F1TENTH Sim Racing with Nav2 MPPI controller project report

Yair Nahum 034462796

July 15, 2025

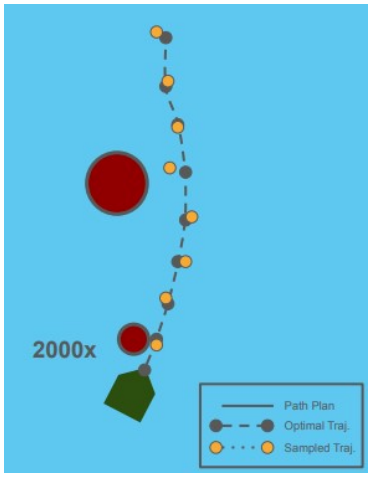
Contents

1 Preface

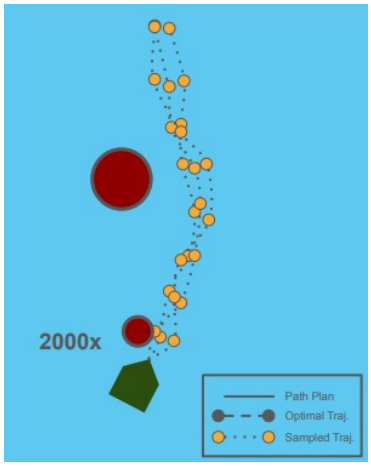
1.1 MPPI controller previous work and overview

The MPPI controller was introduced in Williams et al. [8] and was based on the previous path integral optimal control framework Kappen [3].
Another work by the same authors Williams et al. [9] is a great example of using MPPI with the car dynamics learned by the neural network and RL.
In its base, the concept of the MPPI controller is to explores many possible trajectories by sampling with specified variance and select the one with the minimum cost with respect to a none linear cost function that can be compound from various critics with different weights to tune/configure.
We can break it down to several stages:

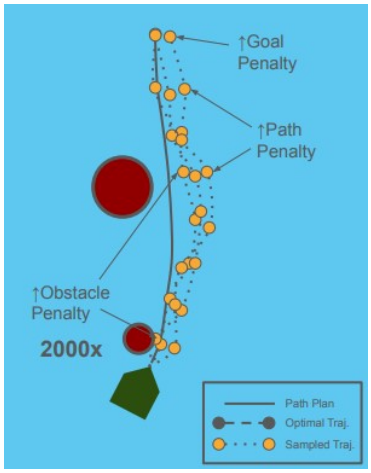
1. Add Noises to Prior Optimal Trajectory's Controls - see yellow points added with respect to given optimal control path (calculated on previous step). In red are the obstacles.



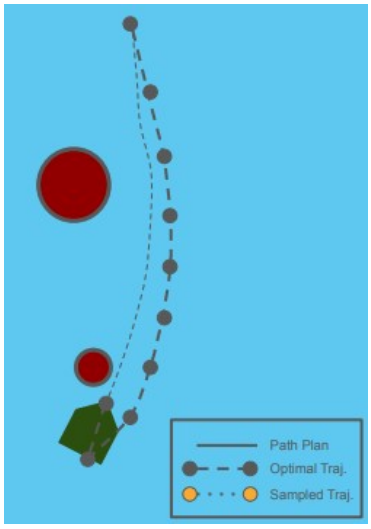
2. Apply Dynamics to Controls & Rollout Trajectories.



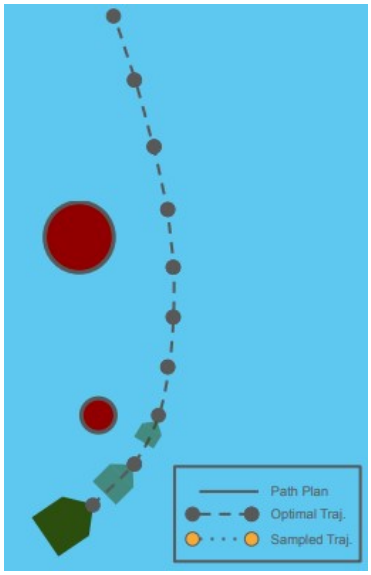
3. Score Noised Trajectories Via Objective Functions - The critics are applied over the states and controls along the rolled out trajectories.



4. Compute New Optimal Control Sequence.



5. Execute First Control, Shift Optimal Control. Repeat.



⇒ No derivatives or convexity of the objectives is required (as required in many MPC controllers).
⇒ Arbitrary dynamics, constraints, and objectives.

The MPPI controller is heavily used due to its simplicity, although it does not provide guarantees about optimality.

1.2 How Nav2 stack implements the MPPI controller?

The main component of the Nav2 MPPI is the optimizer. The main function of the optimizer is the control evaluation which basically does the following steps:

1. Generate noised trajectories
2. Evaluates trajectories scores by the critics manager
3. Update the control sequence - applies constraints on control according to car's parameters and applies the motion model constraints (e.g in Ackerman model the minimum radius which imposes the maximum curvature).

