section.

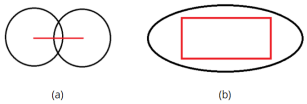

Figure 2: (a) model of autonomous vehicle (b) model of obstacle

2019 Spring 24-785 Course project.

2 Formulation of Optimization Problem

2.1 Problem Statement

The proposed optimization problem is to minimize the total cost of a planned overtaking trajectory, subject to four vehicle states **x** and two control inputs **u**. The decision variable x includes vehicle x, y coordinate (x, y), velocity (v), and heading angle θ. The decision variables u includes acceleration ($\dot{v}$) and derivative of yaw rate ($\dot{\theta}$).

The objective function J(x,u) is to minimize the sum of control effort and differences between vehicle states and reference states. The equality constraint h(x) is the vehicle kinematics model, which guarantees the solution to be physically feasible. Inequality constraints include upper and lower bounds for the two controller inputs, front and rear collision check for ego vehicle [1].

Model parameter table is listed as Table 3.

2.2 Problem Mathematical Formulation

$$\underset{\mathbf{x},\mathbf{u}}{\text{minimize}} \qquad J = \sum_{k=0}^{N-1} \left(C_k^{acc} + C_k^{yawr} + C_k^{ref} + C_k^{vel} \right)$$

minimize:

- acceleration control input
- yaw rate control input
- vehicle coordinates to the reference path
- vehicle velocity to the reference path

$$\text{with respect to} \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ v \\ \theta \end{bmatrix}, \ \mathbf{u} = \begin{bmatrix} \dot{v} \\ \dot{\theta} \end{bmatrix}$$

with respect to:

- state: rear center coordinate x
- state: rear center coordinate y
- state: velocity v
- state: yaw angle
- control input: acceleration $\dot{v}$
- control input: yaw rate $\dot{\theta}$

subject to:

$$\text{subject to} \qquad h1 : \mathbf{x}_{k+1} - f(\mathbf{x}_k, \mathbf{u}_k) = 0, \ k = 0, 1, \ldots, N-1$$

- vehicle dynamics in discrete terms

$$g1 : \mathbf{u}_k^T \begin{bmatrix} 1 \\ 0 \end{bmatrix} - a_{max} \leq 0$$

- maximum acceleration

$$g2 : a_{min} - \mathbf{u}_k^T \begin{bmatrix} 1 \\ 0 \end{bmatrix} \leq 0$$

- maximum deceleration

$$g3 : \mathbf{u}_k^T \begin{bmatrix} 0 \\ 1 \end{bmatrix} - \bar{\dot{\theta}} \leq 0$$

- maximum yaw rate calculated from maximum steering

$$g4 : \underline{\dot{\theta}} - \mathbf{u}_k^T \begin{bmatrix} 0 \\ 1 \end{bmatrix} \leq 0$$

- minimum yaw rate calculated from minimum steering

$$g5 : 1 - (\mathbf{x}_k - \mathbf{x}_o)^T \mathbf{P} (\mathbf{x}_k - \mathbf{x}_o) \leq 0$$

- obstacle distance from the rear center of ego vehicle

$$g6 : 1 - \left(\mathbf{x}_k^f - \mathbf{x}_o \right)^T \mathbf{P} \left(\mathbf{x}_k^f - \mathbf{x}_o \right) \leq 0$$

- obstacle distance from the front center of ego vehicle

2.3 Major Difference from the Project Proposal

Previously, the project proposal proposed a dynamic obstacle avoidance case; however, due to progression and technical issues, the problem was simplified to a static obstacle avoidance case. As a result, optimization was done in a time horizon that spanned the entire scenario instead of in a sliding finite time horizon case. Despite this change, all cost function and constraint functions remained unchanged.

3 Analysis of Problem Statement

3.1 Monotonicity Analysis

The general monotonicity analysis is not appropriate for constraints g_5 and g_6 in this case because of their quadratic nature. However, a more local analysis could be performed by changing the difference to a single variable δx. Besides, analysis could only be discussed for δx that don't change signs. The analysis table is shown in Table 1.

Table 1: Monotonicity Analysis Table

	x	y	v	θ	$\dot{v}$	$\dot{\theta}$
f	+	+	+	+	+	+
g_1					+	
g_2					-	
g_3						+
g_4						-
g_5	-	-	-	-		
g_6	-	-	-	-		

As a result, when $\delta x > 0$, Table 1 suggests that the problem is well defined with g_2 being active on $\dot{v}$, g_4 being active on $\dot{v}$, and g_5 g_6 being active on state vector $\mathbf{x}$. One thing to note is that when δx flips sign, g_5 g_6 would become inactive. Moreover, with state variables having independent signs, g_5 g_6 would have different activeness over variables. However, in general, monotonicity analysis is not an appropriate analysis due to quadratic nature of this problem.

