

Project Group, IRF, May 6th 2024

Chronos Miniature Car

Wayne Paul Martis, Shubhankar Kulkarni, Aashiv Shah, Rohit Sahasrabuddhe, Nicolás Duque, Eman Shahid,
Nabil Miri and Josh Kannemeyer

apl. Prof. Dr. rer. nat Frank Hoffmann
Lehrstuhl für Regelungssystemtechnik

Outline

Motivation & Introduction

Trajectory Tracking

Reinforcement Learning

RL in Matlab - Python

Analysis

Conclusion

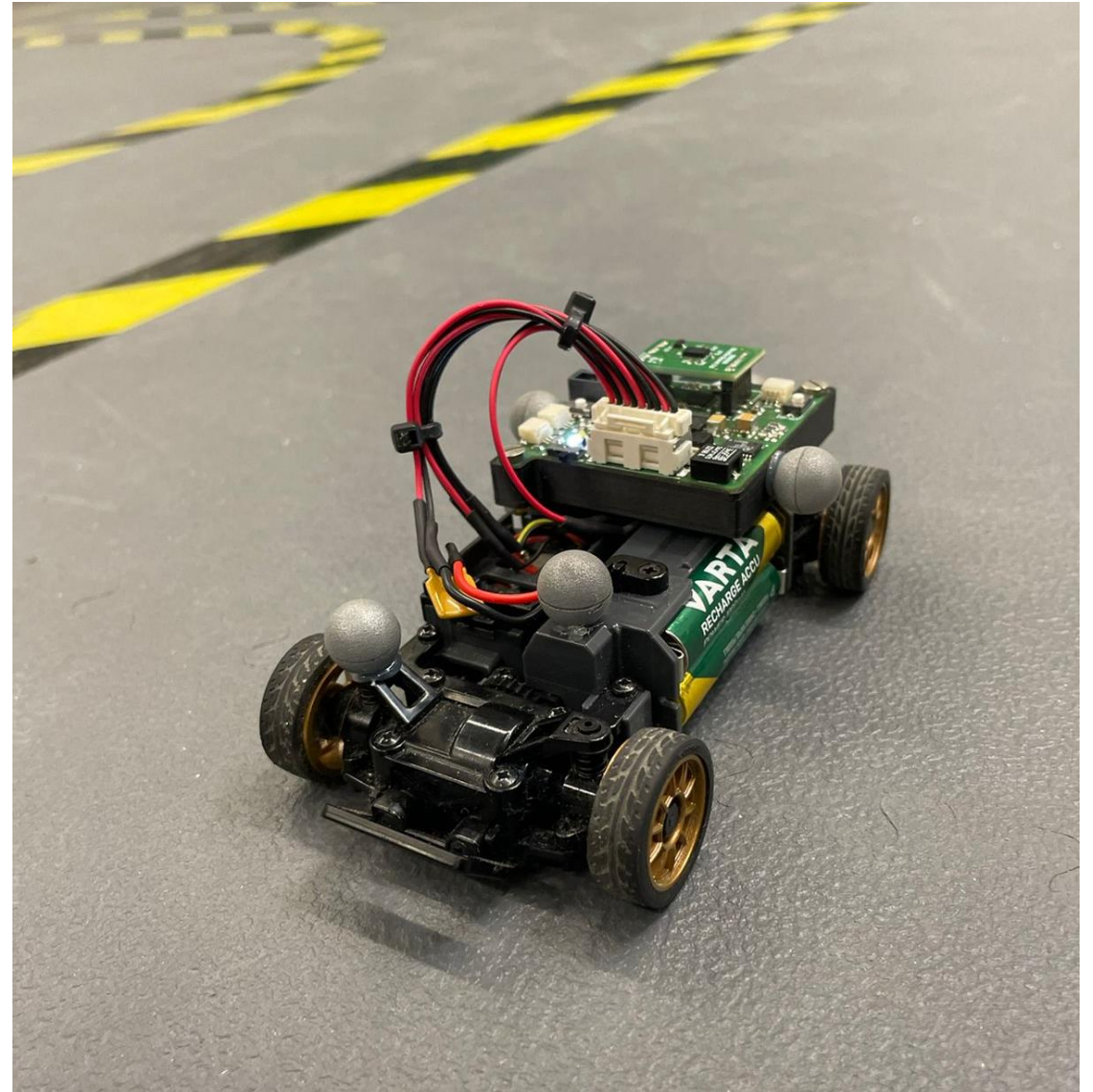
Motivation

- Understanding Reinforcement Learning algorithms.
- Improving accuracy and efficiency
- Teamwork
- Diving deeper into control systems and software development
- Trajectory tracking with empirical dynamic car models.
- Mocap system and hardware



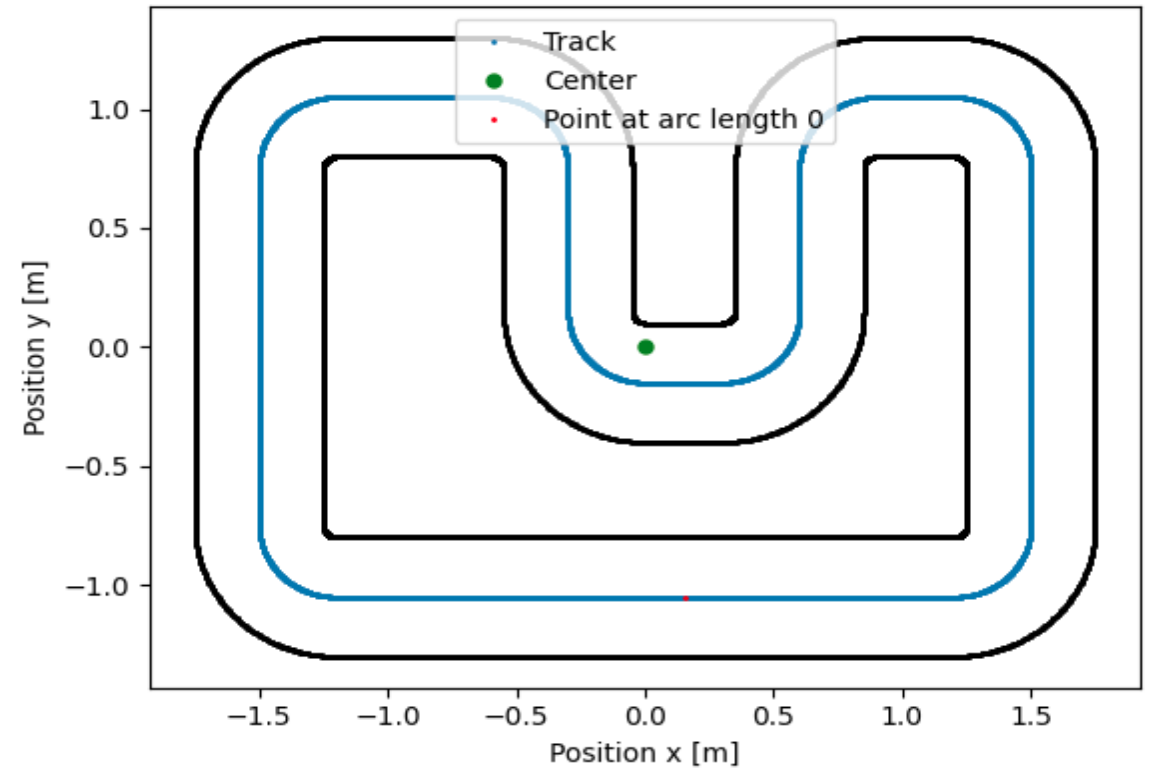
Introduction – Chronos car

- Agile Car-like robot (1/28th scale of a car)
- Uses the Ackermann steering mechanism.
- Equipped with open-source CRS software framework
- Allowing the implementation of custom algorithms in control, localization etc.

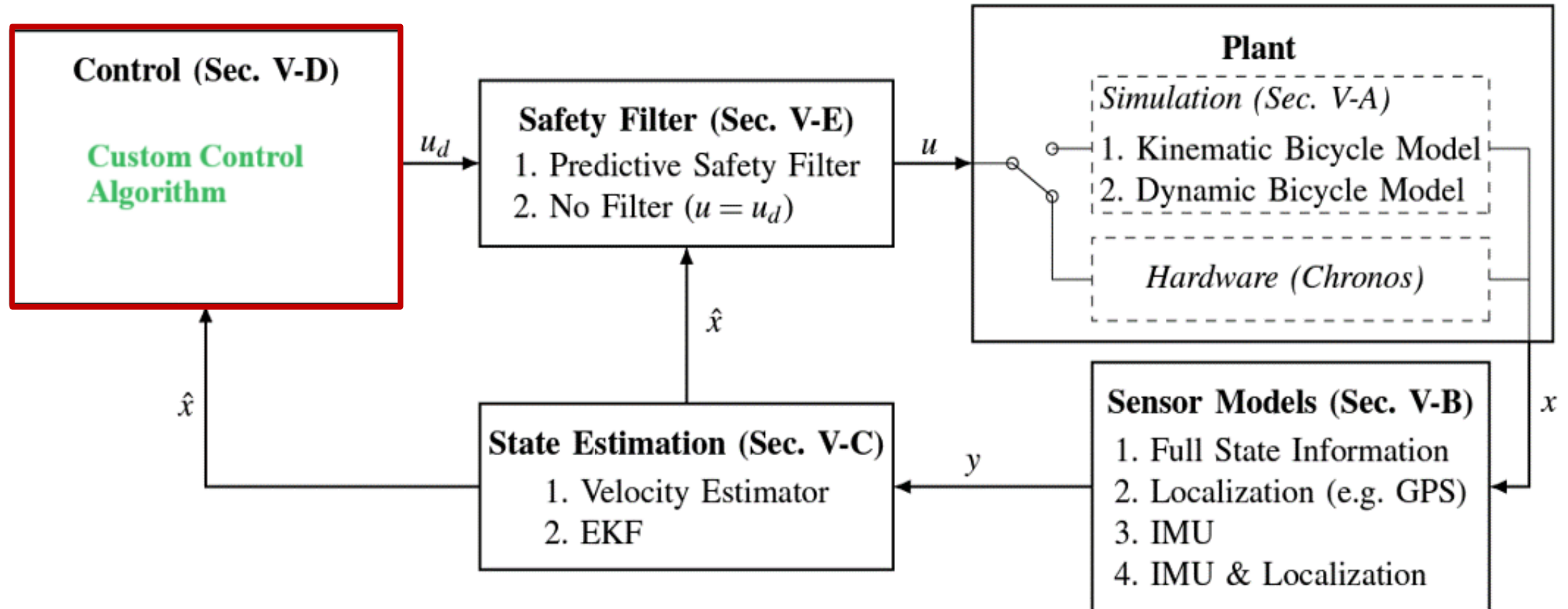


Objective

- Apply various Control and Reinforcement Learning approaches to autonomously drive the car on the track.



