# ECE 346 Final Project: A Robust iLQR-Based Algorithm for Guiding Autonomous Trucks in the MiniCity Environment

Alex Pirola
*M.Eng. Mechanical & Aerospace Engineering*
*Princeton University*
Princeton, NJ USA

Daniel Yang
*B.Sc. Computer Science*
*ETH Zurich*
Zurich, Switzerland

Matthew Collins
*M.Eng. Electrical & Computer Engineering*
*Princeton University*
Princeton, NJ USA

Xiaorun Wu
*A.B. Mathematics*
*Princeton University*
Princeton, NJ USA

Jordan Bowman-Davis
*B.S.E Electrical & Computer Engineering*
*Princeton University*
Princeton, NJ USA

William Makinen
*B.S.E. Electrical & Computer Engineering*
*Princeton University*
Princeton, NJ, USA

Chukwuagoziem Uzoegwu
*B.S.E. Computer Science*
*Princeton University*
Princeton, NJ USA

***Abstract*—In this project, we implemented an iLQR-based algorithm that allowed the self-driving trucks to complete adaptive cruise control and overtaking tasks. This was done by adapting the cost functions provided in previous labs and interpolating reference trajectories and velocities to ensure higher efficacy and robustness of the algorithm. These iLQR adaptations were tested by implementing them on the autonomous trucks in the MiniCity environment. The algorithm adaptations showed excellent performance on these tasks, as was seen during the in-class demonstration. This work is described here as well as discussions on shortcomings of the implementations and future experiments that could further boost the performance of autonomous navigation.**

## I. Introduction

The purpose of this project was to apply the concepts learned in ECE346 this semester to the autonomous trucks and the MiniCity environment introduced during lab. Three tasks were to be completed: two core tasks including adaptive cruise control (ACC) and overtaking, and a third stretch task. The ACC task required a client truck to autonomously follow a host truck that was manually controlled by the user while the overtaking task required the client truck to autonomously pass the lead car. The stretch task chosen was platooning, which was an extension of ACC, in which three trucks were required to drive together while following a similar pattern, with the lead car being controlled manually. Two other stretch tasks were also explored; negotiating an intersection and parallel parking. Although they were ultimately not implemented due to time constraints, our theoretical explorations of them as well as some simulations are laid out here for completeness.

## II. Core Task: Adaptive Cruise Control

### A. Goal

The goal of the Adaptive Cruise Control (ACC) task is to follow a vehicle along its path and to adjust the speed based on the leading vehicle while keeping a safe distance. For simplicity of the project, the following vehicle receives the full state estimation of the leading vehicle through ROS, from which we can obtain the current state

$$x = [\mathrm{x}, \mathrm{y}, v, \psi]$$

of the leading vehicle with position $(\mathrm{x}, \mathrm{y})$, velocity $v$ and heading angle $\psi$.

### B. Naive Approach

To obtain an initial working baseline, we first try to implement ACC with minimal effort by using the code base provided to us. The idea is to record the received state estimations of the leading vehicle in a list and to fit a continuous spline to this recorded, discrete $(x, y)$-trajectory using the **pySpline** package [1]. Since the given code for a **Model Predictive Controller (MPC)** [1] based on **iterative Linear Quadratic Regulator (iLQR)** is already able to work with splines as its reference trajectory, we can minimize the amount ofadditional code that needs to be written and reduce the chances of introducing unintentional bugs.

The original usecase of the given MPC is to let a vehicle to follow a fixed reference trajectory along a static track. However, in our case, the reference trajectory is constructed over time based on the past motion of the leading vehicle. This results into several drawbacks for this method making it difficult to implement.

[1] https://github.com/mdolab/pyspline

- First, it is not always possible to fit a spline to the recorded trajectory of the leading vehicle. Assuming the leading vehicle does not move, all recorded positions are placed in the same location. Similarly, if the leading vehicle does not follow a "well-behaved" trajectory (e.g. driving back and forth), pySpline is also not able to fit a suitable spline to the given trajectory.
- Second, this method requires us to fit a spline for each iLQR computation, which can be computationally expensive and not suitable for real-time planning.