4 Optimization Study

4.1 iLQR

The base line numerical method of this method is the linear quadratic regulator (LQR) which solves an unconstrained problem with quadratic cost. The formulation is shown in Equation 1.

$$\begin{aligned} \underset{\mathbf{x},\mathbf{u}}{\text{minimize}} \quad & J = \frac{1}{2}\sum_{k=0}^{N-1}\left\{\mathbf{x}_k^T\mathbf{Q}\mathbf{x}_k + \mathbf{u}_k^T\mathbf{R}\mathbf{u}_k\right\} + \mathbf{x}_N^T\mathbf{Q}\mathbf{x}_N \\ \text{subject to} \quad & \mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k = 0,\ k = 0, 1, \ldots, N \end{aligned} \tag{1}$$

To convert the defined optimization problem into an LQR problem, the detailed derivation is presented in the Appendix.

By using this value iteration method, reaching global minimal solution was not guaranteed as the cost function after modification was not convex, which made all descending direction non-uniform. If the obstacle lies in the direct center of the lane as the vehicle does, then there existed two optimal trajectories which were mirrored about the center line. In other non-symmetric cases, optimal solutions given a set of initialization were unique.

Since a complex cost function was used, analyzing KKT conditions numerically was very difficult. Instead of checking KKT for termination, the algorithm convergence was satisfied if two consecutive iterations resulted in total costs that were within a small tolerance range.

To initialize the algorithm, a polynomial over-fitting was applied over the obstacle. Figure 3 shows the planned trajectory with vehicle poses at each step. The planned trajectory was smooth and satisfied the vehicle kinematics constraint. The vehicle didn't enter the ellipse for any step during the process, which satisfied the obstacle constraint.

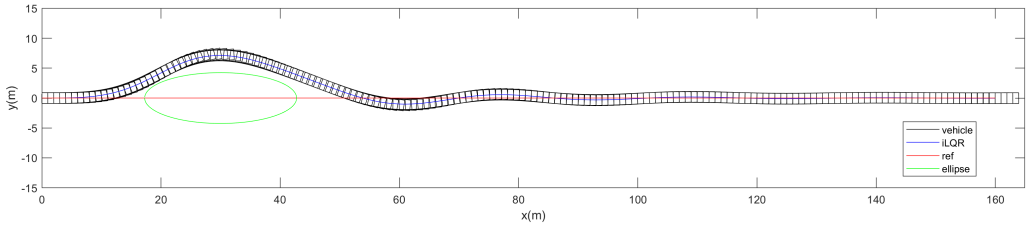

Figure 3: Planned Trajectory with Vehicle Pose

In Figure 4, it could be seen that there were no clippings in control effort, which indicated that control efforts were within constraints as well.

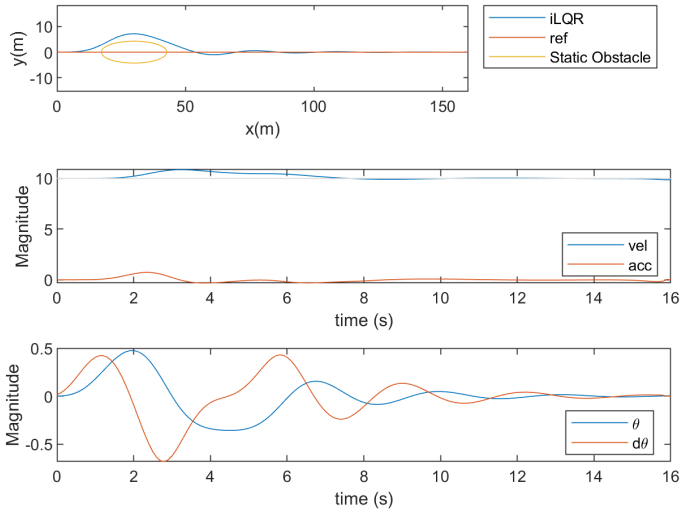

Figure 4: Planned Trajectory with Control Effort

One thing to note is that the planned trajectory had oscillations after passing the obstacle ellipse. Also, when passing the obstacle, vehicle velocity was higher than the reference velocity. The reason behind these two points was suspected to be because of the reference point setting. Currently, there existed 160 reference point over the entire 16 seconds planning horizon. A one-to-one penalty was used in iLQR, meaning that the k^{th} point on the planned trajectory corresponded to the k^{th} point on the reference trajectory. When avoiding the obstacle, the vehicle needed to cover more distance within the same time interval. As a result, once it passed the obstacle, higher velocity meant larger steering needed and potential oscillating for settling were possible.