Introduction – CRS (Control and Robotics Software)



[1] CRS Framework

Edited image from [1] Andrea Carron, Sabrina Bodmer, Lukas Vogel, René Zurbrügg, David Helm, Rahel Rickenbach, Simon Muntwiler, Jerome Sieber and Melanie N. Zeilinger, "Chronos and CRS: Design of a miniature car-like robot and a software framework for single and multi-agent robotics and control," 2023 IEEE International Conference on Robotics and Automation (ICRA), London, United Kingdom, 2023, pp. 1371-1378

Setup of Testing Arena

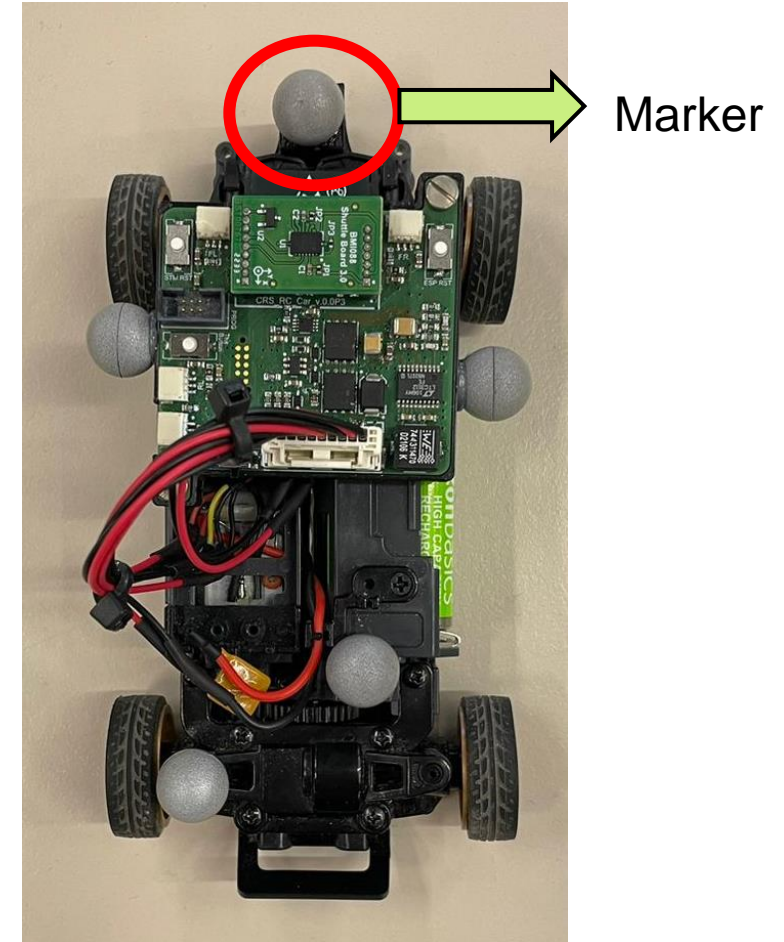
- Localization for hardware experiment via the Optitrack motion capture system.



Testing
Arena



Infrared Cameras
(tracks the markers)

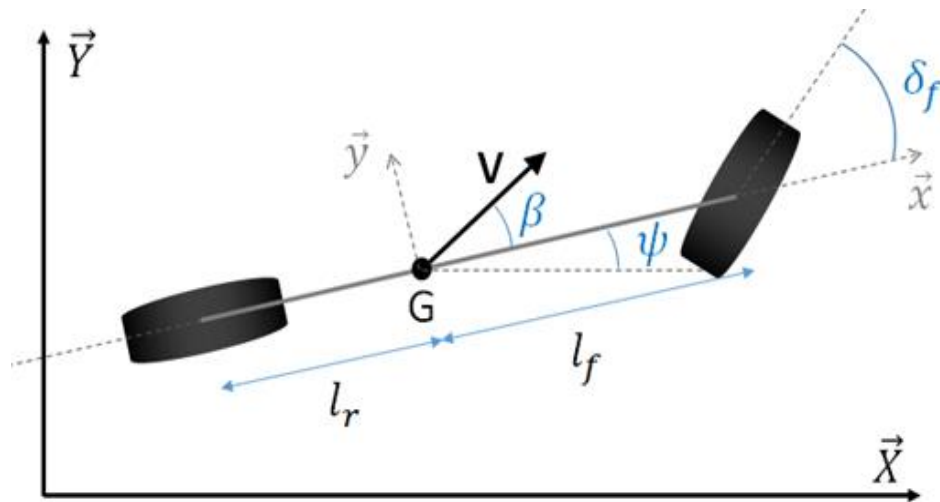


- The state of the car is communicated to the host PC and then sent across the ROS topic.

Car Simulation Model

Kinematic Model

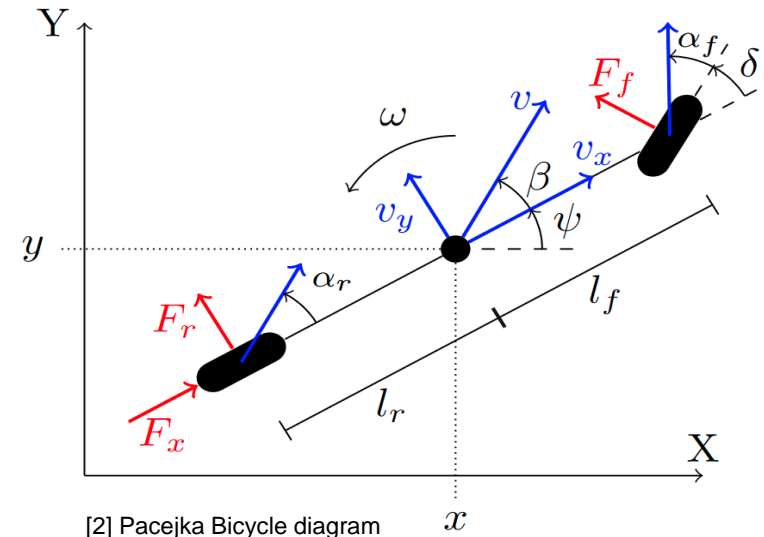
- Front and rear wheels share a common center of rotation
- It neglects lateral slip and other dynamic parameters.



[1] Kinematic Bicycle diagram

Pacejka Model

- Empirical simulation model of the car that considers the slip, various tire forces and friction.
- It realistically depicts the behavior of the car on the track.



[2] Pacejka Bicycle diagram

[1] The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Kinematic-bicycle-model-of-the-vehicle_fig1_318810853 [accessed 23 Apr, 2024]

[2] Fröhlich, Lukas & Küttel, Christian & Arcari, Elena & Hewing, Lukas & Zeilinger, Melanie & Carron, Andrea. (2021). Model Learning and Contextual Controller Tuning for Autonomous Racing.

Outline

Introduction & Background

Trajectory Tracking

Reinforcement Learning

RL in Matlab - Python

Analysis

Conclusion

Path Following Feedback Control

- Reference path frame: $[X^\sigma, Y^\sigma, \psi^\sigma]$
- Vehicle frame: $[X, Y, \psi]$

- Error can be calculated as follow:

- Lateral error

$$e^y = \cos(\psi^\sigma)(Y - Y^\sigma) - \sin(\psi^\sigma)(X - X^\sigma)$$

- Orientation error

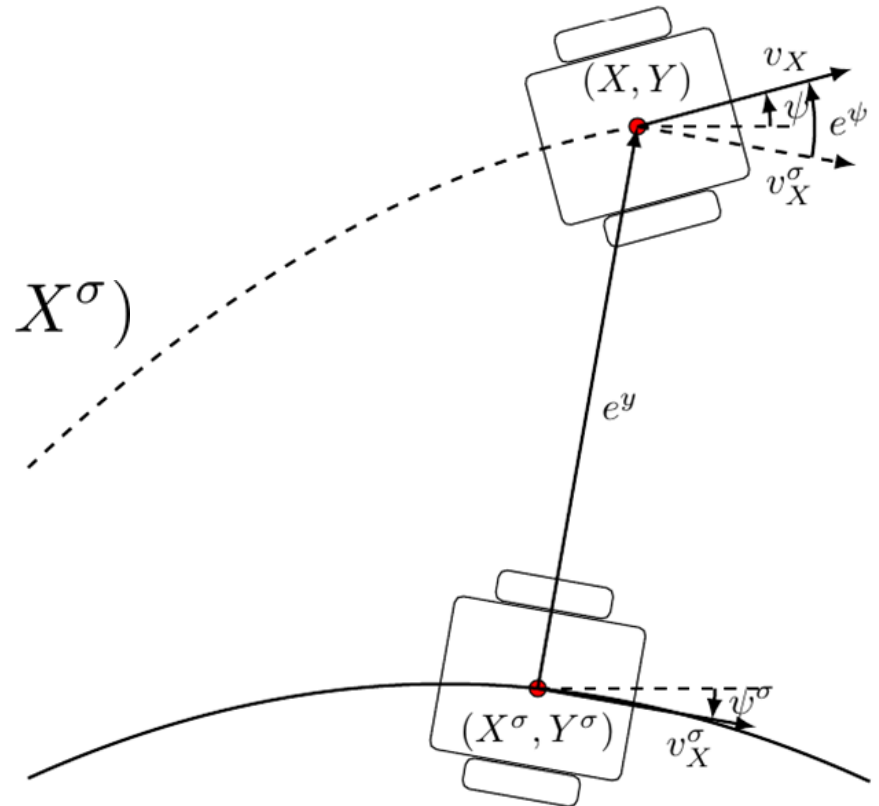
$$e^\psi = \psi - \psi^\sigma$$

- Vehicle pose in the global frame

$$X = X^\sigma - e^y \sin(\psi^\sigma)$$

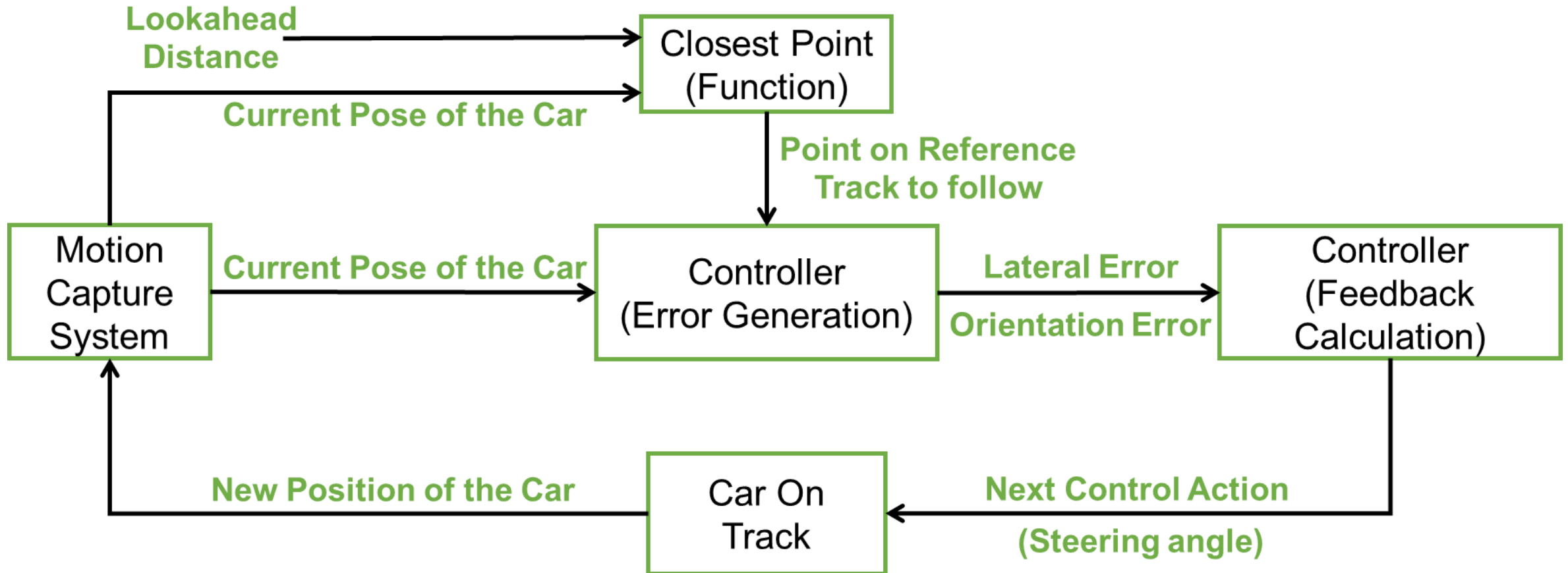
$$Y = Y^\sigma - e^y \cos(\psi^\sigma)$$

$$\psi = \psi^\sigma + e^\psi$$



Source: RST

Custom Controller Node in Matlab/Python



Steering angle feedback for P/PD Controller

- Proportional (P) Controller

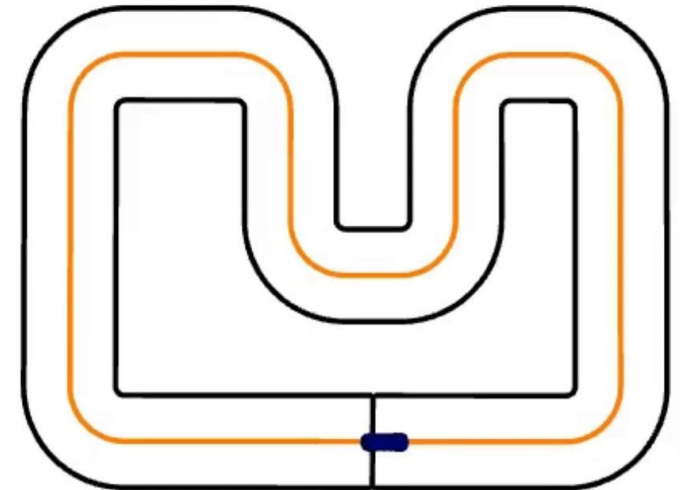
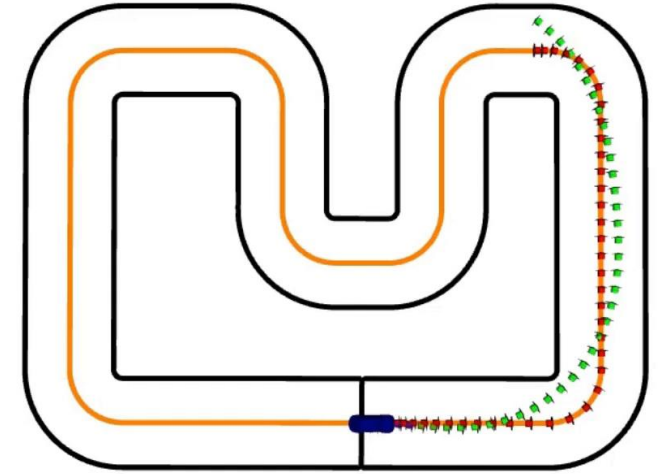
- Considering constant velocity
- Steering angle feedback control :

$$\delta_f = -k_y e^y - k_\psi e^\psi$$

- Proportional-derivative (PD) Controller

- Considering constant velocity
- Steering angle feedback control :

$$\delta_f = -k_{py} e^y - k_{p\psi} e^\psi - k_{dy} \frac{de^y}{dt} - k_{d\psi} \frac{de^\psi}{dt}$$



Controller with Feedforward action

- It anticipates the track curvature and incorporates the equivalent steering angle as a feedforward action

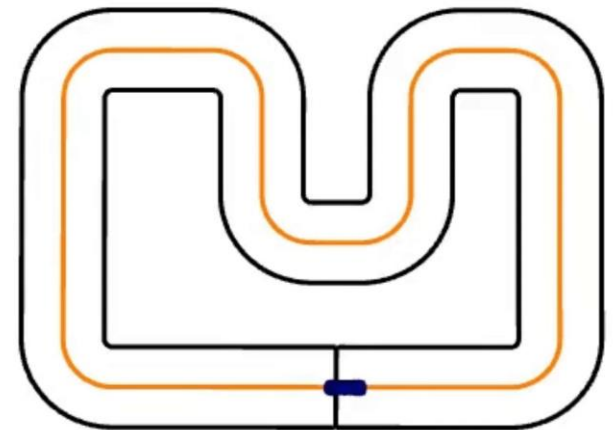
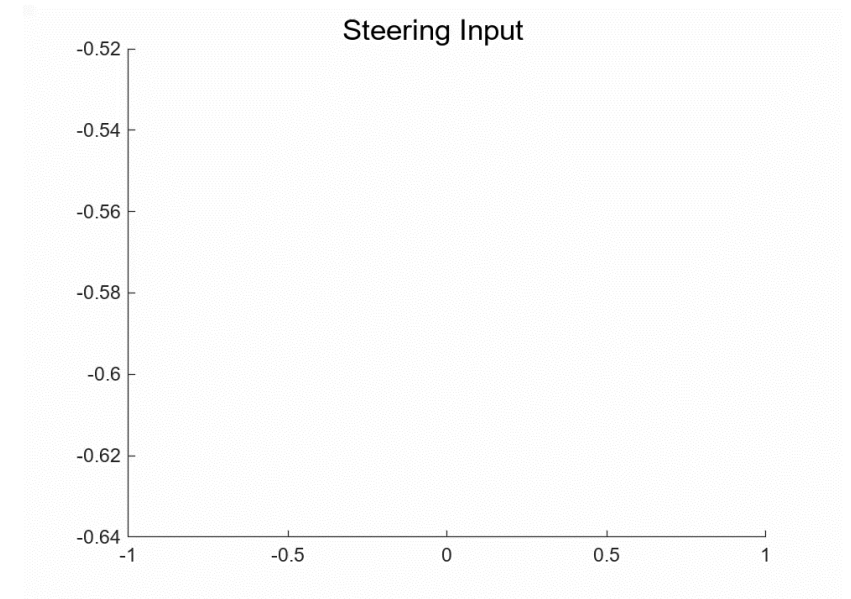
- Steering angle feedforward control

$$\delta_f = -k_y e^y - k_\psi e^\psi + \Delta\delta_{ff}$$

$$\Delta\delta_{ff} = \operatorname{sgn}(\kappa^\sigma) \tan^{-1} \left(\sqrt{\frac{l_f^2 \kappa^{\sigma 2}}{1 - l_r^2 \kappa^{\sigma 2}}} \right)$$

