# CS 378H Autonomous Driving Drifting: Final Report

## Goals

For passenger-carrying autonomous vehicles, maintaining control of the vehicle at all times is paramount for safety. This includes conditions where the wheels may slip. For our final project, we modified the autonomous car to perform high-speed drifting, and investigated methods to maintain autonomous perception, navigation, and control in this scenario. Our proof of concept demonstration was to autonomously perform a controlled drift while maintaining localization, and our final demonstration was to perform a controlled continuous drifting pattern of a lemniscate, AKA a figure-eight.

## Mathematics/Kinematics

This section includes formulas, equations, and other theoretical considerations we researched and discussed. However, not all of these were included in the final demonstration or code.

### Lemniscate Curve

We defined our lemniscate curve parametrically (units in meters):

$$x(t) = \frac{2.5\cos(t)}{1+\sin^2(t)} + \frac{2.5}{1.65}$$

$$y(t) = \frac{2.5\cos(t)\sin(t)}{1+\sin^2(t)} + \frac{2.5}{2.8}$$

This is the ideal path we want the car to take while completely drifting (after an initial ramp-up period. Here's how that path looks when graphed. Note when processing we broke this up to a series of points and tried to space them approximately 0.05 m apart.

### High Level Implementation Approach

For the most part, we generalized our model input parameters to be fed directly in from our IMU and lemniscate curve calculations. However, our choice to do so was based on Acosta and Kanarachos' paper on using neural nets to generalize most of the drifting criteria. In their paper, they discuss combining three models to combine path following, drift control, and terrain detection all into one. For drift control in particular, they describe their success in using neural nets to avoid dealing with details about the car's specific structure/tire traits.

We decided to follow a similar approach using neural nets but limited our model to achieve drift control and path following by training it on multiple figure-eight paths we manually constructed. Additionally, we trained on a single terrain that we could reliably reproduce findings from in the GDC. Though this made our model less generalizable, it allowed us to avoid some of the hard physics behind the mechanical parts of the car that Acosta and Kanarachos' paper detailed.

# Hardware Modifications

## Wheel Tape

To simulate a reduction in tire grip, we first switched the tires on the car to racing tires that had smoother treads. This effect did not appear strong enough to provide a successful demonstration in an enclosed space, so we added clear Scotch tape to the tires. This combined with testing on the smooth GDC floor at high speeds made wheel slippage more pronounced.

## Increased Acceleration, Speed, and Turn (and Fuse)

The maximum speed limit was increased from 1 meter per second to 5, and the maximum curvature was increased from +/-1 to the maximum allowed by the clipping function in vesc_driver (in our case +1.5ish and -1.8ish). This instigates wheel slippage for both autonomous and manual drifting. We found, however, that keeping the same acceleration rate with this increased speed limit causes the fuse to short out, so while the speed and curvature were increased, the acceleration was decreased. During demo day, we learned that larger fuses were available, thus allowing for much faster acceleration and reducing the need for a speed ramp-up prior to commencing drifting.

## Inertial Measurement Unit (IMU)

A Vectornav inertial measurement unit containing accelerometers and gyroscopes was added to the car. This is necessary because the odometer no longer provides useful information as wheel turning no longer tracks with the actual travel direction and distance of the car. The IMU can provide similar movement information but based on actual forces the car experiences. As the localization system developed in the class was built on odometer and LIDAR information, we switched to an external SLAM system (Cartographer) that supports IMU.

The IMU was initially installed at the front of the car away from the base-link frame (which raised the concern about the lever arm effect) and was installed with the axes upside down (see IMU "calibration" challenge). This led to some…interesting behavior from Cartographer, but after addressing the issues, the IMU-SLAM system was found to provide remarkably accurate localization within the large, landmark-devoid GDC atrium even during drifting.

