# CARHAB SOLUTIONS:

## AUTONOMOUS DRIVING

CARHAB

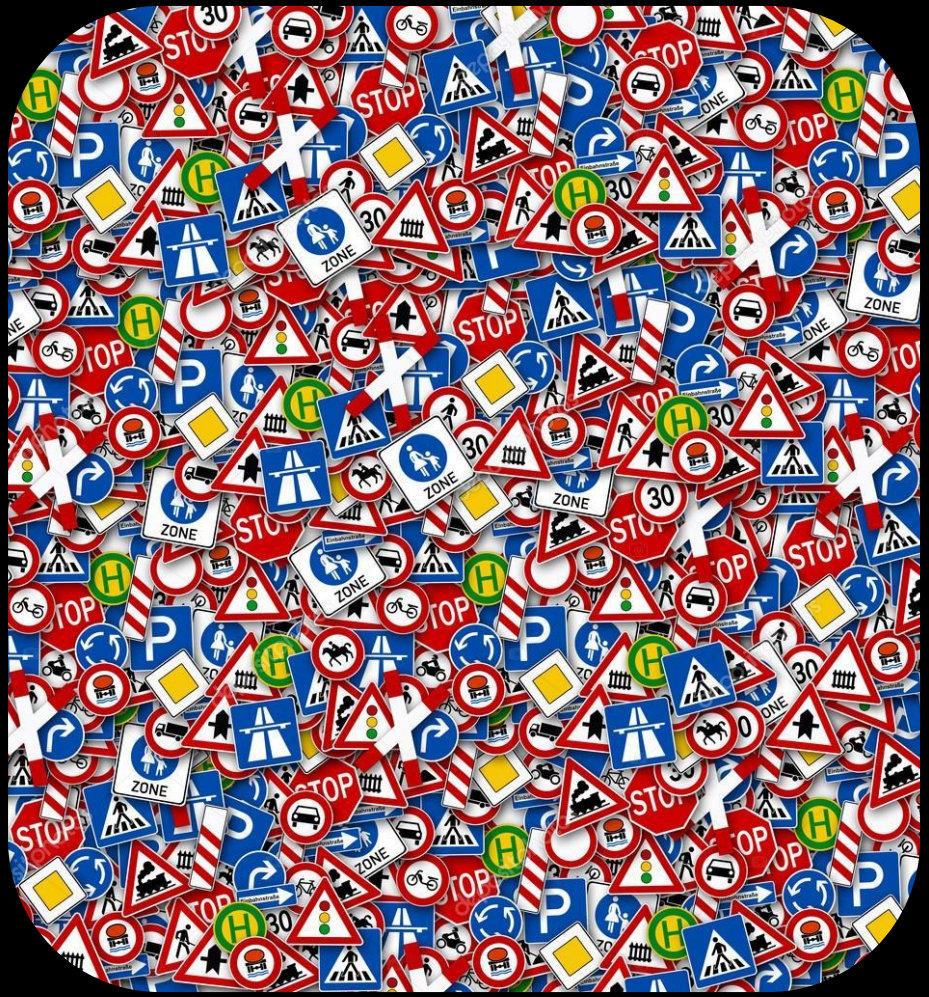
كارهاب

**Members:**

Logan Field, Pranav Nallapaneni, Nasih Al-Barwani, Mohammad Alburaidi

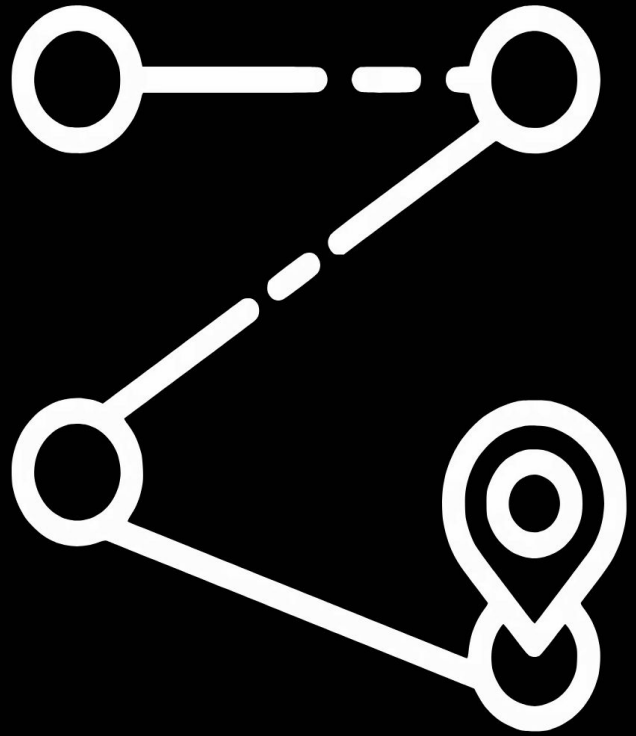
# 1

# PROJECT GOAL



# WAYPOINT BASED SELF DRIVING

- **Self-driving** vehicle that navigates using a **point-to-point** algorithm.
- Navigation **based exclusively on sign classification and depth** measurements.
  - Vehicle approaches the closest identified sign and executes instruction.
  - After executing an instruction, the vehicle locates the next sign.
  - **Signs act as waypoints** to define a course.
- Alternative to standard lane-following.
- Modular solution for small-scale autonomous vehicles



# Motivations:

## MODULARITY

- **Adaptable** to various small-vehicle applications
- **Easily modified** for different tasks (inventory management, indoor delivery)
- **Hardware agnostic** - can operate on laptops or specialized edge devices
- **Transferable skills** and code across different platforms

## ROBUSTNESS

- Functions independently of **lane markings**
- No reliance on specific environmental conditions
- Operates in **diverse settings**, indoor and outdoor
- Handles **variable surface** conditions

## NOVELTY

- **Departure from traditional** lane-following approaches.
- Sign-based instruction allow us to explore provide richer navigation commands.
- **Demonstrates consistency** in a more difficult aspect of self driving.
- **Contributes something new** to the project base of RC self driving.

**Detection and Classification**

A real time **classification model** returns bounding boxes and type for all signs in a frame.