- Curvature of the path

$$\kappa^\sigma = \frac{1}{\rho^\sigma}$$



Hardware Test with Controller with Feedforward



Outline

Introduction & Background

Trajectory Tracking

Reinforcement Learning

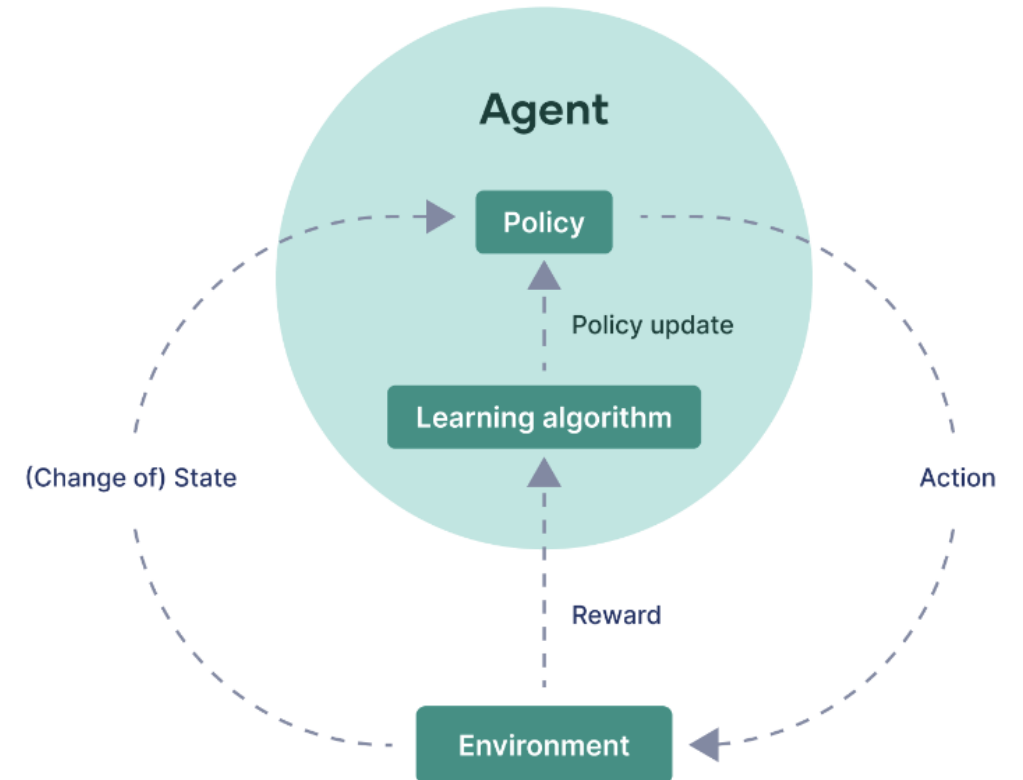
RL in Matlab - Python

Analysis

Conclusion

Why Reinforcement Learning ?

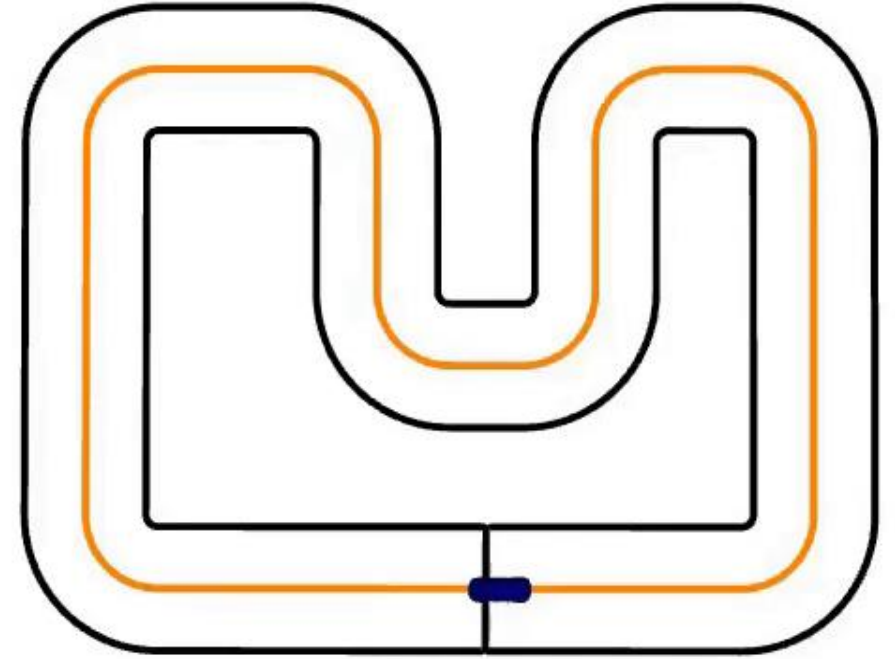
- Reinforcement learning (RL) learns by interacting with the environment, handling complexity without an explicit model.
- Generalization Ability
- Error Minimization
- Optimize the tracking performance.



Source: <https://www.scribbr.com/ai-tools/reinforcement-learning/>

Implementation of Reinforcement Learning

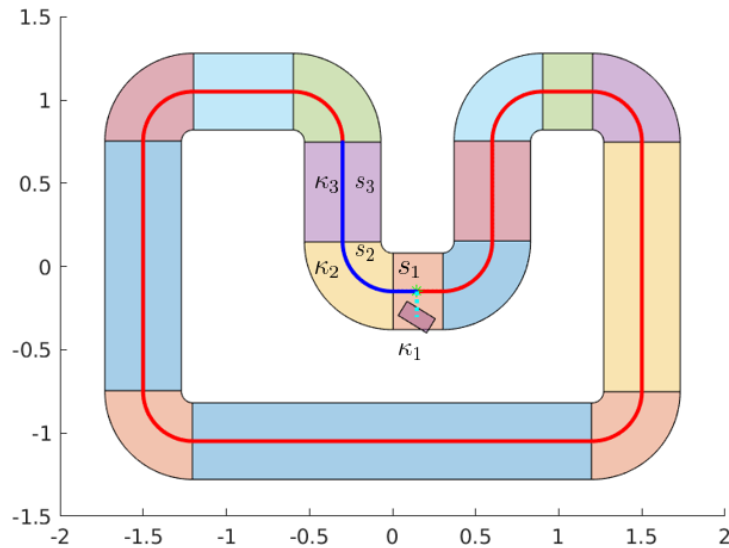
- **Goal**
 - Learn through interaction
 - Enable multiple goals through training
[Lap time - Centerline Tracking - Navigate from any starting position]
- Simulations used:
 - CRS Simulation for **Offline** Training
 - Matlab Simulation for **Online** Training
- Exploring Discounting and ϵ -Greedy strategies
- Exploring learning from interaction using RL and Imitation Learning



Randomized start poses for varied experience buffer

Parameters for Algorithms

Track Segmentation



State Vector

$$state = [e_y, e_\psi, s_1, \kappa_1, s_2, \kappa_2, s_3, \kappa_3]$$

Rewards

- + Progress over track
 - Progress in each segment
 - Extra reward for crossing each segment
- Penalizing lateral & orientation errors
- Penalties for collision
- Penalty for very sharp turns

Action Spaces

- Constant torque, variable steering angles
- Discrete Action Space: Variations in size & resolution of the action space (DQN)



- Continuous action space: $[-\delta, +\delta]$ (DDPG)

