

AutoDrive Design Specification

Team Pink Flamingo

Jianjie Liu (CS)
Michael Morscher (ENP)
Phoebe Yang (CS)

Sponsors

JK Pollard
Jivko Sinapov

January 31, 2019
Version 1.1

Table of Contents

Table of Contents	1
Version Updates	2
Introduction	3
Limitations	4
Research Approach	5
System Safety & Behavior	5
Self-Driving Model Frameworks	6
Stakeholders, Needs & Requirements	7
Materials & Technologies	8
Software Modules Developed	9
Data API	9
Control API	9
Deep Learning Model	10
Functionality	11
Current Functionality	11
Planned Features	11
Proof-of-Concept Product Design	12
Car Prototype	12
Simulation	12
Proof-of-Concept Integration	13
References	16

Version Updates

Version 1.1 (February 1, 2019)

The updates to this version include a modified System Safety & Behavior diagram reflecting planned changes in the scope of Spring 2019 research, updated functionalities reflecting the completion of features that were previously planned and changes to future planned features, clarifications to stakeholder needs and requirements, and minor updates to more accurately reflect the current state of the project.

Version 1.0 (November 16, 2018)

The original version of this document contained documentation for all project-related activities through November 2018.

Introduction

The mission of AutoDrive is to **build and train a miniature self-driving car to investigate real-world problems in safety, behavior, and computation.**

This project was designed with three initial parallel efforts executed in Fall 2018 — building the hardware of the physical car, training the self-driving model using a simulator, and conducting human factors research to inform the ideal autonomous driving behavior. These efforts will now be integrated to form our proof-of-concept self-driving car, enabling us to conduct further research with real-world implications and experiment with variations on the self-driving model.

This design specification provides an overview of our project approach and implementation choices for the three parallel efforts conducted and the present unified effort to enhance the autonomous vehicle prototype and use it to demonstrate research findings. This document is intended to be updated with any project developments, and contains sufficient depth of information to inform a software and hardware testing plan.

Each team member brings their technical expertise to contribute to this project — Jianjie in self-driving models and hardware engineering, Michael in transportation safety research and systems engineering, and Phoebe in software architecture and machine learning. We can be reached at the following email addresses with any questions, concerns, or comments:

Jianjie.Liu@tufts.edu

Michael.Morscher@tufts.edu

Yu.Yang@tufts.edu

Limitations

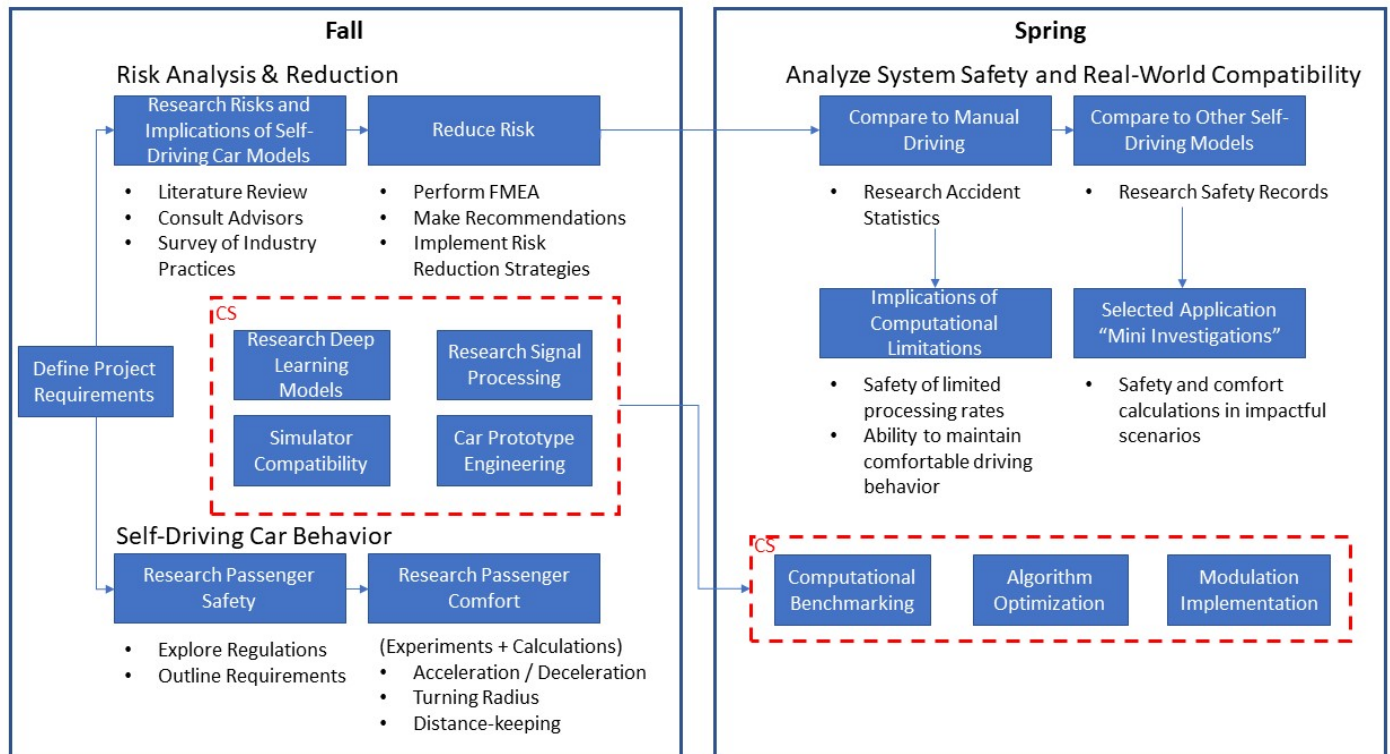
This document is intended as an overview of the current state of the AutoDrive project. For this reason, it does not include in-depth research results, data, or implementation details. Further, some of the prototype enhancement activities for Spring 2019 are still being planned, so certain components are limited to overviews in this document.

