Model Predictive Control of Skid-Steered Mobile Robot with Deep Learning System Dynamics

Zhan Dorbetkhany, Matteo Rubagotti, Almas Shintemirov

Outline

1. Introduction.
2. Literature.
3. Methodology.
4. MPC designs and testing.
5. Conclusion.

Introduction

Introduction

Objectives:

1. Develop a path following MPC based on spatial kinematic model of skid-steered mobile robot.
2. Create a framework for development data-driven model predictive control for mobile robots.

This work was inspired by:

G. Huskic, S. Buck, M. Herrb, S. Lacroix, and A. Zell, “High-speed path ´ following control of skid-steered vehicles,” The International Journal of Robotics Research, vol. 38, no. 9, pp. 1124–1148, 2019.

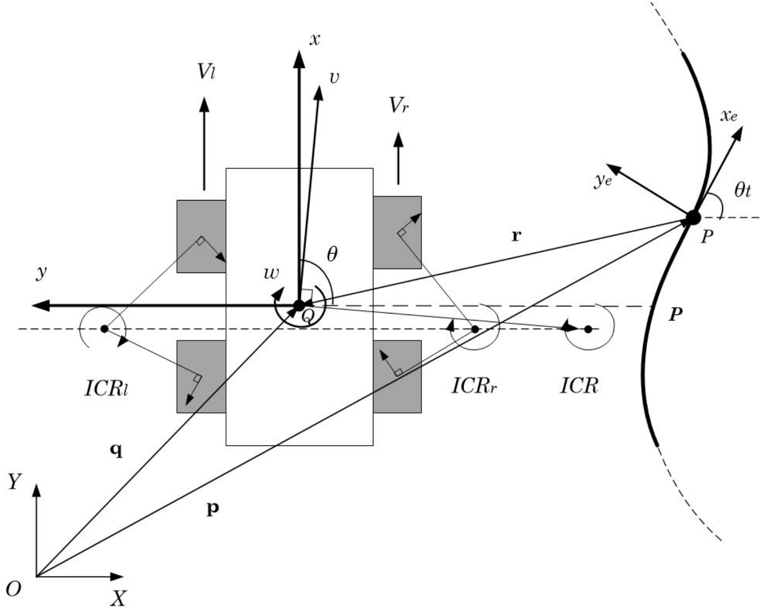
Tim Salzmann, Elia Kaufmann, Jon Arrizabalaga, Marco Pavone, Davide Scara-muzza, and Markus Ryll. Real-time neural mpc: Deep learning model predictive control for quadrotors and agile robotic platforms. IEEE Robotics and Automation Letters, pages 1–8, 2023.

Literature review: MPC for mobile robots

MPC:

1. Used in control of many types of autonomous vehicles.
2. Relies on vehicles motion model.
3. Allows definition of system or obstacle related constraints.
4. Used in path following control using spatial motion models.

SSMR Kinematics



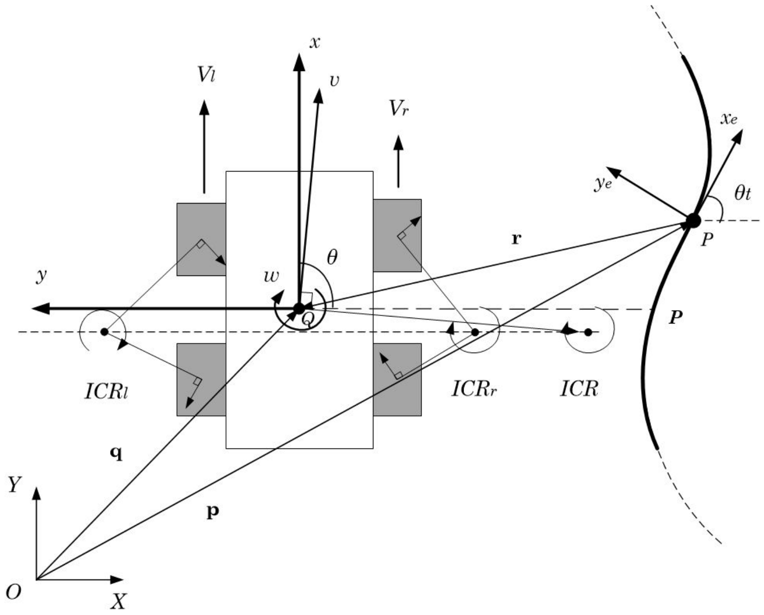
SSMR model in global coordinates

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & x_{ICR}\sin\theta \\ \sin\theta & -x_{ICR}\cos\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ \omega \end{bmatrix}$$