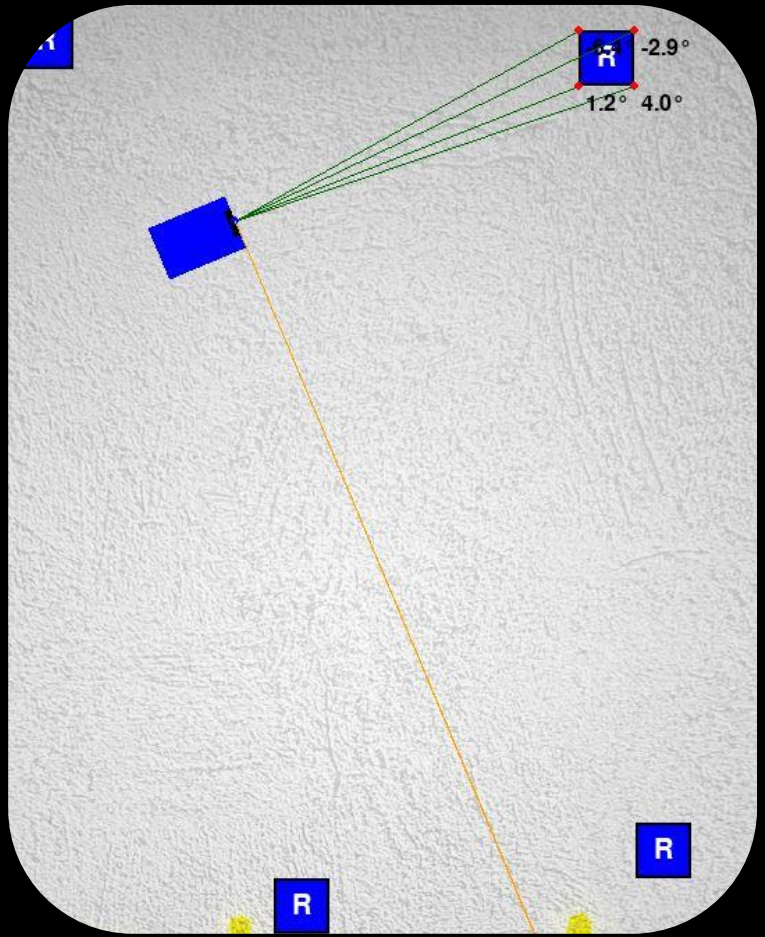
**Instruction Interpretation**

Bounding boxes, sign type, and ID are sent in the form of **SORT** trackers to an algorithm that converts the **sign type, depth and location into instructions** for the vehicle.

**Navigation**

The driving algorithm's centering and executing states are tuned so that signs can be **easily positioned** to create a **point to point route** for the vehicle to its destination.

# Fundamental Software Elements



# Scope and Limitations

## Scope

- **Small-scale** autonomous vehicle operation
- **Sign-based navigation** in controlled environments
- Support for **6 sign types** (left, right, forward, stop, caution, u-turn)
- **Point-to-point navigation** between signs
- Basic **obstacle detection** and avoidance

## Intentional Limitations

- **No lane following** functionality
- **Low speed** implementation
- Limited to **pre-existing signage**

## Technical Constraints

- **CPU-based** processing (no GPU acceleration)
- Limited by processing speed of laptop hardware
- Simplified environment with clear sign **visibility**

# Success Metrics

## Detection Performance

- **Accuracy** > 95%
- **Detection distance** ~4m
- **Consistent classification**
- **Low false positive rate**

## Navigation Performance

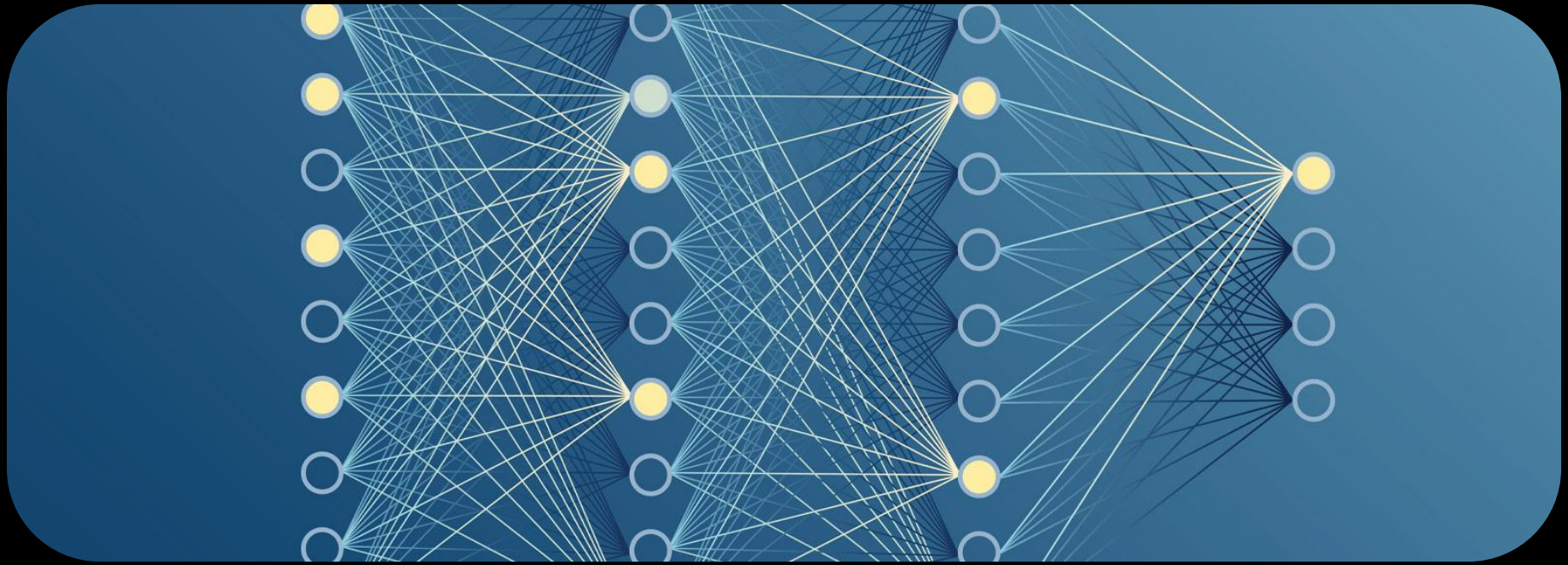
- **Completion** of multi-sign course
- **Execution** of dest sign instructions
- **Modifier signs** affect behavior
- **Obstacle detection** and avoidance

## System Integration

- **Reliable communication** between laptop and vehicle
- **Low inference time** ~0.1s
- **Real-time** processing of camera data ~10-15fps

## Implementation Quality

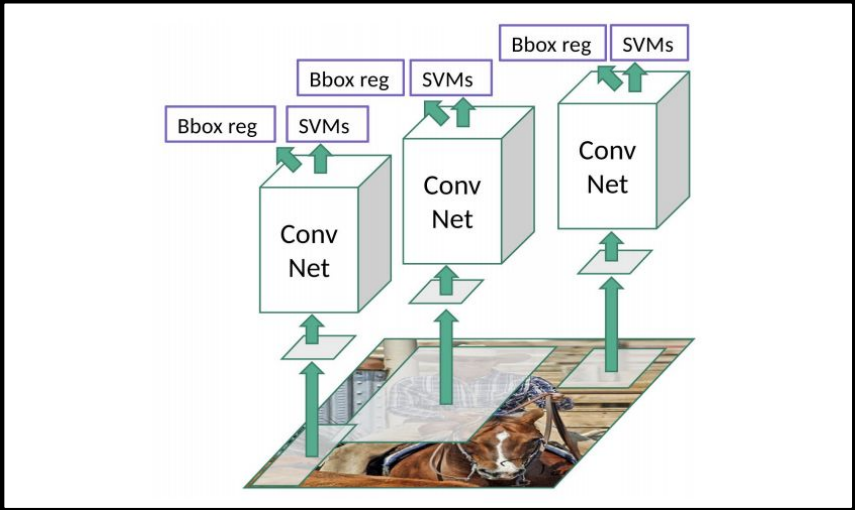
- **Modular** codebase
- **Minimal modifications** to base vehicle
- **Transferable** to other small-scale applications
- **Proper Documentation**



