# Robust Lane Following with V2X Traffic Management

Luis Escamilla, Michael Evans, Beñat Froemming-Aldanondo,
Rickey Johnson, Marcial Machado, Tatiana Rastoskueva, and Anna Vadella

## 2024 LTU REU

# Contents


## 01 Introduction

- Motivation
- Importance

## 02 Lane Following

- Algorithms
- Performance Analysis

## 03 RSU

- Establishment
- Integration

## 04 Architecture

- Construction
- Modularization

## 05 Speed Algorithm

- Algorithm Analysis
- Waypoints and GPS

## 06 Results

- Figure 8 Simulation
- Lot H Simulation
-

## 07 Conclusion

- Results
- Future Work

# 01 Introduction

Motivations and Importance

# Why Use Automated Vehicles?

Multitude of problems can be solved with automated vehicles:

- **Minimizes traffic** and road congestion
- Improved routing **reduces energy**
- **Increases safety** for both pedestrians and drivers

# Our Approach

- Develop a V2X Intersection Control System to improve traffic efficiency, comfortability, and safety
  - Create reliable lane following algorithms
    - Which performs best?
  - Handle traffic light detection and messaging via Road Side Unit (RSU)
    - Simulate V2X in simulation before real-life testing

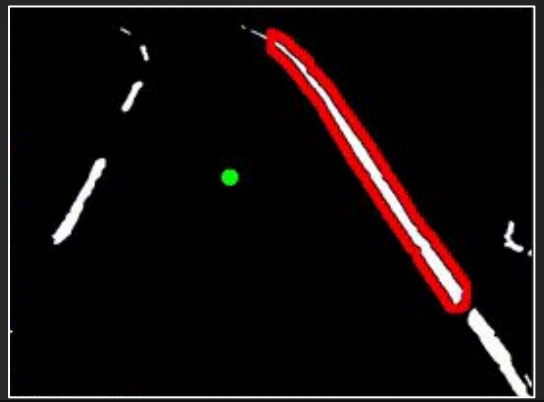
# 02 Lane Following

Analysis of Lane Following Algorithms

# Lane Following Algorithms

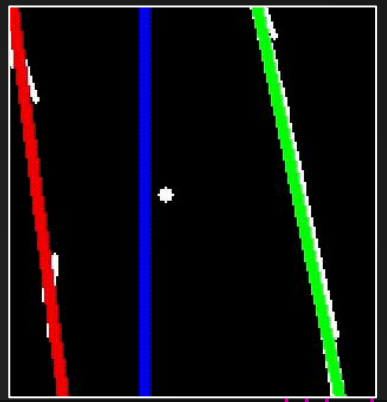
## Largest White Contour

- Detects only one line (the largest white contour)
- Adds a center offset
- Sensitive to lane line quality (cracks, shades, etc.)



## Least Squares Lane Estimation

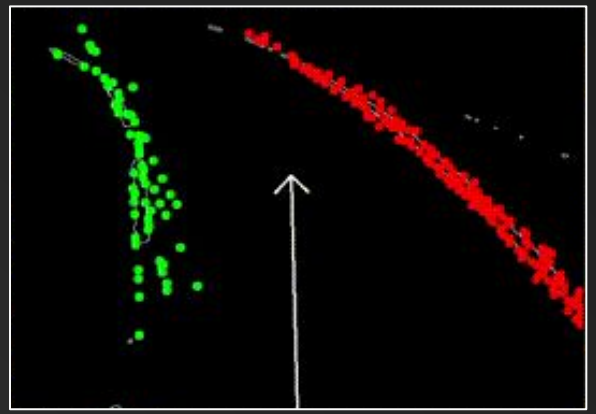
- Change image perspective to detect lines easier
- After perspective change, lane lines are straight and parallel
- Divide image into two parts and calculate least squares regression line to determine center of lane



# Lane Following Algorithms

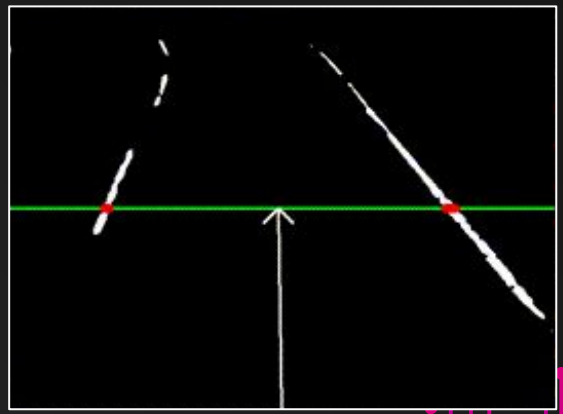
## Lane line discrimination with DBScan

- Uses unsupervised learning
- Detects and separates both lane lines
- Sensitive to parameters

