Autonomous Self Driving Vehicular Project 2023

RASCAL: Robotic Autonomous Scale Car for Adaptive Learning

RUTGERS
WINLAB | Wireless Information Network Laboratory

Adarsh Narayanan (UG), Joshua Menezes (UG), Christopher Sawicki (UG), Tommy Chu (UG), Brandon Cheng (UG), Ruben Alias (UG), Aleicia Zhu (HS), Ranvith Adulla (HS), Arya Chhabra (HS), Suhani Sengupta (HS)

RASCAL Overview

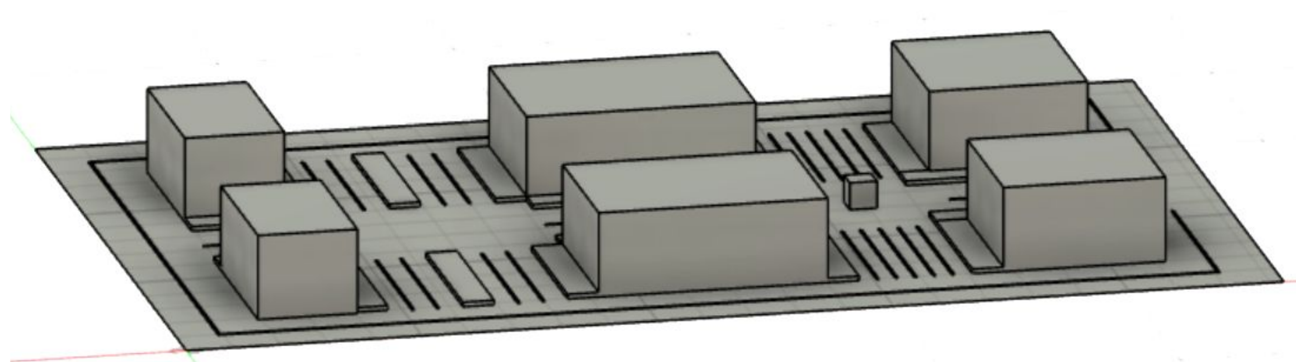
In our project, we hope to provide greater simulation accuracy with our design for a ~1/14 scale platform for miniature smart cars for repeatable tests in the smart city environment. Our design includes mechanisms such as Ackerman steering and differential gearing system to emulate the behavior of real cars, as well as sensors to calculate feedback. Using this car, we trained an imitation learning model to drive using camera input.

Goal

Create a self-driving car that emulates the mechanics of a real car, controlled autonomously by an imitation learning algorithm

Methodology

- Prototype 3D printed car
- Control hardware components
- Record training data
- Train machine learning model to associate control to image



