F1TENTH Sim Racing
using Nav2 MPPI controller
project status report

Yair Nahum 034462796

June 18, 2025

Contents

1 Preface

This project aims to explore the Nav2 MPPI controller in the F1TENTH simulation racing league. This simulator, based on the AutoDrive ecosystem, is used as an environment in which teams around the world can learn

autonomous driving algorithms and explore new ones in addition to competitions between teams.
The main website link
We define the project targets as follows:

1. Run the simulator together with a simple wall follow + SLAM toolbox to map the given racetrack.

2. Integrate the nav2 stack and use it over the simulator with its default components.

3. Minimize the nav2 stack to the minimum needed for race needs.

4. Replace the nav2 default controller's plugin with the MPPI plugin.

5. Clone and build the MPPI plugin, learn its design and implementation, and try to tweak its parameters for race requirements.

6. Explore and change if needed the source of the MPPI controller, low-level control, motion model, etc.

2 Development steps overview

We first needed to have our environment set up for the project. After some setup problems with the Ubuntu 22.04 machine with ROS2 humble, native install was not successful for some reason, so the containerized approach was tried (submission to the competition requires it anyway).

2.1 First simple controller and SLAM

We took the following first steps in development:

1. Create a wall follow controller for exploring the track.

2. Adding a low-level PD controller over the throttle and steering where the desired steering and speed are supplied to the controller. see clips' links: rviz , sim.

3. Learn, install, and use the slam_toolbox to map the racetrack. see clip slam Also, played with the localization and applied odometry node based on wheel encoders as if we don't have the accurate location (not needed basically, as the localization is given accurately by the simulator "world"→"f1tenth" transform).

2.2 Nav2 MPPI controller integration

1. Integrate nav2 stack and have it working with goal definition with the AutoDrive simulator. see clip nav2 default
2. Reduction of nav2 components to minimum required:
   (a) Removed localization AMCL node from nav2 as the localization is given. For this to work, a static transform from "map" → "world" was added.
   (b) Removed redundant components from nav2 like smoother sever, velocity smoother, waypoint follower, behavior server. Keep the planner server and the controller.
   (c) Removed planner by replacing it with a predefined reference race-track calculated offline with some simple CV program and a publisher that publishes the relevant part of it for the car to follow.
   (d) Replaced default behavior tree of nav2 with a minimal one that has only an action node that gets the plan from previous publisher to the behavior tree blackboard for the controller to follow.
3. Replaced the controller server default DWB local planner plugin (implementation of the FollowPath algorithm) with the MPPI plugin.
4. Cloned and built the MPPI controller in the container to be able to change/debug it if needed. Had some issues that were found to be related to swap memory size needed in WSL (over windows OS).

2.3 Raceline Optimization

Found Steven Gong's website about race line optimization for optimal minimum curvature/time reference trajectory.
My forked version of the github: https://github.com/ynahum/Raceline-Optimization
Sub steps done to create the optimal minimum curvature reference:

1. Configure the vehicle parameters needed (mass, max speed,steering, minimum radius, etc..).
2. CV operations to get the center line of the track.
3. Run optimization over minimum curvature + velocity profile (iterative process of these 2 steps). At the end, we get a csv formatted file reference points with x,y coordinates and velocity.