# CLASSIFICATION MODEL

2

# An Obvious Choice...

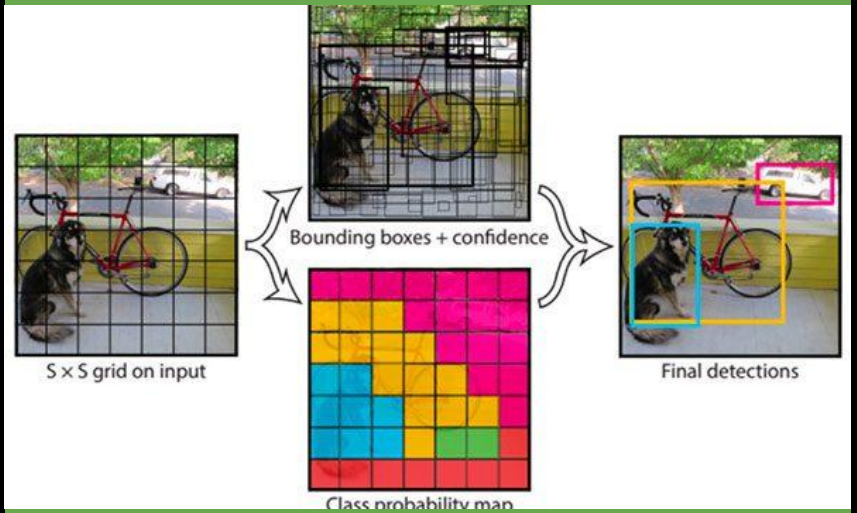



## R-CNN (Region-based Convolutional Neural Network)

- Two stages: Region Proposal, Proposal Classification.
- Pro: High Precision
- Con: Low speed

vs.

# Winner: Speed is Paramount!




## YOLO (You Only Look Once)

- Single Stage: Divides an image into a grid, assigns class probabilities in one forward pass
- Pro: Lower Precision
- Con: High speed

| | |
|---|---|
| **Lightweight** | Second lightest model in the YOLOv5 series Suitable for real time performance. |
| **Accurate Classification** | v5s has similar accuracy as the heavier models, but far better accuracy than v5n. |
| **Documentation** | YOLOv5 commonly used in object detection projects<br>Easy to access, reference, and adapt to our project. |

# YOLO Model Choice: YOLOv5s

# Dataset Classes

## Destination Signs

First define waypoints, then at depth threshold, movement instructions.

U-TURN

RIGHT

LEFT

## Modifier Signs

Modify how the above destination signs are approached.


STOP

CAUTION

FORWARD

01

## INITIAL DATASET

- GTSRB (German Traffic Sign Recognition Benchmark) Modified to only include the 6 signs used.
- 8000 Training images, 12,000 Validation images.

02

## CUSTOM DATASET

- Captured Frames of Printed signs in indoor environment through realsense camera.
- Annotated using Roboflow.
- 517 training images, 222 validation images.

03

## AUGMENTATIONS

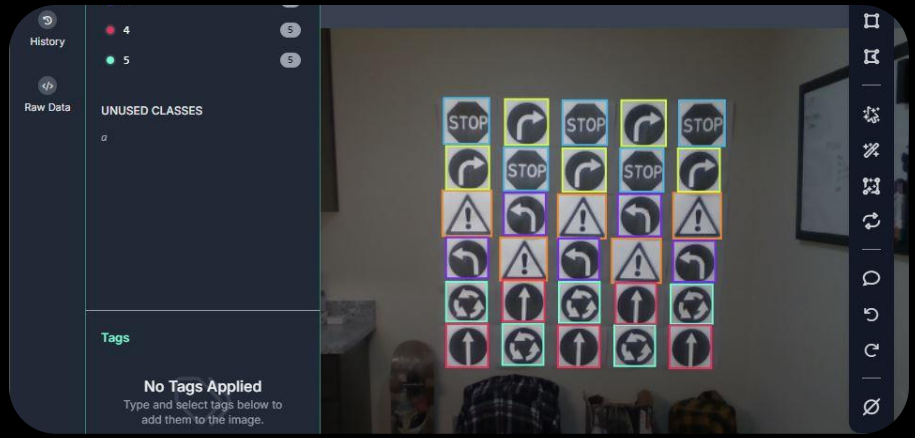
- GTSRB training done with scale, grayscale, rotation, saturation, and perspective augmentation.
- Synthetic data generated from custom dataset through crop, saturation, exposure, blur to generate a total of 1773 images for training

04

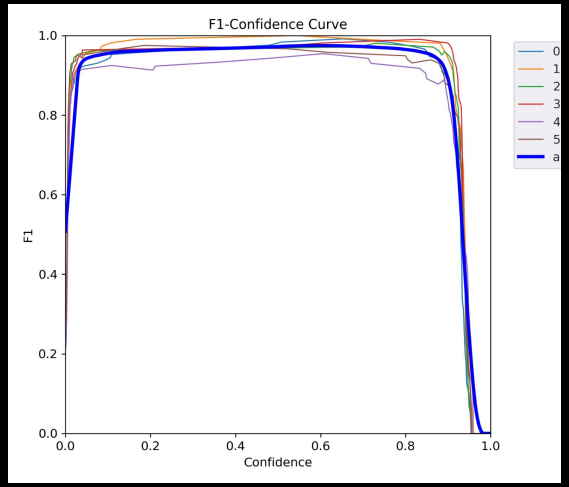
## MODEL TRAINING

Initial Dataset
- Image resolution (640)
- Batch size (16)
- Epochs (100)
- Starting weights: yolov5s.pt

Custom Dataset
- Image resolution (640)
- Batch size (16)
- Epochs (100)
- Starting weights: best.pt from initial dataset.

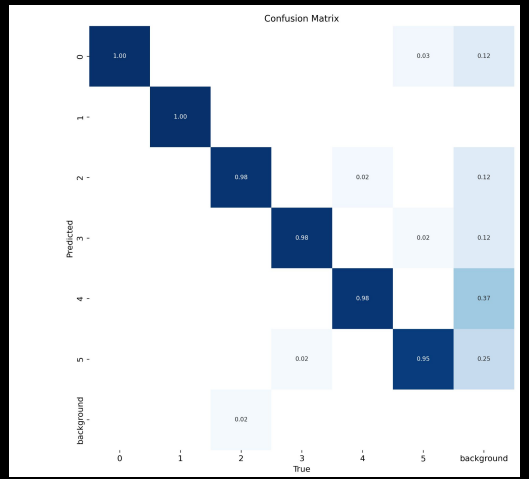


# MODEL PERFORMANCE



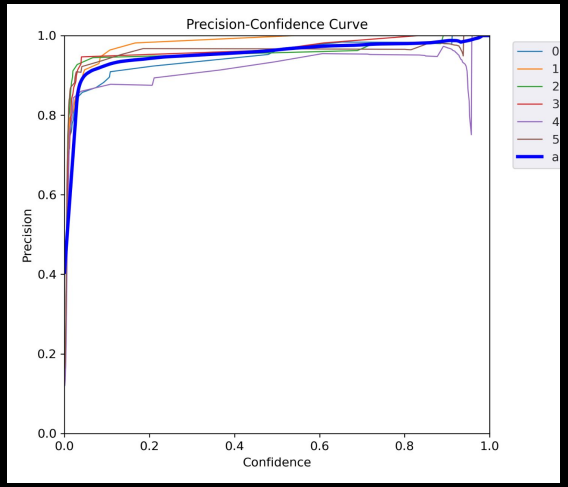

## F1 CONFIDENCE CURVE

Demonstrates consistent recall and precision across all confidence levels, representing low probability of false positives and negatives.