Smart City Model

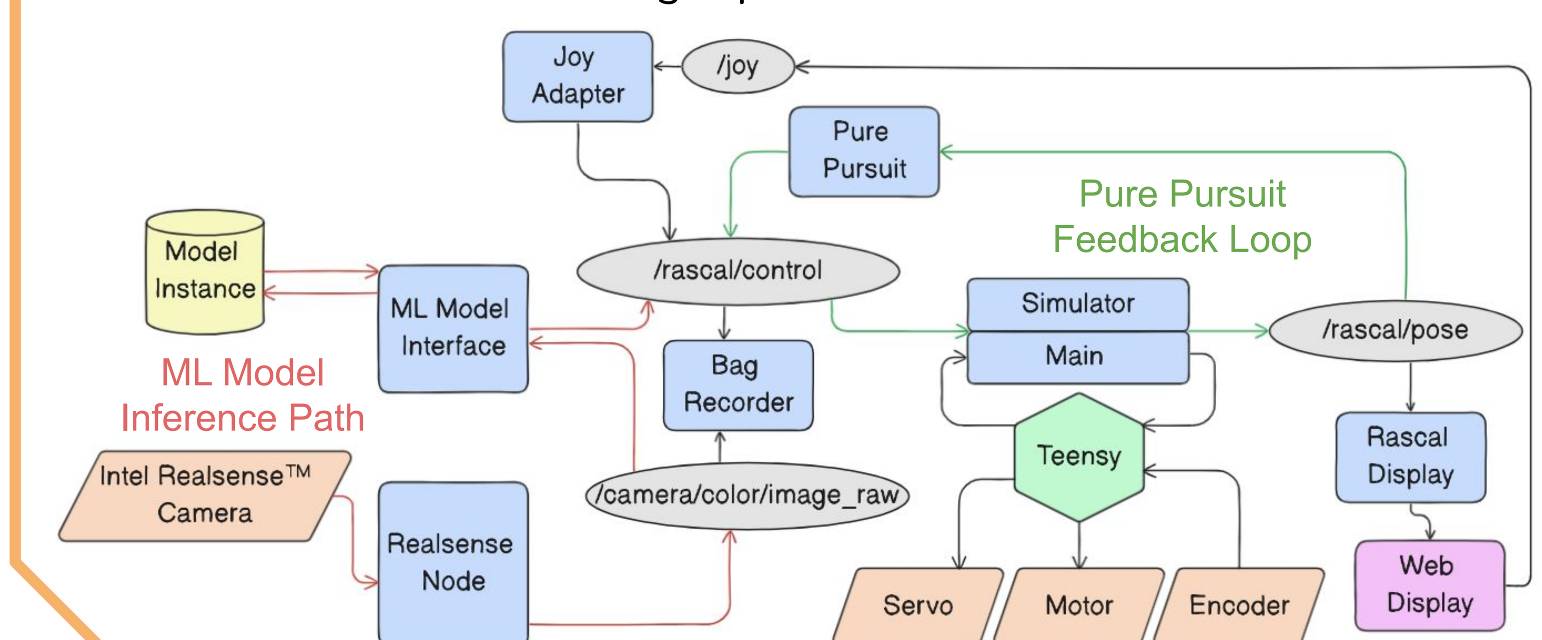
Software Architecture

ROS - Robot Operating System

- Project runs code in ROS **nodes**, which communicate using **topics**

Modularity

- Nodes can be interchanged while others are kept the same
- **Main** can be replaced by the **Simulator** node, for parallel development without hardware
- Specifics for Arduino hardware can be found in the 2022 "Mini Smart Car Hardware Design" poster



Software Architecture Diagram

Imitation Learning Model

Goal

Train a PyTorch machine learning model to drive using video input

Data Collection

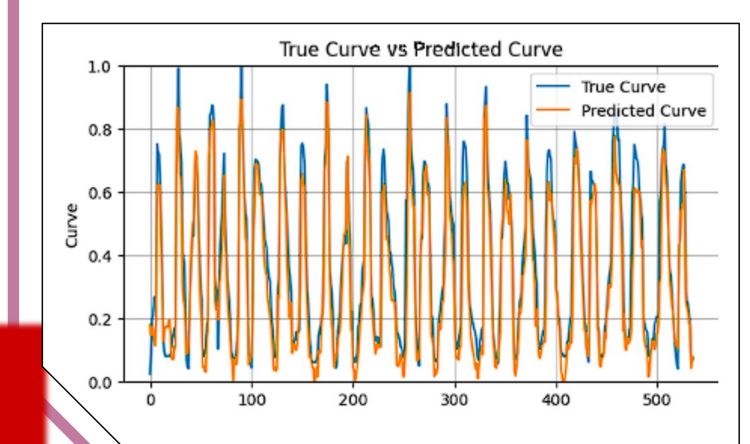
- Drive with pure pursuit algorithm
- Collect training data using the camera and control input
- Synchronize data to associate image frames with control input
- Clean the data and split it into training and test sets