This is also not an full technical report on each project milestone. We will refer to our [GitHub repository](#) for the complete source code produced and for more details on technical challenges and solutions.

Research Approach

System Safety & Behavior

The following diagram summarizes the scope of our research into system safety and behavior:



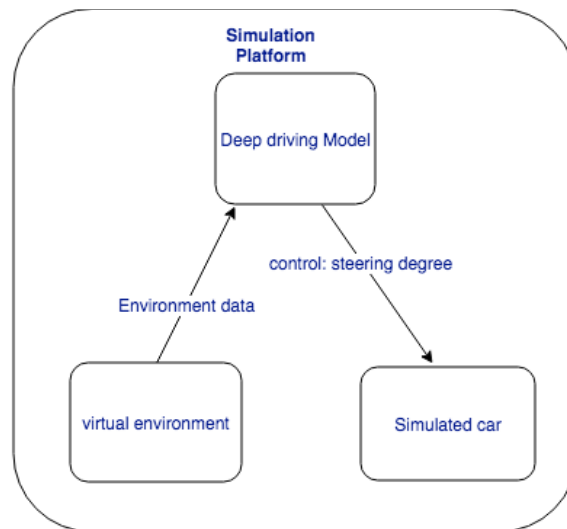
ENP System Engineering Diagram - Team Pink Flamingo

For each research component or group of components above, the following approach will be taken:

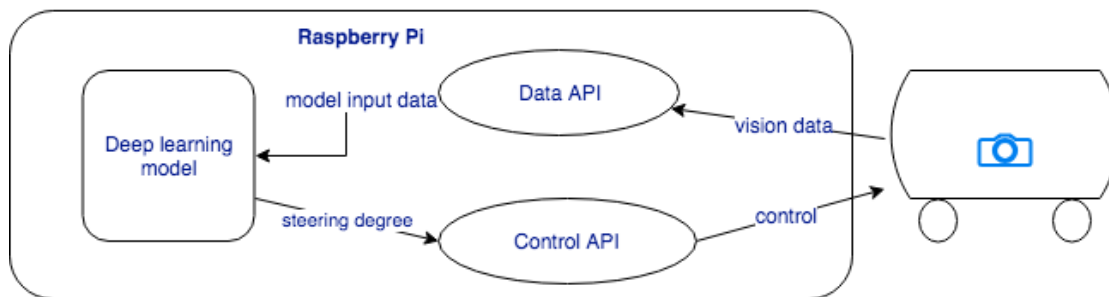
1. Conduct a literature review of human factors and transportation industry sources
2. Determine gaps in previous research
3. Consult sponsors / advisors on filling in these gaps
4. Fill in the gaps by running an independent investigation, interviewing industry experts, and/or pulling in relevant findings from other fields
5. Organize results as a deliverable, in the form of a formal write-up or proof-of-concept demonstration

Self-Driving Model Frameworks

Research and implementation of the self-driving model will be organized under two frameworks. The first framework involves **training and testing** the self-driving model:



The second framework involves **deploying** the self-driving model:



The research approach in building the self-driving model follows the order of the required system functions as indicated by the arrows in the diagrams above.

Stakeholders, Needs & Requirements

AutoDrive is an interesting project in that the end result is not a software application or physical product. **The “products” of this project are research findings which will introduce valuable new knowledge to the public domain.** This will be accomplished through building a sensor-connected miniature vehicle supported by a self-driving model, which is a significant deliverable but not a consumer product in and of itself. It will be foremost a proof-of-concept of building an extensible self-driving model, and later on, a tool for both implementing research conclusions and discovering new insights.