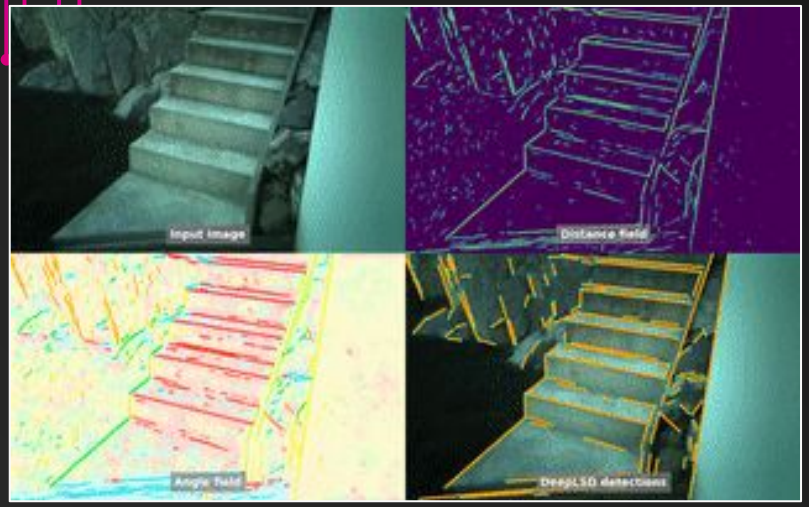


## Linear Lane search with K-Means

- Uses unsupervised learning
- Robust to losing lanes
- Sensitive to noise



# Lane Following Algorithms

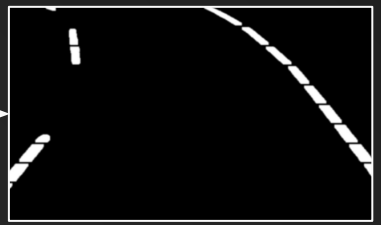



## DeepLSD lane detection

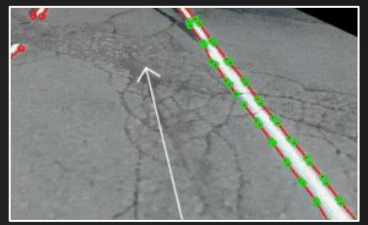
- Uses supervised learning
- Model trained to detect straight lines
- Inference time is slow (~5 images processed per second) on our laptops
- Can't find curves

Curve not detected

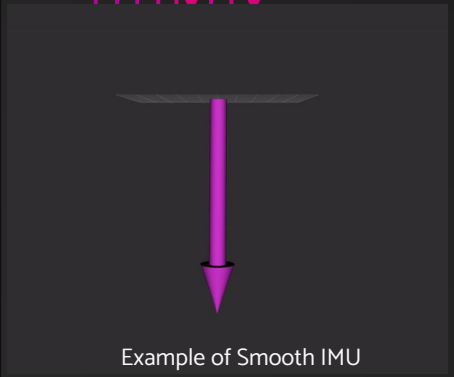
Draw horizontal lines



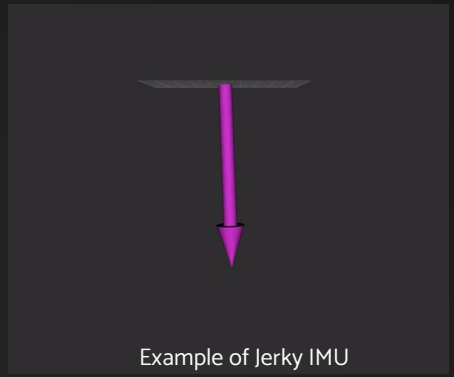
Curve detected



# Comfortability



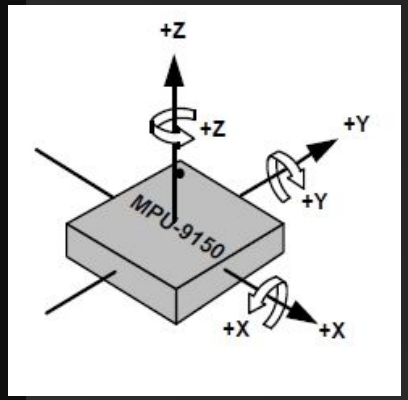
Example of Smooth IMU

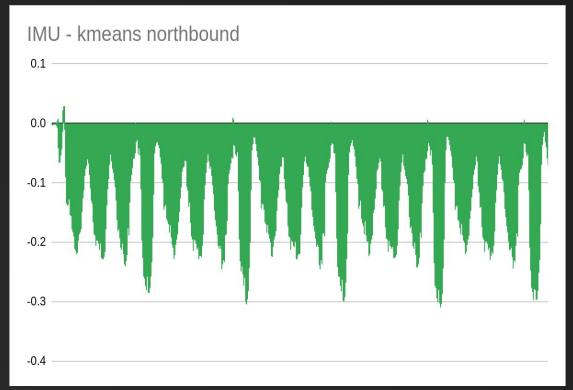


Example of Jerky IMU

## IMU

- Inertial Measurement Unit
- Measures forces acting on the ACTor vehicle
  - Angular Z: “Side-To-Side Force”
- Indicates “smoothness”