## CONFUSION MATRIX

Demonstrates low misclassification for signs, needs improvement in background misclassification




## PRECISION CONFIDENCE CURVE

Demonstrates precision that increases consistently as confidence threshold increases. Represents low probability of false positives.

# CHALLENGES ENCOUNTERED IN TRAINING

## POOR VALIDATION SET

- Lacking diversity in GTRSB dataset.
- Annotation inconsistencies
- Class skew

## YOLOV8 AND YOLOV11

- Compatibility issues
- Optimization challenges

## MISCLASSIFICATION

- Ambiguous Object Features
- Contextual Bias

## SCALE HYPERPARAMETERS

- Augmentation gaps
- Anchor box mismatch

# 3. PRACTICAL IMPLEMENTATION

# EXPLORATIONS IN COMPUTE PLATFORM

## NVIDIA JETSON TX2

Runs Ubuntu based Jetson OS
Not powerful enough to run Yolov5s model.
Able to control car, reflash, and run lighter model at slow framerates


## NVIDIA JETSON AGX XAVIER

More powerful platform, runs the same Jetson OS as the TX2
Not able to reflash or install SDK components

## ASUS ZEPHYRUS LAPTOP (FINAL)

Able to run Yolov5s model at ~15fps
Windows required different libraries and file pathing.
Overall more likely to be supported in the future.

# EXPLORATIONS IN VEHICLE CONTROL

## DIRECT CONNECT TX2 TO PCA9685

Pros: Simple, Edge AI implementation, no latency
Cons: TX2 not powerful enough to run models locally.

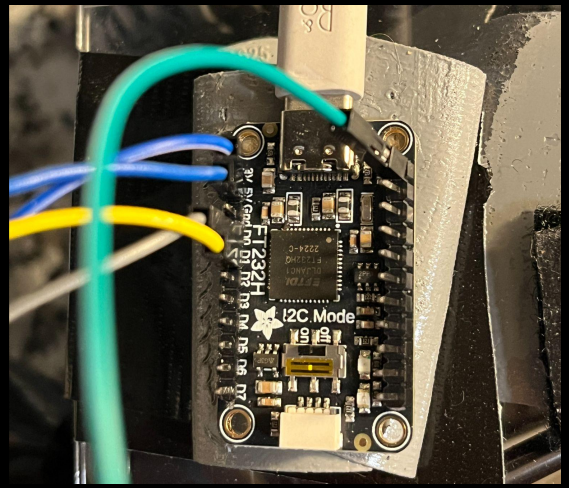
## UDP COMMUNICATION BETWEEN LAPTOP AND TX2

Pros: Allows for use of Laptop for running model, represents a distributed computing platform
Cons: High latency, overly complex

## LAPTOP TO BREAKOUT BOARD TO PCA9685 (FINAL)

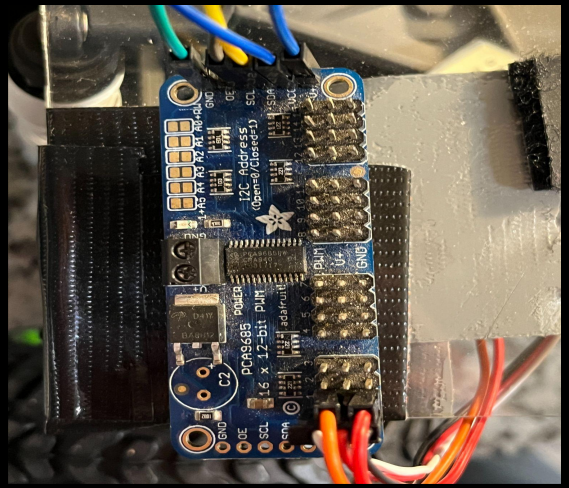
Pros: No latency, Laptop can run model, simple
Cons: Does not represent product level edge-AI product.

# VEHICLE SIGNAL PIPELINE




## FT232H

Breakout board from the computer.
Converts I2C usb signal to individually addressed wired PWM signals.
Provides power and ground pins.




## PCA9685

Carries over PWM signal from FT232H, sends throttle PWM to TRAXXAS ESC. Directly drives steering motor with USB 5V power rail.