We tried to address the first problem by ending the iLQR planning thread in the MPC whenever the spline could not be constructed and restarting it again when the trajectory was updated and a new spline could be fitted. This naturally prevents the following vehicle to imitate the "bad behavior" of the leading vehicle. However, the method was still unstable regarding runtime errors, in particular during the initialization phase, and the performance was insufficient. In addition, the overall complexity of the tools used made it more difficult to debug and resulted in strange behaviors for the following car.

### C. Simpler Cost Function 1

To improve the understanding of our controller and eliminate inexplicable behaviors, we aim to simplify the cost function provided to iLQR. To overcome the issues of using splines, we work directly with the discrete trajectory as our reference path. However, this hinders us from using costs like the contour error or lap progress, which require functions like computing the closest point or slope along the trajectory. As a substitute, we came up with a cost function, which uses the distance from a given state $x_t$ over the closest point $\tilde{x}_c$ on the trajectory of the lead vehicle along the trajectory $\{\tilde{x}_i\}_{i=1}^{M}$ as the cost for each state

$$c_{state}(x_t) = \|x_t - \tilde{x}_c\|^2 + \sum_{i=c+1}^{M} \|\tilde{x}_i - \tilde{x}_{i-1}\|$$
$$\text{with } c = \underset{i=1,\dots,M}{\operatorname{argmin}} \|\tilde{x}_i - x_t\|$$

with index $c$ of the closest data point to $x_t$. In addition, we also regularize our control actions using two tuneable parameters $w_{accel}$ and $w_{delta}$

$$c_{control}(u_t) = w_{accel} \cdot u_t[0] + w_{delta} \cdot u_t[1]$$

The idea is to provide incentive for iLQR to find a plan which both sticks to the given reference trajectory and increases its progress along the reference trajectory. One subtle problem is that the closest point $\tilde{x}_c$ on the trajectory to a given state $x_t$ can lie behind instead of in front of $x_t$, such that minimizing the distance between these two points result into a negative progress. Thus, for a given state $x_t$ we find the closest midpoint between two succeeding points on the reference trajectory instead and minimize the distance between $x_t$ and the latter of the two succeeding points. In that way, $x_t$ is always moved towards positive progress when minimizing the cost.

Since iLQR does not depend on the construction of splines anymore, this approach was more reliable in producing at least some plans. However, the overall performance was still not satisfying and unstable. One possible, but unverified reason might be that the closest (mid-)point for each state of the iLQR plan can change in each iLQR iteration, which could have a negative impact on the stability of iLQR.

### D. Simpler Cost Function 2

To simplify the cost function further, we fix the assignment of each planned state $x_t$ to a point on the reference trajectory $\tilde{x}_t$ instead of finding the closest one. Thus, we are asking iLQR to find the sequence of control actions such that the planned trajectory exactly matches the reference trajectory "statewise". The corresponding cost function for each state is

$$c_{state}(x_t) = \|x_t - \tilde{x}_t\|^2$$

Our initial approach was to record the state estimation of the leading vehicle with a sampling rate corresponding to the iLQR time step size $\Delta t$ and then supply the first $N$ recorded states to iLQR. We also remove all points along the recorded trajectory that come before the closest point to the current position of the vehicle such that iLQR matches the correct part of the past trajectory.

### E. Interpolation of reference trajectory

Another approach is to sample the reference trajectory again more densely and independent of the iLQR time step size, but then to compute the iLQR reference points by interpolating between the points on the discrete trajectory. The spatial separation $\Delta s$ between each of the reference points can be computed based on some reference velocity $v_{ref}$ and the iLQR time step size $\Delta t$

$$\Delta s = v_{ref} \cdot \Delta t$$

For example, running iLQR with a time horizon of $T = 2s$ and $N = 11$ steps, the time step size of iLQR is $\Delta t = \frac{T}{N-1} = 0.2s$. Hence, computing reference points for iLQR with a reference velocity of $v_{ref} = 1\frac{m}{s}$ amounts to finding points on the reference trajectory which are separated by $\Delta s = 0.2m$.

This also means that we implicitly encode some reference velocity in the reference points provided to iLQR. The closer the reference points are to each other, the lower the reference velocity and vice versa. The advantage of this approach is that it allows us to specify some reference velocity relative to the current velocity of the leading vehicle and based on the distance $s_{front}$ to the leading vehicle and some safety distance $s_{safety}$. For simplicity, we use the following function to set the reference velocity:

$$v_{ref} = \text{clip}(v_{front} + (s_{front} - s_{safety}), v_{min}, v_{max})$$

This increases or decreases the velocity of the following vehicle by how much the current distance between the leading and following vehicles deviates from the safety distance and set this as our reference velocity.