# Software Modifications

## Cartographer SLAM

The primary challenge with cartographer slam was figuring out how to set up our own files to specify the details of our car to Cartographer. Ultimately we created a launch file that contained our desired topics, a Lua file describing the sensors to be used from the car, and a urdf file containing transformations from the baselink to the car's sensors. We also made an in-software fix for the IMU not being oriented correctly (adding roll to the urdf file), but we ended up just physically flipping the IMU over. Debugging this was quite difficult and required reading deeply into the documentation and ensuring that the cartographer was interacting with each of the rostopics correctly.

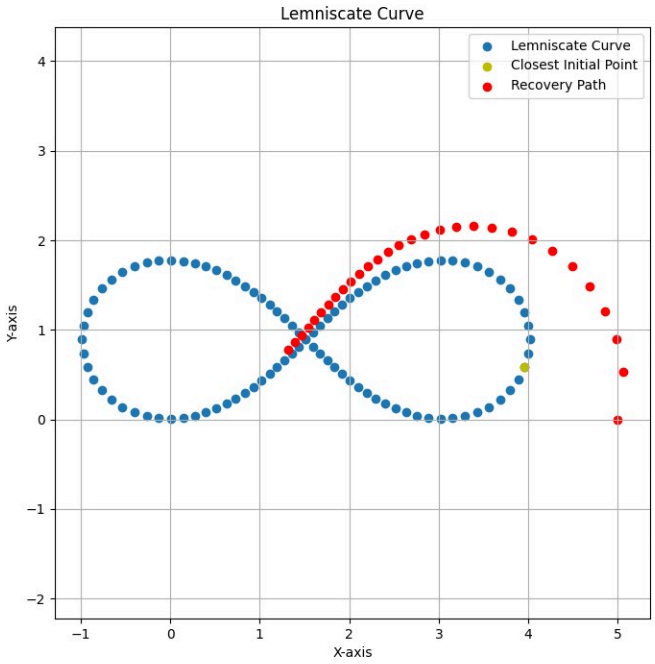
## Visualization

In order to display the cartographer's position at any given moment, we added some code to the websocket that would listen to the /tracked_pose topic and display a green x corresponding to the cartographer's tracked position. This was relative to the car's starting position. We tried to get a transformation to the map up at the last minute but ran into some bugs, which were apparent in the angle of the arrow that was intended to represent the car. However, the localization provided by the cartographer was super accurate and the actual position of the arrow was exactly where it should be on the map. When compared to the particle filter, which we left in to see how the two localizers fared, the cartographer was much more accurate. This is both due to the power of adding an IMU as well as the complexity of the cartographer.

## Lemniscate Pathing

To allow us to make our car drift along a figure eight pattern, we needed to represent our path (formally known as a Lemniscate curve) in such a way that we could use it for the model. The other component to this path problem was accounting for correction – as the car drifted more and more, error accumulated much like with the particle filter.

To solve both problems, we first decided to represent the path as a series of fixed discrete points in the global map space. Then, when we needed the next position to feed into the model at the next time step, we'd find the closest point on the path and find the point following it. With both points in hand, we performed smoothing to find an intermediary point between the car's current global position and the ideal next point. This allowed us to recover our drifting when we deviated from the path and generally stay in a figure eight motion. This 'smoothing' is visualized in the diagram below:




To actually publish our next point planning to feed into the model, we made a custom ROS node with topics to make a pub-sub channel from our C++ code to our Python model. This involved changing some settings in our catkin workspace but allowed us to keep all our message streams in ROS.