SSMR longitudinal/angular velocity to wheel speed mapping

$$v_x = \frac{\alpha_l y_{ICR_r} v_l - \alpha_r y_{ICR_l} v_r}{y_{ICR_r} - y_{ICR_l}}, \quad \omega = \frac{\alpha_l v_l - \alpha_r v_r}{y_{ICR_r} - y_{ICR_l}}.$$

SSMR Spatial Kinematics



Spatial conversion scheme:

$$\dot{x_e} = \begin{bmatrix}\cos\theta_v & \sin\theta_v\end{bmatrix}\begin{bmatrix}\dot{X}\\ \dot{Y}\end{bmatrix} - \dot{s}(1 - c(s)y_e),$$

$$\dot{y_e} = \begin{bmatrix}-\sin\theta_v & \cos\theta_v\end{bmatrix}\begin{bmatrix}\dot{X}\\ \dot{Y}\end{bmatrix} - c(s)\dot{s}x_e,$$

$$\dot{\theta}_e = \dot{\theta} - c(s)\dot{s},$$

SSMR spatial state-space model

$$\dot{x_e} = v_x\cos\theta_e + x_{ICR}\omega\sin\theta_e - \dot{s}(1 - c(s)y_e),$$

$$\dot{y_e} = v_x\cos\theta_e - x_{ICR}\omega\sin\theta_e - c(s)\dot{s}x_e,$$

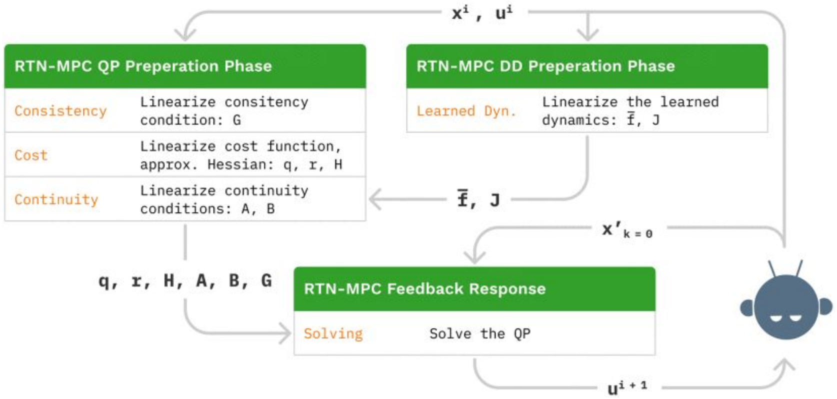
$$\dot{\theta}_e = \omega - c(s)\dot{s}.$$

Path curvature: $c(s) = \dfrac{x'_f(s)y''_f(s) - y'_f(s)x''_f(s)}{(x'_f(s)^2 + y'_f(s)^2)^{\frac{3}{2}}}.$

Literature review: data-driven MPC

Data-driven MPC approaches: parameter inference, residual learning, full dynamics learning.

Tim Salzmann et. al. propose a method for using large neural networks in real-time:



$$f^*_{\mathcal{D}}(\mathbf{x},\mathbf{u}) \approx \bar{\mathbf{f}}^i_{\mathcal{D}} + \mathbf{J}^i_{\mathcal{D},k} \begin{bmatrix} \mathbf{x}-\mathbf{x}^i_k \\ \mathbf{u}-\mathbf{u}^i_k \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \mathbf{x}-\mathbf{x}^i_k \\ \mathbf{u}-\mathbf{u}^i_k \end{bmatrix}^\top \mathbf{H}^i_{\mathcal{D},k} \begin{bmatrix} \mathbf{x}-\mathbf{x}^i_k \\ \mathbf{u}-\mathbf{u}^i_k \end{bmatrix}.$$

Methodology

The project involved several stages:

First stage: path following MPC implementation using ACADO optimization toolkit and MATLAB environment.

First stage has been published in a conference.

Second stage: development of framework for design for time-domain and path following MPC with data-driven component using Python environment, ACADOS optimization toolkit, and Pytorch-Casadi integration.

All testing was done using WeBots simulation environment. Version R2020b for MATLAB, modified R2023a for Python.