Besides the oscillation, the algorithm was also sensitive to initial guess. Figure 11 in Appendix shows a sample result from a less favorable initialization. In this initialization, acceleration term was a vector with each entry having a value of 0.03 while the yaw rate vector being a zero vector. In this case, in order to achieve good results, all limits on control inputs were removed.

4.2 GA

Unlike the cost function mentioned above, cost function used for GA was different.

$$\begin{aligned} &\underset{\textbf{path}}{\text{minimize}} && F = \sum_{i=0}^{n} C_{dis}^{i} + \sum_{i=0}^{n} C_{turn}^{i} + \sum_{i=0}^{n} C_{safety}^{i}, \\ &\text{with respect to} && \textbf{path}, \end{aligned} \tag{2}$$

In the first version, the cost of distance, turn and safety were either polynomial or exponential (left part of equation 3). With these cost functions, 400 initial paths were generated as parents. Two of them were randomly picked and a cross point was random generated. Two new children were created using parts of the parents. After obtaining two children, mutation was applied to points on the path, which was random number influenced by which iteration in the process. Finally, children were added into a saved pool until there were enough children, and then parents and children were mixed together and picked the better 400 paths to be parents of next iteration.

$$\begin{aligned} &C_{dis}^{i} = |p_{gen} - p_{ref}|, && C_{dis}^{i} = |p_{gen} - p_{ref}|, \\ &C_{turn}^{i} = \begin{cases} deg * 9/\pi & deg < \pi/9 \\ (deg * 9/\pi)^4 & deg \geq \pi/9 \end{cases} && C_{turn}^{i} = \begin{cases} 0 & deg < \pi/9 \\ 10^9 & deg \geq \pi/9 \end{cases} \\ &C_{safety}^{i} = \begin{cases} e^{100} - e & x < 0.01 \\ e^{x^{-1}} - e & 0.01 \leq x < 1 \\ 0 & x \geq 1 \end{cases} && C_{safety}^{i} = \begin{cases} 10^9 & x < 1 \\ 0 & x \geq 1 \end{cases} \\ &\text{where x = (point on path - ellipse center) /} && \text{(point on ellipse - ellipse center)} \end{aligned} \tag{3}$$

Working with the method above, results were shown as Fig. 5:

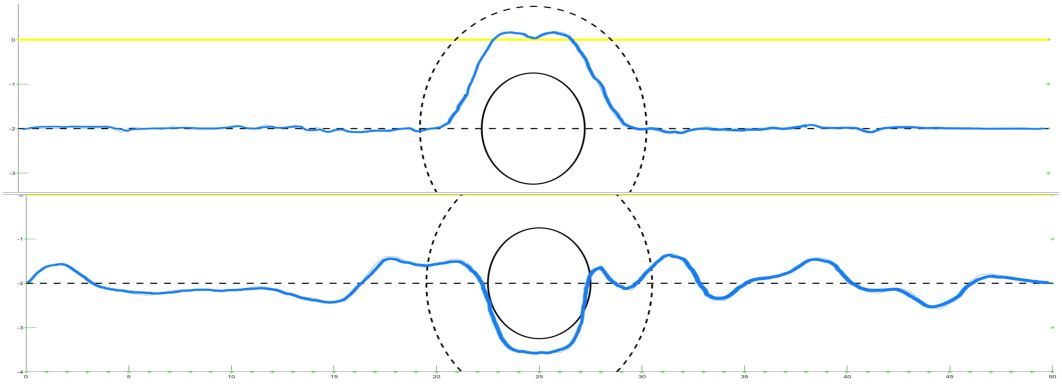

Figure 5: Trail for GA Ver.1

In these trails, generated paths were all suffered from either intersecting with safety ellipse or overturning. Also, initial path is really important. So with this thought, a second version of GA was tried and the result is shown as Fig. 6:

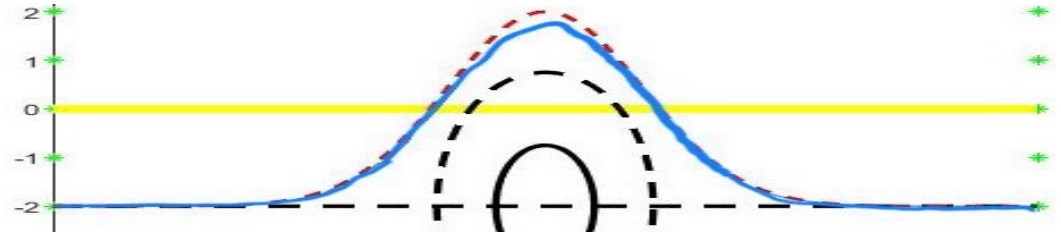

Figure 6: Trail for GA Ver.2

This time, an initial guess of the route was added, it is a smooth path and never contains overturn and have no intersection with the safety ellipse. So, as a path, it will only have cost term of distance to the center lane. Based on this, the cost function was modified and it looks like right part of equation 3.

So, by using these cost terms, the distance of the generated path to the center lane was trying to be minimized, and when it came to overturn or intersect with the safety ellipse, the cost would be extremely large and thus it would never be the optimal solution.

Running this new version of GA, it still suffered from divergence. With the population of 400 and iteration of 400, an average of 5 minutes using version 1 and 3 minutes using version 2 was too long to achieve the task of path planning. So adding more population or more iteration was not possible in this scenario. Several reasons are proposed why GA was not performing well:

1. Generating new children using cross point is likely to produce two paths that cannot be used. Since the point for next step was not generated from the previous point using vehicle kinematics, overturn can be easily brought into paths when crossing between two different parents.

2. Mutating through random noise may not have good results. Since the case is generating paths that are feasible for vehicles, adding noises into path generating can bring non-smoothness into paths. So many overturns were found in paths generated.

3. GA is a gradient free method, which means that it needs to sample enough points to find the smaller ones, and pick them as parents for next step. Unlike gradient descent, GA cannot apply much penalty when gradient is large, so it is really slow to get into the local minimum. And due to its gradient free property, it cannot guarantee that costs of new generated paths are smaller than the initial ones. Even the initial ones are not optimal. Therefore this is not a good method when we can calculate the gradient of the cost function.

And this method has many other disadvantages, such as: 1. it is really slow without parallel programming [2]; 2. when using large population, it will consume large computational power and memory compare to other methods.

4.3 IPM

Different from iLQR and GA mentioned above, IPM computed the optima locally, in other word, it optimized each step instead of the global path. The cost function for each step did not sum up the whole path, which was

$$\begin{aligned} &\underset{\mathbf{u}}{\text{minimize}} && J = C_k^{acc} + C_k^{yawr} + C_k^{ref} + C_k^{vel}, \\ &\text{with respect to} && \mathbf{u} = \begin{bmatrix} \dot{v} \\ \dot{\theta} \end{bmatrix}, \end{aligned} \tag{4}$$

but was subject to the same constraints. The iteration for each step and its previous step followed the vehicle kinematics constraint and is listed in Appendix Section A.

The MATLAB built-in function *fmincon* and the algorithm specified as *interior-point* was used to calculate the result.

The first version of the code would generate trajectories that are not smooth enough, as shown in figure 7. It is because the yaw rate was too high when it turned to its reference path from the left path. The vehicle had to pullback the high yaw rate and make the non-smooth turn, or it would hit the obstacle.

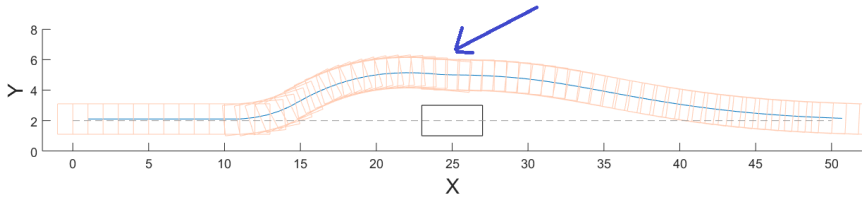

Figure 7: An non-smooth path generated by first version of IPM. The blue arrow points out where the path is not good enough.

To improve the solution, the weight w_{yawr} of yaw rate cost term in the cost function was added to decrease the changes in vehicle yaw. The second version of IPM could generate trajectories that were smooth enough for real implementation. A typical trajectory with vehicle poses at each step is shown in figure 8. It lead the vehicle to the left of the obstacle, and then gently backed to the reference path. The overall solution had a good balance between aggressiveness and submissiveness.

