Introduction to F1Tenth Racing Research 2024

`Version 1.0`

Welcome to the ESL in 2024 to do research into F1Tenth autonomous racing.

This document aims to provide you with a head start with getting used to autonomous racing algorithms. The first few months of your masters are normally focused on learning new skills and getting used to the concepts that will carry you through the rest of your masters.

This guide is divided into two sections, general F1Tenth skills and specific skills for each of you.
I propose that you work together in understanding the general skills and that will put you in a good place to develop the specific skills that you will need for the rest of your masters.

I am always available for any questions you might have. My normal office hours are 8am-4pm, with lunch between 1-2. Otherwise, Slack is also an effective means of communication.

General Skills

We use the F1Tenth platform for most development. There are Python and ROS simulators and the physical vehicle uses ROS.

General software skills:

- Python programming
- Git - this is essential
- Robot operating system (ROS2)
- Clean code - other people should be able to understand
- JIT -jist in time compiled code. This makes python code a lot faster. Look at the JAX or Numba libraries
- Good data visualisation - when you read an article, look at how the results are presented. You should learn to make clean graphs that clearly make a point

Note taking
Another key skill for the duration of your masters is effective note taking. When you learn something, write it down. At the start, it seems like you will remember everything, but you will forget. So take notes.
I use Obsidian for notes, but you can use any software that you like. The most important thing is take notes. If you want specific advice on what I do, feel free to ask.

Reading
Part of doing research is reading papers to understand what other researchers have done. While lots of engineers do not enjoy reading, it is essential. I included a list of papers to start reading related to each of your topics. I propose you **aim to read 2-3 papers per week**. If you that consistently, overtime you will build up a feel for what the literature says on a topic.

When you read a paper, you do not have to read every single detail. Rather, aim to have an overview so that if your friend asks what you read about, you can explain to them in simple terms. Keep notes of the papers you read and what you learn in them.

Routine
This is more of a personal skill than anything else, but one that is important. There are no rules while you do masters. You may work as much or as little as you like. For many people, the large amount of freedom can be a challenge to manage. I strongly propose setting up a routine for yourself. It can be anything, and you should tailor it to your own strengths and weaknesses. For example, mine is something like this:

- Arrive at 8am and make a todo list
- Work hard from 8:30-10:30, often on writing an article (my most difficult activity)
- Take a short walk for 15 minutes
- Work hard from 11:00-13:00, normally on programming tasks for my current project
- Lunch from 1-2
- Arrange meetings for the afternoon, or else continue with programming or else learn new skills that I require

F1Tenth skills

Start by watching courses to understand the F1Tenth platform. The website, and YouTube playlist have a lot of useful content on. The following lessons are essential to watch and understand:

- L05 - Follow the Gap for Obstacle Avoidance
- L06 - Vehicle States, Vehicle Dynamics and Map Representations
- L09 - Particle Filters
    - https://www.youtube.com/watch?v=aUkBa1zMKv4 - this is a cool video that gives an intuitive understanding of the particle filter
- L10 - Pure Pursuit
- L22 - Raceline optimisation

Python Simulator

Once you understand the basic concepts, the next step is to get to grips with the Python Simulator. The official Python simulator is available at https://github.com/f1tenth/f1tenth_gym

I have a modified version of the simulator, and many algorithm implementations at https://github.com/BDEvan5/f1tenth_sim
This repo includes:

- A simple particle filter
- Raceline optimisation
- Pure pursuit trajectory tracking
- Follow the gap
- Deep reinforcement learning agents - these are interesting, but not so relevant to your projects.

You should clone the repository and verify that you are able to run the follow-the-gap algorithm, generate a racing line and track it with the pure pursuit algorithm.

Take time to study this repo

Most of you work will look similar to what I have done in this repo. Take time to understand how these algorithms work, how they are implemented and how to run experiments.

- A good mindset is to study the repo as if you were going to write an exam on it.

Introductory Tasks

Most software skills are developed through practice and trial and error, so take time to experiment and try things out. Therefore, I put together two tasks to get you up and running in the simulator

- Implement your own version of follow the gap
  - Do not copy my implementation, but start from scratch and code up a follow the gap algorithm.
  - Once it works, compare the performance to my version and see if you can beat me.
- Implement a pure pursuit controller to follow the centre line of the track
  - Once again, implement your own version of pure pursuit that tracks the centre line at a constant speed.
  - Tune the lookahead distance until the best performance is achieved.
  - Once it works, evaluate how accurately it tracks the centre line.
  - Run an experiment to see how the tracking accuracy changes based on the constant speed of the vehicle.
  - Between the three of you, see who can achieve the best tracking accuracy with a speed of 5 m/s

For both tasks, try to use good scientific practice to plan experiments, collect data, and present the data to draw a conclusion.

ROS skills

While ROS skills are not essential to start with, you will require them at some point.