Path following MPC in MATLAB

Control Objective: navigate a predefined path and avoid obstacles.

System state: $x = (x_e, y_e, \theta_e, v_l, v_r, s)$

Control inputs: $u = (a_l, a_r)$

System model:
$$
\begin{aligned}
\dot{x}_e &= v_x \cos\theta_e + x_{ICR}\omega \sin\theta_e - \dot{s}(1 - c(s)y_e), \\
\dot{y}_e &= v_x \cos\theta_e - x_{ICR}\omega \sin\theta_e - c(s)\dot{s}x_e, \\
\dot{\theta}_e &= \omega - c(s)\dot{s}, \\
\dot{v}_l &= a_l, \\
\dot{v}_r &= a_r, \\
\dot{s} &= v_x \cos\theta_e + x_{ICR}\omega \sin\theta_e.
\end{aligned}
$$

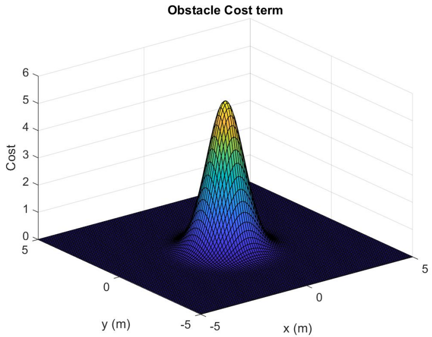
MPC design

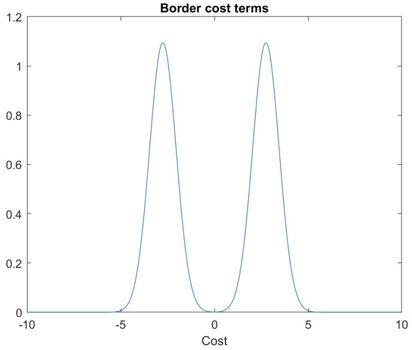
Obstacle avoidance terms:

$$ao = e^{-((y_e - y_{e_{obs}})^2 + (x_e - x_{e_{obs}})^2 - (r_{obs} + r_{car})^2)},$$

$$aiB = e^{-((y_e + track\_width/2)^2 - r_{obs}^2)},$$

$$aoB = e^{-((y_e - track\_width/2)^2 - r_{obs}^2)},$$



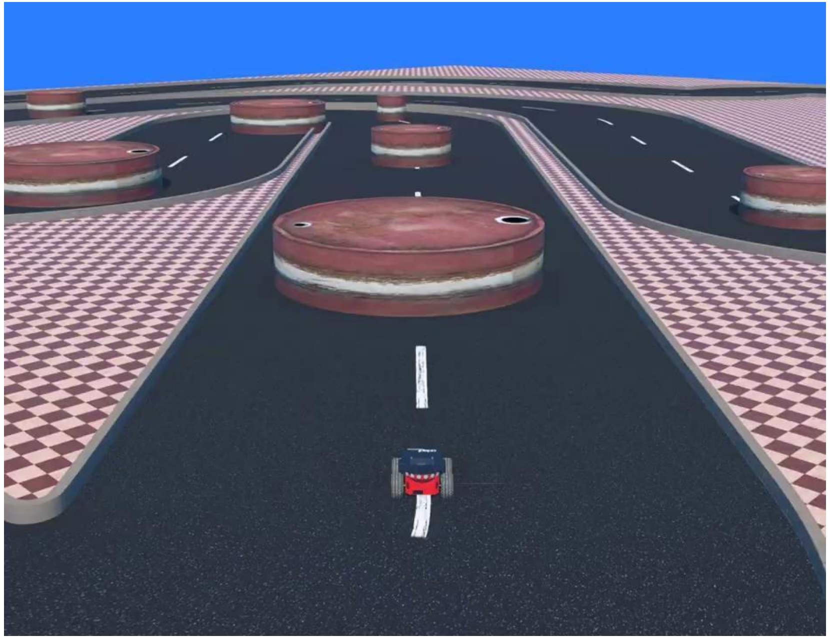


MPC design

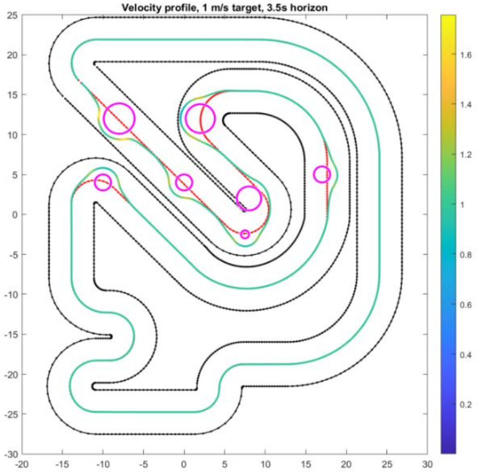
Optimal control problem:

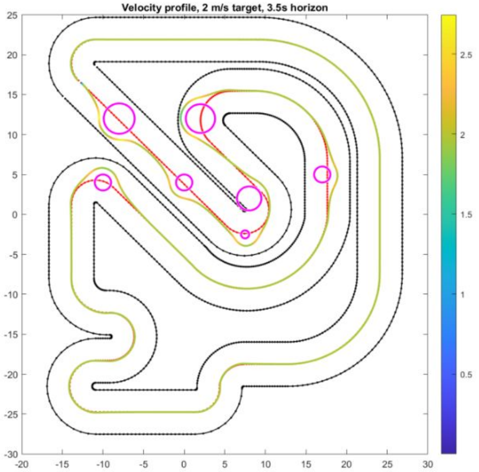
$$
\begin{aligned}
h =& (x_e, y_e, \theta_e, v_l, v_r, v_s, ao, aiB, aoB), \\
h_{ref} &= (0,0,0,0,0, v_{s_{ref}}, 0,0,0). \\
\min \frac{1}{2} &\int_{t_0}^{t_n} ||h(\tau) - h_{ref}(\tau))||_W^2 \\
&\text{s.t. } \dot{x} = f(x,u), \\
&v_{min} \leq v_l \leq v_{max}, \quad v_{min} \leq v_r \leq v_{max}, \\
&a_{min} \leq a_l \leq a_{max}, \quad a_{min} \leq a_r \leq a_{max},
\end{aligned}
$$