The following diagram depicts the direct and indirect stakeholders of the AutoDrive project:



Our direct stakeholder is essentially “the field of research”, with significant implications in extensibility to the real world.

The needs and requirements of our stakeholders can be summarized as follows:

Our stakeholders need:

- Research findings on best practices for autonomous models from an objective, academic perspective.
- Open sharing of best practices knowledge in system design, behavior, and safety, unlike closed knowledge of private autonomous vehicle manufacturers.

Our research investigations and findings shall:

- Extend trusted previous research with new approaches.
- Address ethical and safety implications of the findings.
- Be formally documented.
- Be openly accessible to inform objective safety and behavior standards.

Materials & Technologies

This project utilizes both hardware and software as follows:

Physical Materials

1. Radio Wave Controlled Car (Tamiya TT02)
2. Raspberry Pi 3b
3. Raspberry Pi Camera
4. HC-SR05 Ultrasonic Sensor
5. Power Bank
6. Breadboard
7. Wires

Software Dependencies

1. Python3.6
2. Matplotlib 2.2.2
3. Numpy 1.14.1
4. Keras 1.2.1
5. Tensorflow 1.8
6. pigpio 1.38 (Natively installed on Raspberry Pi OS)
7. picamera 1.13 (Natively installed on Raspberry Pi OS)
8. Udacity Self-Driving Car Simulator (<https://github.com/udacity/self-driving-car-sim>)

Software Modules Developed

The source code of all the modules mentioned here can be found on [GitHub](#).

Data API

This interface is responsible for processing raw images from the physical camera and converting them to 'PIL.image' (Python Imaging Library) objects for further processing. 'PIL.image' can be easily converted to other library objects such as a 2D 'numpy.array'. Much of this module relies on the library 'picamera', an SDK natively installed on the Raspberry Pi operating system.

Control API

We designed two separate objects to separately control the longitudinal (forward) motion and transverse (steering) motion of the car:

Servo.py

Defines the "Servo" object that controls the physical servo in the car with pulse width modulation (PWM). This module can turn the servo in a steering angle from a continuous range of $[-18.5^\circ, 18.5^\circ]$. This is established by utilizing a linear relationship between steering angle and the pulse width needed to induce the angle on the servo. This linear relationship was found experimentally (see [GitHub](#) for more details) and is defined as follows:

$$y = 17.9x + 1439$$

Where 'y' is the pulse width needed to induce the desired steering angle, 'x', in degrees.

We define the axis of orientation with respect to the car, so left turns have negative steering angles and right turns have positive steering angles. We use the library method ([link to documentation](#)) 'pigpio.pi.set_servo_pulsewidth()' to physically simulate the signal with the desired pulse width to control the servo. However, the library method does not specify the units of the pulse width, but we suspect it to be in milliseconds.

Methods:

- 'turn()': turn the servo within the range $[-18.5^\circ, 18.5^\circ]$.
- 'left()' : turn the servo into the minimum steering angle (-18.5°) .
- 'right()' : turn the servo into the maximum steering angle (18.5°) .
- 'neutral()': set the servo to neutral position (0°) steering angle).

Motor.py

Defines the “Motor” object that controls the physical motor in the car with pulse width modulation (PWM). At the current stage, this module provides only a trivial method to move the vehicle forward at a constant minimum velocity. Again, we use the library method ([link to documentation](#)) 'pigpio.pi.set_servo_pulsewidth()' to physically simulate the signal with the desired pulse width to control the motor.

Methods:

- 'calibrate()' : calibrate the motor to the appropriate range of signal.
- 'moveForward()' : move the motor in positive direction at a constant velocity.
- 'stop()' : stop the motor.

Deep Learning Model

The complete source code of this module can be found on [GitHub](#).

We instantiated and trained an end-to-end convolution neural network (CNN) model using Keras. The model takes in a 64x200 image and outputs steering commands in units of radians to keep the vehicle in the middle of a track. We use the CNN architecture published on Nvidia's [End to End Learning for Self-Driving Cars](#) paper with small modifications:

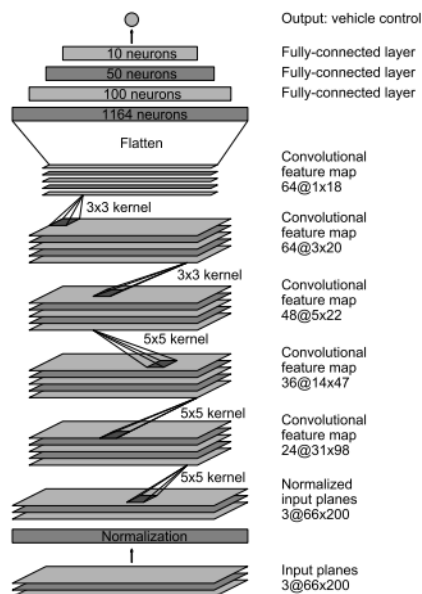


Figure 4: CNN architecture. The network has about 27 million connections and 250 thousand parameters.