Outline

Introduction & Background

Trajectory Tracking

Reinforcement Learning

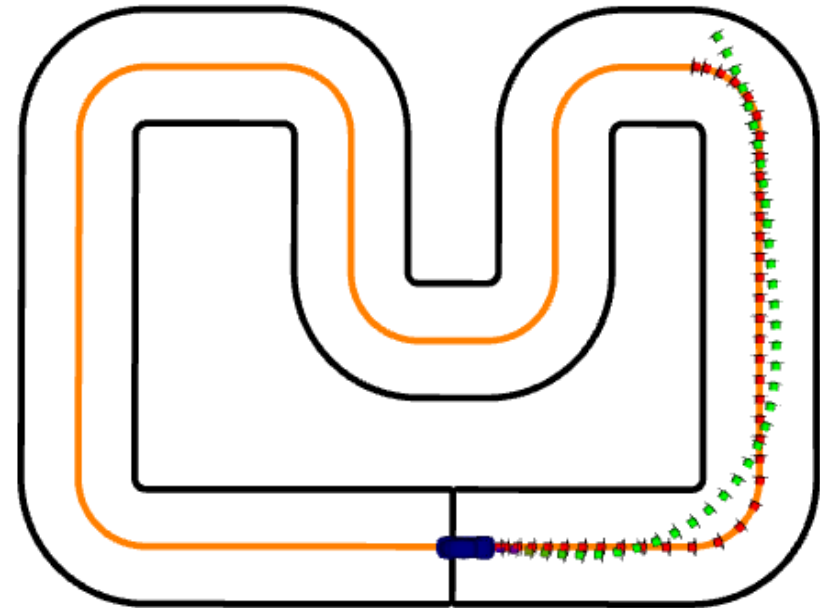
RL in Matlab / Python

Analysis

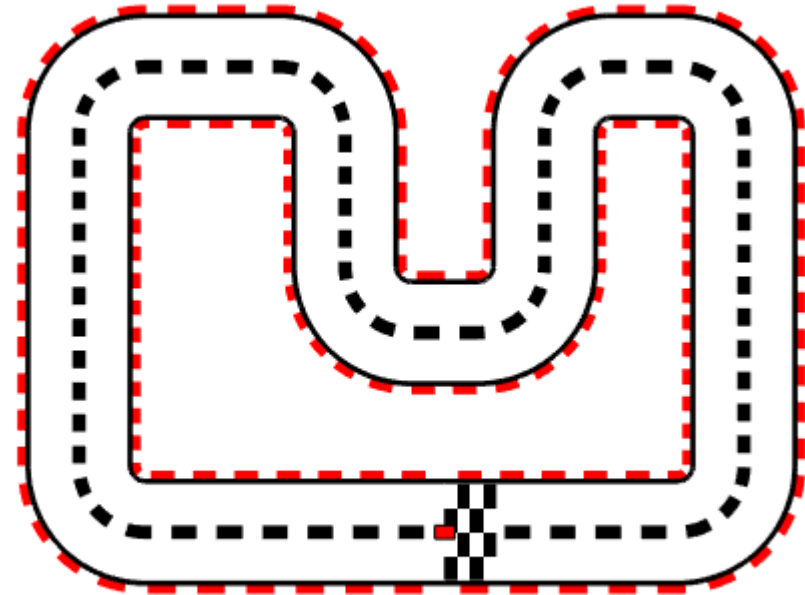
Conclusion

Matlab Simulation [Offline]

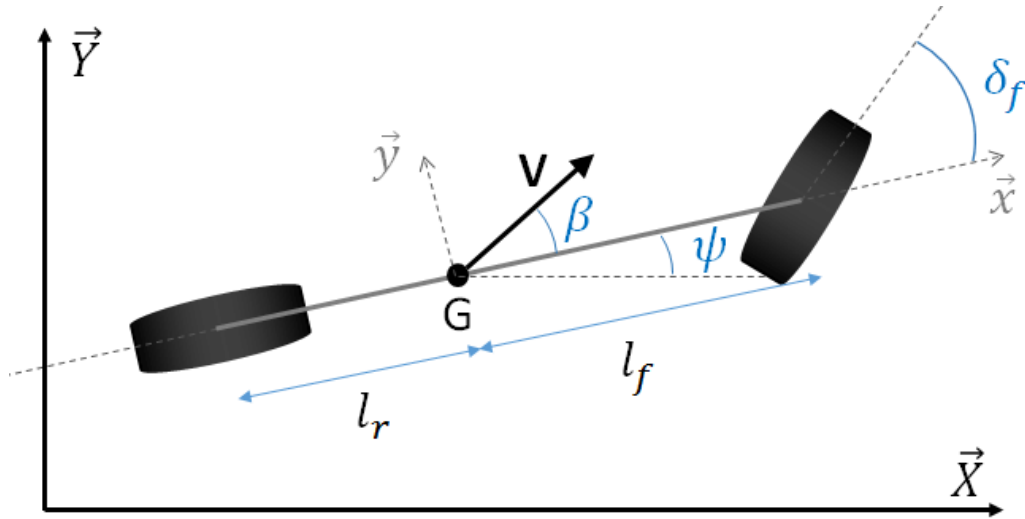
- Pros
 - Accurate representation of vehicle behavior using the Pacejka Model.
 - Enables exploration closer to real-life vehicle dynamics.
- Cons
 - Computationally expensive for Online Training with Matlab



- Pros
 - Able to collect more samples quicker
 - Easier to implement certain functions
- Cons
 - Less accurate model



Online Matlab RL Model



[1] Kinematic Bicycle diagram

Given constants:

$$a = 6.1 \quad l_r = 0.038$$

$$b = 0.2 \quad l_f = 0.052$$

$$\tau = 0.6$$

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v \cdot \cos(\psi + \beta) \\ v \cdot \sin(\psi + \beta) \\ v \cdot \frac{v}{l_r} \sin(\beta) \\ -\frac{v}{\tau} + \frac{u_{kin}}{\tau} \end{pmatrix}$$

$$\beta = \text{atan2}(\tan(\delta) \cdot l_r, l_f + l_r)$$

$$u_{kin} = a \cdot u + b$$

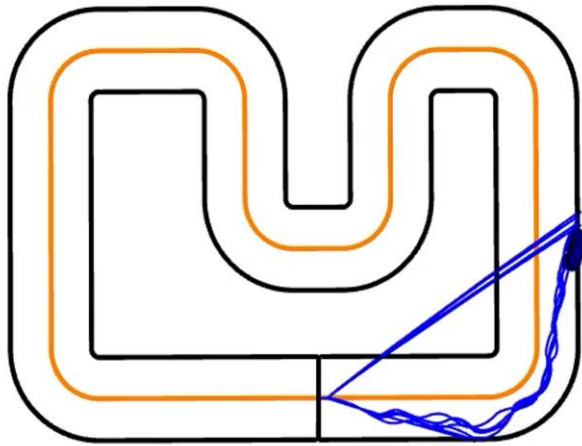
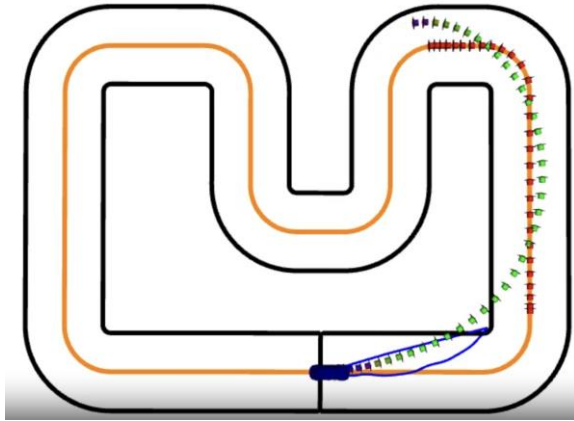
[2] Kinematic equations obtained from CRS Project wiki

[1] The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Kinematic-bicycle-model-of-the-vehicle_fig1_318810853 [accessed 23 Apr, 2024]

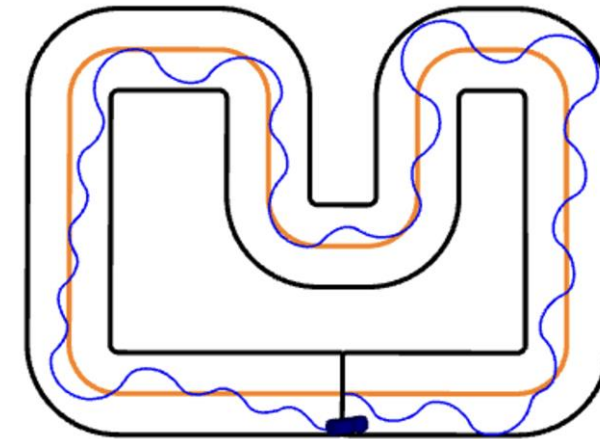
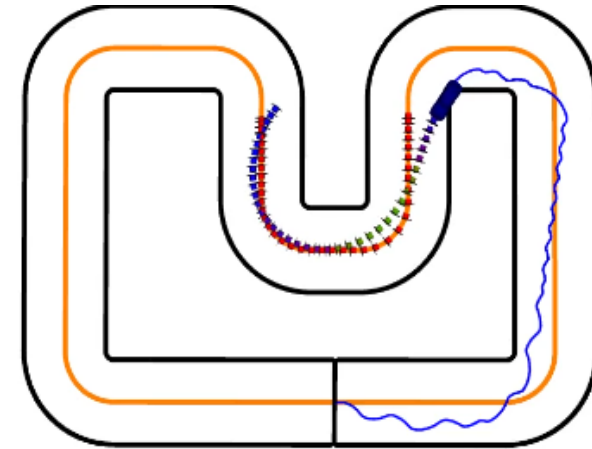
[2] https://gitlab.ethz.ch/ics/crs/-/tree/main/software?ref_type=heads

Offline Learning Results

DQN Agent



DDPG Agent

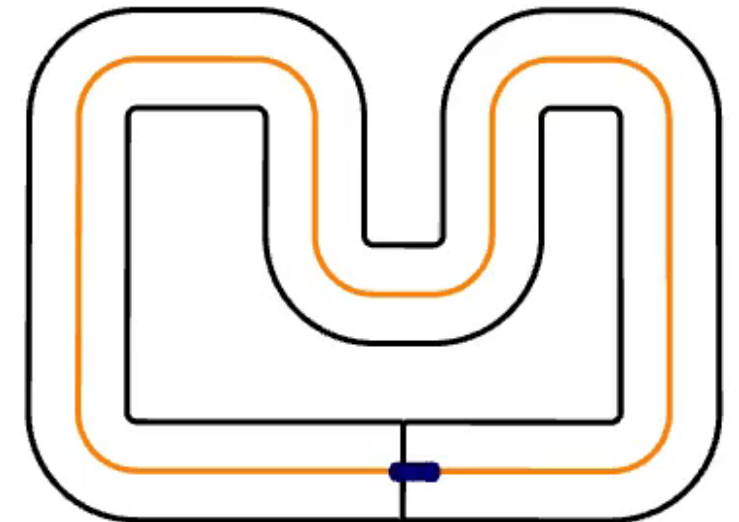
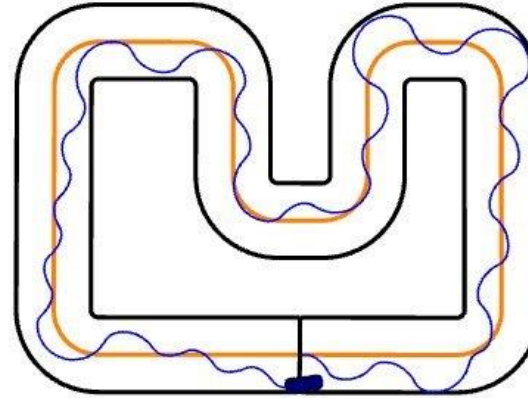