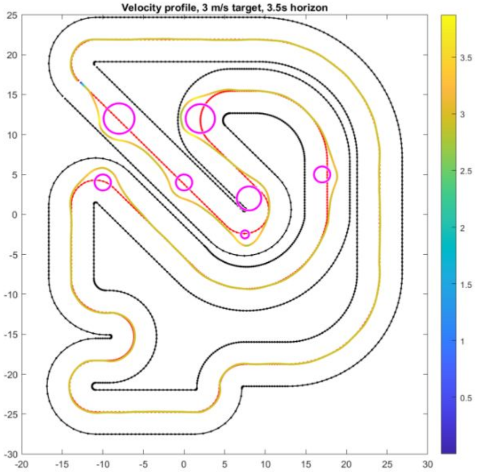
Results: Simulation in WeBots



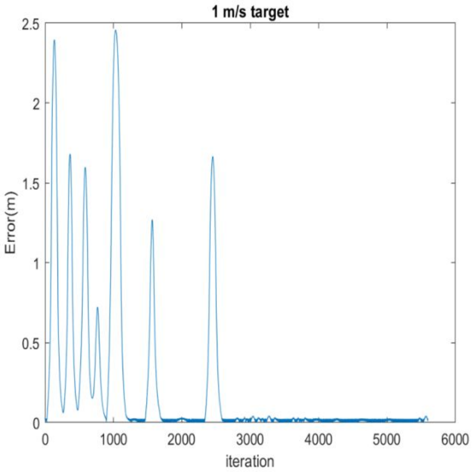
Results: velocity profiles

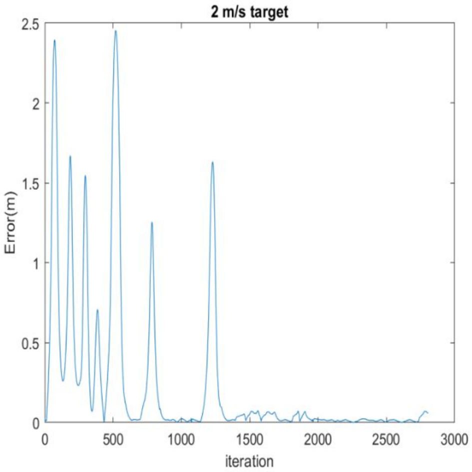


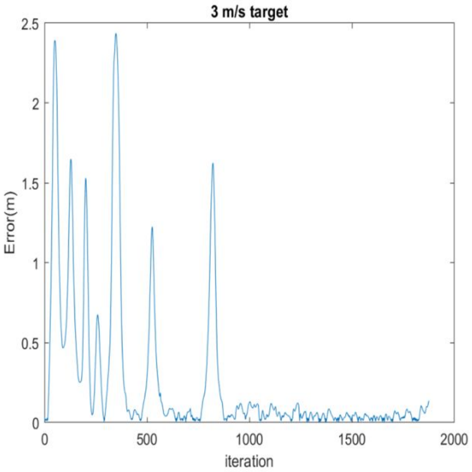




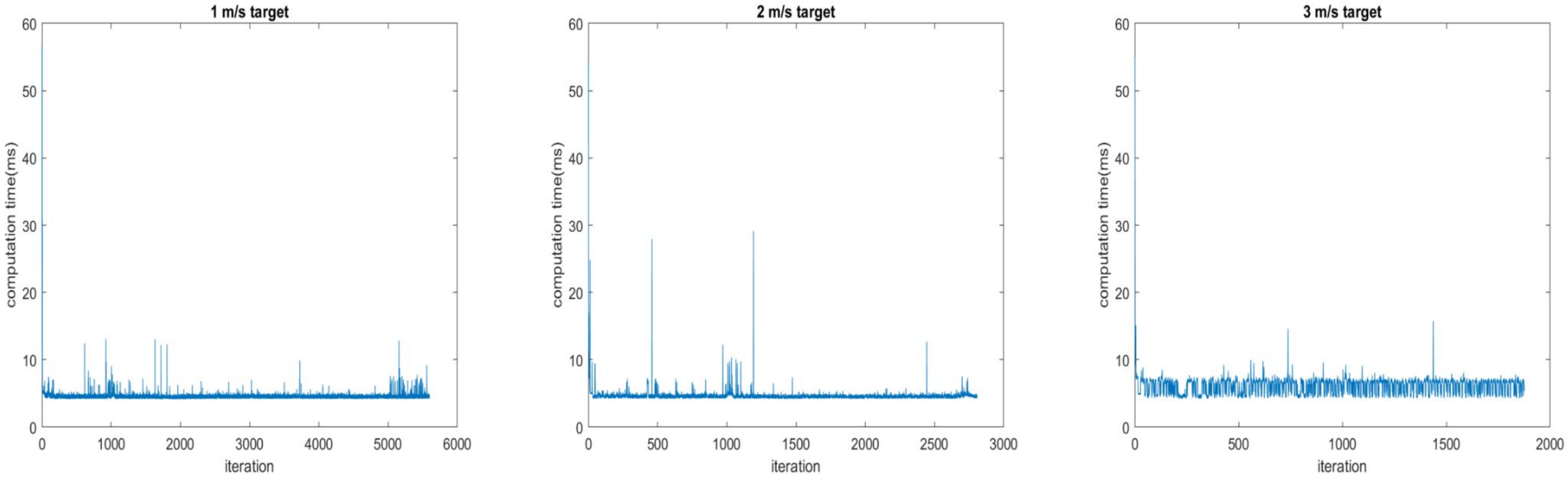
Results: errors



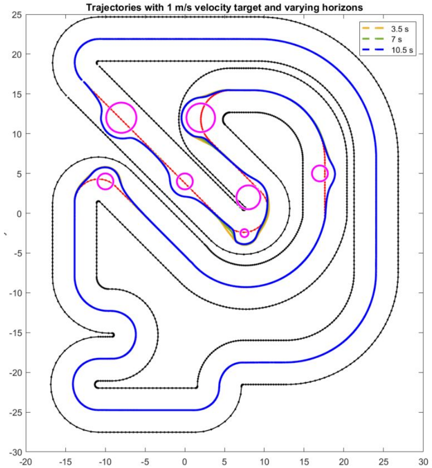


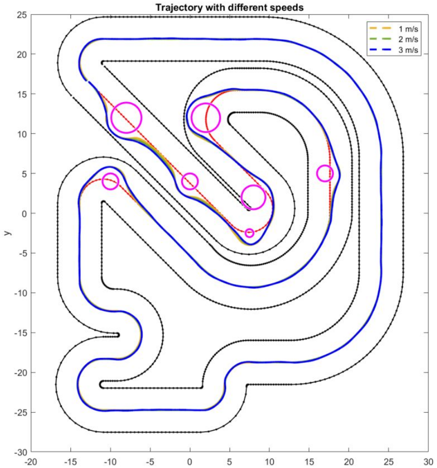


Results: computation times



Results: comparison





Target speed	1m/s	2m/s	3m/s
Average error	0.0155m	0.0255m	0.0501
Average computation time	4.56ms	4.66 ms	6.10ms

Time-domain MPC in Python

Control Objective: reach a reference point.