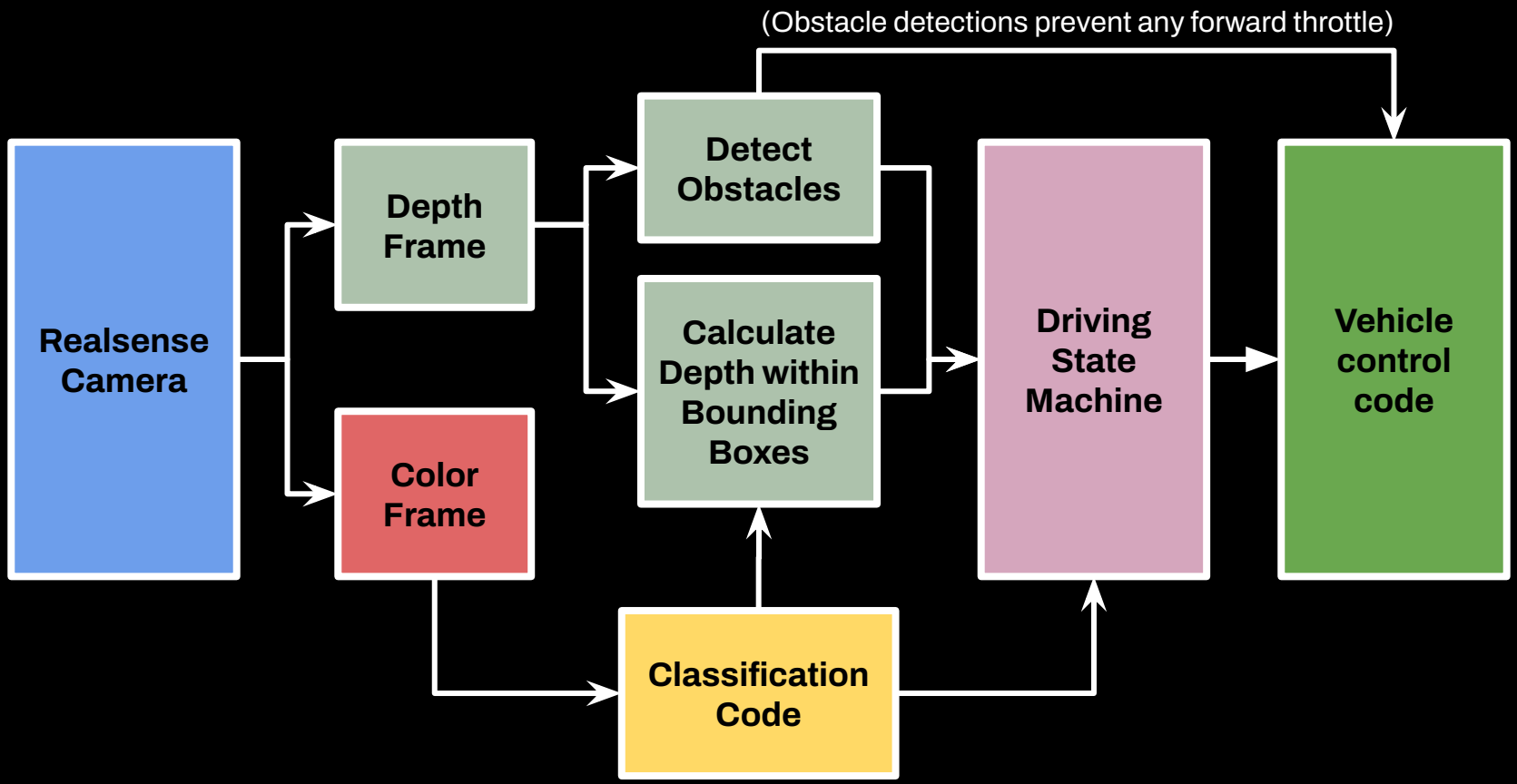

## TRAXXAS XL5 ESC

Converts PWM to throttle values for rear motor.
Requires initialization upon startup.