Best Result for Offline Learning using Matlab

- **Parameters:**
- 4000 episodes with 200 steps in each
- State Vector:
 - Current pose
 - Lateral & Orientation Errors
 - Curvature of current & next segment
- Training from 2 poses on the track
- Continuous Action Space (with DDPG Agent)
- Increased sampling time
- Reward

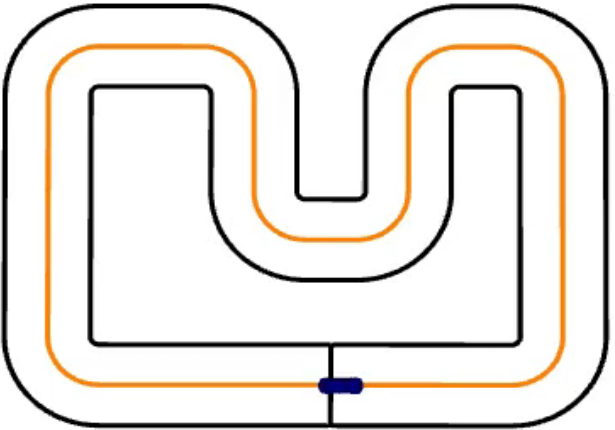
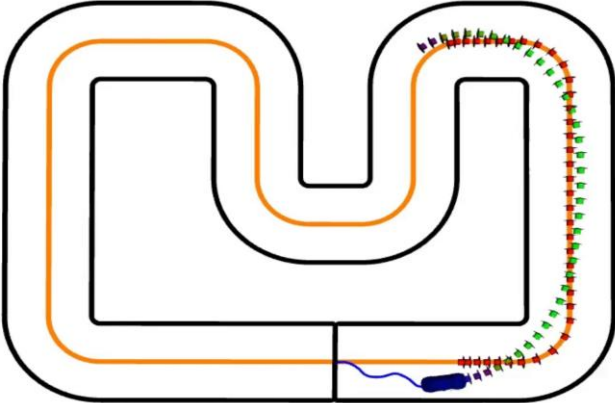
$$R_c = -2 \times (\text{maxStepsPerEpisode} - \text{currentStep})$$

$$R_s = \text{trackProgress} - 20 \times (e^y) - 20 \times (e^\psi)$$

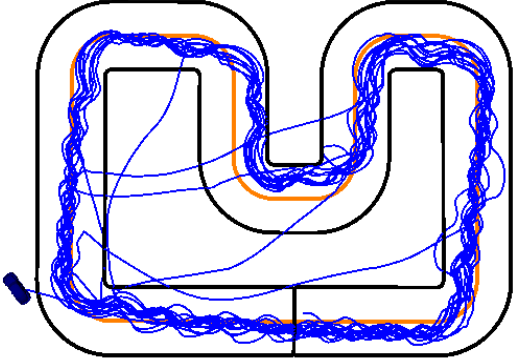


Progress

From: CRS Simulation testing

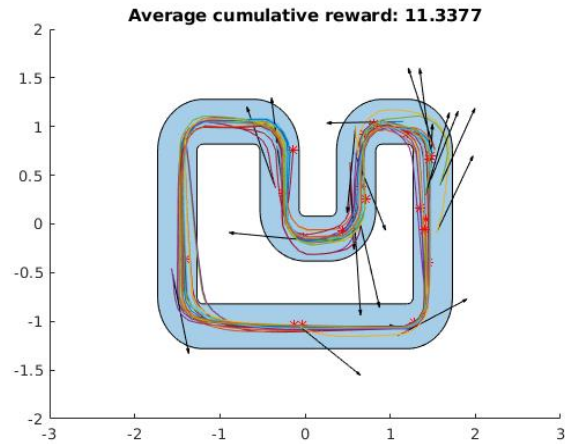


To: Chronos Car Experiment

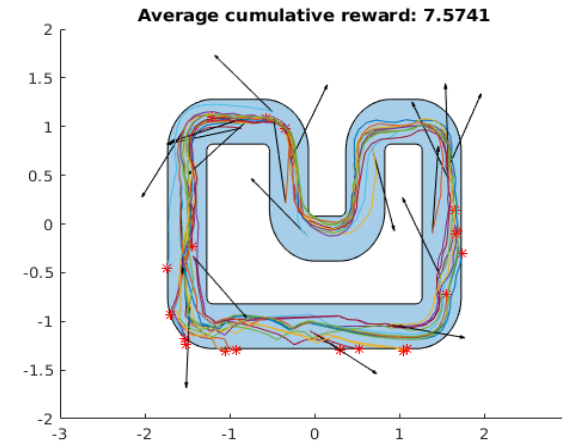


Online Learning Results

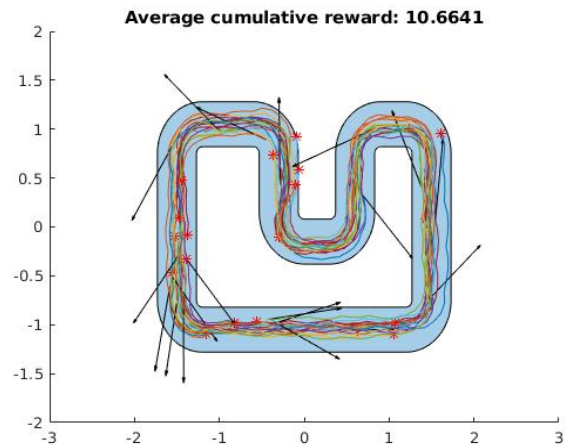
PID Reference



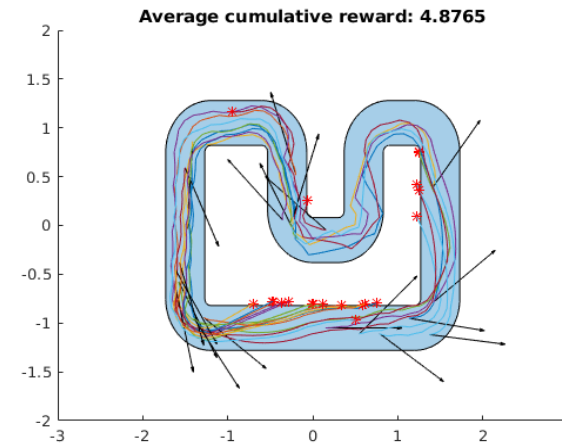
DQN (discrete actions)



PPO (continuous actions)



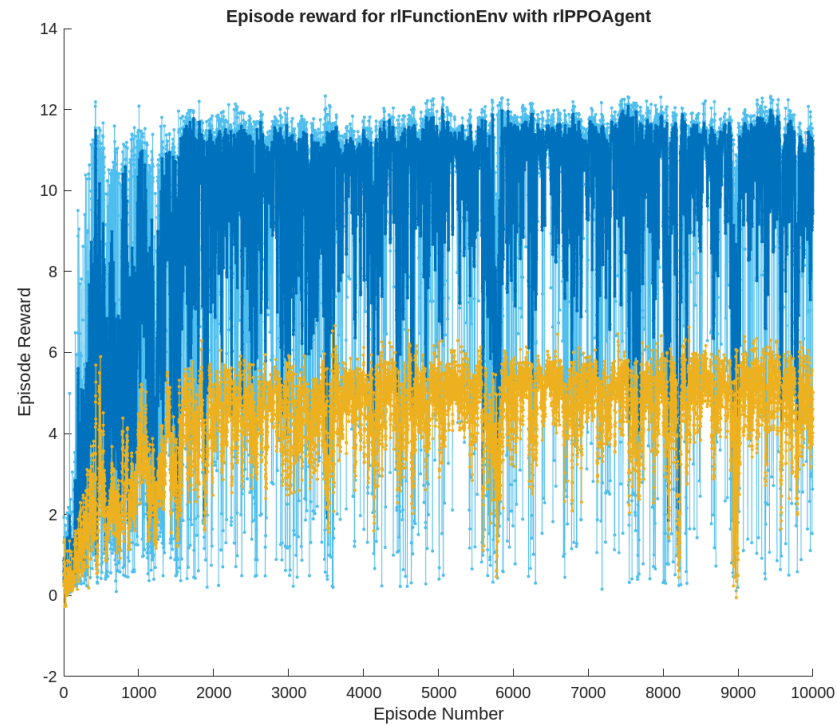
TRPO (continuous actions)