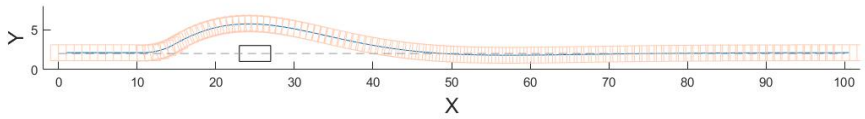

Figure 8: Trail for IPM

IPM was robust for different obstacle positions. But limitations also exist that it could stuck in local minima in typical situations. Further analysis on its sensitivity is conducted in section 5.

5 Sensitivity Analysis

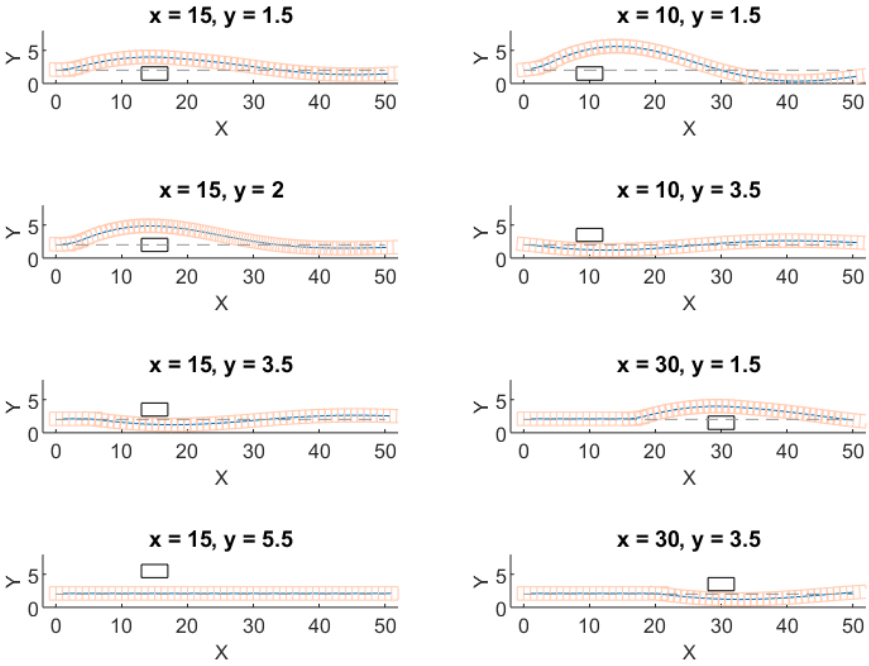

Figure 9: Sensitivity analysis of IPM. Obstacle position is titled in each sub-figure.

The sensitivity analysis was mainly performed with IPM. There existed a major reason why the sensitivity analysis for iLQR was not quite feasible over a wide distribution of obstacle placements. As mentioned above in its own section, iLQR suffered a problem of initialization. With the same set of initialization, resulted paths from a small distribution of obstacles were reasonable. However, if the obstacle position were to change rapidly, initialization would have be drastically different, which would convert sensitivity issue to robustness problem. On the other hand, when the obstacle was kept steady and initial guesses were to be disturbed by a small noise, then the algorithm could still be feasible. However, if the initial guess were to be changed from for instance a polynomial to a straight line, then the algorithm would fail. Figure 11 supports this claim.

Since GA failed drastically on this problem, the sensitivity analysis was not conducted.

Results from IPM are listed and described below.

While obstacle position changed, IPM remained robust for most situations. The left column in figure 9 shows how the generated path changed when y position of obstacle differed. The vehicle would be directed to either left or right side of the obstacle depending on their relative y position - if $y_{vehicle} > y_{obs}$, the vehicle would turn left, and vice versa. The right column in figure 9 shows the change in path when x position of obstacle changed. If the obstacle was too close to the vehicle, as shown in first figure, a larger oscillation would be achieved after passing the obstacle. A recommended relative distance in x position would be larger than 12 meters.

Though IPM worked for most situations, figure 10 shows the possibility that IPM stuck in a local optimun, which generated a path that goes outside the road. Since the method itself has the limitation of being stuck in local minima, an attempt tos solve the problem was not made.