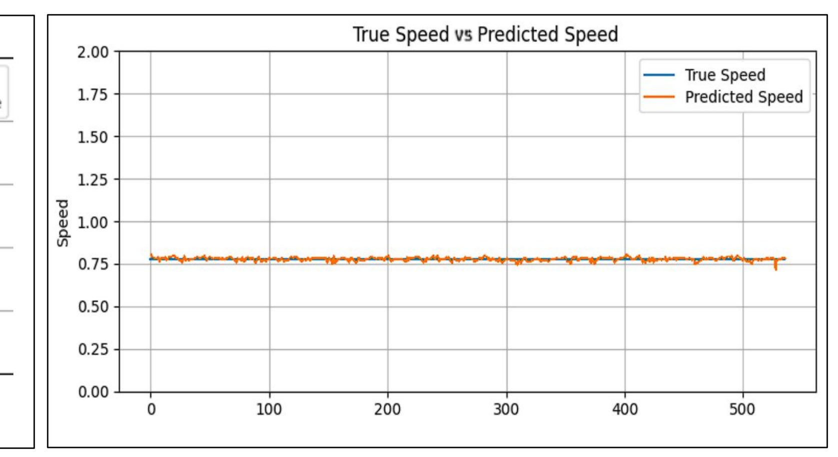
Training the Model

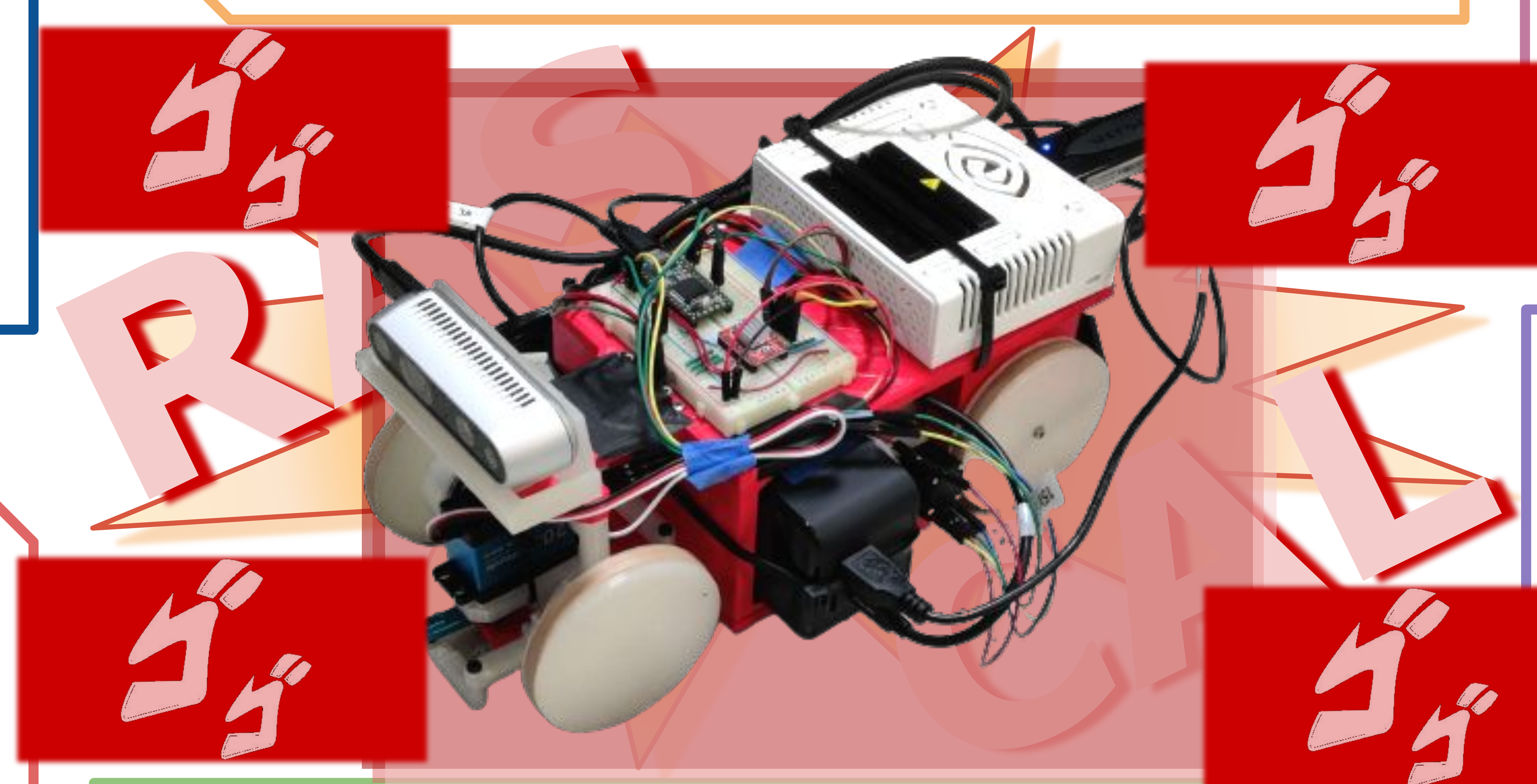
- Train model to learn speed and curve associated with an image
- Output speed and curve when given a **new** image
- Compare with expected value (output from pure pursuit)

Running the Model

- Give the model live image feed and receive signals to control car



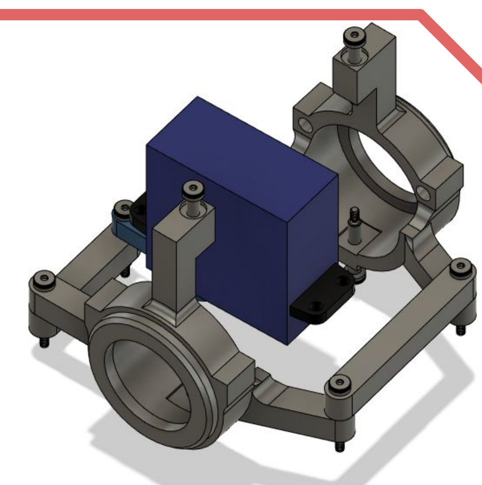


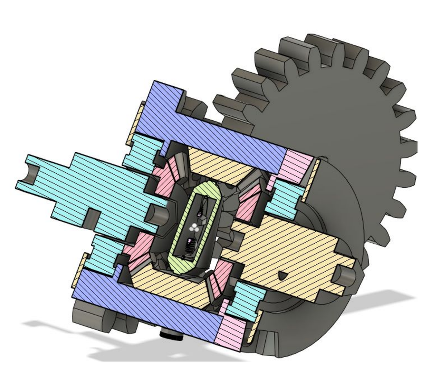


Physical Design

Ackerman Steering

- Inside wheel travels in an arc with a smaller radius during turns
- Allows car to turn front wheels at correct angles
- Eliminates skidding during turns





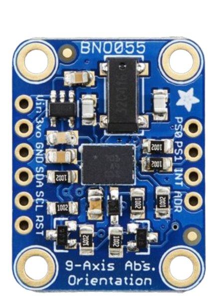
Differential Gear System

- Back wheels are powered by stepper motor
- Allows left and right wheels to turn at different speeds, to accommodate different turn radii

Additional components added since *2022 Smart Car Hardware Design* project:

Adafruit BNO005 Absolute Orientation Sensor

- Orientation data
- Used for odometry

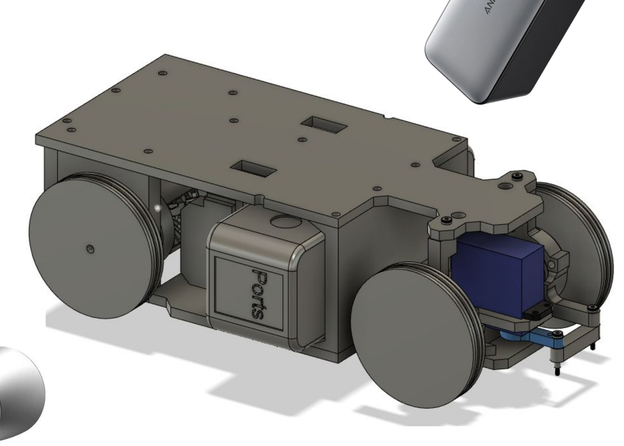


Anker 737 Power Bank

- Improved battery for addition of camera

Intel RealSense D435

- RGB and depth camera

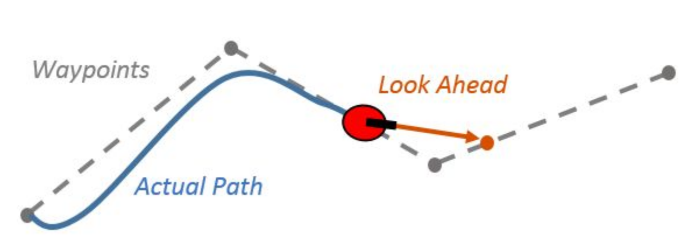


3D Model of RASCAL 2.0

Pure Pursuit & Odometry

Motivation

- Pure pursuit drives much more consistently than manual control
- Useful for gathering training data

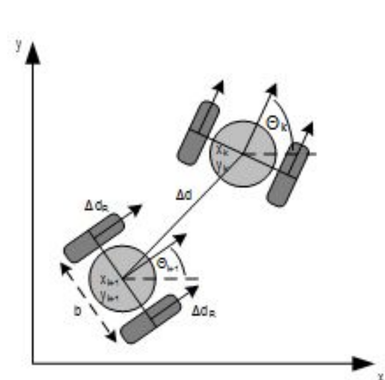


Pure Pursuit

- Finds a lookahead point on a path given the path and current position
- Internal calculations drive the car in arcs to the lookahead point

Odometry

- Uses car angle (from IMU) and distance traveled (from encoder) to calculate the car's position with arcs
- Use this feedback to constantly adjust back on to the path



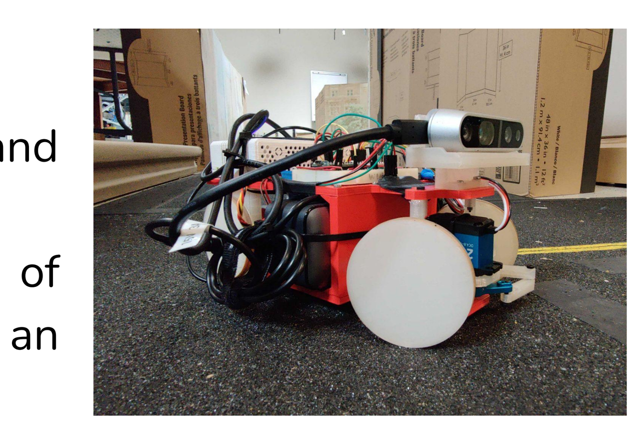
Conclusion & Future Work

Conclusion

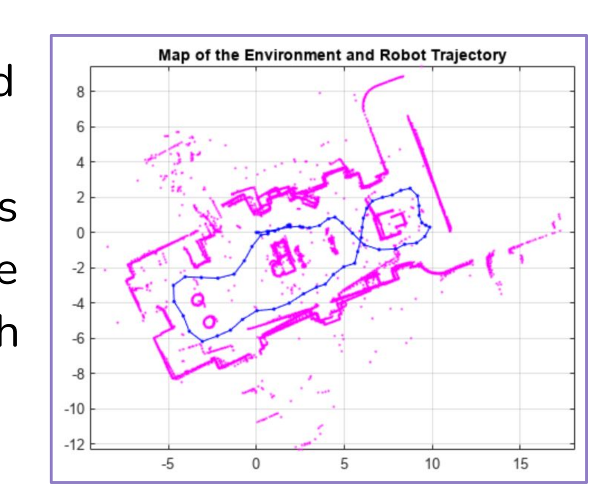
Our hardware and software redesigns resulted in vastly improved performance in operation. Using pure pursuit, we were able to accurately follow a defined path with manageable drift, and gather data to train our imitation model to drive in a loop in the smart city.

Future Work

- Improve design for easier assembly and mass production
- Incorporate an additional input of desired action, such as left or right at an intersection
- Process data with time or previous state, like stopping at stop signs or waiting for traffic lights
- Train to output a desired point instead of direct controls and use pure pursuit
- Implement SLAM (Simultaneous localization and mapping) to improve accuracy of odometry using depth camera



Driving in the Smart City



SLAM

This work was supported in part by the NSF REU program and the donation from nVERSES CAPITAL

