Current Progress of V2X Research

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

Lawrence Tech and Michigan State University

An Overview

Questions and Motivations:

- Efficient traffic management
- Viable and scalable V2X model

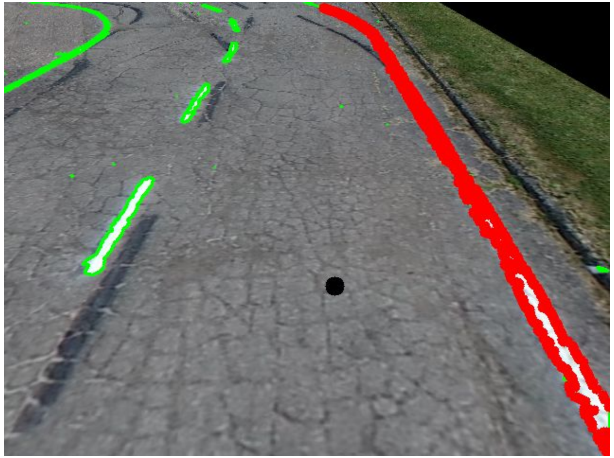
How We've Been Answering Them:

- Creating reliable lane-following algorithm(s)
- Developing viable intersection detection
- Prototyping light-RSU-vehicle connection
- Establishing a modular node architecture
- Simulating V2X capabilities with gazelle

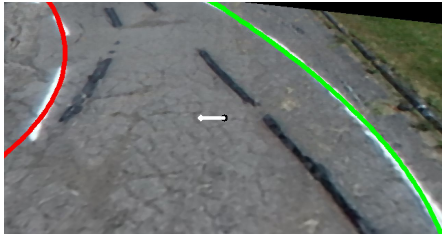
Lane Detection and Following

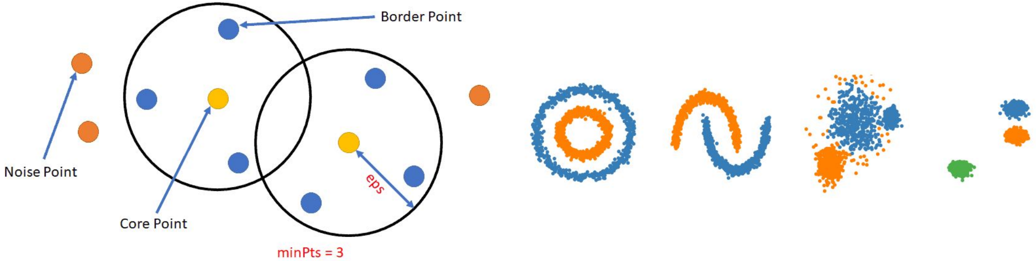
Tested Approaches on the ACTors:

1) Largest White Contour With an Offset.



2) Density Based Clustering (Unsupervised Learning)





Lane Following Demo

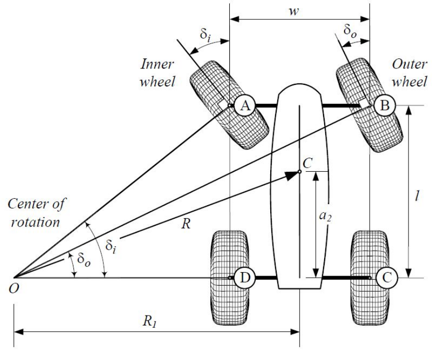
Notable Characteristics:

- Successfully lanekeeps in extreme environments (shown: shadow, glare)
- Centered properly within the lane
- Speed retention on turns
- Minimal jerk on straight drives

Lane Following

Achievements:

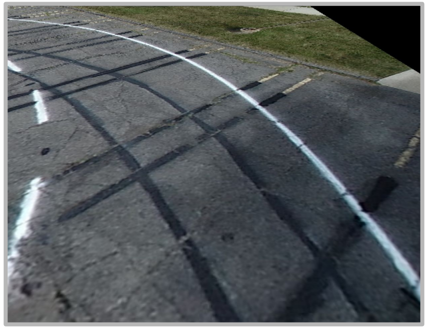
- 3 laps in both inner and outer lanes using each algorithm
- Improved the advanced clustering approach by introducing DBW Native Commands: Actuator Messages and Unified Controller Messages
- IGVC regulatory speed of 5 mph
- Detect the yellow intersection line, stop for a bit, cross the intersection, and ignore the next yellow line



Future Work:

- Birds-eye-view implementation
- Test other approaches: Blob, Hough Lines, and Deep Learning Lane Detection
- Create a table to compare all approaches for the paper
- Choose the best for V2X final demo

Lane Following: Birds-Eye-View



Idea:

- Theoretically, easier curve detection

How it works:

- First using cv2 function to change the image perspective
- Image processing - cropping the image ,threshold, canny
- Gets points of the left and right and computes a least squares function through them
- Computes middle point between left and right lines

Future Improvements

- Tune algorithm to deal with faster speeds and sharp curves
- Connect algorithm to a Deep Learning lane detection

Computing Speed and Detecting Intersections

```
# detect intersection
Create yellow mask
Compute yellow pixel percentage

# lane detection
compute ideal center using CV

if drive:

      # linear speed
      if yellow percentage > threshold:
         Set target speed to 0 for a set time
         Start moving again and ignore yellow detection
         for  a set time
      else:
         set target speed to speed limit

      # angular speed
      if mid is left of center by a threshold:
          turn steering wheel left
      elif mid is right of center by a threshold:
         turn steering wheel right
      else:
         go straight

else:
      set target speed to 0
```

Establishing an RSU Strategy: Initial Connection

Our Introduction:

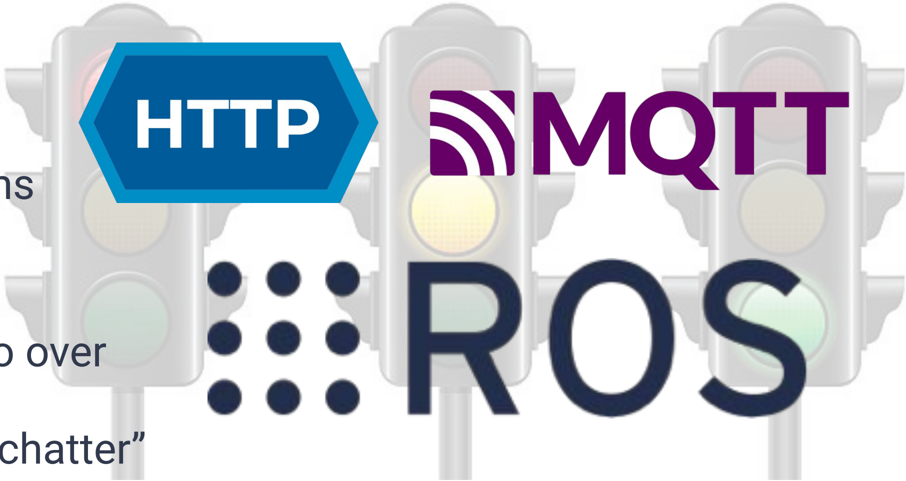
- Understanding sample GPT HTTP Arduino code

The Process:

- Look into alternatives: MQTT, ROSSerial
- Practice with alternative timing algorithms

Where We Are:

- Successfully connected RSU and Arduino over RSU-hosted ROSSerial connection
- Prototyped communication over model “chatter” topic



Establishing an RSU Strategy: Message Types

Traffic light implementation

- The traffic light ‘brain’ implemented on arduino
- After each switch publishes a messages to topics:
    - Direction, State, time until next change

```
$ rosmsg info TrafficMessage
[custom_msg_pkg/TrafficMessage]:
bool state
uint16 time
```