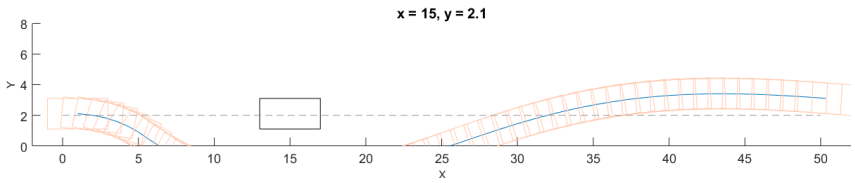

Figure 10: IPM stuck in local optima.

6 Conclusion

The computational time for three methods are listed in Table 2. It can be concluded that iLQR had the most efficient computational speed due to the dynamic programming utilized. GA suffered from the unpredictable stochastic process. IPM had the balance of computational speed and robustness which was discussed above.

Table 2: Computational Time for Three Different Methods

Computation Time			
Algorithm	iLQR	GA	IPM
Time (s)	2.07	297	20.0

Based on experiments conducted, the advantage of iLQR is that it can inherently incorporate vehicle dynamics into the algorithm. Yet it is sensitive to initial guess, meaning "bad" initialization can lead to local optimal which is unfeasible in practice. For GA, it should be faster than searching the entire feasible domain theoretically. However, in practice it performs horrendously. Possible reasons can be cost functions should be fine tuned to make the algorithm work efficiently. And IPM is the most robust algorithm among all three. It can converge to a solution with few iterations. Yet as many other optimization algorithms, it can get stuck in local optimal is the problem contains to much local minima.

References

[1] J. Chen, W. Zhan, M. Tomizuka. Constrained iterative lqr for on-road autonomous driving motion planning. *IEEE International Conference on Intelligent Transportation Systems*, 2017.

[2] V. Roberge, M. Tarbouchi, G. Labonte. Fast genetic algorithm path planner for fixed-wing military uav using gpu. *IEEE Transactions on Aerospace and Electronic Systems*, 2018.

A Problem Statement Terms Explanation

where:

$$C_k^{acc} = w_{acc}\mathbf{u}_k^T \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \mathbf{u}_k$$

- acceleration cost term

$$C_k^{yawr} = w_{yawr}\mathbf{u}_k^T \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{u}_k$$

- yaw rate cost term

$$C_k^{ref} = w_{ref}\left(\mathbf{x}_k - \mathbf{x}_k^r\right)^T \mathbf{Q}_{ref}\left(\mathbf{x}_k - \mathbf{x}_k^r\right)$$

- reference tracking cost term

$$C_k^{vel} = w_{vel}\left(\mathbf{x}_k - \mathbf{x}_k^r\right)^T \mathbf{Q}_{vel}\left(\mathbf{x}_k - \mathbf{x}_k^r\right)$$

- velocity tracking cost term

$$\bar{\dot{\theta}} = \frac{\mathbf{x}(3)}{L}\tan\bar{\delta}$$

- maximum allowed yaw rate given velocity

$$\underline{\dot{\theta}} = \frac{\mathbf{x}(3)}{L}\tan\underline{\delta}$$

- minimum allowed yaw rate given velocity

$$P = \begin{bmatrix} \frac{1}{a^2} & & & \\ & \frac{1}{b^2} & & \\ & & 0 & \\ & & & 0 \end{bmatrix}$$

- obstacle avoidance matrix based on ellipse equation

$$a = L_o + v_o t_{safe} + d_{safe}$$

- obstacle ellipse major axis length

$$b = W_o + d_{safe}$$

- obstacle ellipse minor axis length

$$\mathbf{x}^f = \mathbf{x} + \begin{bmatrix} \cos\mathbf{x}(3)L & \sin\mathbf{x}(3)L & 0 & 0 \end{bmatrix}^T$$

- state of front vehicle center in relation to state of the vehicle

$$\begin{aligned} \mathbf{x}_{k+1} &= f(\mathbf{x}_k, \mathbf{u}_k) \\ &= \mathbf{x}_k + \begin{bmatrix} \cos\theta_k\left(v_k dt + \frac{\dot{v_k}}{2}dt^2\right) \\ \sin\theta_k\left(v_k dt + \frac{\dot{v_k}}{2}dt^2\right) \\ \dot{v_k}dt \\ \dot{\theta_k}dt \end{bmatrix} \end{aligned}$$