There are also the path_handler for handling the path transformations from different frames and the visualizer component for visualizing over RViz.

2 Goals of this project

This project aims to explore the Nav2 MPPI controller from Nav2 Development Team [4],[5] and apply it in the F1TENTH simulation racing league on the AutoDrive ecosystem created by Samak et al. [6], [7].
The above ecosystem is used as the environment in which teams around the world can learn autonomous driving algorithms and explore new ones in addition to competitions between teams.
The competition details can be found at AutoDRIVE Ecosystem [1].
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.

3 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).

3.1 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, tuned with the localization and applied odometry node based on wheel encoders as if we do not have the accurate location to mimic real-world scenario. However, in the sim racing the odometry is not needed, as the localization is given accurately by the simulator "world"→"f1tenth" transform.

3.2 Nav2 MPPI controller integration

1. Integrate nav2 stack and have it working with goal definition with the AutoDrive simulator [6]. 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 racetrack 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 default DWB local planner plugin from the controller server (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).

5. Sync Nav2 MPPI controller to its latest version - Synchronized with the latest Nav2 MPPI controller 1.3.7 as it was updated in ROS2 Jazzy compared to its ROS2 Humble version 1.1. At some locations (costmap layers) had to revert to back to Humble. Kept the additions of debug info gathering. This introduces max longitudinal acceleration for example and had to increase its default max rotational acceleration according to sim.

3.3 Raceline Optimization

Found Gong [2] about race line optimization for optimal minimum curvature/time reference trajectory.
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.