Custom message type on arduino | Error

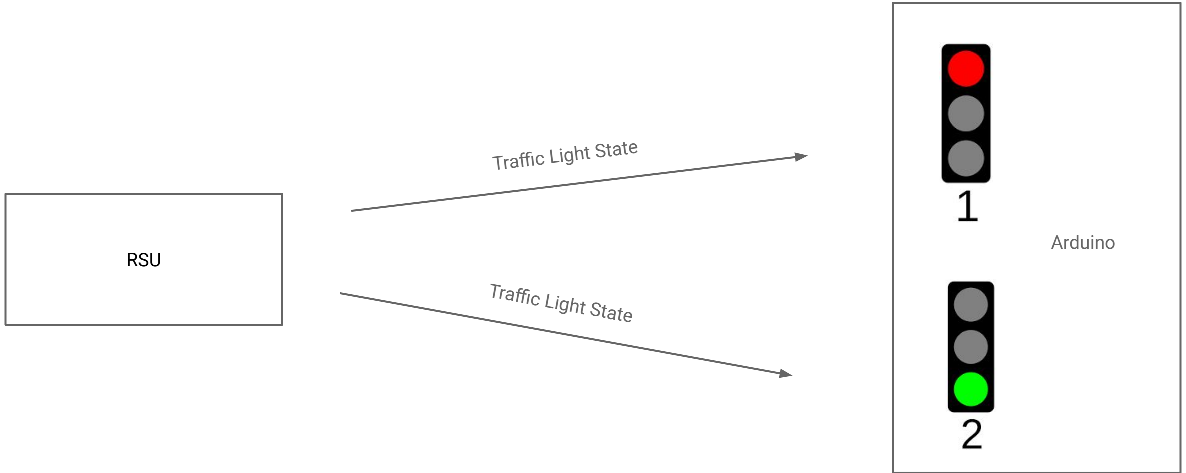
Idea: publish a custom ros topic with state and time from arduino to RSU

- Made a custom ros topic on ros
- Problem: too much work with installing both ros on windows and arduino on linux
    - Tried WSL, but linux and windows have different file systems -> wouldn’t be able to connect ros message to arduino
- Decided to come up with another system

```
state: True
time: 256
---
state: True
time: 256
---
state: True
time: 256
---
state: True
time: 256
---
state: True
time: 256
---
state: True
time: 256
---
```

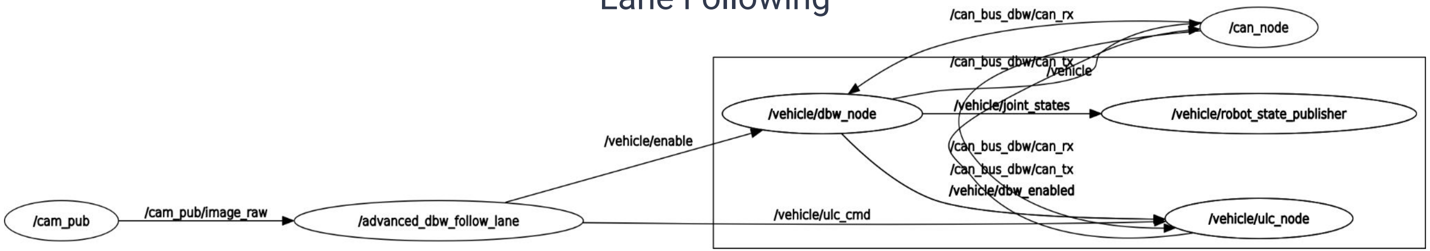
Going Forward: RSU “Host” → Light “Client”

- RSU will run the state and timer of traffic light internally
- RSU only publishes state changes to traffic light node
- Each side of the light will subscribe to topics in that side’s ROS namespace