WINLAB

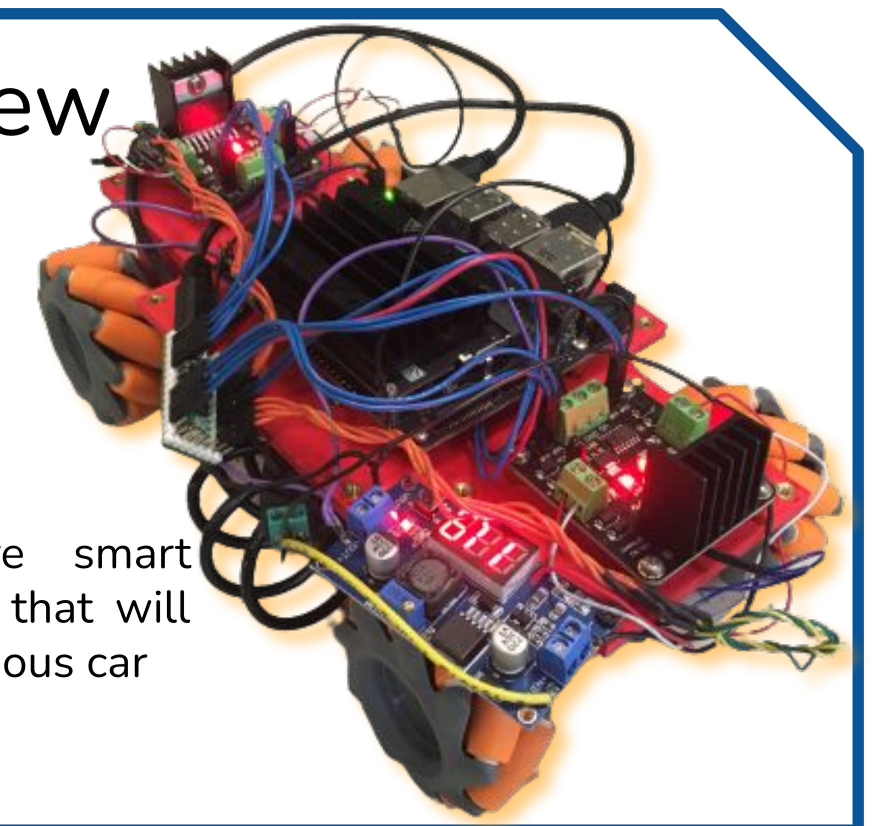
RUTGERS
WINLAB | Wireless Information Network Laboratory

Autonomous Self Driving Vehicular Project 2023

SCAMP: Self-guided Computer Assisted Mecanum Pathfinder

Adarsh Narayanan (UG), Joshua Menezes (UG), Christopher Sawicki (UG), Tommy Chu (UG), Brandon Cheng (UG), Ruben Alias (UG), Aleicia Zhu (HS), Ranvith Adulla (HS), Arya Chhabra (HS), Suhani Sengupta (HS)

SCAMP Overview

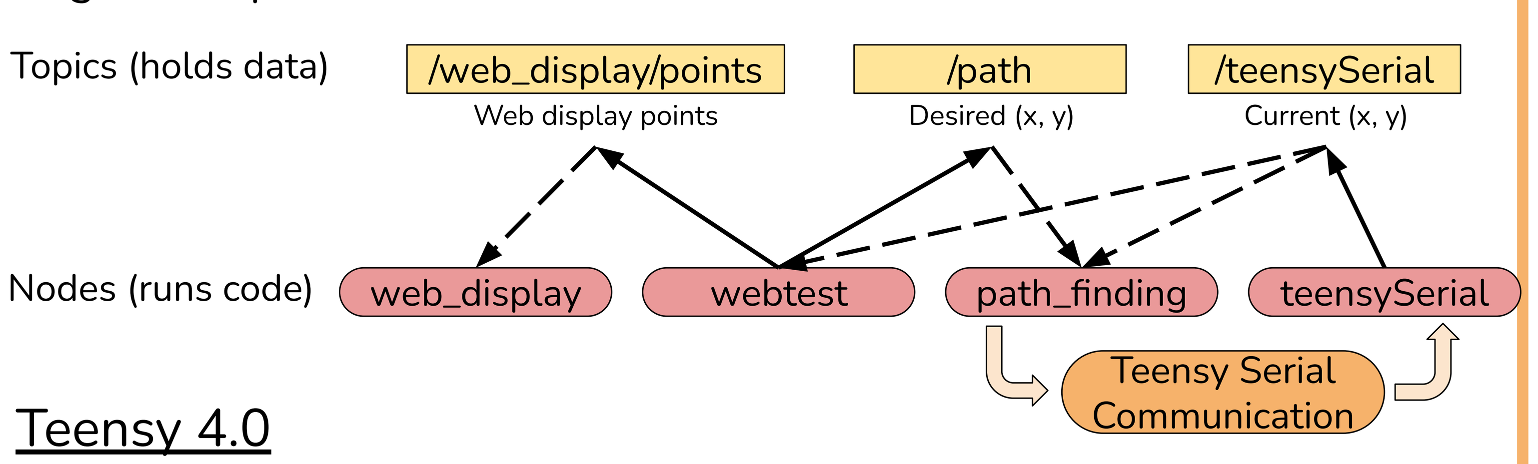


Goal
Develop a remote-controllable car to mimic the path taken by a car in a real intersection.

Purpose
Bring life to the miniature smart intersection with dummy cars that will simulate traffic for our autonomous car RASCAL.

Software

Jetson Nano
The software framework of the car is built on the Robotics Operating System (ROS) environment, which uses “nodes” to run different sections of code in parallel. Each node is responsible for its own concern and communicates with other nodes by “publishing” data and “subscribing” to global topics.