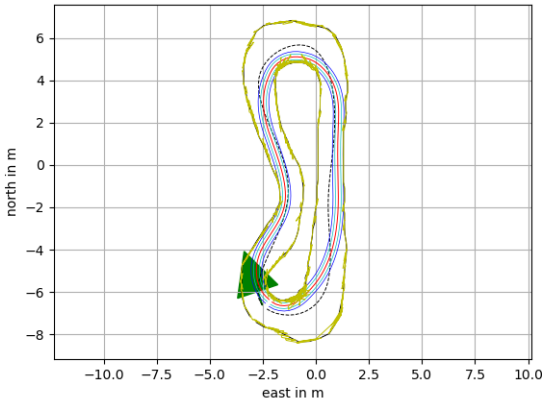


Figure 1: Optimized minimum curvature reference trajectory

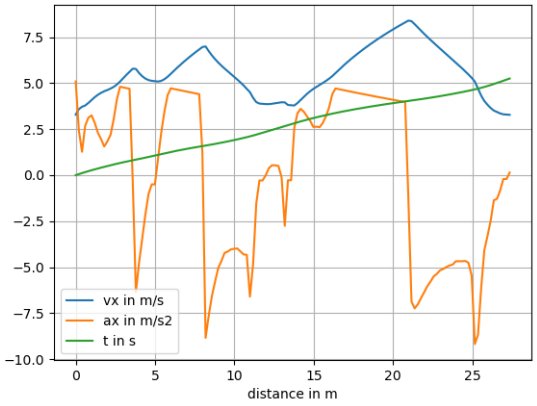


Figure 2: Velocity and acceleration reference

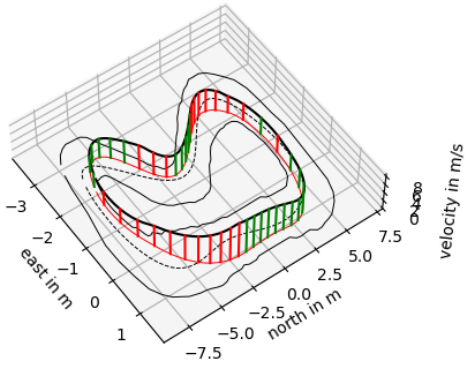


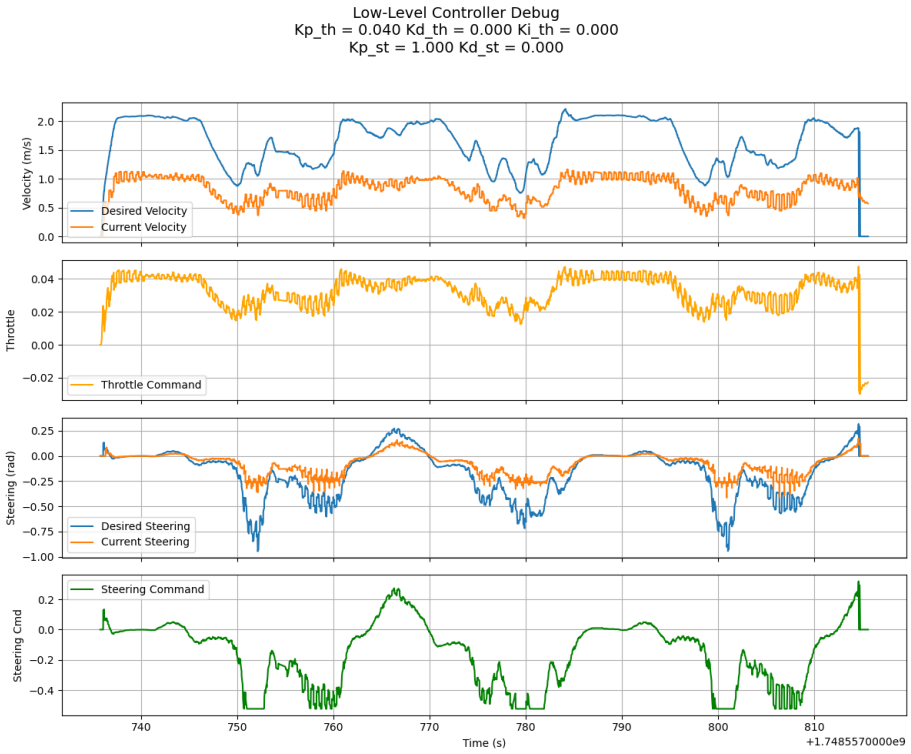
Figure 3: Acceleration reference in 3D

3.4 Debug and Visualizations

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

3.5 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.