We also improved our results by taking the current position of the following vehicle into account. By computing the

reference points starting from its current position instead of the first position of the recorded trajectory, we are able to obtain useful reference points even if the following vehicle does not exactly follow the past trajectory of the leading vehicle.

Another idea was to interpolate and penalize deviations in the heading angle at each reference point, since we do not only want to match the exact positions, but also the orientation, such that we can continue to follow the trajectory in the future. However, this turned out to be not ideal in general, since iLQR still tries to match the orientation even though it might cannot match the positions exactly. In this case, it is undesired to have this specific orientation but at another position.

### F. Interpolation of Reference Velocity

We achieved another improvement in our results by taking the current velocity of the following vehicle into account and interpolating between the current and reference velocities when computing the reference points. Initially, we first applied simple linear interpolation. Since we still saw large deviations between the computed reference points and planned iLQR states for a few points, we instead used B-splines with derivatives set to zero for a smoother interpolation of the velocity at the beginning and end.

### G. Understanding of iLQR

Using this approach of minimizing the deviations between the reference trajectory and trajectory planned by iLQR "state-wise", we only use iLQR to convert the desired trajectory into a matching sequence of control actions, instead of providing some cost function to iLQR and hoping that iLQR finds a good trajectory itself.

We observed that it is important to provide a feasible reference trajectory to iLQR, in order for it to produce reasonable control inputs. If a non-feasible reference trajectory is provided, it is possible that the calculated cost is of a similar magnitude to that of a feasible, desired trajectory, resulting in the infeasible trajectory being chosen. In these cases, we experienced that iLQR would compute strange plans, causing our vehicle to behave in an undesired way. For example, providing a reference trajectory without interpolation between the current and reference velocity corresponds to asking iLQR to apply infinite acceleration, which it can never reach.

In addition, we made the observation that reducing the number of iLQR planning steps $N$ (e.g. from 11 to 7) has a improves the stability of the motion execution. This might regularize the control itself to fewer over-corrections.

### H. Future Work

In this section, we offer some brief discussion on some potential further optimizations that could be implemented to optimize the agent performance. We will have some further discussions in chapter VII.

First as we mentioned in the previous section, we see that there's mild delays relating to the reaction time for the truck to catch up with the truck in front–we observe if the leading truck suddenly accelerates or is too far away from the truck controlled by the algorithm, then the following truck can lose its way. To solve this, we might need to tune the functions, hence to better set reference velocity based on deviation from safety distance (we used a linear interpolation, however higher-order approximations may prove more effective).

Also, we can see that the truck tends to have less awareness towards the surrounding environments–while the truck follows closely to the car in front of it, it is unaware of the surrounding environments. This could be solved by making the truck more aware of its surroundings–this could be done by tuning the objective function to stay more closely to the desired trajectory or implementing object detection.

## III. Core Task: Overtaking

### A. Goal

The goal of overtaking is to overtake a lead cruising vehicle with a following vehicle, with the behavior of the lead vehicle being unknown due to it being manually controlled.

### B. Track and iLQR

For this task, we use the two lanes of the outer-loop for performing the overtaking. The center line of the track between the two lanes is given in a CSV file. To compute the trajectory on the right or left lane, we apply the track offset by computing the vector perpendicular to the vector connecting the previous and the point after the current point. The planning part uses the same methods as for ACC and computes reference points over the reference trajectory.

### C. Implementation of Cruising Vehicle

The cruising vehicle drives along the outer loop and changes its lane every $2s$ with some fixed probability. The velocity is hardcoded to some low number, such that the following car is able to catch up.

