CONNECTED AUTONOMOUS VEHICLE

IMPLEMENTATION OF MOTION PLANNER USING PYTHON AND CARLA SIMULATOR

GROUP 23

Adil Iqbal - 110089140

Nidhi Bhavesh Desai - 110072124

Sharath Srivathsan Ramesh - 110079675

Ezhil Arasu Padmarani Ravichandran - 110087772

SUBMITTED TO: Prof. Ning Zhang

PRESENTATION FLOW

- THE MISSION PLANNING PROBLEM
- HIERARCHICAL MOTION PLANNING
- MISSION PLANNING
- BEHAVIOR PLANNING
- COLLISION CHECKING
- PATH GENERATION AND SELECTION
- CARLA SIMULATOR
- IMPLEMENTATION DIAGRAM
- SIMULATION RESULTS
- CONCLUSION
- CHALLENGES

THE MISSION PLANNING PROBLEM

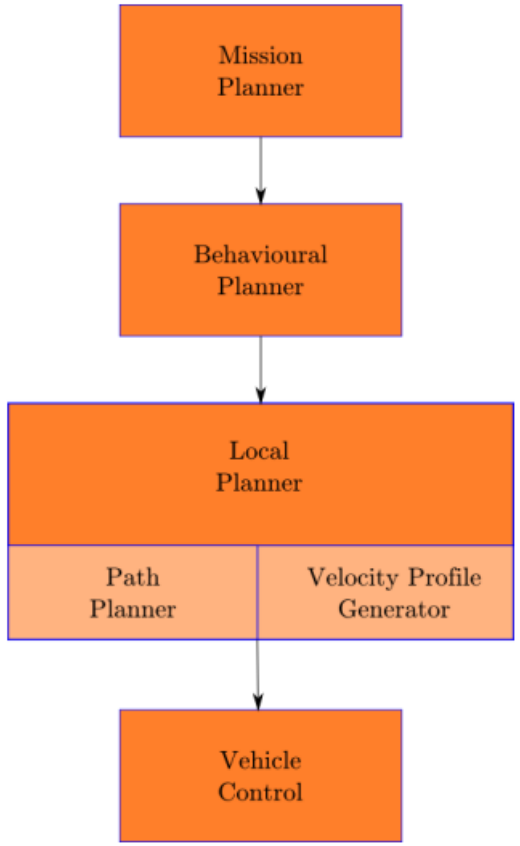
- The mission of self driving cars is to reach from origin to destination safely following traffic rules and minimizing risk due to other obstacles.

- However, the task of mission planning is a complex one, which requires environment perception from several sensors and perceiving the ego vehicle’s state and its surrounding environment in various scenarios.

- Therefore, mission planning is broken into a set of optimization problems known as hierarchical structure.

HIERARCHICAL MOTION PLANNING

- The motion planning process has three key sub-components, which defines the ego vehicle's **prediction, behavior, and trajectory**.[1]
- The hierarchical motion planner is based on the kinematic bicycle model which has the dynamics and kinematics that resembles an ego vehicle in consideration.[2]
- **Mission Planner**: To derive the shortest path from source to destination.
- **Behavioral Planner**: To decide maneuvers to be taken by an ego vehicle.
- **Local Planner**: Generates a collision free path with an appropriate velocity profile.



HIERARCHICAL STRUCTURE

MISSION PLANNING

- Computes the shortest path from point A to point B.
- Implemented using Dijkstra’s or A* algorithm.
- Does not take lower-level details in consideration.
- Outputs a set of future lane level sub-missions such as lane & intersections to be taken.[3]
- Transfers the above constraints and outputs to the lower-level planner.[3]
- The Mission Planner for this project is based on Dijkstra’s algorithm, which is implemented in Python.