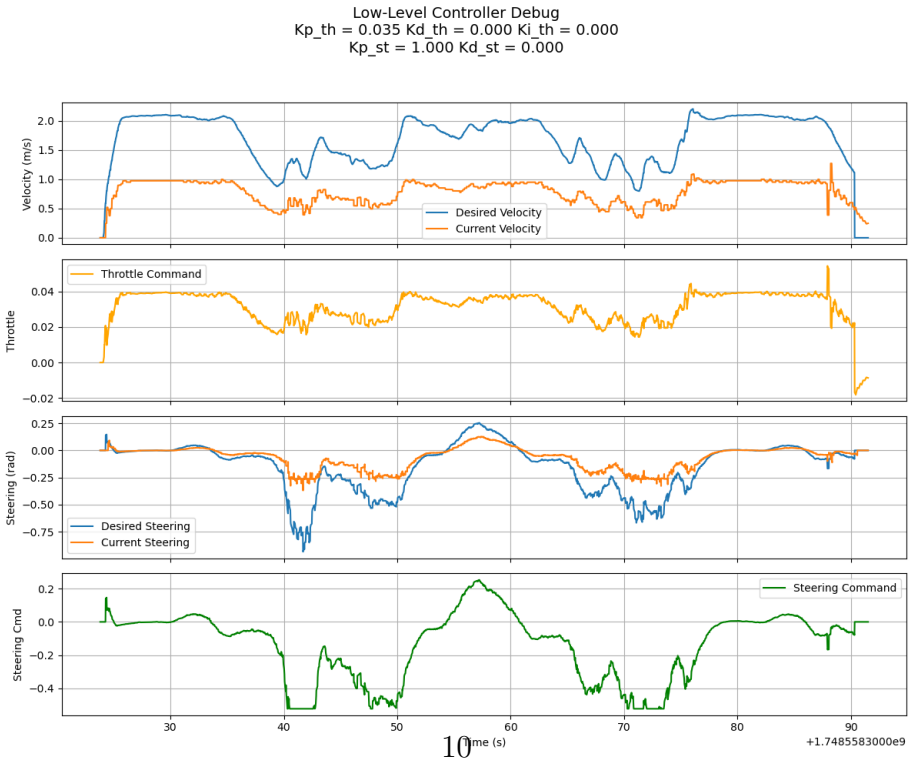


Figure 4: Different Throttle velocity Kp values

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

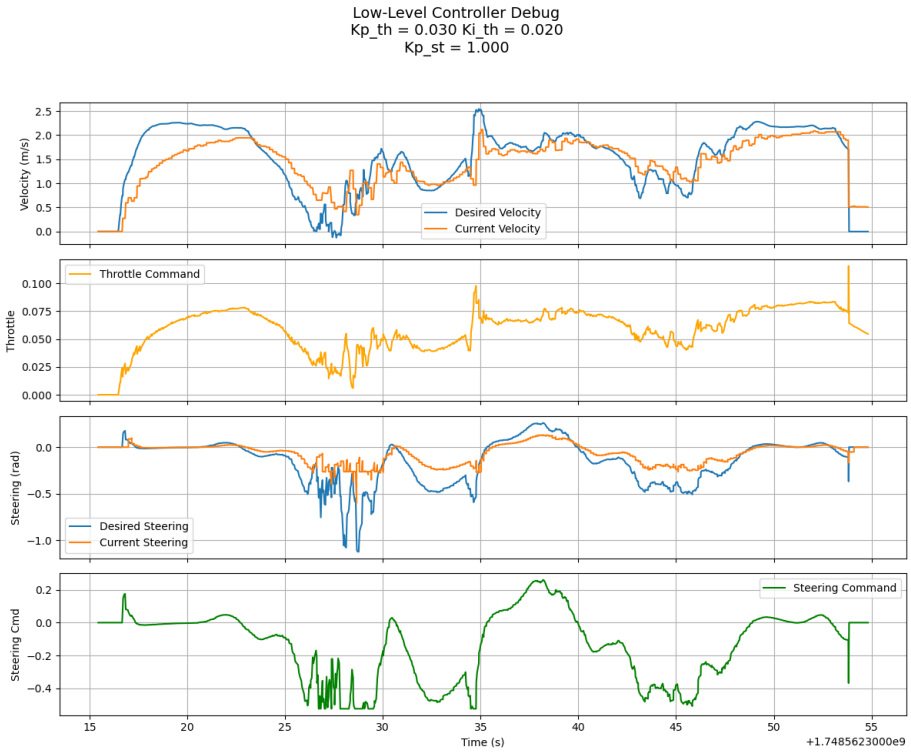


Figure 5: Throttle Kp=0.035, Ki=0.02

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.
The low level PI controller tuning has improved performance dramatically by reducing oscillations thus avoiding crashes and also driving in higher speed closer to the suggested speed by the MPPI controller search for optimality.

4 Current performance and analysis

At first, the performance of the nav2 stack with MPPI and its default configuration were 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 tune the rather big amount of parameters, 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).
Deeper learning of vehicle dynamics is needed in order to get better performance.
The average lap time is still rather poor and stands about 25 seconds per lap.
It took a while to stabilize the controller parameters and low level controller to avoid crashes and complete the laps with a decent time.

5 Conclusions and next steps

1. The Nav2 MPPI controller is not equipped with more accurate dynamic models of a car (like single track dynamic model) and thus needs to be changed for race use case.

2. Deeper learning of the MPPI controller interface parameters, tuning (dt size, amount of time steps, state constraints, etc..), the different critics used for its cost and their own parameters. Try to tune the different weight over the critics. Moreover, the latest MPPI controller version introduced more parameters which can be tuned.

3. New critic can be created for the path align, as the raceline optimization now gives not only (x,y,yaw) states along the ref path, but also speeds that the controller needs to align to. In other words, give high penalty to deviation from reference speeds along the ref track as well.

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. We should consider/explore whether introducing a custom dynamic model of the car would improve performance without incurring too high of a computation time penalty as the MPPI should be run in 20Hz (current configuration but can be changed also depending on batch size and time horizon).
In addition, a dynamic single track model that accounts for slip angles, forces and many other parameters, 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.

6 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

My forked github of the reaceline optimization repo forked from Gong [2]:
https://github.com/ynahum/Raceline-Optimization

References

[1] AutoDRIVE Ecosystem. F1tenth sim racing @ cdc 2024. `https://autodrive-ecosystem.github.io/competitions/f1tenth-sim-racing-cdc-2024/`, 2024. Accessed: 2024-12-15.

[2] Steven Gong. Raceline optimization. `https://stevengong.co/notes/Raceline-Optimization`.

[3] Hilbert J Kappen. Linear theory for control of nonlinear stochastic systems. *Physical Review Letters*, 95(20):200201, 2005.

[4] Nav2 Development Team. Model predictive path integral controller. https://navigation.ros.org/configuration/packages/configuring-mppic.html, 2023. Accessed: 2024-12-15.

[5] ROS Planning. Navigation2. https://github.com/ros-planning/navigation2, 2023. Accessed: 2024-12-15.

[6] Tanmay Samak, Chinmay Samak, Sivanathan Kandhasamy, Venkat Krovi, and Ming Xie. Autodrive: A comprehensive, flexible and integrated digital twin ecosystem for autonomous driving research education. *Robotics*, 12(3), 2023. ISSN 2218-6581. doi: 10.3390/robotics12030077. URL https://www.mdpi.com/2218-6581/12/3/77.

[7] Tanmay Vilas Samak, Chinmay Vilas Samak, and Ming Xie. Autodrive simulator: A simulator for scaled autonomous vehicle research and education. In *2021 2nd International Conference on Control, Robotics and Intelligent System*, CCRIS'21, page 1–5, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450390453. doi: 10.1145/3483845.3483846. URL https://doi.org/10.1145/3483845.3483846.

[8] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Information theoretic model predictive control: Theory and applications to autonomous driving. *arXiv preprint arXiv:1509.01149*, 2015.

[9] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic mpc for model-based reinforcement learning. *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.