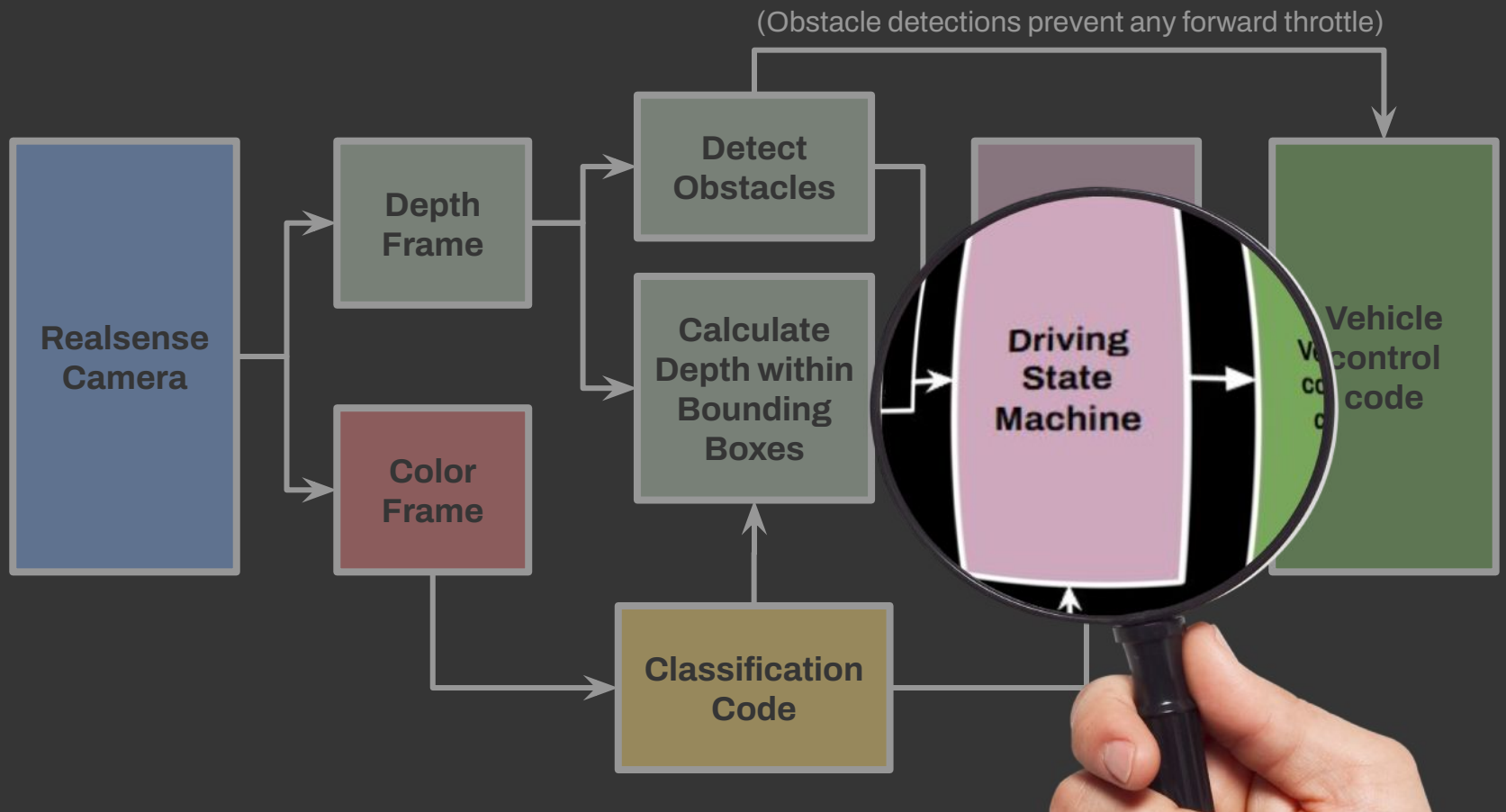
# DRIVING LOGIC

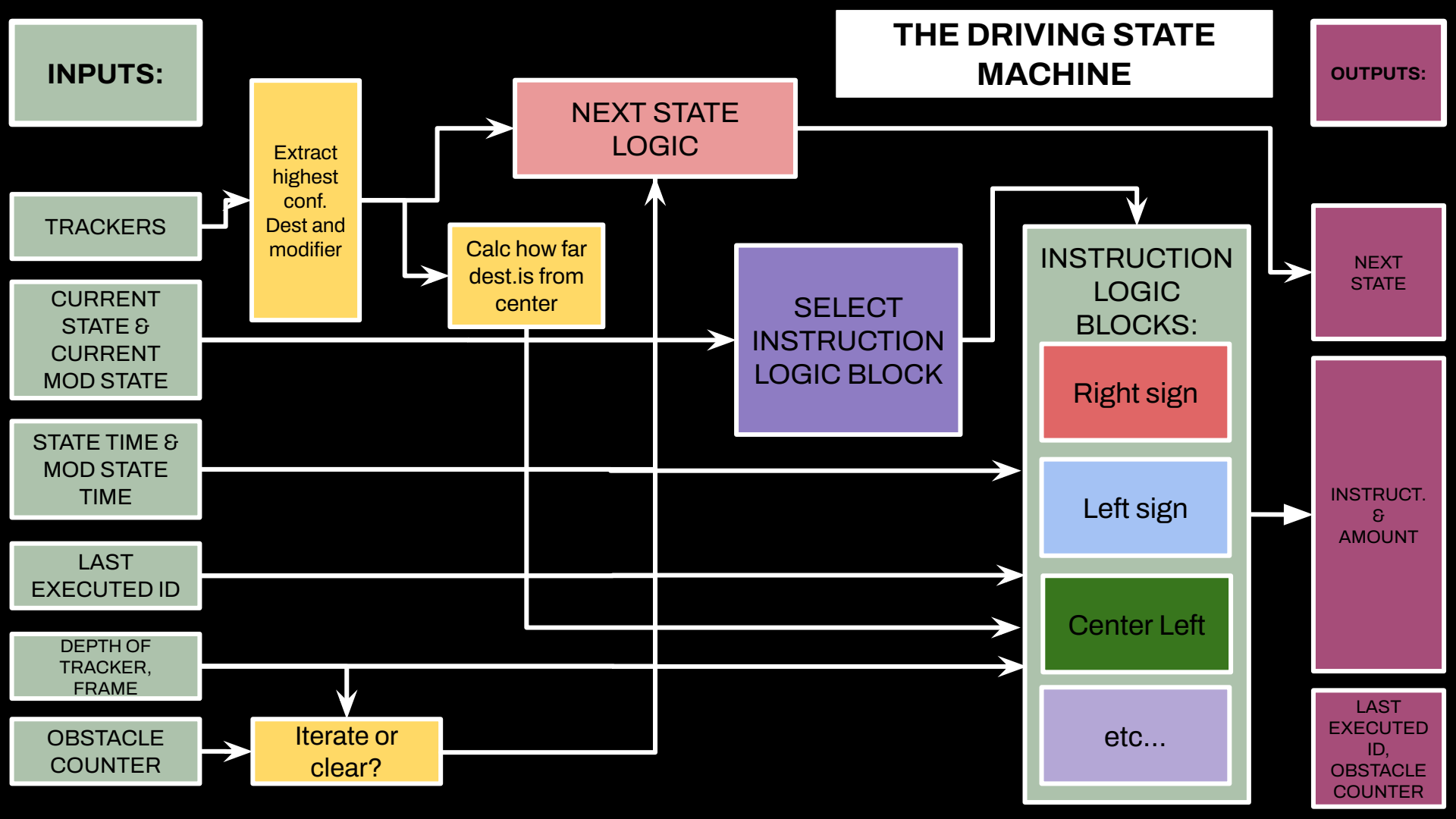
4

# INFORMATION PIPELINE



# INFORMATION PIPELINE




THE DRIVING STATE MACHINE
INPUTS:
TRACKERS
CURRENT STATE & CURRENT MOD STATE
STATE TIME & MOD STATE TIME
LAST EXECUTED ID
DEPTH OF TRACKER, FRAME
OBSTACLE COUNTER
Extract highest conf. Dest and modifier
Calc how far dest.is from center
Iterate or clear?
NEXT STATE LOGIC
SELECT INSTRUCTION LOGIC BLOCK
INSTRUCTION LOGIC BLOCKS:
Right sign
Left sign
Center Left
etc...
OUTPUTS:
NEXT STATE
INSTRUCT. & AMOUNT
LAST EXECUTED ID, OBSTACLE COUNTER

Q: Why use a state machine?
A: To understand, we must look one level deeper

# THE DRIVING STATE MACHINE

**INPUTS:**

- TRACKERS
- CURRENT STATE & CURRENT MOD STATE
- STATE TIME & MOD STATE TIME
- LAST EXECUTED ID
- DEPTH OF TRACKER, FRAME
- OBSTACLE COUNTER

Extract highest conf. Dest and modifier

Calc how far dest.is from center

NEXT STATE LOGIC

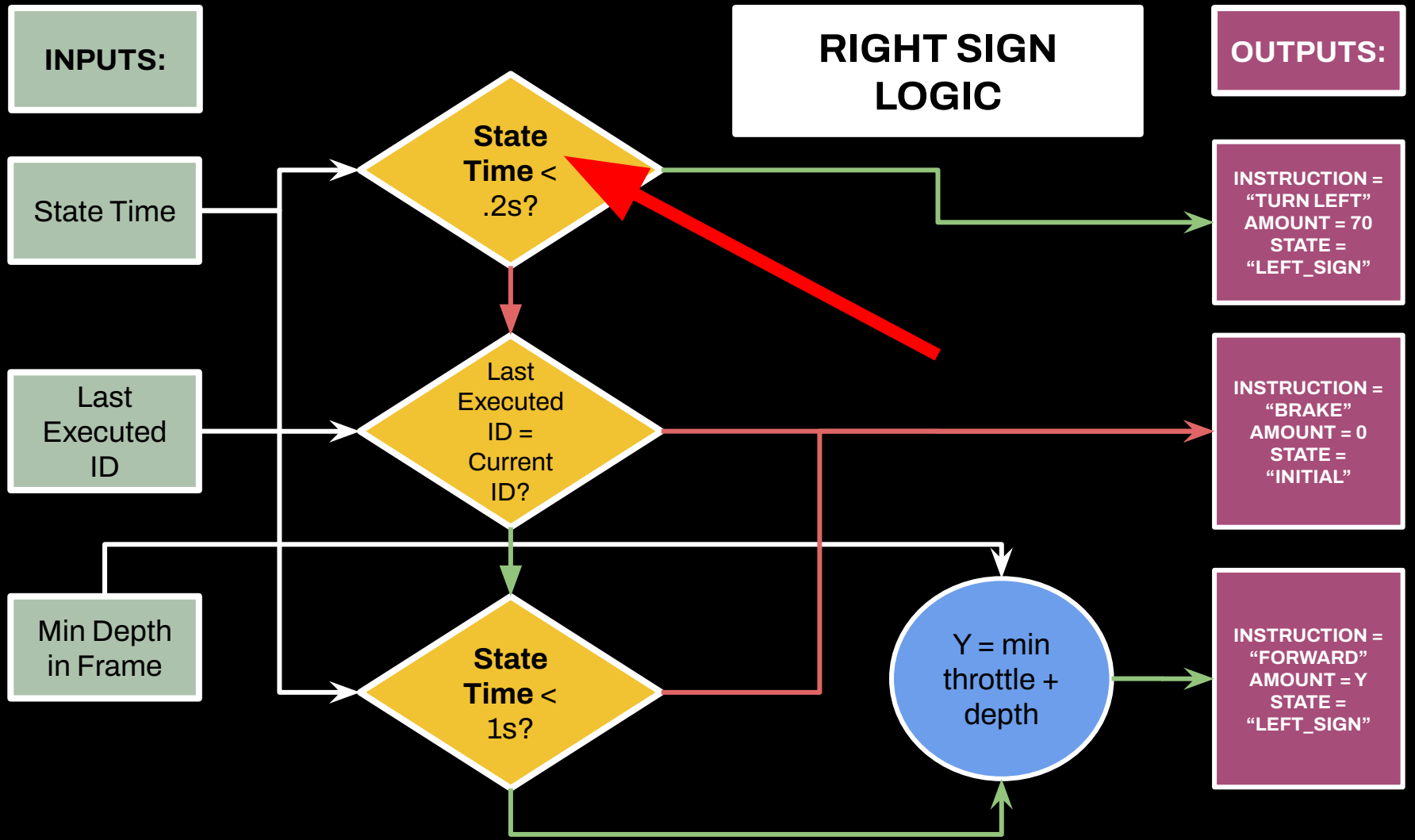
Iterate or clear?