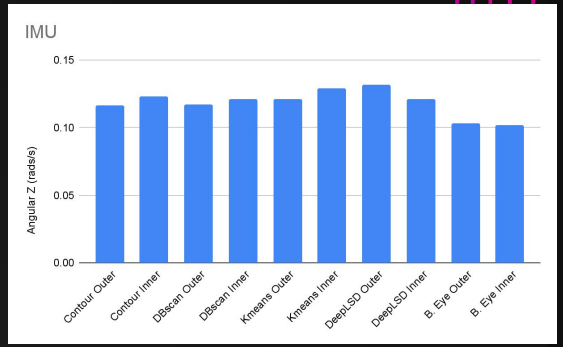

## Acceleration

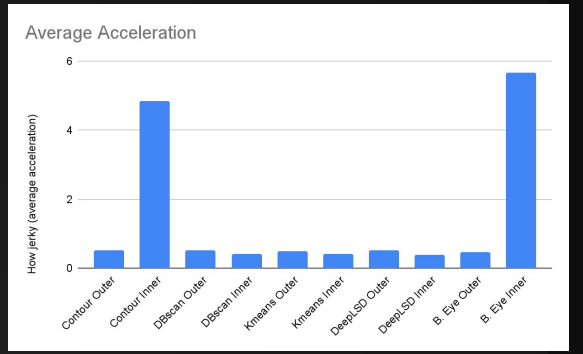
- Indicates “jerkiness”

$$\bar{a} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{v_{i+1} - v_i}{t_{i+1} - t_i}\right)$$

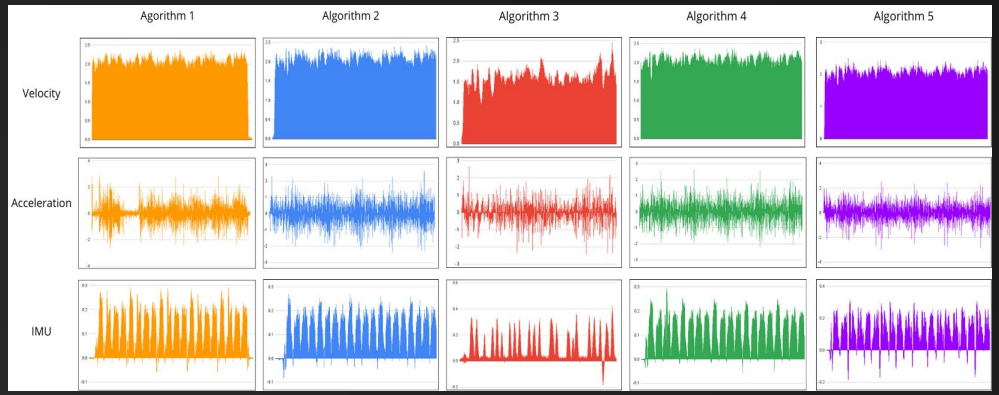
# Performance & Comparison

| Algorithm Name | Lane | # of successful laps | Time (s) | Speed (m/s) | Jerkiness: Acceleration (m/s/s) | Zig Zag Factor: IMU |
|---|---|---|---|---|---|---|
| Contour | Inner | 5/5 | 257 | 1.53 | 4.857873894 | 0.1231172873 |
| Contour | Outer | 5/5 | 249 | 1.96 | 0.5089629265 | 0.1168738107 |
| Birds Eye View | Inner | 5/5 | 289 | 1.52 | 5.666396926 | 0.1018033431 |
| Birds Eye View | Outer | 5/5 | 320 | 1.36 | 0.4709566919 | 0.1033114171 |
| K-means | Inner | 5/5 | 240 | 1.64 | 0.4106803578 | 0.129166573 |
| K-means | Outer | 5/5 | 250 | 1.95 | 0.4963880041 | 0.1210361303 |
| DBscan | Inner | 5/5 | 257 | 1.53 | 0.4075250789 | 0.1212667764 |
| DBscan | Outer | 5/5 | 251 | 1.94 | 0.5125099194 | 0.1173864182 |