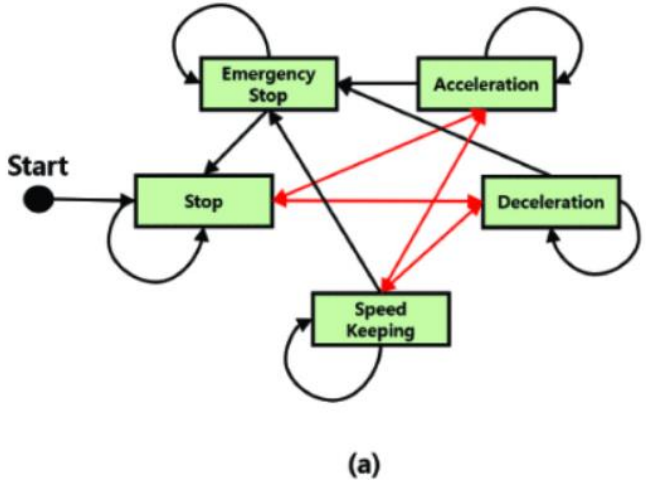
BEHAVIOUR PLANNING

- The Behavioral Planner receives a set of constraints and objectives from the Mission Planner.
- The goal of the Behavioral Planner is to navigate the selected route following the driving convention and rules of the road.[4]
- This module take in considerations rules of the road, Static and dynamic objects around the vehicle.
- Various driving rules are considered such as tracking speed, following the vehicle ahead of the ego vehicle, decelerate to stop, Stop, Merge and few more.

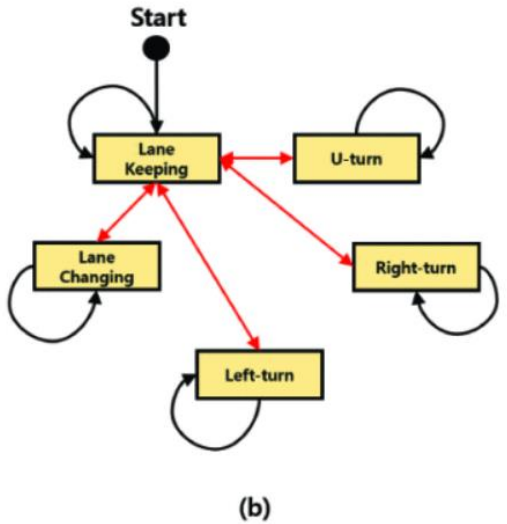
FINITE STATE MACHINES

- Finite state machines is the traditional approach used to represent set of rules required for behavior selection of the ego vehicle.

- The two types of finite state machines are **Control Finite State Machine and Motion Finite State Machine**.[5]



C-FSM



M-FSM

- The black line means a one-way state change, and the red line means a two-way state change.

COLLISION CHECKING

- To decide states in the Finite State Models, collision checking has to be done.
- Collision checking is the process of ensuring that decided planned path when traversed by ego vehicle does not collide with any static or dynamic obstacles along the way.[6]
- Collision checking in computationally intensive and requires perfect information to generate safety.[6]
- The 2 main types of collision checking methods are **Swath-based** collision checking and **Circle based** collision checking**.**

SWATH BASED COLLISION CHECKING

- Swath based collision technique requires rotating and translating the footprint of a vehicle along every point of a given point.

- Swath based collision checking computes the union of the path that vehicle has travelled with that of occupancy grid map and localization map for dynamic objects.

- This requires recursive computation which increases its complexity for larger projects.

Swath based collision checker

CIRCLE BASED COLLISION CHECKING

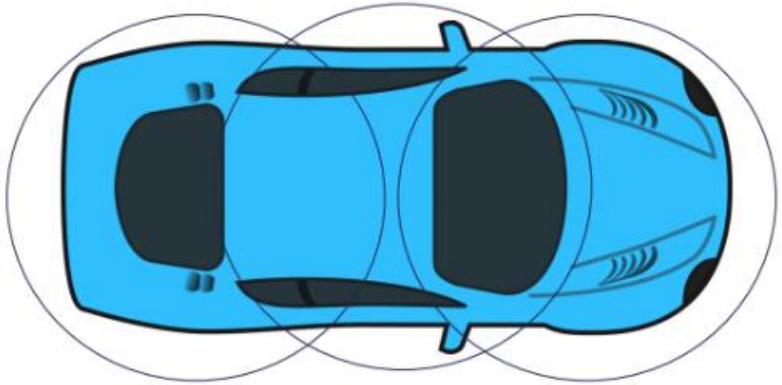
- In this method, the ego vehicle is localized and is estimated by forming a circle around it.