### D. Implementation of Overtaking Vehicle

We assume that the cruising vehicle has constant velocity with a constant heading angle and we use ellipsoid objects to add the forward-reachable set (FRS) [2] of the cruising vehicle to the cost of the overtaking vehicle. We use a **hybrid system** to encode the following two modes with their respective dynamics:

- **Cruise:** vehicle cruising on right track with medium velocity
- **Overtake:** vehicle overtaking on left track with high velocity

We define the parameters $d_{overtake}$, below which the overtaking vehicle starts overtaking the cruising vehicle, and $d_{mergeback}$, above which is a lower bound of the distance to merge back. To switch between the two modes, we use the following guard conditions

$$G_{\text{cruise}\to\text{overtake}} = \{x \in \mathbb{R}^4 \mid \text{status}(x) = 1, \text{dist}(x) < d_{overtake}\}$$
$$G_{\text{overtake}\to\text{cruise}} = \{x \in \mathbb{R}^4 \mid \text{status}(x) = -1, \text{dist}(x) > d_{mergeback}\} \cup \{x \in \mathbb{R}^4 \mid \text{status}(x) = 1, \text{dist}(x) > d_{overtake}\}$$

where status indicates whether $x$ is in front of $x_{front}$ or not.

### E. Future work

The environment where the overtaking occurred was highly constrained since it only involved two cars on the road at a time. Therefore, a logical extension would be to expand the environment to include multiple vehicles. Assuming we maintain the same model of a singular overtaking vehicle and multiple cruising vehicles, and assuming that each of the cruising vehicles abides by the same constraint of the current cruising vehicle, the method for doing this would be very similar to our current implementation. We can utilize the forward-reachable set of each of the cruising vehicles when computing the cost of the singular overtaking vehicle.

A more interesting extension would be to emulate a situation where we have multiple vehicles that are attempting to overtake a singular cruising vehicle or multiple cruising vehicles. Ideally, each of the overtaking vehicles would have the incentive to avoid a crash while trying to achieve the goal of overtaking. This would involve a form of multi-agent planning.

## IV. Stretch Task: Multi-Vehicle Platooning

### A. Goal

The goal of Multi-Vehicle Platooning is to have several vehicles following a lead vehicle, matching its trajectory and speed.

### B. 3-car extension of ACC

The intention of investing a large amount of effort into optimizing the performance for ACC is to reuse its methods for platooning. Our first approach was to allow the two following cars to follow the trajectory of the leading car, but only with different safety distances. The problem however was that the two following cars could not communicate with each other, which resulted in conflicts in cases one of them became immobile.

Our second approach was to have the second car follow the trajectory of the first leading car and the third car follow the trajectory of the second car. At the beginning, we were first concerned about how the noise and inaccuracy might propagate over the second to the third car. In the end, the performance of this approach was more convincing than the one of our former approach, mainly because of the stability of ACC itself and thus this is the solution that was used.

Due to time constraints during the demo period, we were unable to showcase this. Instead, a video of the platooning maneuver can be found here.

### C. Future work

One of the potential future work to try to have even more than 3 cars. In theory, the same system could be used with minimal modification, however additional collision avoidance may become necessary as the chain of cars grows.

## V. Stretch Task: Parallel Parking

### A. Goal

The goal of the parallel parking task was to plan and execute a trajectory to park in a boxed region situated between two vertically parked cars. This was never fully implemented but aid out below is the theoretical framework we developed before deciding to focus on platooning.

### B. Approach: RRT and ILQR

In attacking this problem it was helpful to consider it as two distinct problems: trajectory **planning** and trajectory **tracking**. The first task is planning a trajectory from the agent car's current location to the final parking spot. We accomplished this, in simulation, using Rapidly Exploring Random Trees (RRT) - a search-based algorithm that generates a collision-free, fully connected tree of randomly sample nodes until a path to the goal state is found. This is shown in the figure below.

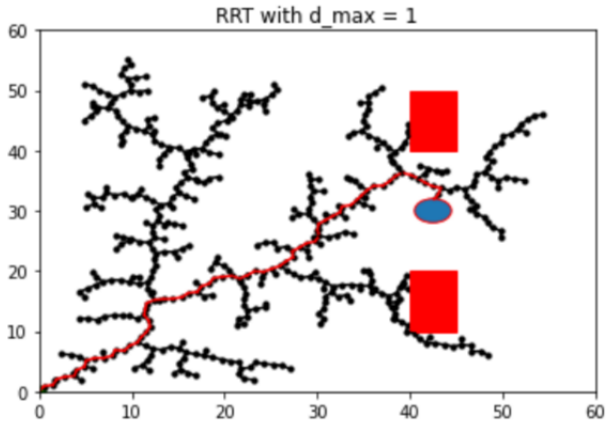


Fig. 1. RRT finds a path from the agent's current state ((0,0) in the above example) to the parking spot. Note that the path does not consider the truck's radius of curvature and that it is non-optimal. It would be ILQR's job to follow this path as closely as possible while also figuring out some way to reverse into the parking spot