Control inputs: $u = (a_l, a_r)$

System model:

$$\dot{x} = v_x \cos\theta + x_{ICR}\omega \sin\theta,$$

$$\dot{y} = v_x \cos\theta - x_{ICR}\omega \sin\theta,$$

$$\dot{\theta} = \omega,$$

$$\dot{v}_l = a_l,$$

$$\dot{v}_r = a_r$$

Time-domain MPC in Python

Optimal Control Problem:

$$h = (x, y, \theta, v_l, v_r, a_l, a_r),$$

$$h_{ref} = (x_{ref}, y_{ref}, 0, 0, 0, 0, 0).$$

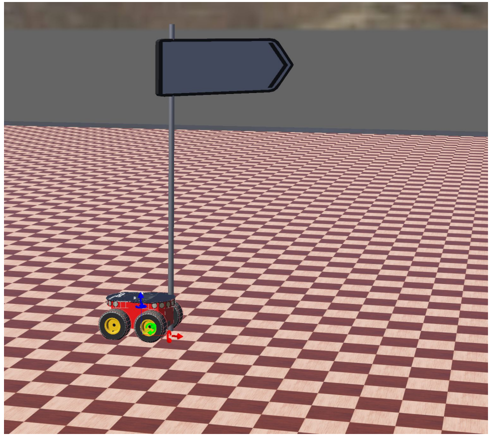
$$\min \frac{1}{2} \int_{t_0}^{t_n} ||h(\tau) - h_{ref}(\tau))||_W^2$$

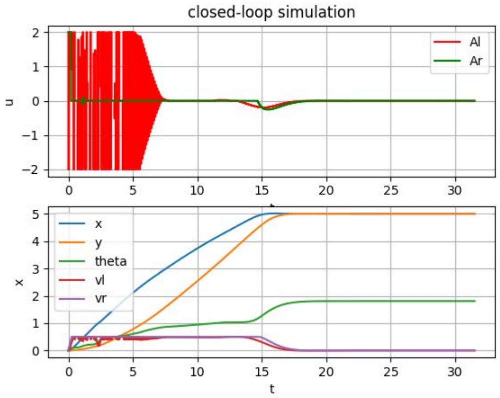
$$\text{s.t. } \dot{x} = f(x, u),$$

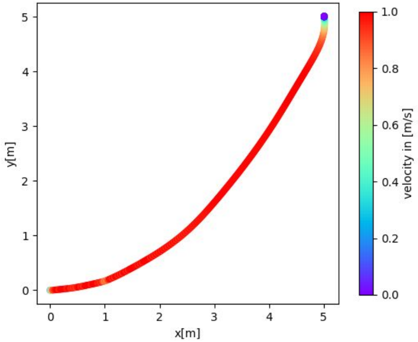
$$v_{min} \leq v_l \leq v_{max}, \quad v_{min} \leq v_r \leq v_{max},$$

$$a_{min} \leq a_l \leq a_{max}, \quad a_{min} \leq a_r \leq a_{max},$$

Time-domain MPC in Python: testing







Data-driven Time-domain MPC

Control Objective: reach a reference point.

Control inputs: $u = (a_l, a_r)$

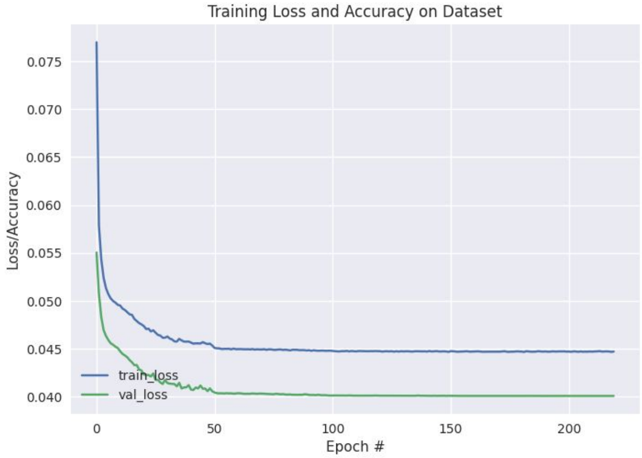
System model: $\dot{x} = f_{NN}(x, u)$, approximation of a two-layer neural network