- The radius of the circle can be adjusted as per the vehicle's specification and collision requirements.

$\|(x_i, y_i) - (x_c, y_c)\| \leq r$

$\|(x_i, y_i) - (x_c, y_c)\| > r$

- May report a collision that won't occur but never misses a chance to detect the collision that will occur.



LOCAL PLANNING - PATH GENERATION AND SELECTION

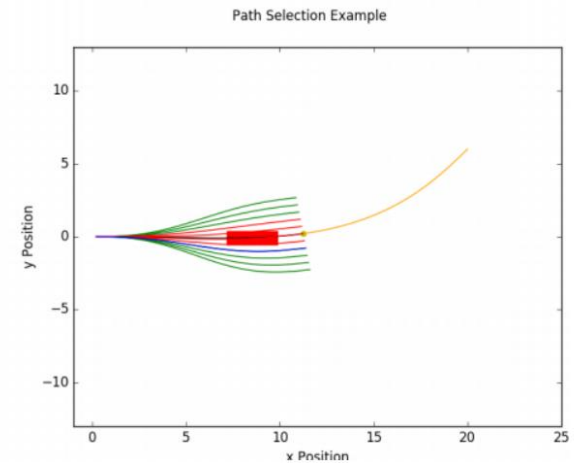
- The goal of the path planning is to plan a feasible collision free path to the destination.
- Goal points on the way are found by lookahead technique and are calculated based on speed and other factors.
- There are two methods for path trajectory: **Quintic splines** and **Cubical spiral**.
- For each of our goal states, spiral can be optimized to the goal point.
- A final collision free path is selected from all the generated spirals.

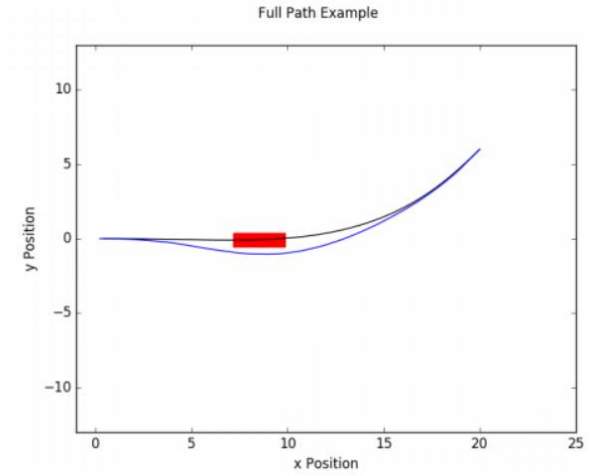
CUBICAL SPIRALS

- Defined as a polynomial curvature function with respect to arc length.

- Since a spiral is a polynomial function of curvature, the curvature value will not change extremely quickly like it can in the case of quintic splines.

- Closed form solution does not exist, hence integrals need to be evaluated numerically using Simpson's rule.