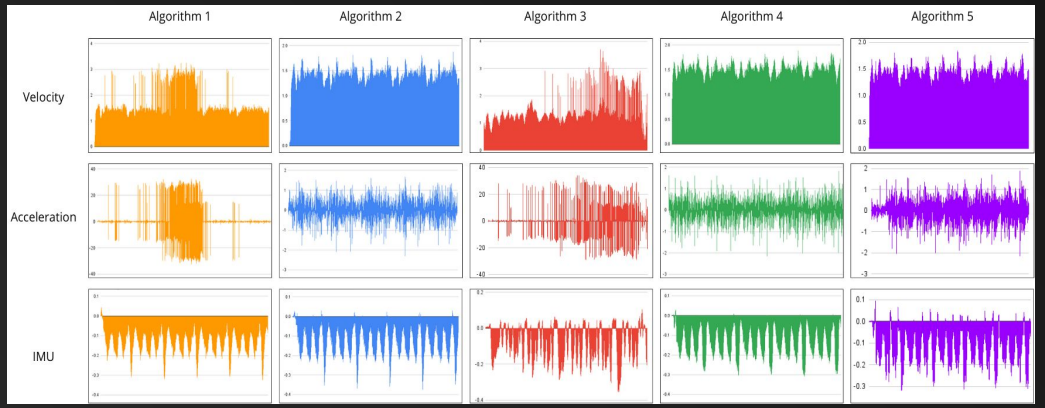



# Performance & Comparison




Outer Lane Results

Inner Lane Results

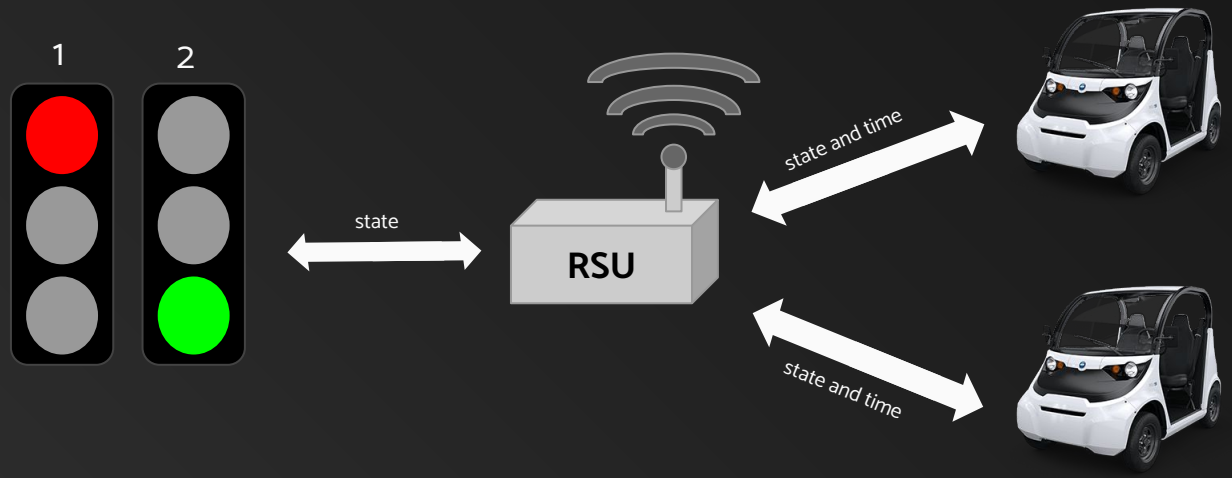


# 03 RSU

Establishing and Integrating a Connection

# Traffic Light Establishment

- Requirements
  - Centralized “brain” → vehicles focus on following lanes and expect to receive prepared-and-relevant information
  - Minimal hassle in establishing connections across nodes & computers
  - Potential for scalability
    - provide an effective yet simple-and-extensible proof of concept