Data-driven Time-domain MPC

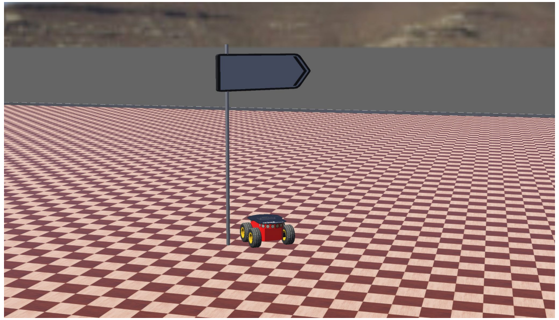
OCP: same as in time-domain MPC, but with learned state-space model.

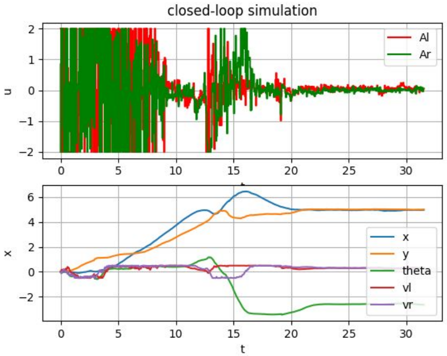
Dataset: mix of simulation data and artificially generated data from nominal equations.

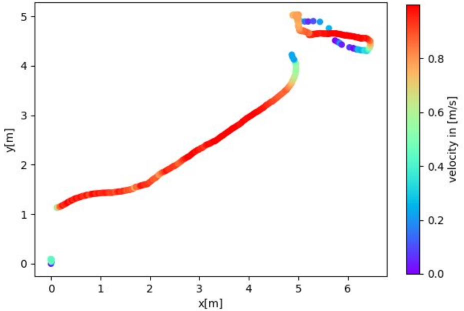
Model training:



Data-driven Time-domain MPC: testing







Path following MPC in Python

Control Objective: navigate a predefined path and avoid borders.

System model: same as in MATLAB implementation.

OCP changed:

$$h = (x_e, y_e, \theta_e, v_l, v_r, s, a_l, a_r),$$

$$h_{ref} = (0, 0, 0, 0, 0, s_{ref}, 0, 0).$$

$$s_{i_{ref}} = s_{current} + iT_s v_{s_{ref}}, i \in [1, N]$$

$$\min \frac{1}{2} \int_{t_0}^{t_n} ||h(\tau) - h_{ref}(\tau))||_W^2$$

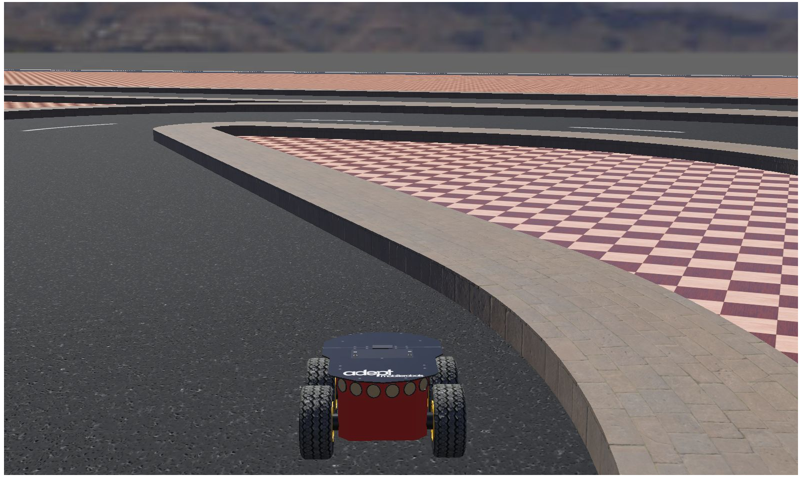
$$\text{s.t. } \dot{x} = f(x, u),$$

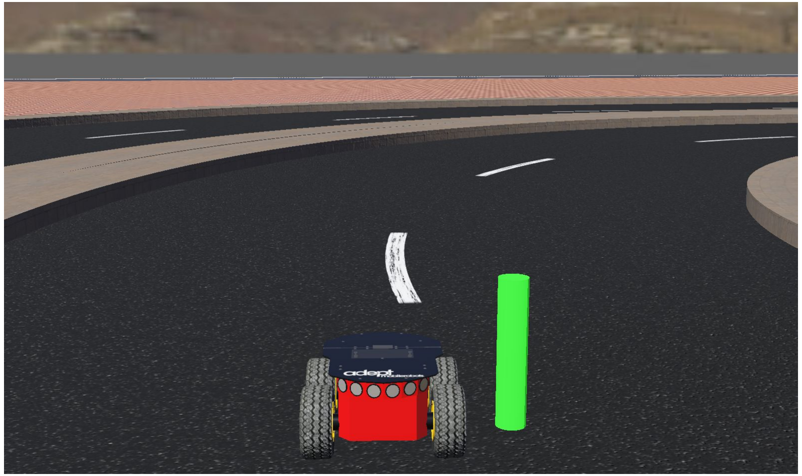
$$v_{min} \leq v_l \leq v_{max}, \quad v_{min} \leq v_r \leq v_{max},$$

$$a_{min} \leq a_l \leq a_{max}, \quad a_{min} \leq a_r \leq a_{max},$$

$$y_{e_{min}} \leq y_e \leq y_{e_{max}},$$

Path following MPC in Python:testing





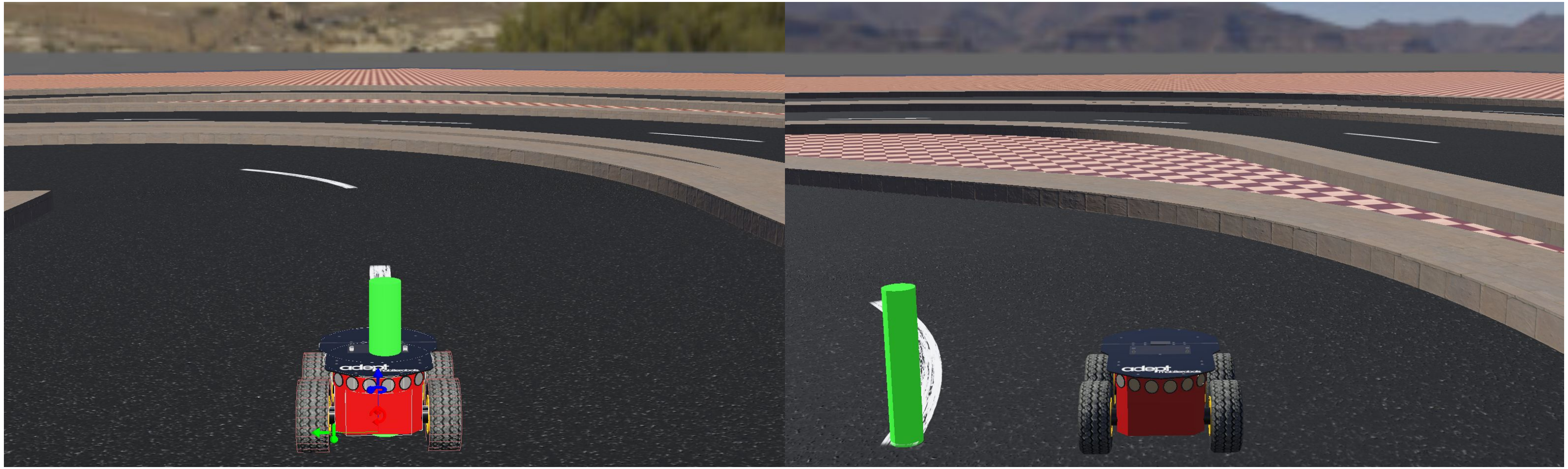
Data-driven Path following MPC

Control Objective: navigate a predefined path and avoid borders.

System model: $\dot{x} = Kf(x,u) + (1-K)f_{NN}(x,u),$

OCP: unchanged from python version of path following MPC

Data-driven Path following MPC



Conclusion

Project results:

1. A path following MPC based on spatial kinematic model of SSMR with static obstacle avoidance.
2. A framework for development of data-driven MPC of SSMR.
3. Several Python implementations of MPC for SSMR.

Future work

1. Add path following contouring control variant.
2. Develop an effective data-driven MPC of SSMR.
3. Experiment using more advanced simulation and real robotic platforms.