- discrete system dynamics

B Model Parameters

Table 3: Model Parameters

Symbol	Description	Base Value	Sensitivity Values	Units
a_{max}	maximum acceleration	4	-	m/s^2
a_{max}	maximum deceleration	-6	-	m/s^2
$\bar{\delta}$	leftmost steering angle	$\frac{\pi}{6}$	-	rad
$\underline{\delta}$	rightmost steering angle	$-\frac{\pi}{6}$	-	rad
L_f	distance of center of front axle to center of gravity	1.5	-	m
L_r	distance of center of rear axle to center of gravity	1.5	-	m
v_o	relative velocity difference between obstacle and the vehicle	10	-	m/s
L_o	max obstacle length	4.5	$\{4, 5\}$	m
W_o	max obstacle width	3.5	$\{3, 4\}$	m
x_o	obstacle center x-coord	25	$\{15, 30\}$	m
y_o	obstacle center y-coord	0	$\{1.5, 2, 3.5, 5.5\}$	m
t_{safe}	safety time margin	1.5	-	s
s_{safe}	safety space margin	2.0	-	m
r_{rad}	vehicle wrapping circle radius	1.7493	-	m

C Additional Resulted Figures

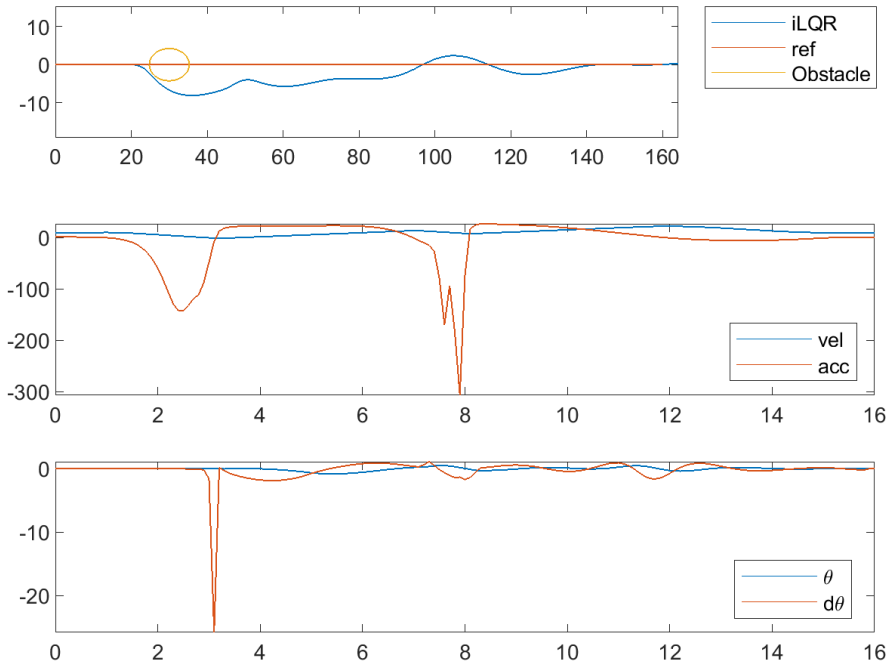

Figure 11: iLQR Result with Bad Initialization

D iLQR Algorithm Derivation

The equality constraint (vehicle dynamics) needed to be linearized about every operation point and inequality constraints needed to be coped into the cost function. The barrier function chosen was in the exponential form as shown in Equation 5

$$b_k^x = q_1 \exp(q_2 g_k^x) \tag{5}$$

Further quadritization was required because each term in the quadratic function needed to be quadratized to fit in the LQR formulation[1].

$$b_k^x(x_k + \delta x_k) \approx \delta x_k^T \nabla^2 b_k^x(x_k) \delta x_k + \delta x_k^T \nabla b_k^x(x_k) + b_k^x(x_k) \tag{6}$$

An example is provided below to illustrate the reasoning behind this choice. With the inequality constraint $g_1 = a - a_{max} \leq 0$, when $a < a_{max}$, $g_1 < 0$, $b = q_1 \exp(q_2 g_1)$ is small given positive q_1, q_2. When $a > a_{max}$, $g_1 > 0$, $b = q_1 \exp(q_2 g_1)$ is large given well tuned positive q_1, q_2. As a result, the exponential barrier function was a suitable choice.

Instead of solving an algebraic Riccati Equation, dynamic programming was proposed to solve this LQR problem. Two terms needed definitions before introducing the algorithm: cost-to-go (V) and a Q-value function (Q).