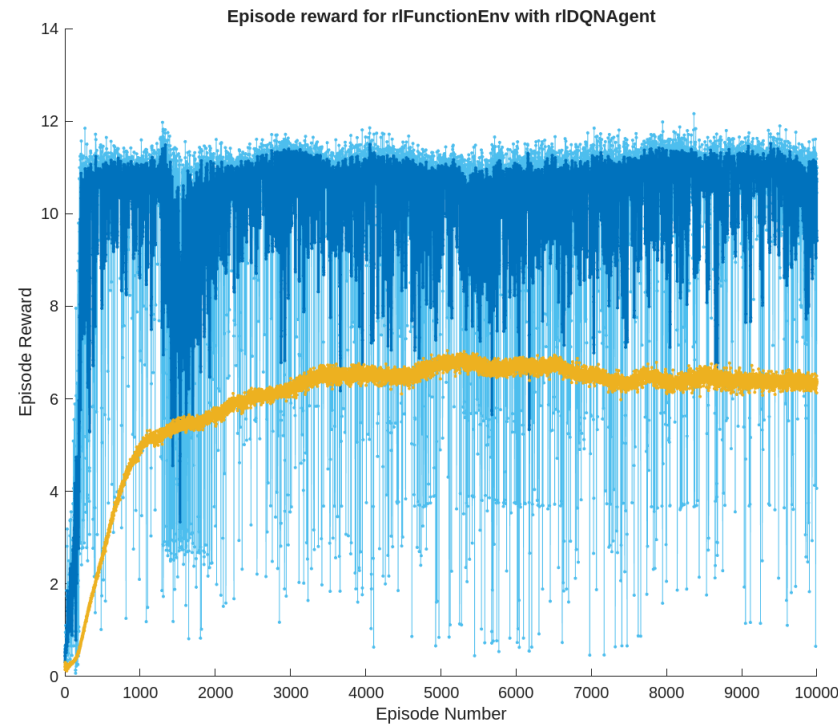
Online Learning Results

- Episode Reward
- Q_0 (the estimate of the discounted long-term reward at the start of each episode)

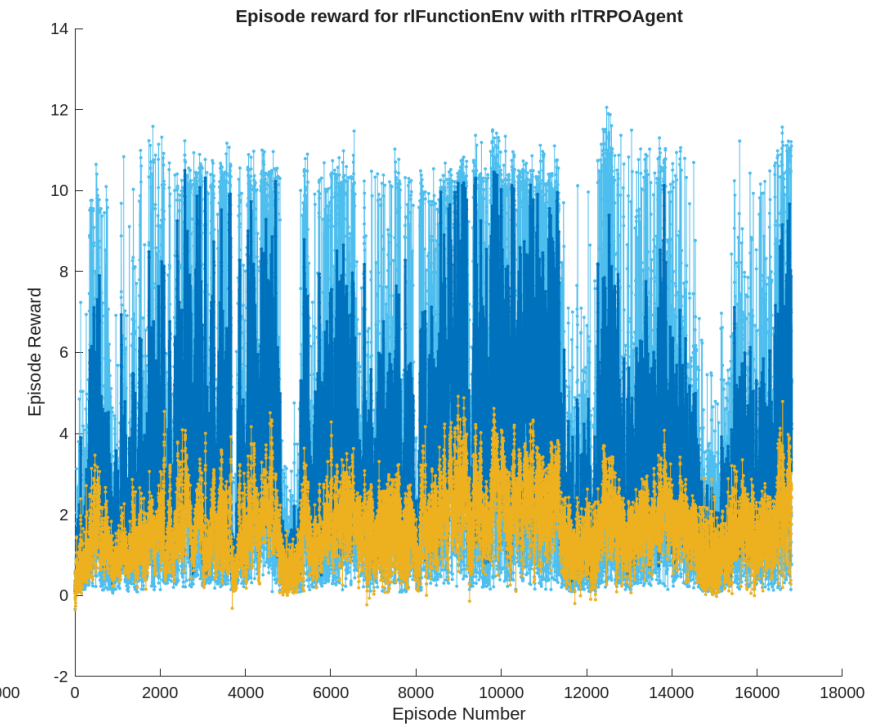
PPO (continuous actions)



DQN (discrete actions)

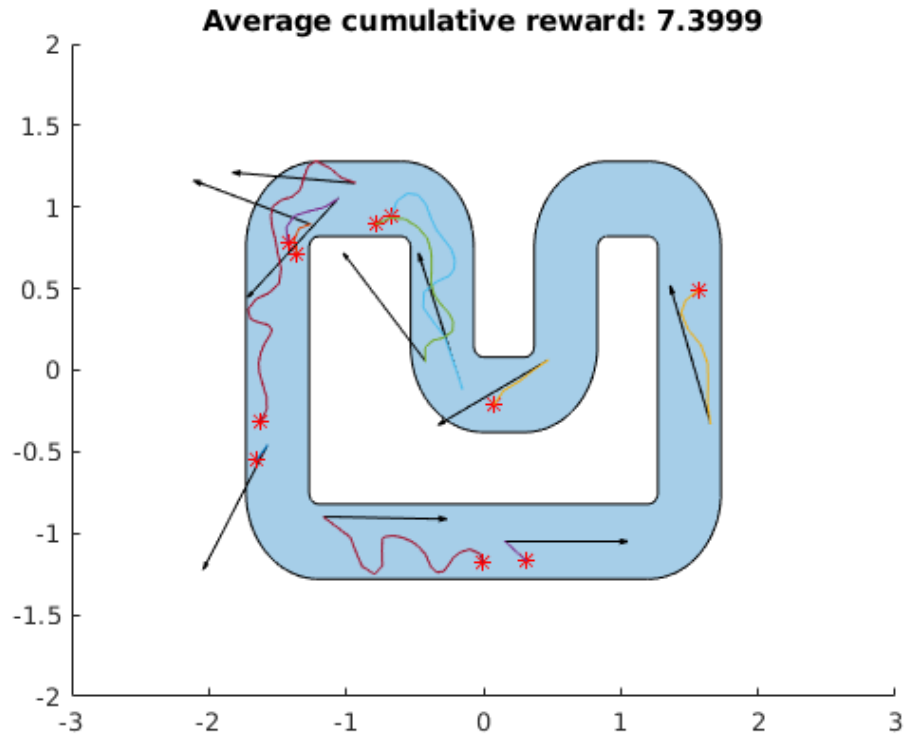


TRPO (continuous actions)

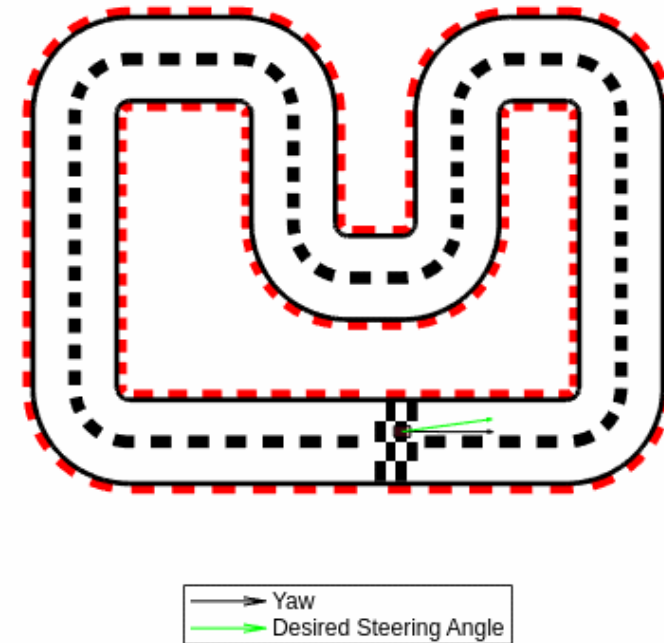


CRS Simulation and Real Car

CRS Simulation (PPO agent)



Real Car (PPO agent)