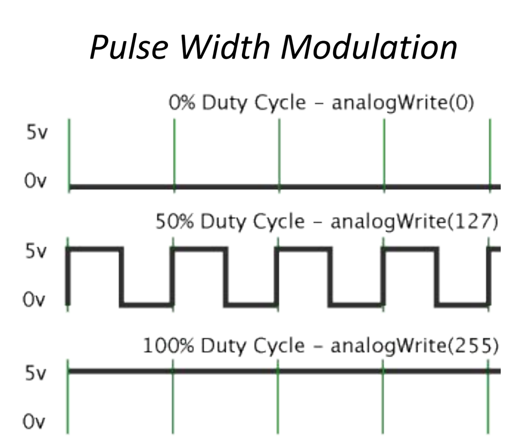


Teensy 4.0
The Teensy runs a main loop that receives desired position data from the Jetson Nano over serial. It then makes calculations based on its current position and angle before sending desired speeds to the four mecanum wheels.

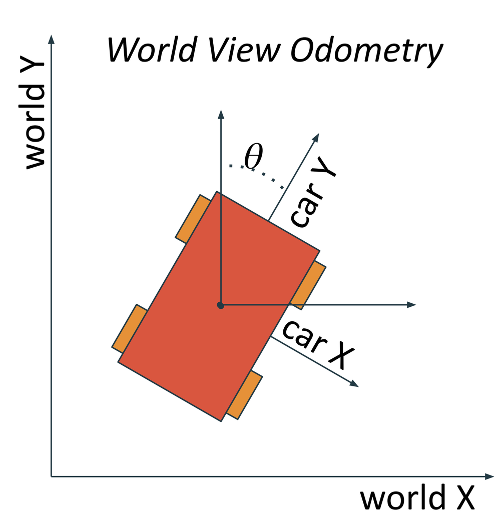
Handling Motor Speed
Motor speed is calculated by finding the time t it took for the wheel to rotate by d encoder ticks (usually small). Using such a small interval for distance is possible because of the Teensy’s microsecond accuracy.

d encoder ticks	1 rotation	1,000,000 μs	60 s
t μs	~1420 encoder ticks	1 s	1 min

The motor controller sends power to the motor based on an analog signal achieved through pulse-width modulation (PWM).



Proportional-Integral (PI) control is used to dynamically control motor power based on desired speed minus current speed. It is necessary because motor power for different motors do not necessarily correspond to the same speed due to manufacturing variability.

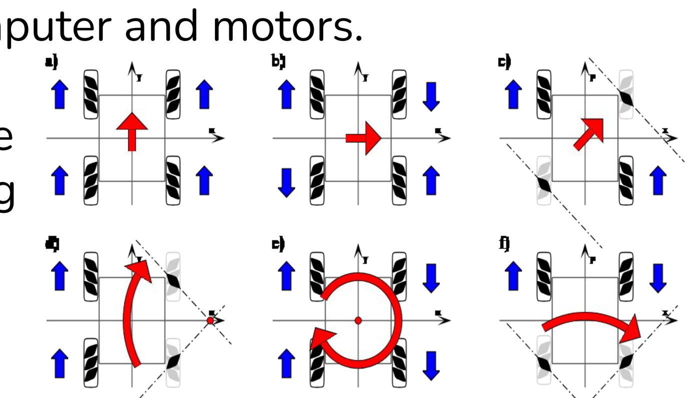


Relative vs World Coordinates
SCAMP is able to calculate its position relative to a world axis using encoders for relative forward-backwards and side-to-side movement and the IMU for orientation data. The world position is used for position adjustment and for display on the web server.

Hardware

Mechanical
The frame of SCAMP employs a 3D printed body with metal screw inserts. The bottom of the frame houses an **Anker 737 power bank**, which delivers power for on-board computer and motors.

Four **65mm mecanum wheels** give the car omni-directional motion, allowing for adjustments during path-following