The model is a convolution neural network that accepts a 64x200 (The original model takes in 66x200) pixel-wide image with 3 channels and outputs one node, which is the steering command. This CNN consisted of a normalization layer and 5 convolution layers followed by 4 fully connected layers.

For more details on training this neural network, please refer to this [GitHub repository](#).

The performance of this model on two simulated tracks is recorded in separate videos: [Track1](#) and [Track2](#).

Functionality

Current Functionality

1. The prototype will autonomously steer itself around a track at constant speed without crossing lane lines.
2. The current algorithm processes at 2 frames per second.

Planned Features

1. Optimization of existing algorithm to minimally process 15 frames per second. This will reduce failure incidences, such as crossing lane lines.
2. Ability to control the velocity and yaw rate of the vehicle. This will enable the following functionalities:
 - a. Ability to bring vehicle to a stop.
 - b. Ability to bring vehicle to cruise speed from stop.
 - c. Ability to modify vehicle handling to conform to optimal bounds on acceleration in the lateral, longitudinal, and vertical directions.
3. Ability to tune the processing speed of the algorithm to simulate different types of latency (decision, actuation, and I/O).

Proof-of-Concept Product Design

Car Prototype

Hardware Components:

1. Radio Wave Controlled Car (Tamiya TT02)
2. Raspberry Pi 3b
3. Raspberry Pi Camera
4. HC-SR05 Ultrasonic Sensor
5. Power Bank
6. Breadboard
7. Wires



Photo of car prototype

Simulation

We trained and tested our model using a Unity simulator which simulates the track environment and supports both the training mode and the autonomous mode. We collect data from the training mode to train a model, then we test the model in the autonomous model to see how it

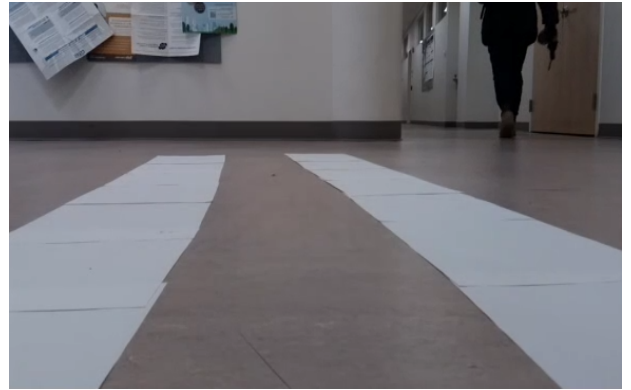
performs. A couple of measures of success include: (1) the car reacted to the change in tracks and steers itself at turns, (2) how long the car can drive itself in the autonomous mode without running off-track, and (3) how stable the car is in terms of velocity, steering speed, and acceleration.



Screen capture of self-driving model running in the simulator

Proof-of-Concept Integration

After training and testing the model in the simulated environment, we want to see how the model performs using real-world data. We collected snapshots of “paper tracks” with the camera on the car and fed the data to the model. Theoretically, the model should output a steering angle close to the proximate angle of the track. We tested the model in 3 basic scenarios: (1) left curve, (2) straight track, and (3) right curve.



Training images for real-world driving track

Here are results corresponding to the sample training images above which exhibit the functionality of our model:

Sample Image	Steering Angle	Throttle
left_1	-0.07146952301263809	0.079644
right_1	0.03908034786581993	0.8582879999999999
straight_1	-0.03622625395655632	1.0337062
straight_2	-0.021753234788775444	0.4809422

We observe that the model can output a correct steering direction based on the track condition. When the track is extending towards the left, the model outputs a left/negative steering angle -0.07; and when the track is turning right, the model outputs a right/positive steering angle 0.03. Based on the results from the test, we are confident that the model performs reasonably on real-world image data.

We plan to integrate the software model with the hardware prototype by deploying the model on the Raspberry Pi. All relevant data transformations will be supported by the Data API and the Control API (see “Software Modules Developed”).

References

AutoDrive Project Repo:

<https://github.com/JohnGee96/AutoDrive>

Deep Learning Model Project Repo:

<https://github.com/JohnGee96/CarND-Behavioral-Cloning>

Simulation Track 1:

<https://www.youtube.com/watch?v=SJyCEcUxbe0>

Simulation Track 2:

<https://www.youtube.com/watch?v=Utz5BzwYJ1Y&t=65s>

Udacity Autonomous Vehicle Simulator:

<https://github.com/udacity/self-driving-car-sim>

Nvidia's End-to-End Learning for Self-Driving Cars:

<http://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf>

PIGPIO Library Documentation:

http://abyz.me.uk/rpi/pigpio/python.html#set_servo_pulsewidth