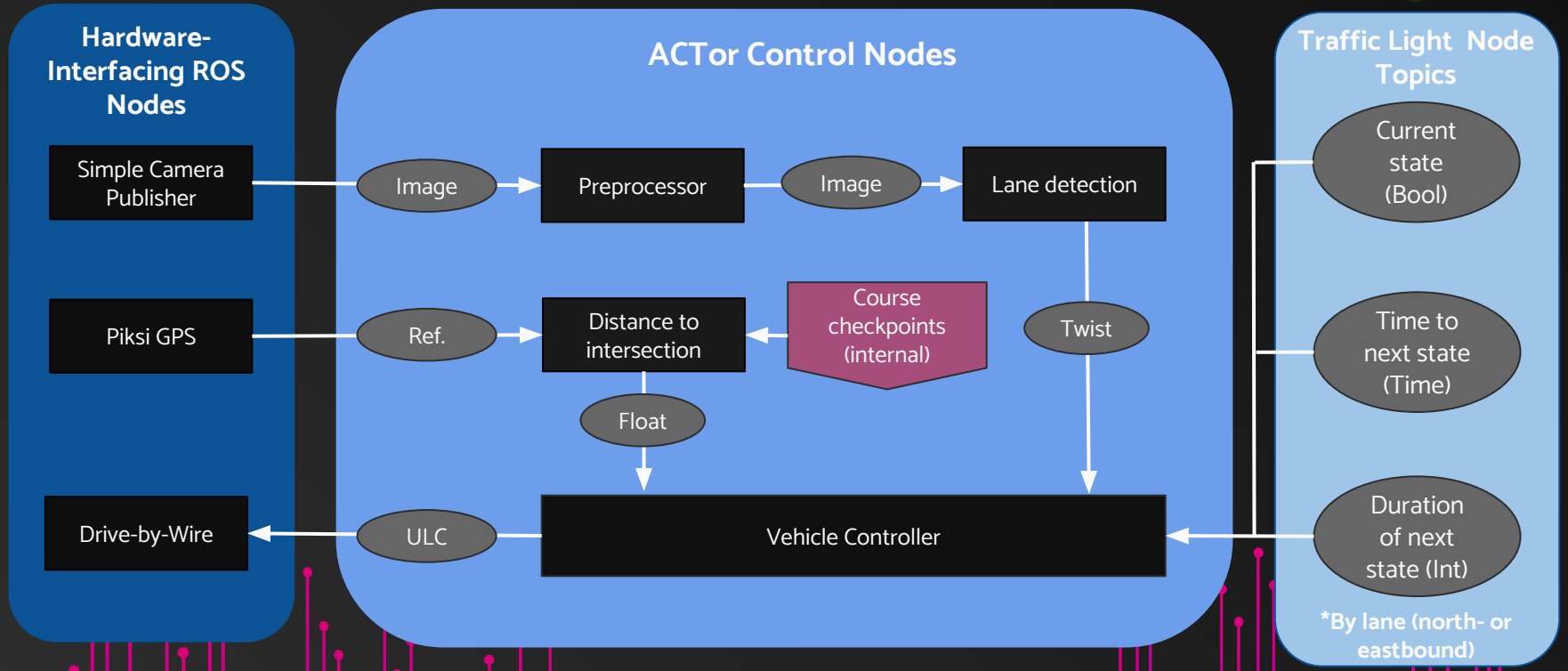
# Traffic Light Implementation

- “Server-client” architecture → traffic light model runs on RSU as host to all network communications
- Light functions as human-facing descriptor of RSU’s current state connected through ROSSerial
  - routinely checks for connection & runs backup light procedure if connection drops
- Cars receive necessary information from RSU for their namespaces only

# 04 Architecture

Construction and Modularization

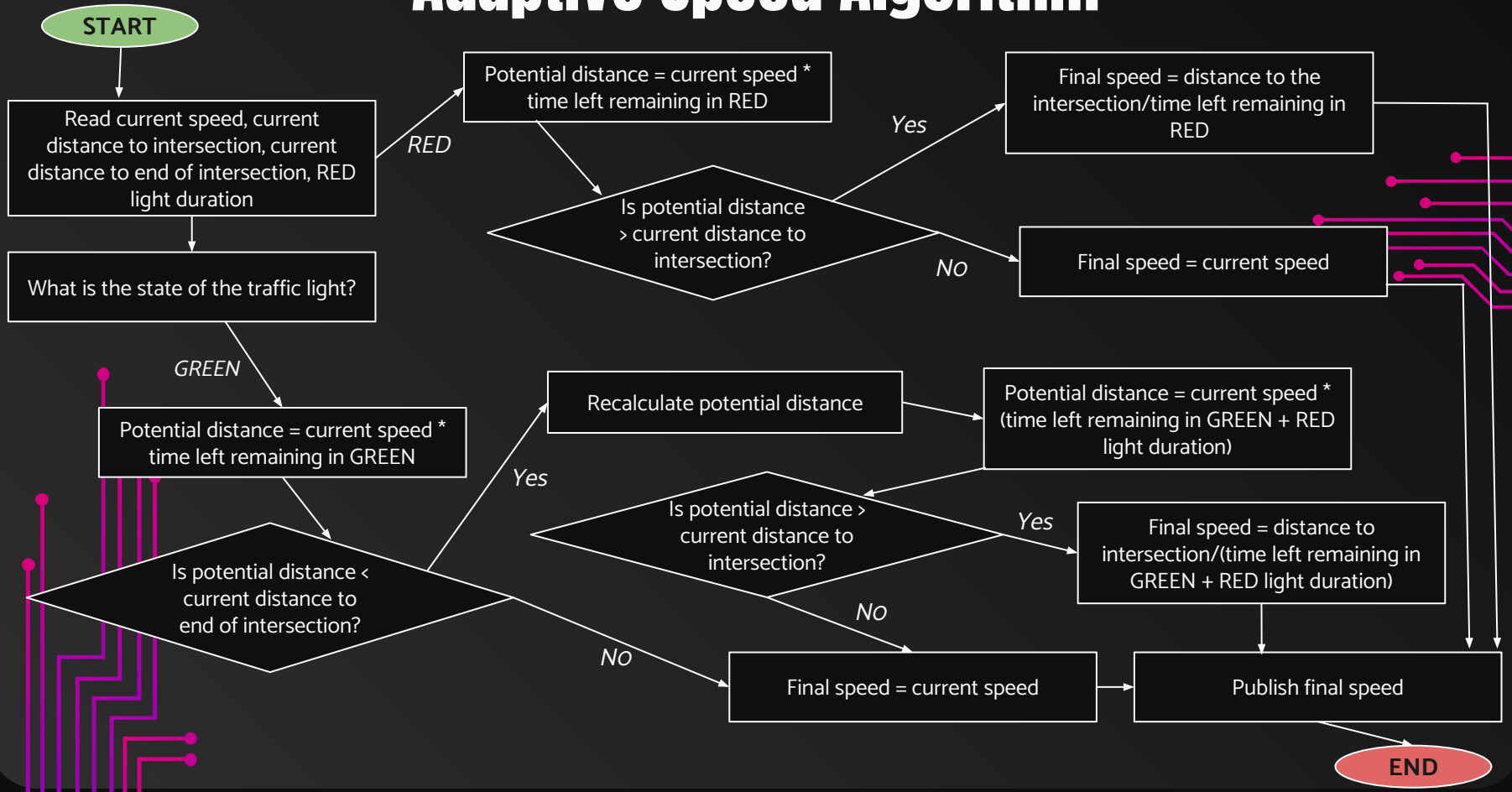
# ROS Node Architecture



# 05 Speed Algorithm

GPS & Algorithm Analysis

# Adaptive Speed Algorithm



# Adaptive Speed Algorithm

| Light State | Going to cross intersection in this phase | Result |
|---|---|---|
| RED | Yes | Slow Down |
| RED | No | Maintain Speed |
| GREEN | Yes | Maintain Speed |
| GREEN | No | Slow Down |