Our RRT implementation does not take into account the vehicle dynamics, so it is up to iLQR to figure out a way to track this reference as closely as possible, given the truck's turn radius. Using iLQR, the agent would have to find a trajectory that minimizes the cost of deviating from the reference while still ensuring that it reaches its goal state, specified by a an (x,y) coordinate and a heading angle. There are several path-planning algorithms we could have used to generate a reference trajectory that we would ultimately feed into iLQR, such as Djikstra [3] or A* [4] or Kinematic-RRT [6] (which accounts for vehicle dynamics). However, no matter how good the trajectory planning stage is, it would still require significant tuning of costs to get the truck to perform the desired behavior, which entails pulling up to the front parked car and reversing into the spot. This led us to consider a different approach.

### C. Approach: ILQR + Geometry

An alternative method to solving the parallel parking problem, which slightly alleviates some of the burden on iLQR to find a fairly complex path, is a series of geometric calculations. Essentially, once the car is vertically aligned with the parking spot (though not yet inside it), the act of parking can be broken down into three distinct maneuvers: pull forward, reverse at steering angle 1, reverse at steering angle 2. Geometrically, this corresponds to a straight line and then two circles connected at a tangent point. As it turns out, if you assume an Ackermann steering model [5], the parking trajectory combining all three maneuvers can be fully defined just from the lateral distance between the two cars, $X_c$, as shown in the figure below.

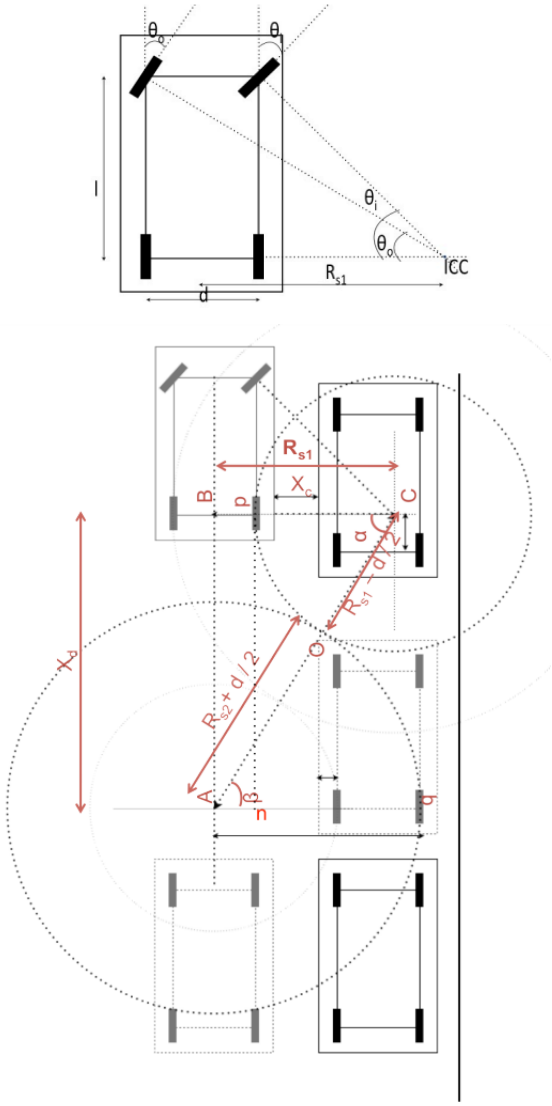


Fig. 2. Purely geometric arguments using an Ackermann steering model can generate the entire parallel parking maneuver.

Then, the task of iLQR is simplified to merely aligning the agent truck vertically next to the desire parking spot. Afterwards, a pre-planned "parallel parking" sequence can take over. Unfortunately, we were not able to implement this behavior on the truck, but our simulation results are promising and given more time, this is another stretch task we would've attempted to integrate with the hardware.