# Deep Learning Model

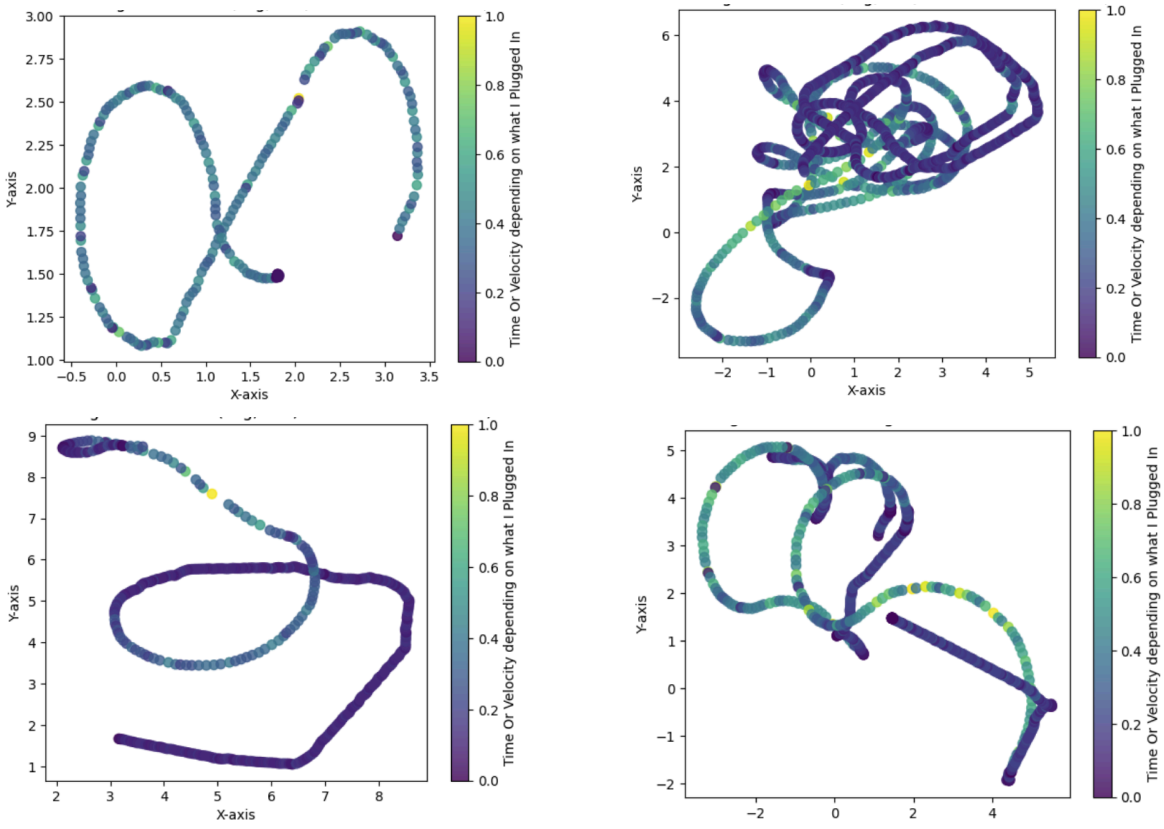
## Data

We collected data through a few different methods. First, we manually drifted the car while we were getting a hold of the best way to do so. Next, we preprogrammed several drift patterns that the car could try (relying on variability from real-life systems to provide diversity to those data sets). To extract relevant data we recorded a rosbag file (IMU data, odometry angle, cartographer published data, lidar data, etc.). We then extracted that from a python file (with continuity correction for all of the time stamps. We essentially grouped data points that were 0.05 seconds apart to match the time delta on the car. There was also the added complexity of how we wanted to orient our data. Did we want to pass every LIDAR reading to the car? Did we want to convert the provided pose quaternion to theta before passing into the model? These were all questions we needed to answer on both ends of data collection: what would be passed into the model for training, and what would be passed into the model at inference.

Data collection also involved driving the car, which meant we also needed to know how to drift the car. Throughout the life of the project, we spent a lot of time drifting the car, both having it

collect data while being controlled by us and the joystick, and also having it collect data while we had the car performing a set autonomous figure-8. These rosbags would get to significantly large file sizes due to the length of the data collection and the diverse amount of data being collected.

Here are some plots of the paths we took with color based on velocity at that moment (you can see where the drifting occurred) when trying to make sure the data we passed our model was good:




## Model

The goal of this model was as follows: given IMU, cartographer, and odometry data from the last k seconds (as collected by our data model), predict the velocity and curvature needed for the next step. It also took in a target point (provided by the curve for the live car, provided by the next data point for training).