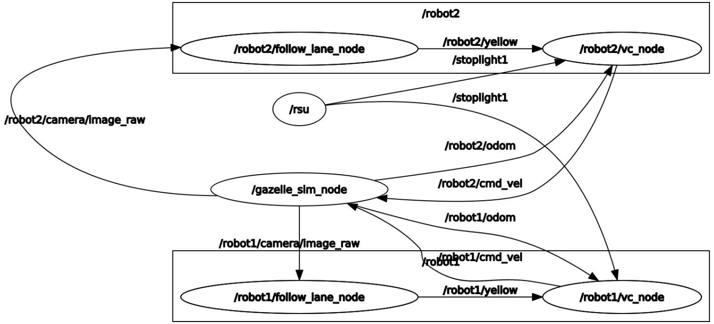


Software Architecture

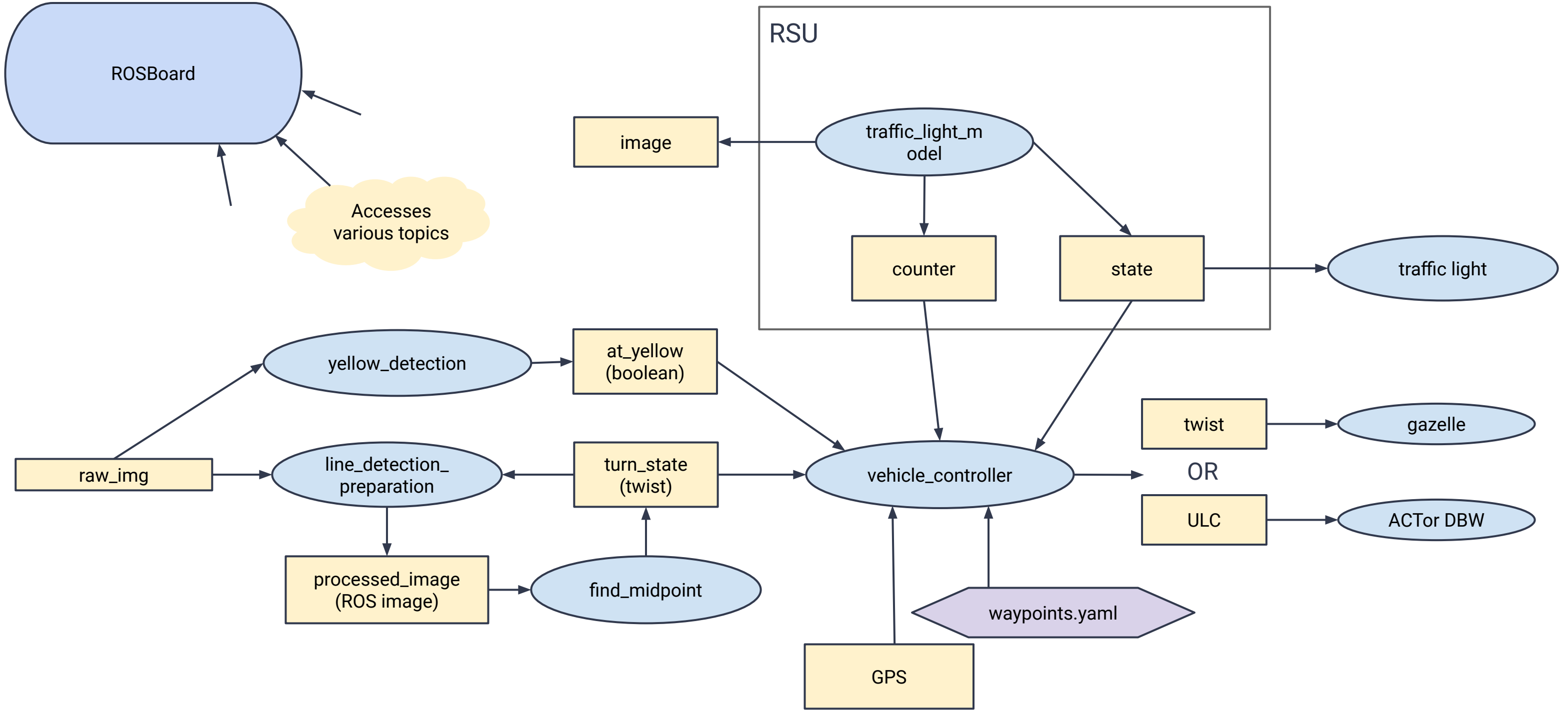
Lane Following



V2X Sim



Conceptualization of Future Architecture



Current Progress on V2X Simulation

- Currently using contour based lane following algorithm
- Developed an adaptive speed algorithm to reduce waiting time at intersections
- Tested algorithm on two different maps with varying number of vehicles
- Created rosboard to visually show stop lights and their times

Capturing Waypoints in Simulation

We use waypoints to solve the “birds eye view” distance calculation problem

Steps:

1. Trace expected path of car on simulator map
2. Ping position of map on simulator through terminal
3. Increment every few centimeters
4. Save positions to .yaml file, separating intersection coordinates
5. Repeat process for each designated path for car to follow

Improvements:

- Reduce number of waypoints to “checkpoint” type structure on corners only
- Compute a known “distance to intersection” at each node

```yaml
intersections:
  - [35.97, 2.14, 0, "E"]       #horizontal - E
  - [30.43, -6.20, 1, "N"]      #vertical - N

waypoints:
  - [30.49, -5.17]
  - [30.43, -4.26]
  - [30.49, -3.34]
  - [30.43, -2.09]
  - [30.37, -0.83]
  - [30.43, 0.43]
  - [30.49, 1.80]
  - [30.60, 2.94]
  - [30.66, 4.26]
  - [30.71, 5.06]
  - [30.71, 6.20]
  - [30.71, 7.11]
  - [30.71, 8.14]
  - [30.71, 9.63]
  - [30.66, 10.89]
  - [30.66, 12.09]
  - [30.60, 13.40]
  - [30.71, 14.60]
  - [30.71, 16.03]
  - [30.77, 17.63]
  - [31.06, 19.00]
  - [31.46, 20.14]
  - [32.09, 21.17]
  - [32.89, 22.09]
  - [33.69, 22.83]
  - [34.77, 23.46]
```

Using Waypoints in Simulation

Finding distance using waypoints:

```
Find the closest waypoint to the car's position:
    for every waypoint:
        calculate the distance from car to waypoint
        if distance < lowest distance so far:
            lowest distance = distance
            record the index of lowest distance waypoint

Sum the total distance from current waypoint to next intersection:
    total distance = 0
    for every waypoint from waypoints[current] to waypoints[intersection]:
        total distance += distance between waypoint i and waypoint i + 1
```

Adaptive Speed Algorithm

```
when stop lights change state:
      calculate distance to closest intersection
      time to intersection = distance / current speed
      # works good on simulation
      if light state is red:
            if time to intersection < time to change state:
                  set speed to distance / time to change state (arrive too soon, SLOW DOWN)
            else:
                  keep speed
      # logic to improve (weird behaviour sometimes)
      if light state is green:
            if time to change state < time to intersection > and time to next change state:
                  set speed to distance / time to change next state (arrive too late, SLOW DOWN)
            else:
                  keep speed
```