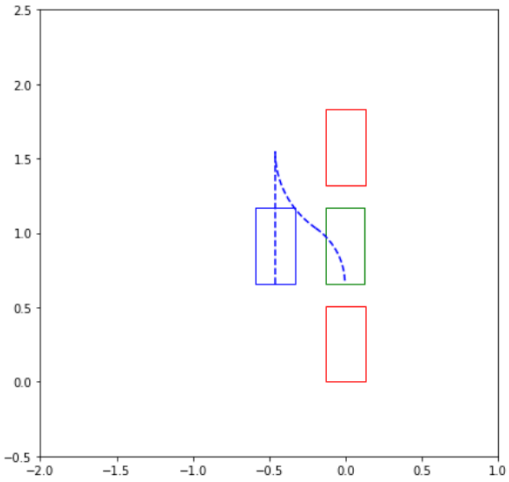


Fig. 3. Simulated results of parallel parking trajectory. The original position of the agent truck is represented by the blue rectangle, while the static trucks are in red. The green rectangle indicates the final goal state. Note that this approach alleviates the pressure on ILQR by simplifying its trajectory. Instead of needing to completely figure out how to reverse into the parking spot, ILQR can simply navigate to the blue starting position and execute the "parallel parking" control sequence.

## VI. Stretch Task: Negotiating an Intersection

### A. Goal

The intersection of focus for this task was exiting the Minicity's roundabout to merge onto the inner ring in which traffic flows clockwise. The goal is for the car seeking to merge onto the inner ring to infer the intention of a (human-controlled) car driving on the inner ring, as to whether it is continuing along the inner ring or turning into the roundabout. Again, this was never implemented fully however here is the framework with which we would have done it.

### B. Problem Formulation

Similar to Lab 3, the intersection negotiation is defined by an MDP involving the two cars. The intent of the human-controlled car (Car B) is modelled as a state within the set of goal states $\mathcal{S} := \{\text{continue along inner ring } (G_1), \text{turn into roundabout } (G_2)\}$. Our autonomous vehicle (Car A) can take actions from the action space $\mathcal{A} := \{\text{merge onto inner ring } (a_1), \text{stop at intersection } (a_2)\}$. Figure 4 gives an overview of the intersection.

The transition probability matrices were adapted from the values used in Lab 3 in order to align with the MDP formulation for this specific intersection, as seen in Tables I and II.

Similarly, the rewards that were ascribed to this MDP formulation are given by the following:

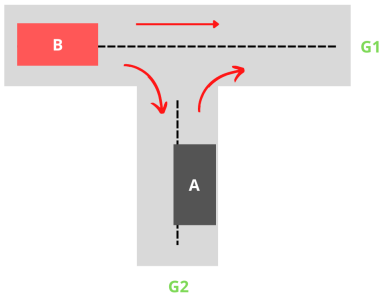


Fig. 4. Overview of Intersection Negotiation

| s\s' | $G_1$ | $G_2$ |
|---|---|---|
| $G_1$ | 0.8 | 0.2 |
| $G_2$ | 0.1 | 0.9 |

TABLE I
TRANSITION PROBABILITY MATRIX FOR ACTION $a_1$

| s\s' | $G_1$ | $G_2$ |
|---|---|---|
| $G_1$ | 0.95 | 0.05 |
| $G_2$ | 0.5 | 0.5 |

TABLE II
TRANSITION PROBABILITY MATRIX FOR ACTION $a_2$

| | |
|---|---|
| $R(G_1, a_1)$ | -100 |
| $R(G_2, a_1)$ | 10 |
| $R(G_1, a_2)$ | -1 |
| $R(G_2, a_2)$ | -10 |

TABLE III
REWARDS FOR THE INTERSECTION MDP IN FIGURE 4

### C. Results

Running Value Iteration in a similar fashion to Lab 3 yielded the values and optimal policy $\pi^*$ shown in Figure 5. As a sanity check, it makes sense that if the goal state is $G_1$ that the optimal policy is $a_2$ (i.e. to stop), and vice versa for goal state $G_2$.

```
Value iteration: 76 iterations
V_star:  [11.05495009 57.86346066]
pi_star:  [1 0]
```

Fig. 5. Value iteration for intersection MDP

Furthermore, Figure 6 reveals that our autonomous vehicle, Car A, will choose to merge onto the inner ring if it's 80% confident in its belief that Car B's goal state is to pull off the inner ring into the roundabout ($G_2$). Once Car A's belief drops below 80%, it will elect to stop at the intersection instead.