- Here is a course you can refer to when the time comes
- https://www.theconstructsim.com/robotigniteacademy_learnros/ros-courses-library/ros2-basics-course/

Our development pipeline normally runs as follows:

- **Ideation:** have an idea that we want to implement
- **Sandbox development:** we implement the most simple version of our idea possible so that we can verify if it works. This is normally done in the Python simulator.

- **Scale implementation:** Now that we know the idea works, we scale up the implementation to include more features. This normally involves converting the code into ROS and testing it in the ROS simulator
- **Evaluation:** Test the algorithm onboard the physical vehicle and record results
- **Write up:** write an academic article or thesis

Individual Projects

You should read through your own project, as well as the other projects to understand where you fit into the research space of the lab.

For each topic, I describe the problem, expected objectives and some starting steps. While these are not set in stone, they hope to provide some direct vision on how to start and what is expected of you. I would propose regularly reviewing this document to monitor your progress and ensure that you remain on track.

Brandon Chetty 2024: Data-driven Model Estimation for an Autonomous Race Car

Context

Classic autonomous racing methods use optimisation planning and control to achieve high performance. Ideal racing behaviour is when the car is at the limit of controllability. This is a knife edge between going too fast and crashing and going to slowly and loosing. One of the most critical parameters in defining the knife's edge is the friction coefficient of the tyres on the track.

Finding the friction coefficient is a chicken and egg problem because you require a model of the vehicle to control the car at high speed, and you require data of the vehicle at high speed to identify a model. While it may be possible to manually tune the parameters in the lab, that is not a feasible strategy for if we take a team to a competition that has a different floor type.

Problem statement

To develop a planning and control strategy that can be used on new surface types to identify the friction coefficient and then use the identified value to control the vehicle at high-speed.

Expected objectives

- Generate a method to collect a real-world data set that could be used for tuning on a new race track
- Develop a method to estimate the friction coefficient from a data set
- Develop a controller that can robustly use the estimated friction value to achieve high-performance. You can either use a current controller, i.e. raceline optimisation and pure pursuit tracking, or implement a new one such as an MPC formulation
- Implement the full method on a physical vehicle of data collection, parameter estimation, and high-performance racing.

Starting steps

- Watch F1Tenth lecture on MPC and understand how model predictive control works. YouTube has some good resources
- Understand trajectory optimisation and how the routines work
    - https://www.youtube.com/watch?v=wlkRYMVUZTs this is the best video I know of.
- Reading
    - Wischnewski, A. (2023). *Robust and data-driven control for autonomous racing* (Doctoral dissertation, Technische Universität München).
    - Becker, J., Imholz, N., Schwarzenbach, L., Ghignone, E., Baumann, N., & Magno, M. (2023, May). Model-and acceleration-based pursuit controller for high-performance autonomous racing. In *2023 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 5276-5283). IEEE.
    - Data-driven science and engineering: Machine learning, dynamical systems, and control
        - This is a great book with a lot of good numerical methods for model identification
    - Minimum curvature trajectory planning and control for an autonomous race car
    - Indy autonomous challenge-autonomous race cars at the handling limits
    - A Survey of Vehicle Dynamics Modeling Methods for Autonomous Racing: Theoretical Models, Physical/Virtual Platforms, and Perspectives
- Coding
    - Create a dataset (of vehicle odometry (position, orientation and speed)) that can be used to identify a model.
        - Start with creating a simulated dataset and then later a dataset from the physical vehicle.
    - Develop a method to estimate the friction coefficient from the data.
        - Start with a simple linear regression to find the best fit value, and then expand to use more advanced numerical methods.
    - So for the first task, create the data set and then try to estimate the friction coefficient used to create the data set
        - Once you can do that, add some noise to the friction coefficient, create a data set and try to filter out the noise.

Ruan Delport 2024: Efficient Robotic Perception using Low-level Environmental Features

Context

Robotic perception is the task of building a map of the environment and then localising the robot on that map. Current methods for perception, focus on using occupancy grid map representations, which are computationally expensive and inefficient.

This project proposes using low-level environmental features to improve the efficiency of robotic perception algorithms. Low-level features are properties that describe the environment and are easy to identify.
For example, a room wall can be represented either by many points on an occupancy grid, or by a

straight line. Representing a room with lines requires less information to be stores, thus significantly improving efficiency.

I have developed a method to race using local maps (currently under review). Your project is to expand this method into a localisation and mapping algorithm. The first test platform (and sandbox) will be F1Tenth racing. Once we have established that it works there, we can expand to a different robot, possibly in a mining or agricultural context.