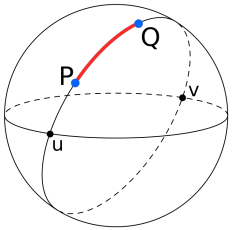
# Capturing Waypoints with GPS

- Need waypoints for an accurate approximation of true distance from vehicle to intersection
- Record GPS coordinates from the Piksi: *reference/piksi/position_receiver_0/sbp/pos_llh*
- Generate .yaml file of waypoints for *both lanes*
- **Course Dimensions:**
  - Inner lane center length: ~80 meters
  - Outer lane center length: ~100 meters

# Calculating Waypoint Distance with GeoPy

**Definitions:**

Let $v = (\phi_v, \lambda_v)$ be the GPS coordinates of vehicle $v$.
Let $w_i = (\phi_i, \lambda_i)$ be the GPS coordinates of waypoint $w_i$.
Let $W = \{w_1, w_2, \ldots, w_n\}$ be the set of all waypoints.
Let $I \subset W$ be the set of all intersections.
The distance between points of a sphere $h(\phi_1, \lambda_1, \phi_2, \lambda_2) =$




$$2r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right)$$

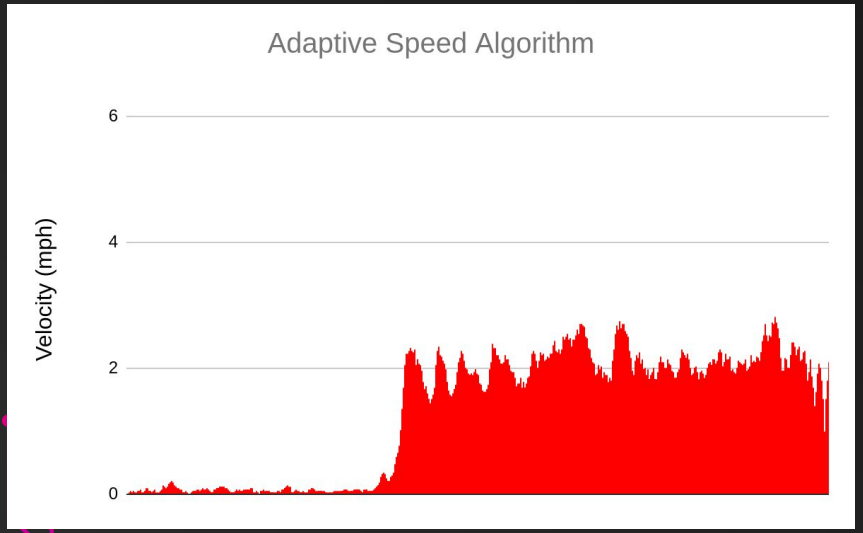
**Find the waypoint $w_i$ with the shortest distance to $v$:**

$$\text{Let } p = \min_{x \in W} h(x, v)$$

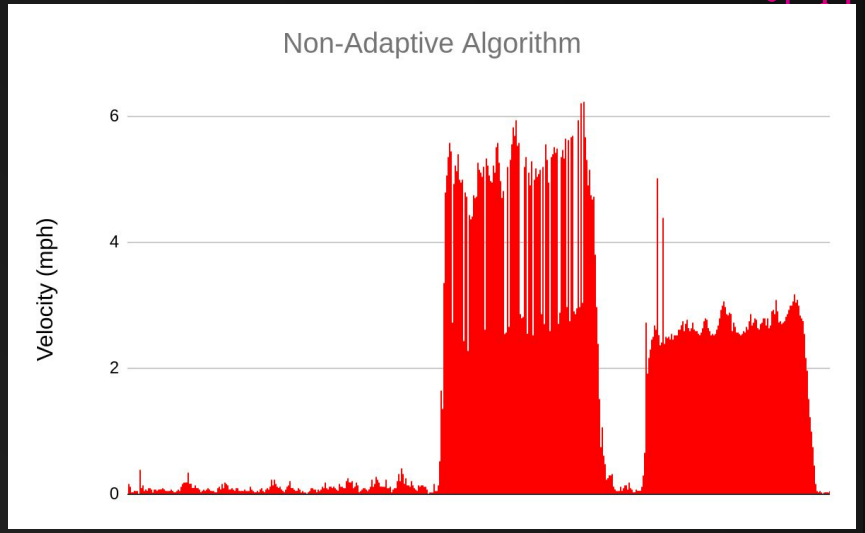
**Sum $h(w_i, w_{i+1})$ from $p$ to the next intersection:**