### D. Future Work

With more time, it would be exciting to implement the intersection on the physical trucks. To do so, Car A would

```
Belief: [0.00, 1.00]    Action to take:  forward
Belief: [0.10, 0.90]    Action to take:  forward
Belief: [0.20, 0.80]    Action to take:  forward
Belief: [0.30, 0.70]    Action to take:  stop
Belief: [0.40, 0.60]    Action to take:  stop
Belief: [0.50, 0.50]    Action to take:  stop
Belief: [0.60, 0.40]    Action to take:  stop
Belief: [0.70, 0.30]    Action to take:  stop
Belief: [0.80, 0.20]    Action to take:  stop
Belief: [0.90, 0.10]    Action to take:  stop
Belief: [1.00, 0.00]    Action to take:  stop
```

Fig. 6. Optimal action based on belief of goal state

be required to execute Bayes rule for human actions, given by the following formula:

$$P(\theta|x, u_H) = \frac{P(u_H|x;\theta)P(\theta)}{P(u_H|x)}$$

in which $\theta$ is the intention of Car B that Car A is trying to infer given Car B's state $x$ and action $u_H$. The state $x$ of Car B could be communicated to Car A over ROS, while the probability of Car B taking a certain action $u_H$ given state $x$ could be calculated using the assumption that Car B is a Boltzman rational agent. This implies that Car A knows that Car B will choose actions in the following way:

$$P(u_H|x) = \frac{e^{\beta Q(x,u_H)}}{\sum_{\tilde{a}} e^{\beta Q(x,\tilde{a})}}$$

In order to fully implement this on hardware, the Q-function for Car B would have to be estimated, likely by taking into consideration Car B's position (and motion based on action taken) relative to a trajectory that would lead Car B to state $s$. Naturally, more time would be required to implement these methods on the physical vehicles; nevertheless, for the purposes of this project, it was beneficial to identify what future next steps would be.

## VII. DISCUSSION AND CONCLUSION

In this section, we offer a brief discussion about the results obtained so far, and state our main highlights of the results as well as the take-aways. Then we making some remarks about some of the possible further improvements in general, in connection with our next experiments to be tried on.

### A. ACC

For Adaptive Cruise Control, we see that in general, the car follows closely to the trajectory made by the car in front of it, so that it was performing an excellent job.

Two main problems we see in the previous section is that: a) it's not delay-proof to sudden actions. b) it was not sufficiently aware of the local environments. All those has caused the car sometimes to lose track, or derail from the front car.

To solve this problem, we need quicker reaction to sudden change of the speed/reaction–hence instead of a linear approximations, we might need higher-order ones to better keep track of the trajectory.

### B. Overtaking

For the overtaking task, we see that in general, the rear car was able to overtake the front car relatively smoothly. However, as we see in the previous case, that the rear car was not sufficiently aware of its surrounding environment. This caused the cars to derail significantly from the front car, either colliding with the road barriers or with other objects in the road.

### C. Multi-Vehicle Platooning

For multi-vehicle platooning, we see that the cars were able to follow relatively closely to the guiding car. However, in some cases, the last car of the chain of the vehicle would not follow very closely to the cars in front. This, we attribute towards the chain reaction of delays from the front car, similar to what was seen in the ACC task. And again, the task would become more demanding as the number of cars increases. Improving the overall reliability of the ACC task would significantly improve the performance of the platoon, as each car is individually running the same algorithms, just with a different lead car. Thus, addressing the issues with ACC would also address many of the issues with platooning.

## REFERENCES


[1] A. Afram and F. Janabi-Sharifi. Theory and applications of HVAC control systems – a review of model predictive control (MPC). *Building and Environment*, 72:343–355, Feb. 2014.

[2] A. Bajcsy, S. Bansal, E. Bronstein, V. Tolani, and C. J. Tomlin. An efficient reachability-based framework for provably safe autonomous navigation in unknown environments. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 1758–1765, 2019.

[3] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, Dec. 1959.

[4] M. Likhachev, G. Gordon, and S. Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. In *Proceedings of (NeurIPS) Neural Information Processing Systems*, pages 767 – 774, December 2003.

[5] W. C. Mitchell, A. Staniforth, and I. Scott. Analysis of ackermann steering geometry. In *SAE Technical Paper Series*. SAE International, Dec. 2006.

[6] L. Palmieri and K. O. Arras. A novel RRT extend function for efficient and smooth mobile robot motion planning. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Sept. 2014.