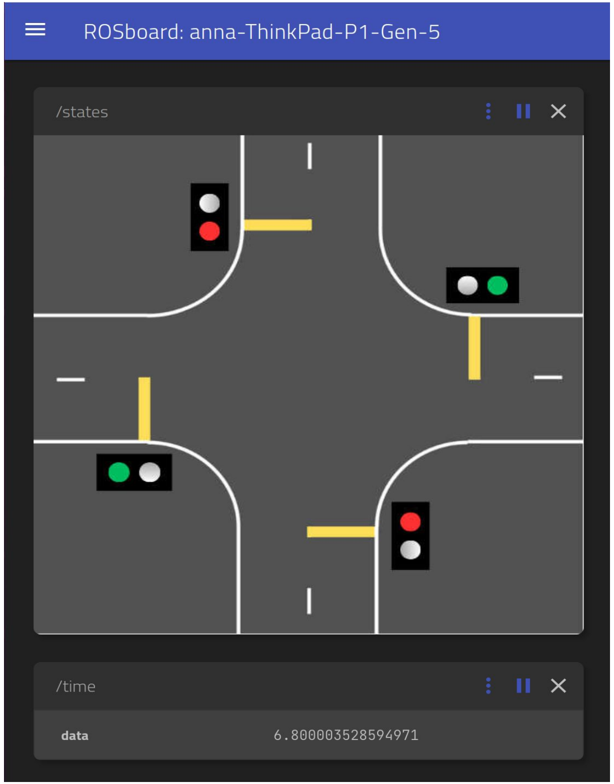
Rosboard

- Initially planned to use Flask
- Then we switched to Rosboard introduced by Devson
- Display stoplight status on the intersection and time left for the state to change

```python
# Red at horizontal intersections, green at vertical intersections
while (time.time() - curr) < 10:
    stop = Bool(data=True)
    left_time = Float64(data=10 - (time.time() - curr))  # Time remaining
    stop_pub1.publish(stop)
    stop_pub2.publish(not stop)
    time_pub.publish(left_time)
    state_pub.publish(state1_msg)
    rate.sleep()

curr = time.time()

# Green at horizontal intersections, red at vertical intersections
while (time.time() - curr) < 10:
    stop = Bool(data=False)
    left_time = Float64(data=10 - (time.time() - curr))  # Time remaining
    stop_pub1.publish(stop)
    stop_pub2.publish(not stop)
    time_pub.publish(left_time)
    state_pub.publish(state2_msg)
    rate.sleep()
```




```python
rospack = rospkg.RosPack()
package_path = rospack.get_path('v2x_sim')
state1_path = f"{package_path}/maps/gr.png"
state2_path = f"{package_path}/maps/rg.png"  # Assuming you meant to have different images

# Load images
state1 = cv2.imread(state1_path)
state2 = cv2.imread(state2_path)
```

Lot H Course Sim

Accomplishments:

- Changing light states
- Light states recognized by cars
- Adaptive speed algorithm

Flaws:

- Lane Following
- Light states at runtime
- Not slowing down enough for red lights
- Stopping before crosswalk