SELECT INSTRUCTION LOGIC BLOCK

INSTRUCTION LOGIC BLOCKS:


**OUTPUTS:**

- NEXT STATE
- INSTRUCT. & AMOUNT
- LAST EXECUTED


INPUTS:
RIGHT SIGN LOGIC
OUTPUTS:
State Time
Last Executed ID
Min Depth in Frame
State Time < .2s?
Last Executed ID = Current ID?
State Time < 1s?
Y = min throttle + depth
INSTRUCTION = "TURN LEFT" AMOUNT = 70 STATE = "LEFT_SIGN"
INSTRUCTION = "BRAKE" AMOUNT = 0 STATE = "INITIAL"
INSTRUCTION = "FORWARD" AMOUNT = Y STATE = "LEFT_SIGN"

# Parallelism!

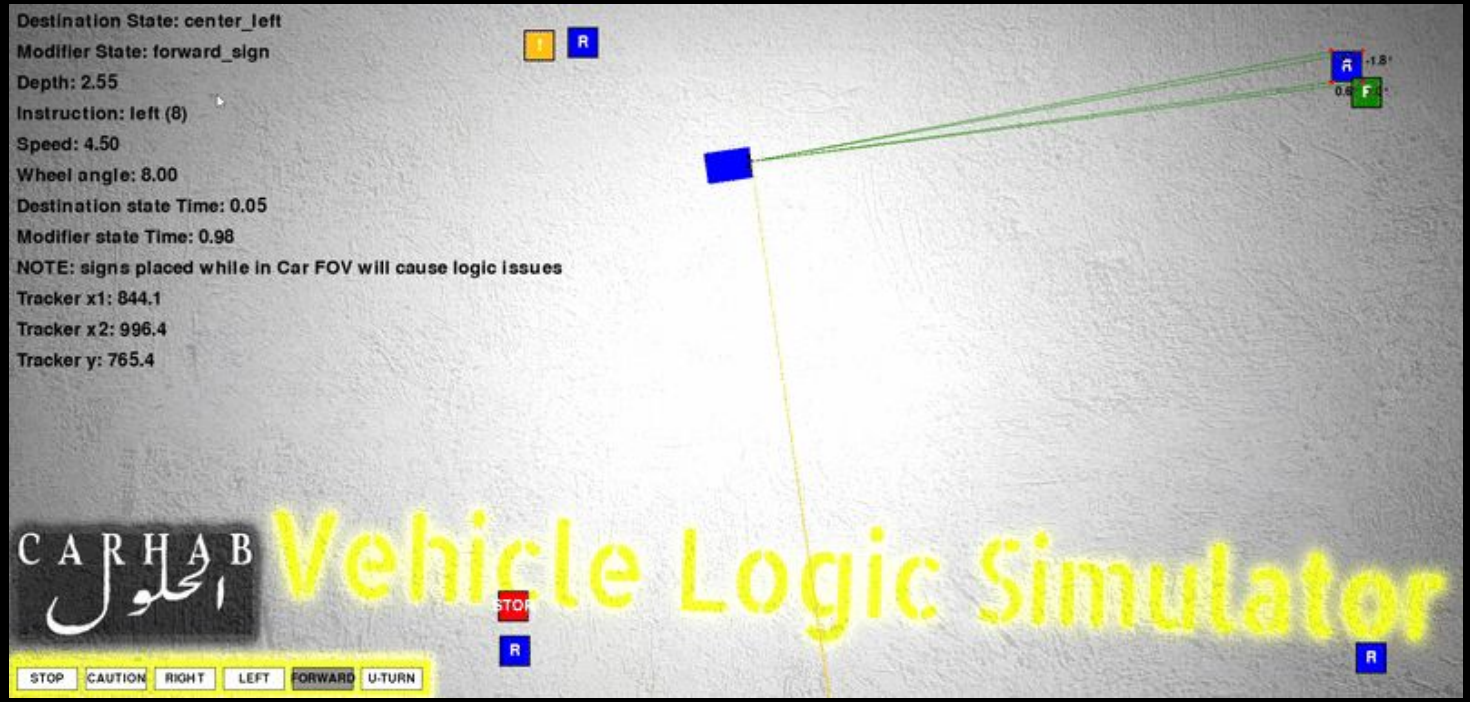
- Using a state machine provides us with **State Time.**
- **State Time** can be used to create delays between vehicle control commands—without having to delay the main function that calls the state machine.

Example: If I want the car to move forward for one second, I have to send instruction = “forward”, wait 1 second, then send instruction = “brake”

- With a standard delay, this would freeze the camera input for an entire second.
- Using state time, every main loop cycle that checks if a second has past, a frame can also be processed, creating almost zero delay.

# Q: How do you develop and test this without getting permanent back pain?

# A: Simulator



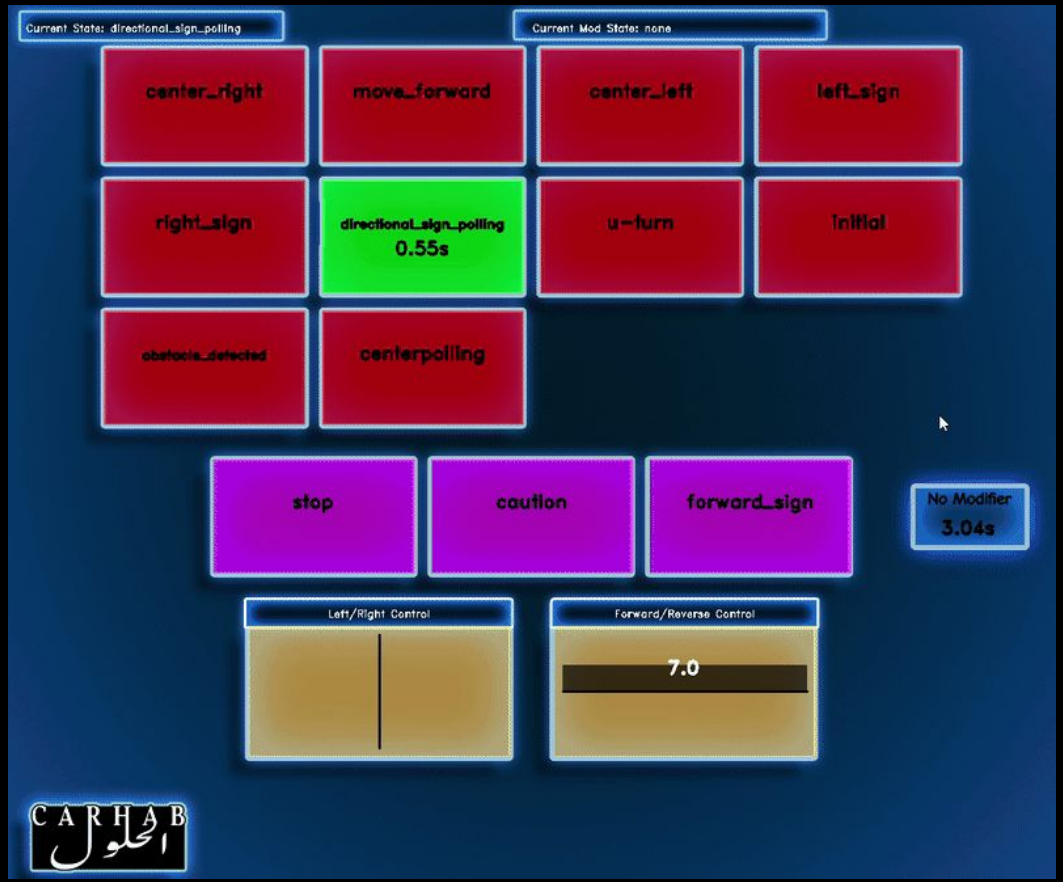

## Features:

Custom user placed signs.
Synthetic forward perspective tracker data generated from placed signs.
Acceleration, braking, turning modeled to match the style of the Traxxas vehicle.