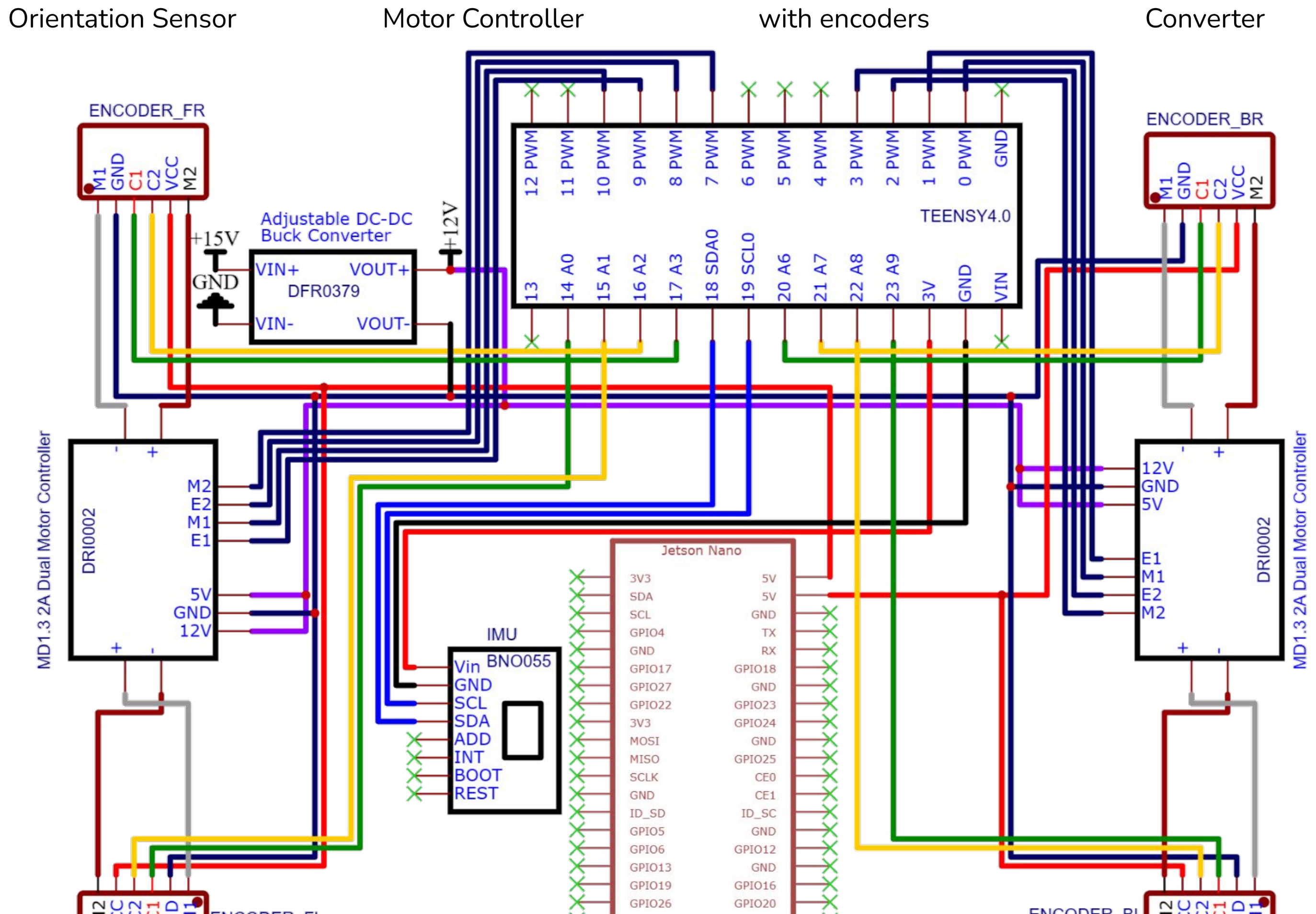
Electronics
The **Jetson Nano**, the on-board computer, runs the ROS software platform, which handles calculations and communication between itself, the microcontroller, and a web display

A **Teensy 4.0 microcontroller** is connected with a micro USB cable, which provides power and allows for serial communication

The **Adafruit BNO005 Absolute Orientation Sensor** is an IMU (Inertial Mass Unit) that sends orientation data to the Teensy through I^{2}C serial communication

Two **DR10002 MD1.3 2A Dual Motor Controllers** each control the front and back pair of **12V Brushed DC Motor with encoders**

Since the motors operate at 12 volts, the **DFR0379 DC-DC Buck Converter** is used to step down 15 volts from the battery



Electrical Schematic of SCAMP

This work was supported in part by the NSF REU program and the donation from nVERSES CAPITAL

Web Display

Problem
Using SSH (Secure Shell) Protocol to access RASCAL and SCAMP’s computers through command-line means that there is no visual interface. It would be better to have a way to display positions and paths visually.

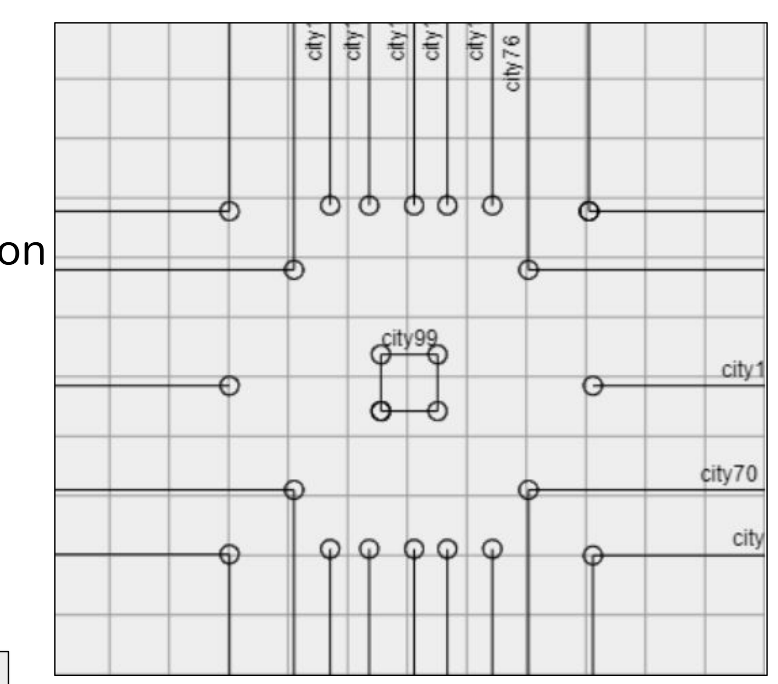
Solution
Host a web server from the car to display a webpage for visual elements. This server can be accessed remotely using a browser through the car’s Internet Protocol (IP) address.