We had two different approaches we were considering trying: the first was a vanilla neural network, the second was a transformer. For the vanilla nn network option, the data that would traditionally be sequential for a transformer (the last *k* seconds of our training data) was just collapsed into the input.

We opted to start with the first option for a few reasons. The first one was simplicity. The second was we heard another group was having trouble getting the gpu on their car to work so we figured we'd save that problem for a day it was required (vanilla nn would also be faster so we could iterate sooner).

We ended up taking input from the last two seconds and feeding it as so to our model. From there, we utilized skip connections to allow for faster model convergence, letting us achieve a MSE loss of about 0.1 on the validation set.

### Deployment

This is where we ran into trouble. We deployed the ML model to our car and it kept outputting that the car should go backward (negative velocity).

Our first thought is we were passing data in incorrectly. We suspect if this is the case it is a problem with the target point (we made sure everything else was uniform). The second option is that the issue was loading the model - something happened to the weights that made it act awry.

Either way, we were unable to get the model deployed. Of note, on the validation set it was predicting the velocity (almost always positive) very accurately so while it could be a problem with the model considering all our data files started from rest we very much doubt it is.

Finally, to try to debug issues with the target point passed in, we made sure the array of points we had representing our lemniscate curve had a curve length of approximately 0.05 m (most common in our training/validation set).

# Challenges

## Wrangling Cartographer/IMU "Calibration"

Although it took some effort, we managed to get Cartographer to output a tracked_pose by the Sunday before Thanksgiving. Yet we did not get Cartographer working until two weeks later… we attempted many different settings in order to finetune Cartographer, but ultimately we realized the issue was that our IMU was upside down. This was also the reason there was so much bias in the readings originally.

## No Simulation

One of the hardest parts of this project was having to write most of our code on the car directly to test our work. Since we didn't have a simulation for our IMU, we couldn't work

asynchronously and test our changes independently – this issue was also more poignant with our larger group size and the incredibly slow startup process we had whenever the car got damaged (see “the Ritual”).

## Crashing and Breaking Stuff

We broke our car a lot. The net total of damages was one lidar base, four screens, and a half dozen blown fuses. The lidar base and screen damages happened mostly due to drifting into walls and other objects, but the blown fuses occurred because we upped our acceleration limit dramatically.

The screen damages were the most annoying to fix (see “the Ritual”) since we couldn’t get our drive system to start without the screen being plugged in. This had the annoying effect of making our startup process pretty long which had to be repeated each time we blew a fuse while testing. The net effect was that it slowed down our development process dramatically.

## The “Ritual”

The “Ritual” was the phrase we coined for starting the car and each of the individual ROS nodes each time the car was shut down. Unfortunately, since we broke one too many screens, we decided to remove the display entirely from the car to avoid further damages. As a result, we had to perform the following steps in exact order otherwise different issues would occur like the drive system crashing or the SSH connection being terminated randomly: (1) plug in the screen’s power/HDMI, (2) boot up the car, (3) SSH into the car and start vectornav/cartographer, (4) disconnect the display and get to work. Since this occurred every time the car turned off, it meant that blowing fuses was particularly annoying as it happened regularly with the 10 AMP fuses we were using.

# Code Organization

Our MS 2 - 3 code is mostly the same as it was before. Navigation.cc / navigation_main.cc contain relevant code for our pre-planned drifting code. The repo contains our model and relevant scripts for running it in /src. The zip files cartographer_ros.zip and lemniscate_next.zip contain code and configurations that were added within our catkins_ws folder as modules.

# Code

Latest Commit: https://github.com/loganvaz/ut-automatic-driving/tree/stuffs2
589c299ff34abba05f580535f1021083f703120d

# Videos

Video of the car drifting manually
Autonomous Drifting Videos:

1. https://drive.google.com/file/d/16H3sK2AtlGHdlBd12cMDKUT82jU1oACZ/view?usp=sharing
2. https://drive.google.com/file/d/1e15lDiBRG9vOTAzD4p59QLsMwjh_H13c/view?usp=sharing