## Most Importantly For Development…

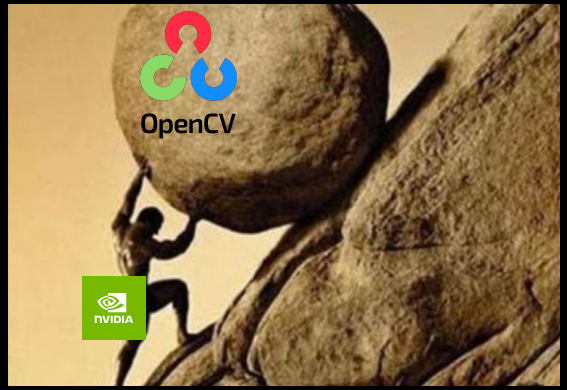
The simulation Interfaces **directly** with the same driving state machine module that the physical vehicle does--Changes to behavior in the simulation will be reflected in reality without the need to port code.

# Additional Debug Program: State Visualizer



# CHALLENGES 5

# JETSON TX2


## Compatibility

Due to the limited support for the device, library installations were a nightmare.
Most recent python version compatible with Jetpack is python 3.6

## Hardware Limitations

Crashed while attempting to run Tensorflow for GPU.
GPU not powerful enough to run Yolov5s.

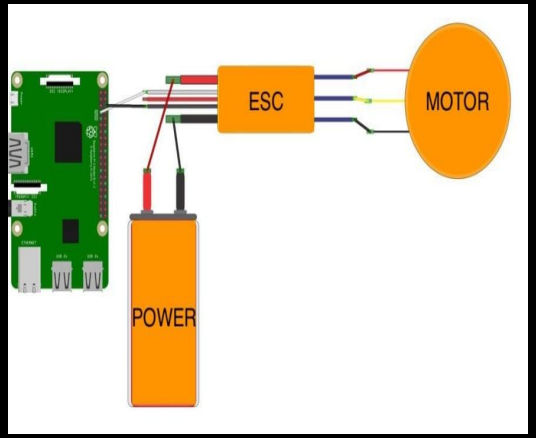
## Storage issues

10GB on board storage caused setbacks in development.
When attempting to change boot to SSD, the OS corrupted and it needed to be re-flashed.

# Vehicle


## Too fast

Car speed is set to high for normal testing.
Low torque gear ratio means car needs to get to speed to carry the weight of the laptop




## GPIO interface issues

Unreliable motor control.
ESC Initialization sequence very sensitive to timing, can change PWM instructions.

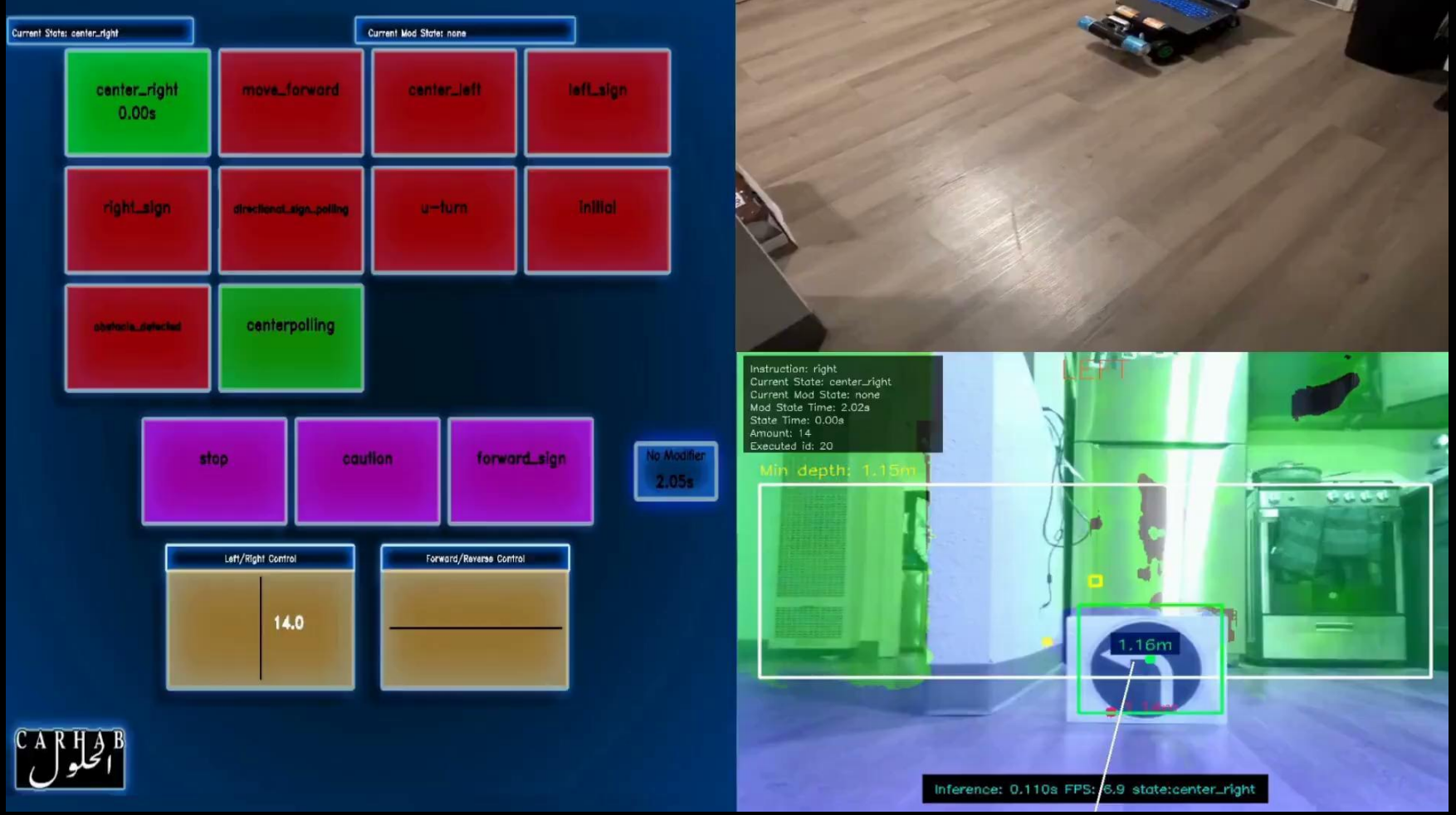
## Inconsistency

Inconsistent car maneuver leading to difficulty programming accurate timing in state machine


(Disclaimer: Picture  is not the Carhab Vehicle)

# DEMO

# 6




Full Course (footage is sped up 2x)

# BONUS DEMO: OBSTACLE DETECTION

Vehicle is able to manage tighter spaces through use of the obstacle detection state.

When depth is below the threshold for a certain number of cycles, straightens wheels and reverses.

Combined with re-detecting signage, this allows it to perform multi-point turns

# 7 Concluding Remarks