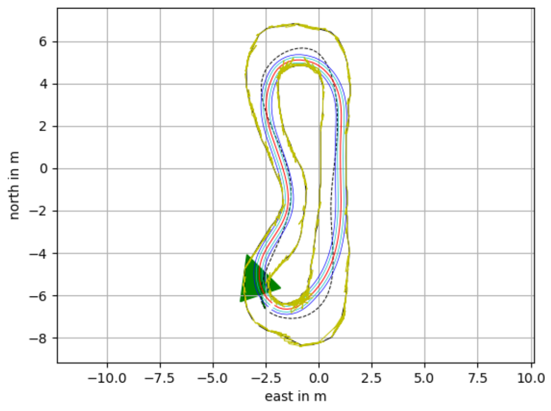


Figure 1: Optimized minimum curvature reference trajectory

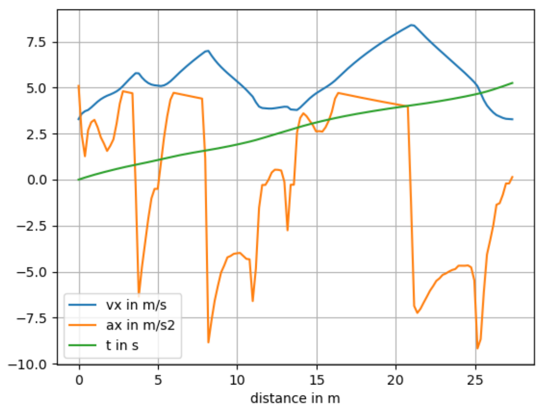


Figure 2: Velocity and acceleration reference

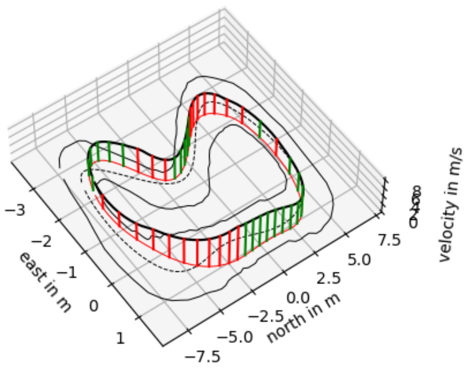


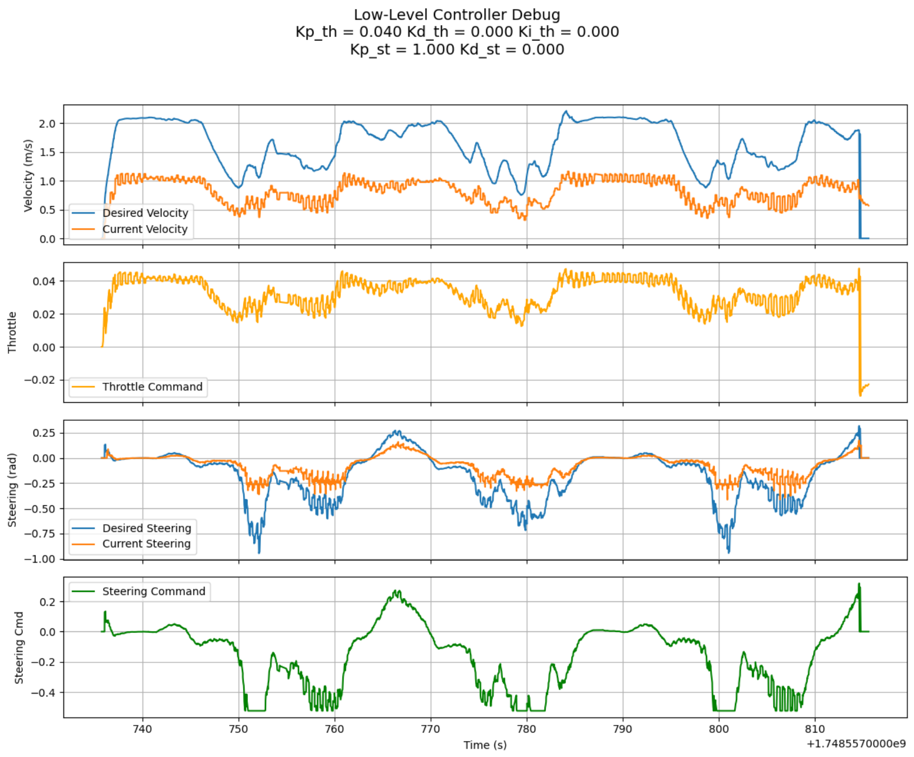
Figure 3: Acceleration reference in 3D

2.4 Debug and Visualizations

Added debug traces to dump as function of time the low level controls (steering,acceleration) after applying PID control + optimizer traces.

2.5 Low level PID controller tuning

Experimented with PID tuning by plotting the reference vs actual velocity/steering. Need some more tuning though as it was done on rather low speeds using the simple Ackerman model.



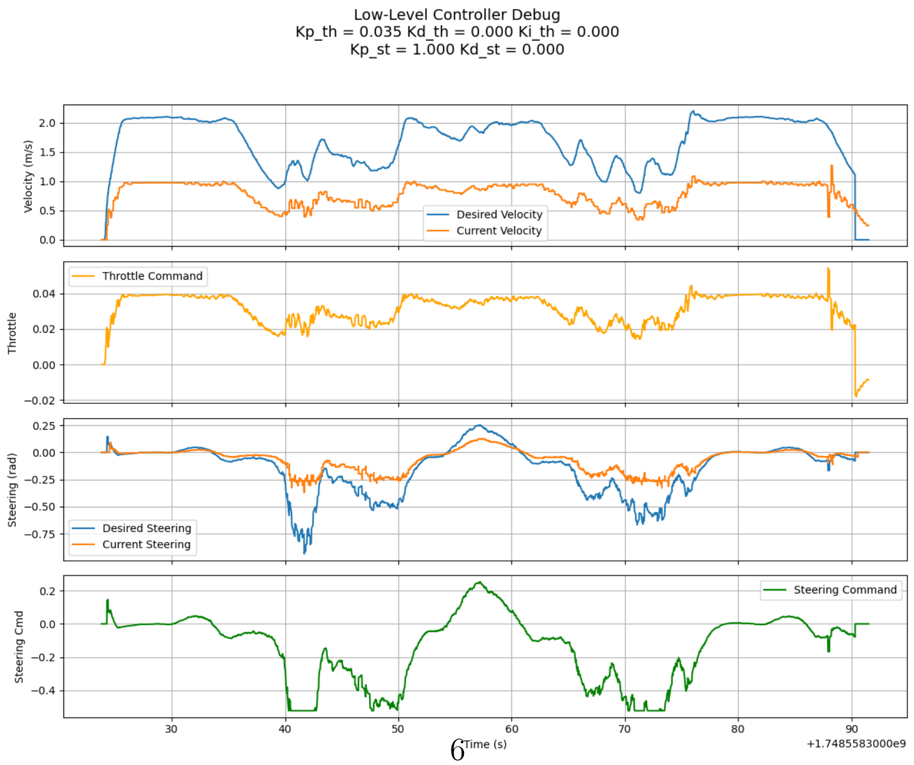


Figure 4: Different Throttle velocity Kp values

There is a steady state error that can be addressed using I component gain that was further tuned:

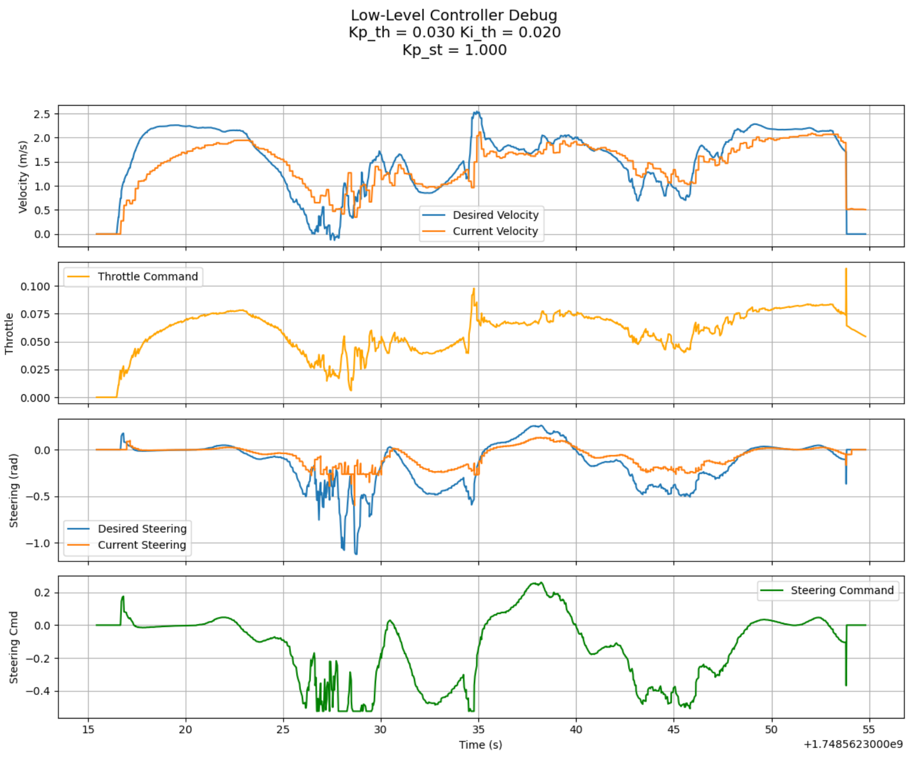


Figure 5: Throttle Kp=0.035, Ki=0.02

The steering needs more tuning. We can already observe the car doesn't react optimaly to the steering even on low speeds. On high speeds, we should probably face under/oversteer behavior and a more accurate dynamics models.

3 Current performance and analysis

At first, the performance of the nav2 stack with MPPI and its default configuration was poor as can be shown in the following clip nav2 mppi default. After improving the low level PID controller, the optimal minimum curvature reference race line and playing with the rather big amount of parameters

that can be changed, it seems that we improved the performance as can be shown in the following clip nav2 mppi current.
There seems to be still a lot of room for improvement as the kinematic Ackerman model seems to be inadequate for high speeds of a race car. In specific, it seems the car is under steering, meaning, the under steer gradient is low due to slip angles when in high speed and high curvature (small radius). When we increase the variance of the speed sampled, the car drives faster but lacks the ability to compensate the under steer and it hits the wall.
To face it, one needs to model dynamics of the vehicle more accurately and calculate the under steer gradient of the car to get the correct control steering angle (based on speed and curvature).
Would like to learn deeper the rather big subject of vehicle dynamics in order to get to better performance. Started doing some courses on the subject on youtube and Udemy

4 What next?

1. Sync with the latest Nav2 MPPI controller as it was updated in ROS2 jazzy compared to ROS2 humble.

2. Deeper learning of the MPPI controller interface parameters (dt size, amount of time steps, state constraints, etc..) and the different critics used for its cost and their own parameters . Try to tune the different weight over the critics.

3. Learn the design and implementation of the MPPI controller cloned.

4. It seems it uses rather simple motion model "DiffDrive" and has also "Ackerman" version which is also rather simple and currently it seems these models only enforce some constraints on the trajectory simulated. Should explore whether we can introduce our own dynamic model which more accurate on one hand but won't require too much overhead as the MPPI should be run in 20Hz (current configuration).
   A dynamic single track model that accounts for slip angles, forces and many other parameters and with acceleration as its input control (speed as state variable) will be much more accurate and robust at high speeds vs a simple Ackerman steering model that has many none real assumptions.

5. ..

5 GitHub and DockerHub repositories

GitHub repository link -
https://github.com/ynahum/mrs_project_container
Docker Hub repository link -
https://hub.docker.com/repository/docker/ynahum/ad_f1_api/general