The cost-to-go function was defined and shown in Equation 7. Clearly the definition was time dependent. To eliminate the time dependency, V(x) was calculated backwards from the last state because control inputs were zeros at the last state as shown in Equation 8, where l_f represented the final step cost and total cost J was the sum of cost at each step l. Working forward, the cost-to-go

could be converted to a purely state and control sequence dependent function as shown in Equation 9.

$$V_t(\mathbf{x}) = \min_{\mathbf{u}_t} J_t(\mathbf{x}, \mathbf{u}_t) \tag{7}$$

$$V(\mathbf{x}_N) = l_f(\mathbf{x}_N) \tag{8}$$

$$V(\mathbf{x}) = \min_{\mathbf{u}} \left[l(\mathbf{x}, \mathbf{u}) + V(f(\mathbf{x}, \mathbf{u})) \right] \tag{9}$$

The Q-value function defined represented the change in value function as a result of small perturbation as shown in Equation 10

$$Q(\delta\mathbf{x}, \delta\mathbf{u}) = l(\mathbf{x} + \delta\mathbf{x}, \mathbf{u} + \delta\mathbf{u}) + V(f(\mathbf{x} + \delta\mathbf{x}, \mathbf{u} + \delta\mathbf{u})) \tag{10}$$

Then, starting from the last step, at each step k, two things that would be used to update control policy were calculated as shown in Equation 11 and Equation 12, the first derivative of V (V_x) and the second derivative of V (V_{xx}) were updated as shown in Equation 13 and Equation 14. One thing to note is that $\mathbf{Q}_{uu} = \frac{\partial^2 Q}{\partial u^2}$, $\mathbf{Q}_{ux} = \frac{\partial^2 Q}{\partial u \partial x}$, $\mathbf{Q}_x = \frac{\partial Q}{\partial x}$.

$$\mathbf{k} = -\mathbf{Q}_{uu}^{-1}\mathbf{Q}_u \tag{11}$$

$$\mathbf{K} = -\mathbf{Q}_{uu}^{-1}\mathbf{Q}_{ux} \tag{12}$$

$$\mathbf{V}_x = \mathbf{Q}_x - \mathbf{K}^T\mathbf{Q}_{uu}^{-1}\mathbf{k} \tag{13}$$

$$\mathbf{V}_{xx} = \mathbf{Q}_{xx} - \mathbf{K}^T\mathbf{Q}_{uu}^{-1}\mathbf{K} \tag{14}$$

Once all steps were covered, a forward propagation was performed to update optimal control policy and new resulted trajectory as shown in Equation 16 and Equation 17. The first step always had the same initial state as defined ini Equation 15.

$$\hat{\mathbf{x}}_0 = \mathbf{x}_0 \tag{15}$$

$$\hat{\mathbf{u}}_k = \mathbf{u}_k + \mathbf{k}_k + \mathbf{K}_k(\hat{\mathbf{x}}_k - \mathbf{x}_k) \tag{16}$$

$$\hat{\mathbf{x}}_{k+1} = f(\hat{\mathbf{x}}_k, \hat{\mathbf{u}}_k) \tag{17}$$

$$g5 : 1 - \Delta_k^T \mathbf{P} \Delta_k \leq 0 \tag{18}$$

$$g6 : 1 - \Delta_{k,front}^T \mathbf{P} \Delta_{k,front} \leq 0 \tag{19}$$

$$\Delta_k = \begin{bmatrix} (\mathbf{x}_k - \mathbf{x}_{obs,k})^T \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \end{bmatrix}^T \\ (\mathbf{x}_k - \mathbf{x}_{obs,k})^T \begin{bmatrix} \sin\theta & \cos\theta & 0 & 0 \end{bmatrix}^T \\ 0 \\ 0 \end{bmatrix} \tag{20}$$

$$\Delta_{k,front} = \begin{bmatrix} (\mathbf{x}_{k,front} - \mathbf{x}_{obs,k})^T \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \end{bmatrix}^T \\ (\mathbf{x}_{k,front} - \mathbf{x}_{obs,k})^T \begin{bmatrix} \sin\theta & \cos\theta & 0 & 0 \end{bmatrix}^T \\ 0 \\ 0 \end{bmatrix} \tag{21}$$

$$\underset{\mathbf{x},\mathbf{u}}{\text{minimize}} \qquad J = \sum_{k=0}^{N} \left(C_k^{acc} + C_k^{yawr} + C_k^{ref} + C_k^{vel} + C_k^{obs} + C_k^{lane} \right) \tag{22}$$