$$x(s) = x_0 + \int_0^s \cos(\theta(s'))ds'$$

$$y(s) = y_0 + \int_0^s \sin(\theta(s'))ds'$$

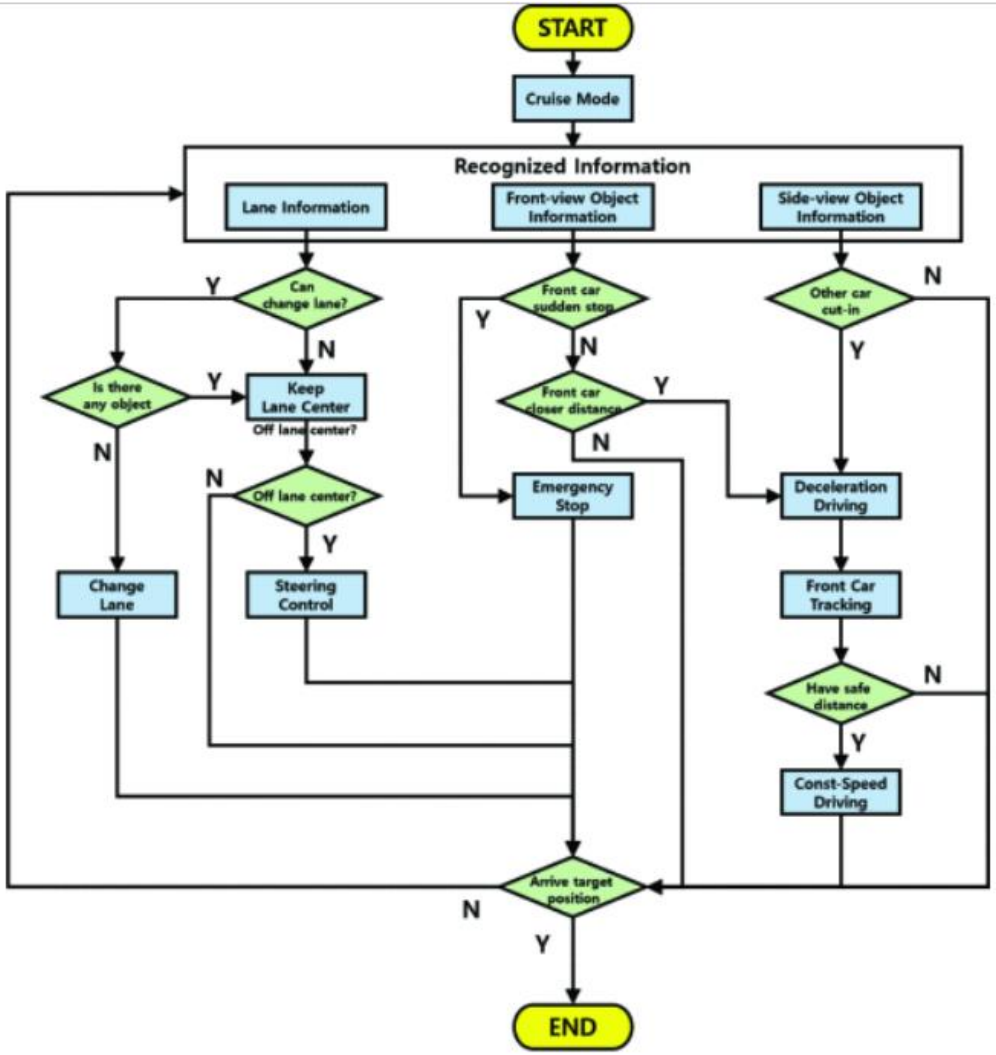
$$\int_0^s f(s')ds' \approx \frac{s}{3n}\left(f(0) + 4f\left(\frac{s}{n}\right) + 2f\left(\frac{2s}{n}\right) + \cdots + f(s)\right)$$