Problem statement

To design and implement a perception system using low-level features (such as lines and shapes) that can exploit an environments known structure to map and localise a robot

Expected objectives

- Expand racing with local maps into a localisation system that can be used to estimate a vehicle's location
- Expand local map racing into a SLAM algorithm that can fuse local maps together to form a race track.
- Test algorithms on the F1Tenth platform
- Implement algorithms on a different robotic platform

Starting steps

- Study the paper, Optimisation-based Control in Unmapped Environments - The Local Map Framework for Autonomous Racing
    - Also study the code in the localmap_racing folder of the f1tenth_sim repo
    - This is the code used to generate the results
- Study state estimation theory
    - This repo has a document I wrote and some test code to demonstrate the concepts https://github.com/BDEvan5?tab=repositories
    - You should be able to roughly recode up the examples in this repo
    - There is a good book called "Probabilist Robotics"
    - Watch the F1Tenth lectures on scan matching
    - YouTube has a lot of resources
- Reimplement the particle filter for F1Tenth racing - NB
    - Run experiments to see how the localisation performance can be maximised
    - Once you have a simple implementation, try to recreate the same sensor model used in https://github.com/f1tenth/particle_filter
    - I propose writing the code using the JAX library that lets you JIT compile Python code to make it fast
    - You can also look at the numba library.
- Start to adapt the local map racing method into a localisation method
    - The way to think about this is to say that if you have a map of the track (specifically the inner and outer boundary lines), and I give you a local map, can you tell me where on

the track I am???
        - If you draw it out, then you will see that it makes conceptual sense to be able to see where the current local map fits best.
        - The question is to come up with a method to do that mathematically.
    - A key will be to use the curvature of the inner and outer boundaries to check where the vehicle is at each time step.
- Reading
    - Vehicle dynamics state estimation and localization for high performance race cars
    - ROS-based localization of a race vehicle at high-speed using LIDAR
    - **Accurate** mapping and **planning** for autonomous racing
    - Fast and **Accurate**: The Perception System of a **Formula Student** Driverless Car
    - A good search term is the "formula student driverless" - there are a lot of algorithms about how they implement perception systems that you should read up about
    - AMZ driverless: The full autonomous racing system
    - TUM autonomous motorsport: An autonomous racing software for the Indy Autonomous Challenge

Christopher Flood 2024: Full stack F1Tenth Autonomous Racing system

Context
The autonomous racing problem is for a vehicle to move around a race track as quickly as possible while avoiding obstacles/opponents.
Racing software stacks require addressing the problems of perception, planning and control.
Various methods exist for each category, for example, a particle filter to estimate the vehicle's position on a map.
Optimisation methods can generate optimal trajectories an pure pursuit and model-predictive-control (MPC) can be used for path following.
Obstacle avoidance and head-to-head racing can be achieved using a graph-planner or re-planning strategy.

Building upon previous work in the ESL, such as trajectory optimization, the project will focus on creating a cohesive, high-performance system. The ultimate goal is to develop a full-stack autonomous racing system capable of participating in international competitions, such as the International Conference on Intelligent Robots and Systems (IROS).

Problem statement

To design, implement and test a full-stack racing strategy for single vehicle and head-to-head racing, with a focus on integrating and testing pipeline components.

Expected objectives

- Implement and test a perception, planning and control strategy for autonomous racing

- Setup benchmark tests that can be used for each component to evaluate how well they perform.
    - i.e. if Brandon designs a new planning method, you should have a test that tells us how well it performs and if it is better or worse
    - If Ruan develops a new perception algorithm, you should tell us if we should use it.
- Set up automated testing that makes it possible to compare different modules easily. This should provide extensive monitoring and feedback. e.g. computational efficiency, low speed performance, high-speed performance
- Test perception, planning and control on a physical vehicle
- Design and implement amended perception for opponent detection from LiDAR
- Implement a head-to-head racing strategy that can take part in competitions

Starting steps

- Reading
    - Autonomous vehicles on the edge: A survey on autonomous vehicle racing - this will give a good overview of all the required components for a racing system
    - Vehicle dynamics state estimation and localization for high performance race cars
    - A deep learning-based radar and camera sensor fusion architecture for object detection
    - A software architecture for an autonomous racecar
    - Indy autonomous challenge-autonomous race cars at the handling limits
    - TUM autonomous motorsport: An autonomous racing software for the Indy Autonomous Challenge
    - AMZ driverless: The full autonomous racing system
- Learn ROS early on and get an entire racing pipeline (probably easiest to adapt the f1tenth_sim repo) working in a ROS simulation
    - Perception with particle filter - discuss best options here with Ruan
    - Planning - racetrack generation
    - Control - pure pursuit tracking - discuss with Brandon if you need help on tuning
- You should evaluate how high you can get the performance and where further improvements are needed
- It is important that you develop set methods for all the required functions. i.e. how do we use the ros slam toolbox to build a map
    - It is very important that you take notes that can form a lab database of how to do anything on the vehicle.
- Transfer your system to the physical vehicle and record some results
- Once you have a high-performance system, start to read up on head-to-head strategies and plan how to race in competition
-