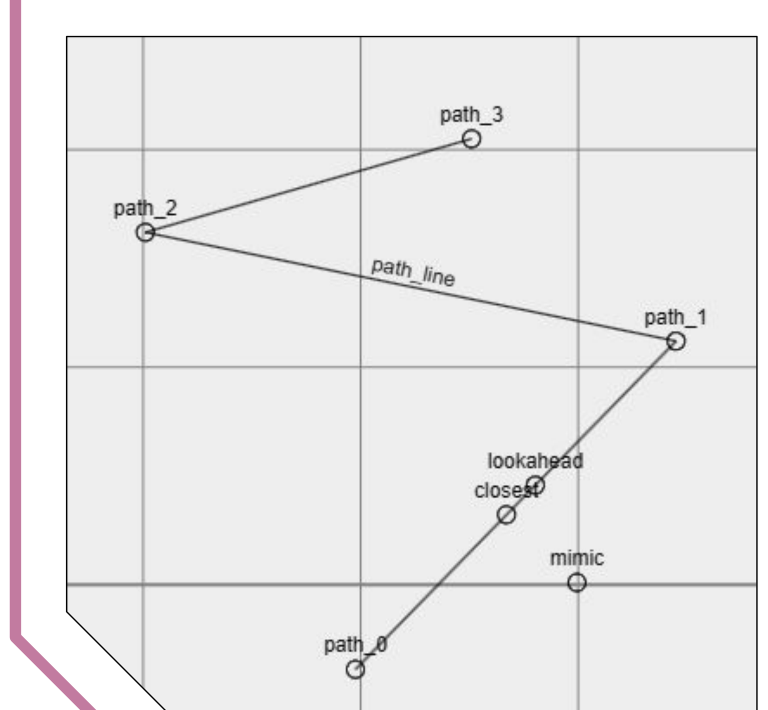
Methodology
The web display runs as a ROS node which starts a Python Flask server. This node listens to various topics that allow any ROS node to publish to and listen from the web display.

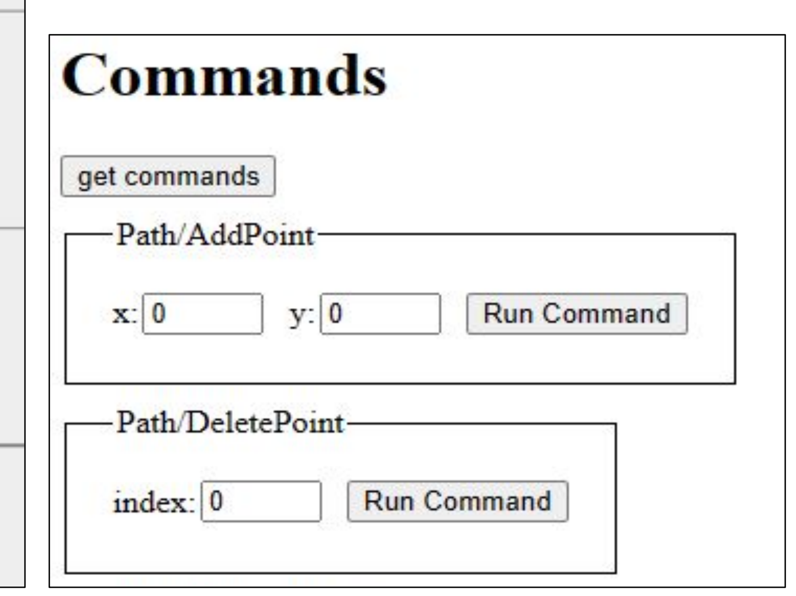
ROS nodes can also publish commands that automatically populate the page with forms for the user to input parameters and run functions.

Features
Interactive x-y graph:
- Display points and lines
- Add, remove, and drag points
- Map of miniature city intersection

In-built and custom commands:
- Save and load points for paths
- Pure-pursuit point service
- Custom commands using ROS







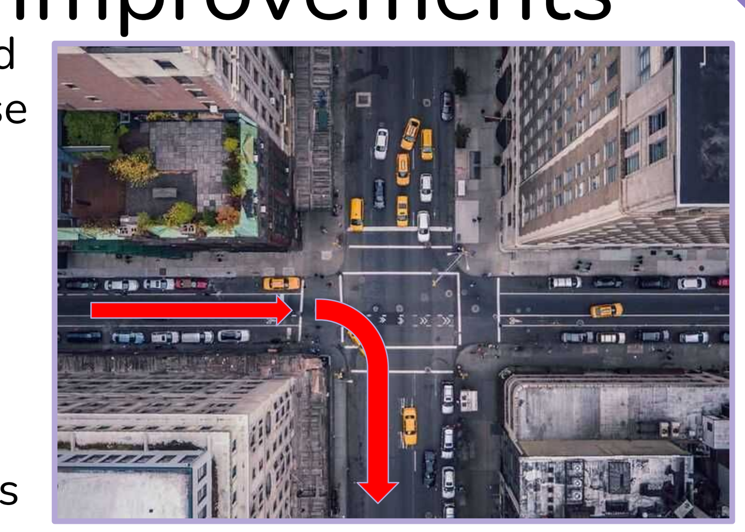
Future Work and Improvements

→ Simultaneous localization and mapping (SLAM) using a Realsense Camera for self orientation

→ Path extraction from video

→ Instantaneous speed parameter

→ Integrate intersection cameras



WINLAB