$$\sum_{i=p}^{k-1} h(w_i, w_{i+1}) \quad : w_k \in I \text{ and } k > p$$

# Adaptive Speed Algorithm Comparison




Adaptive Speed Algorithm

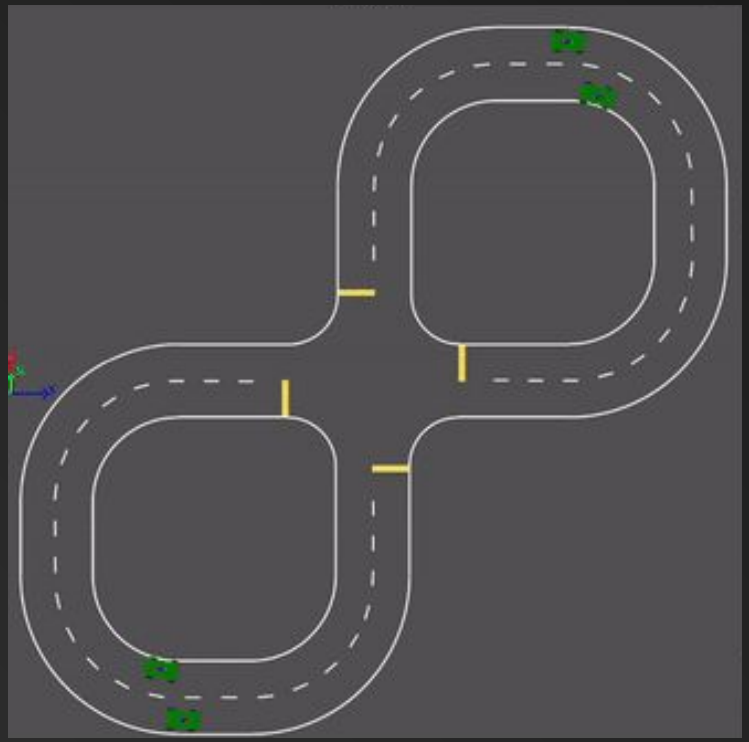


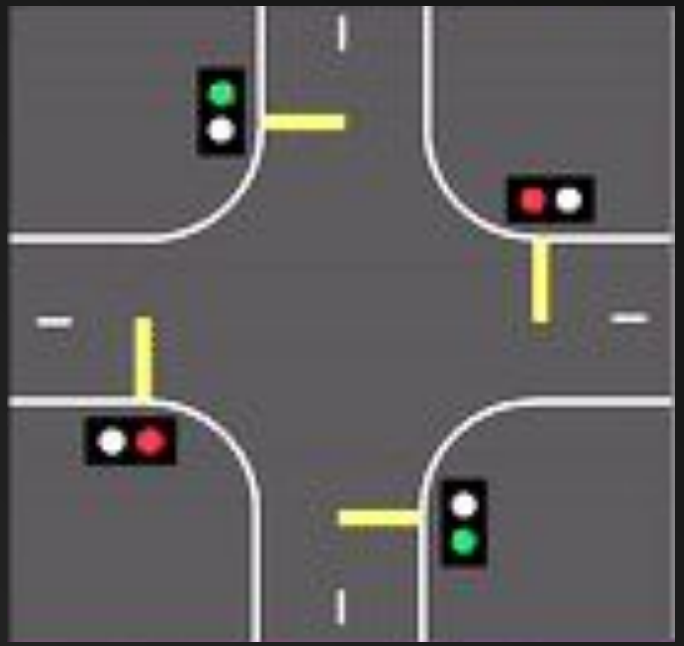

Non-Adaptive Speed Algorithm

# 06 Results

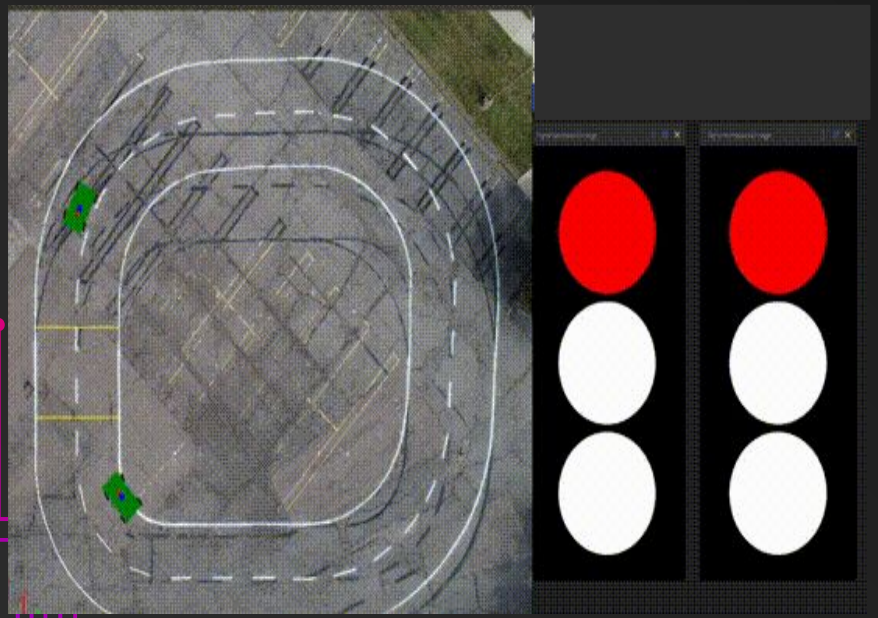
Analyze & Compare Courses

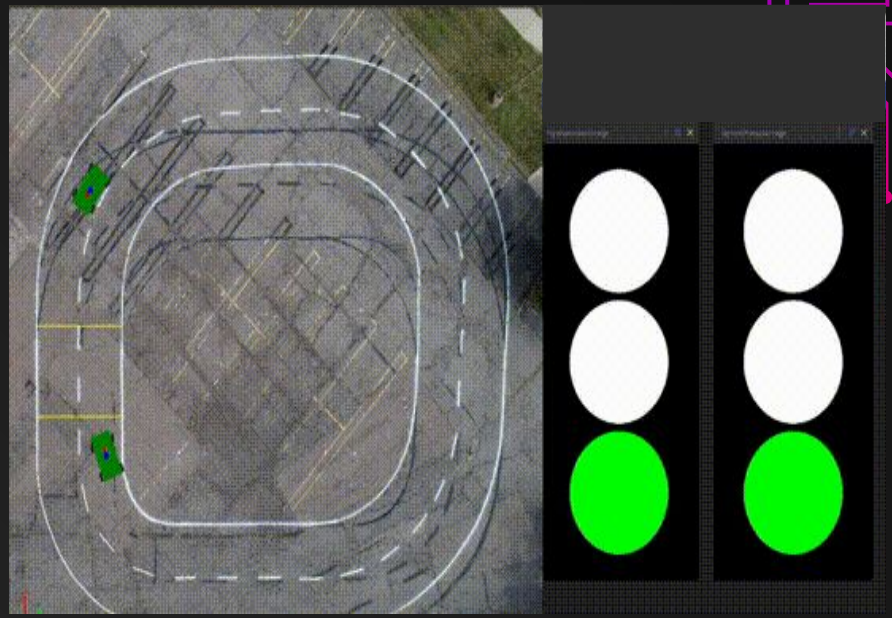
# Course Simulation: Figure 8



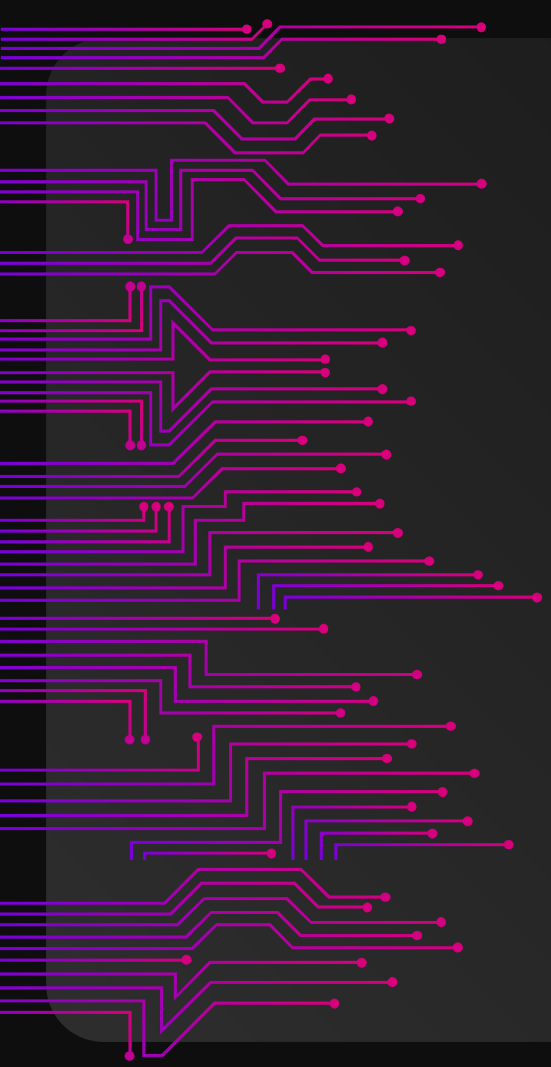


# Course Simulation: Lot H





# Algorithms Outside of Simulation



# 07 Conclusion

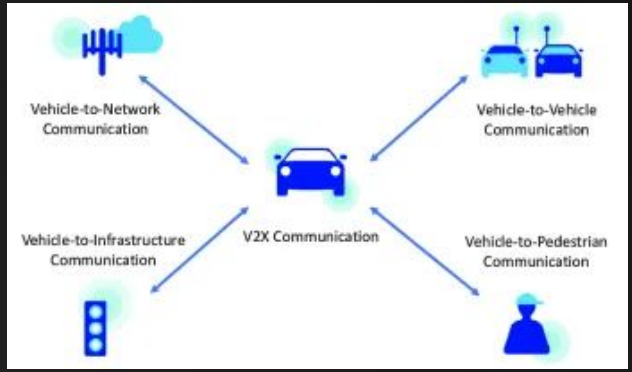
Performance & Future Works

# Future Work

- Consider potential networking issues with RSU implementation and connection
- Implement “speeding up” functionality with adaptive speed algorithm
- Develop adaptive traffic control
  - Traffic light adjustment algorithms
  - Smart vehicle responses
  - Consideration of external safety factors

# Summary

- Analysis of lane following algorithms
- RSU establishment and connection
- Adaptive speed algorithm improves efficiency and comfort
  - Less frequent stopping can improve safety



# Q & A

Thank you!