Simpson's Rule

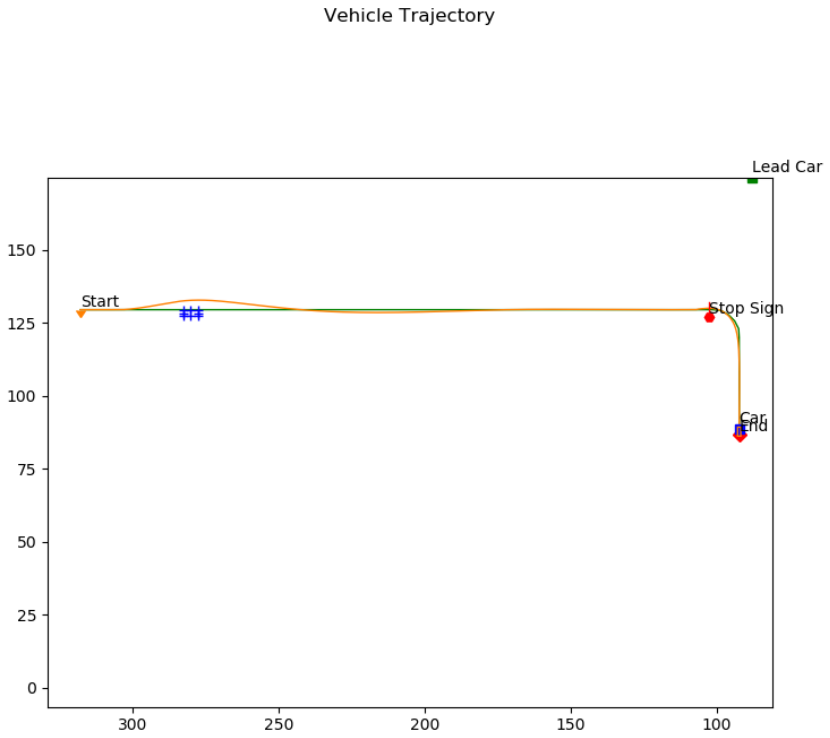
CARLA SIMULATOR

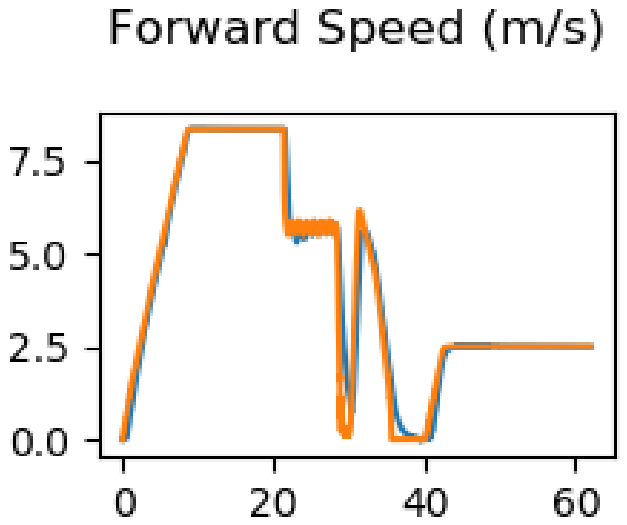
- An open-source simulator for urban driving.
- Developed from the ground up to support training, prototyping, and validation of autonomous vehicles.
- Environment is composed of 3D models of static objects as well as dynamic objects.
- Supports development, training, and detailed performance analysis of autonomous driving systems.[9]