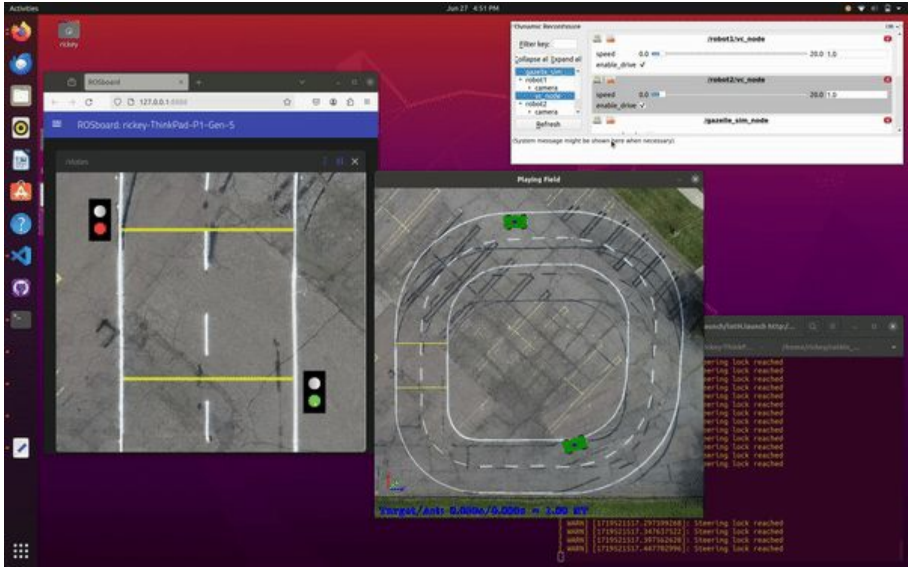


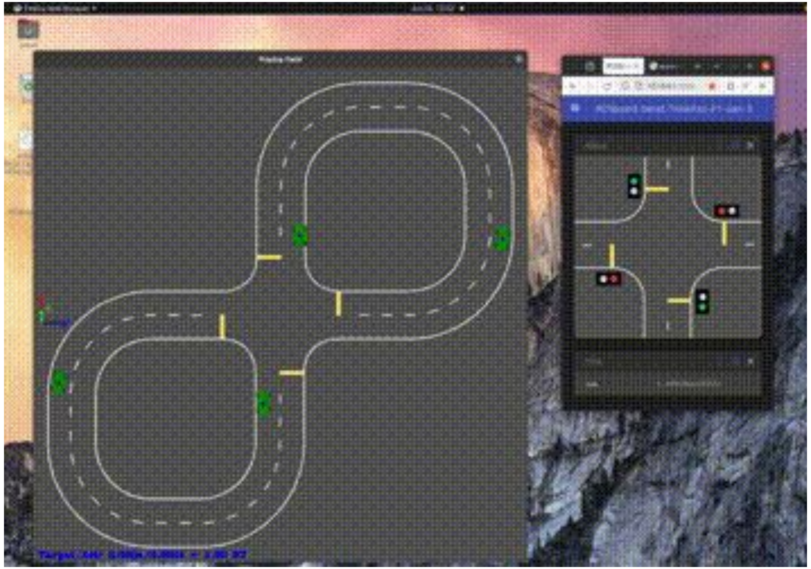
Figure 8 Sim

Accomplishments:

- Changing light states
- Light states recognized by cars
- Adaptive speed algorithm
- Working with 4 vehicles

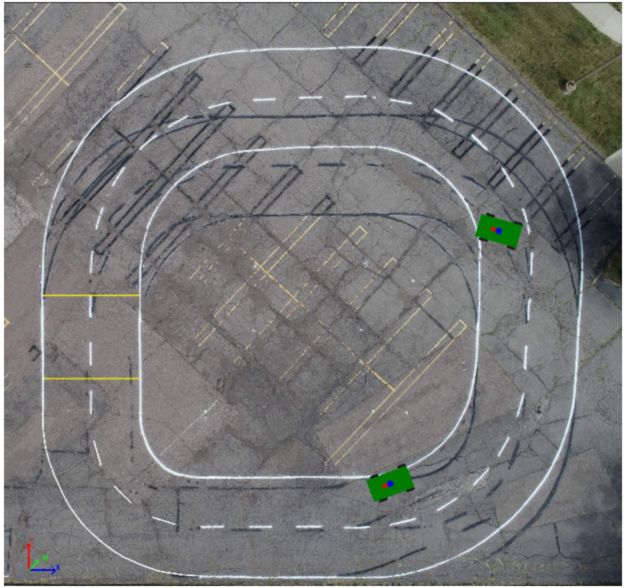
Flaws:

- Lane Following
- Light states at runtime
- Not slowing down enough for red lights
- Stopping before crosswalk



Future Goals for V2X Simulation

- Create a dependable lane following algorithm that works in both simulation and real-life
- Implement a yellow light functionality to avoid really slow speeds
- Adjust adaptive speed algorithm to never stop at red lights
- Begin adapting simulation to real life



Summary and Plans Moving Forward

Summary:

- A robust lane following algorithm developed; viable alternatives on the way
- Proof-of-concept RSU connection established
- Adaptive speed algorithm working within simulation for:
    - Both circle “Lot H” course and figure 8 “intersection” course
    - Red and green lights
    - Up to four cars

Future Plans:

- Modularize codebase to make issues more discrete (single-node architecture likely not maintainable)
- Begin testing all project components (RSU, lane following, adaptive speed algorithm) together outside of the simulation
- Begin working on writing lane following and V2X papers