IMPLEMENTATION DIAGRAM



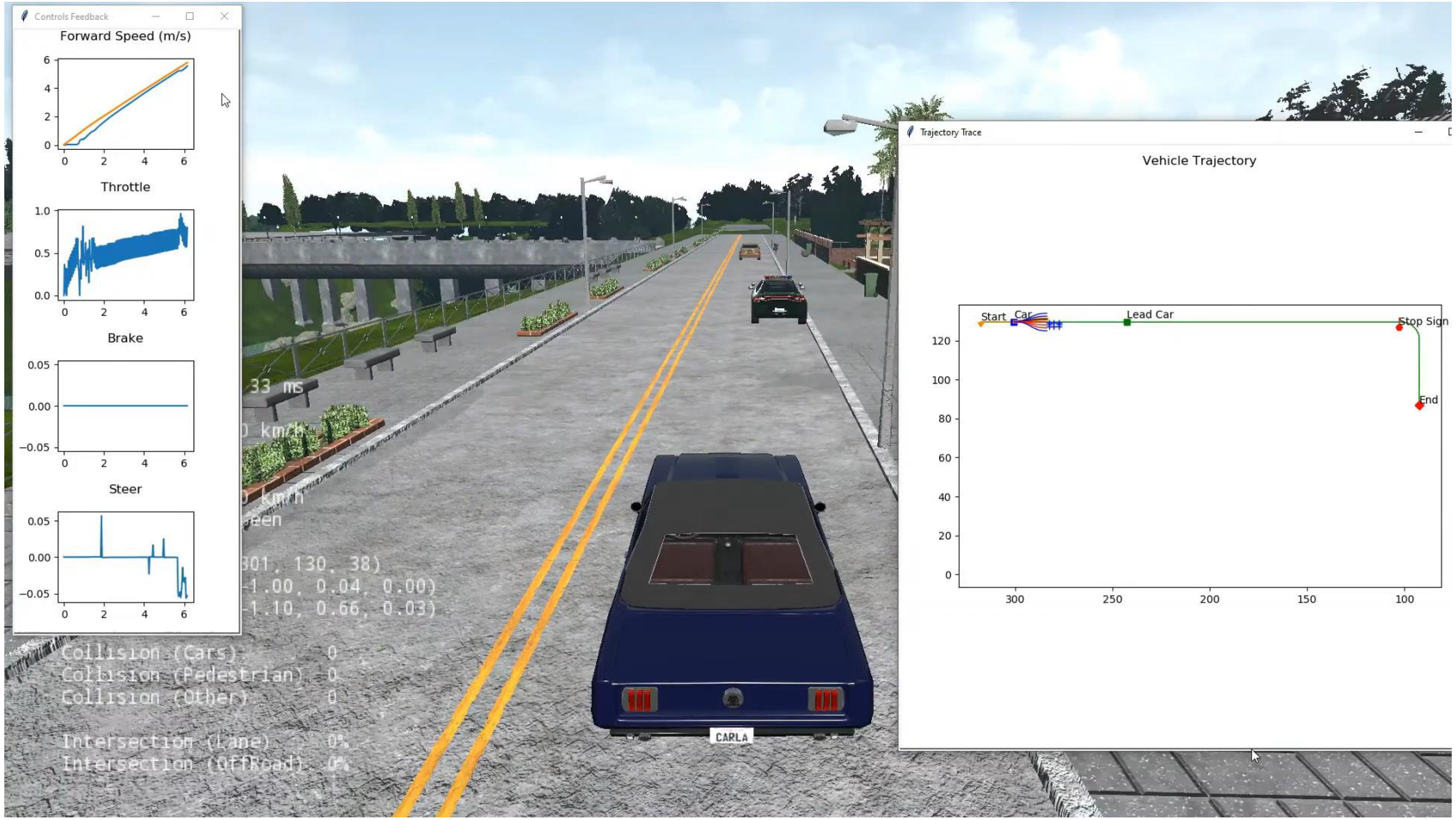
SIMULATION RESULTS





Collision count = 0

SIMULATION RESULTS



CONCLUSION

- Circle based collision checking method is more effective compared to Swath based collision checking.

- Cubic spiral was found to be more effective than Quintic splines for path trajectory.

- It is effective to use Python programming language as there are numerous required built in functions used in various algorithms. For more robustness, C++ can be used.

CHALLENGES

- One of the main challenge was faced in understanding and using CARLA Simulator.

- Apart from CARLA, implementing local planner algorithms for path planning was found to be challenging.

REFERENCES

[1] David Silver, “How Path Planning Works for Self-Driving Cars” Available: https://www.linkedin.com/pulse/how-path-planning-works-self-driving-cars-david-silver/

[2] P. Polack, F. Altché, B. d'Andréa-Novel and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?," *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 812-818, doi: 10.1109/IVS.2017.7995816.

[3] J. Wei, J. M. Snider, T. Gu, J. M. Dolan and B. Litkouhi, "A behavioral planning framework for autonomous driving," *2014 IEEE Intelligent Vehicles Symposium Proceedings*, 2014, pp. 458-464, doi: 10.1109/IVS.2014.6856582.

[4] Brian Paden, Michal Cap, Sze Zheng Yong, Dmitry Yershov, Emilio Frazzoli, “A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles”

[5] S. -H. Bae, S. -H. Joo, J. -W. Pyo, J. -S. Yoon, K. Lee and T. -Y. Kuc, "Finite State Machine based Vehicle System for Autonomous Driving in Urban Environments," *2020 20th International Conference on Control, Automation and Systems (ICCAS)*, 2020, pp. 1181-1186, doi: 10.23919/ICCAS50221.2020.9268341.

[6] M. McNaughton, C. Urmson, J. M. Dolan and J. -W. Lee, "Motion planning for autonomous driving with a conformal spatiotemporal lattice," *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 4889-4895, doi: 10.1109/ICRA.2011.5980223.

THANK YOU