Outline

Introduction & Background

Trajectory Tracking

Reinforcement Learning

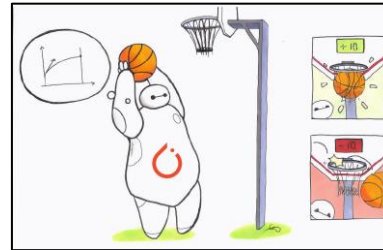
RL in Matlab / Python

Analysis

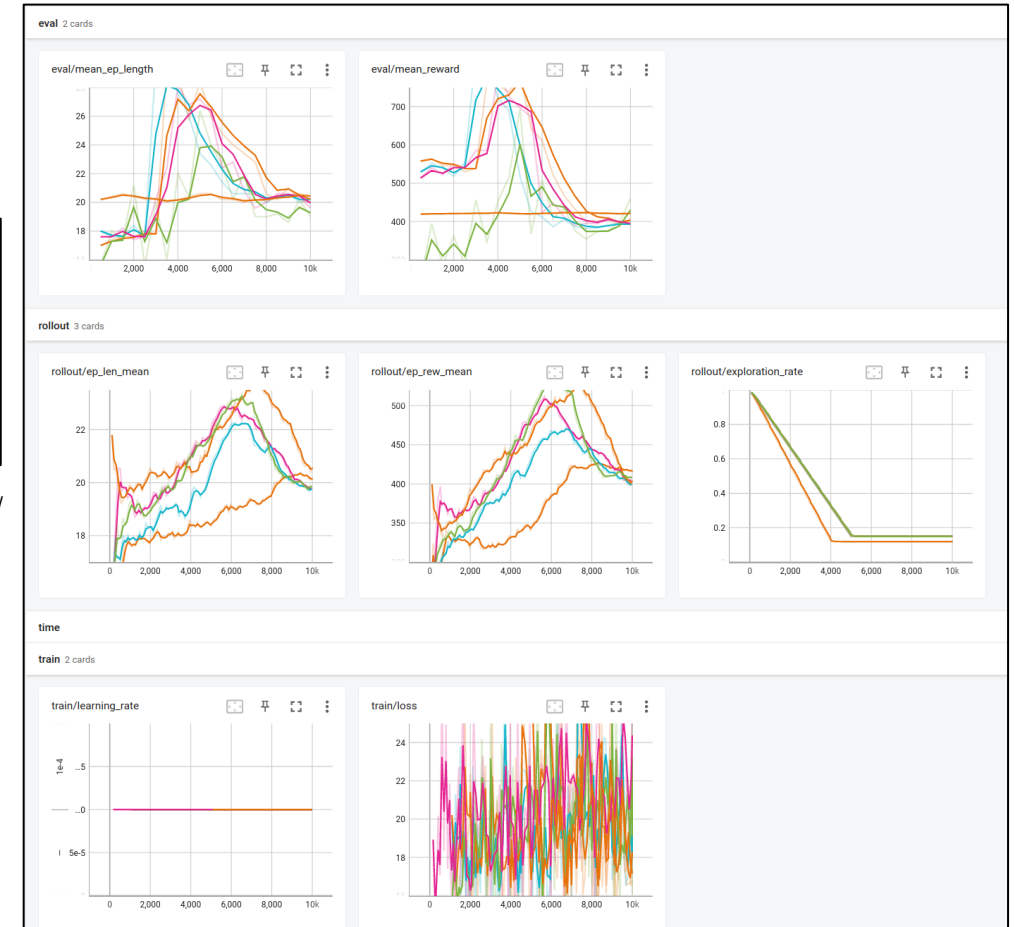
Conclusion

RL framework

- RL Framework: Stable Baselines 3
 - PyTorch implementation of multiple RL algorithms
 - DQN, DDPG, SAC, PPO
 - Features:
 - Unified structure for all algorithms.
 - Tensorboard support.
 - Advantages:
 - Straightforward implementation, training and testing.
 - Disadvantages:
 - Too high level, not possible to see some behaviors.
 - Restrictive for some configurations



Source: <https://stable-baselines3.readthedocs.io/en/master/>




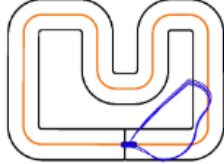



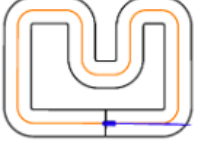

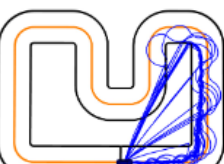
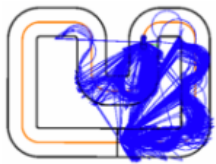

Tensorboard

Environment

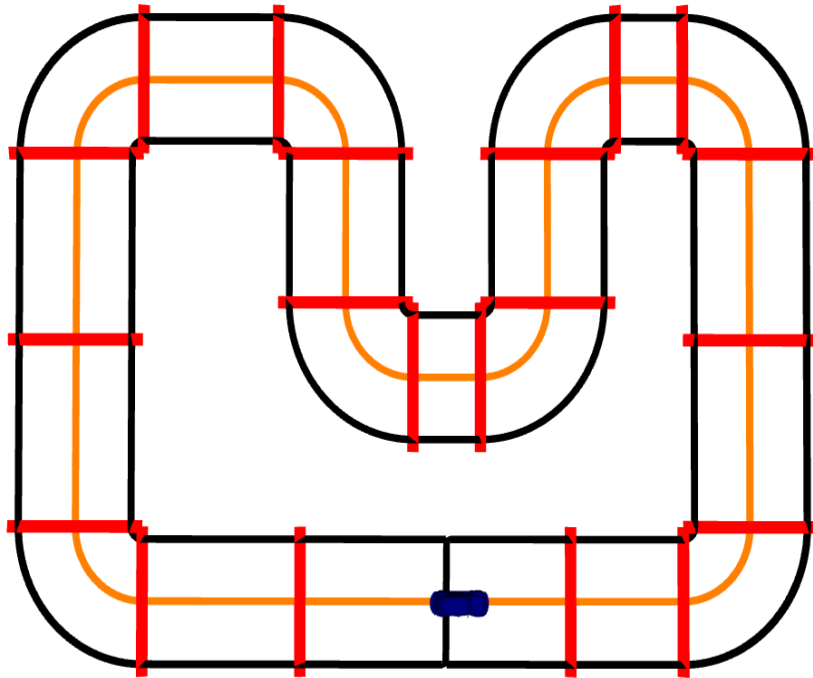
- How to use Stable Baselines 3 with existing CRS simulation (custom) environment ?
 - Gymnasium: API for all single agent reinforcement learning environments
- Structure code to match Gymnasium interface:
 - Implement methods and define attributes from the inherited Env class:
 - Step():
 - Execute action
 - Obtain reward
 - Calculate next state
 - Reset(): Go back to initial state after finishing episode
 - Action_space: All possible actions that the agent can execute - Discrete or continuous (Box)
 - Observation_space: Space where all states are valid

DQN & DDPG

- DQN and DDPG training iterations
- Action Space:
 - DQN1: 15 possible steering angles between -0.4 and 0.4
 - DQN2: 5 possible steering angles between -0.4 and 0.4
 - Common: batch_size = 32
 - DDPG: Continuous between -0.6 and 0.6
 - Common params: net_arch = [10,10], batch_size = 32

Num	Training Image	Params	Prediction Image	Reward
DQN 1		Number of episodes = 20000, learning_rate = 0.0001, gamma = 0.99, exploration_fraction = 0.1, exploration_finalEps = 0.05, exploration_initialEps = 1, train_freq = (4, "steps"), learning_starts = 10000		$R_t = \text{Car_Track_idx}(\text{mod } 3466) - \text{Prev_Reward}$
DQN 2		Number of episodes = 20000, learning_rate=0.0001, gamma = 1, target_update_interval = 100, exploration_fraction = 0.6, exploration_finalEps = 0.1, exploration_initialEps = 1, train_freq = (1, "episode"), learning_starts = 5000		$R_t = -e_y^2 - e_{psi}^2$ $P = -\text{NoStepsRemaining}\left(\frac{\text{lanewidth}^2}{2} + \frac{\pi^2}{2}\right)$
DDPG 1		timesteps = 10000, gamma=0.99, learning_rate=0.05, train_freq=(1,"episode"), learning_starts=1000,		$R_t = \text{Car_Track_idx}(\text{mod } 3466) - \text{Prev_Reward}$
DDPG 2		timesteps = 20000, gamma=0.99, learning_rate = 0.01, train_freq = (1, "episode"), learning_starts = 5000		$R_t = -e_y^2 - e_{psi}^2$ $P = -\text{NoStepsRemaining}\left(\frac{\text{lanewidth}^2}{2} + \frac{\pi^2}{2}\right)$
DDPG 3		timesteps = 10000, gamma = 1, learning_rate = 0.0005, train_freq = (10, "step"), learning_starts = 1000		$R_t = -e_y^2 - e_{psi}^2$ $P = -\text{NoStepsRemaining}\left(\frac{\text{lanewidth}^2}{2} + \frac{\pi^2}{2}\right)$

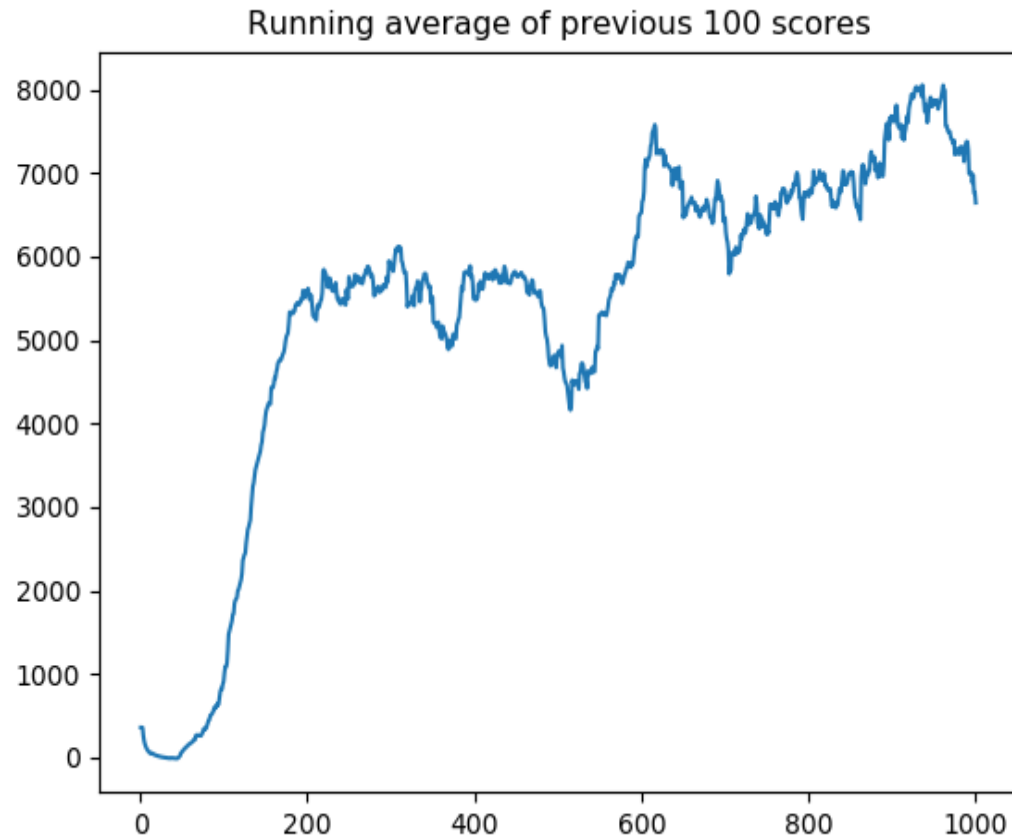
Reward Gates



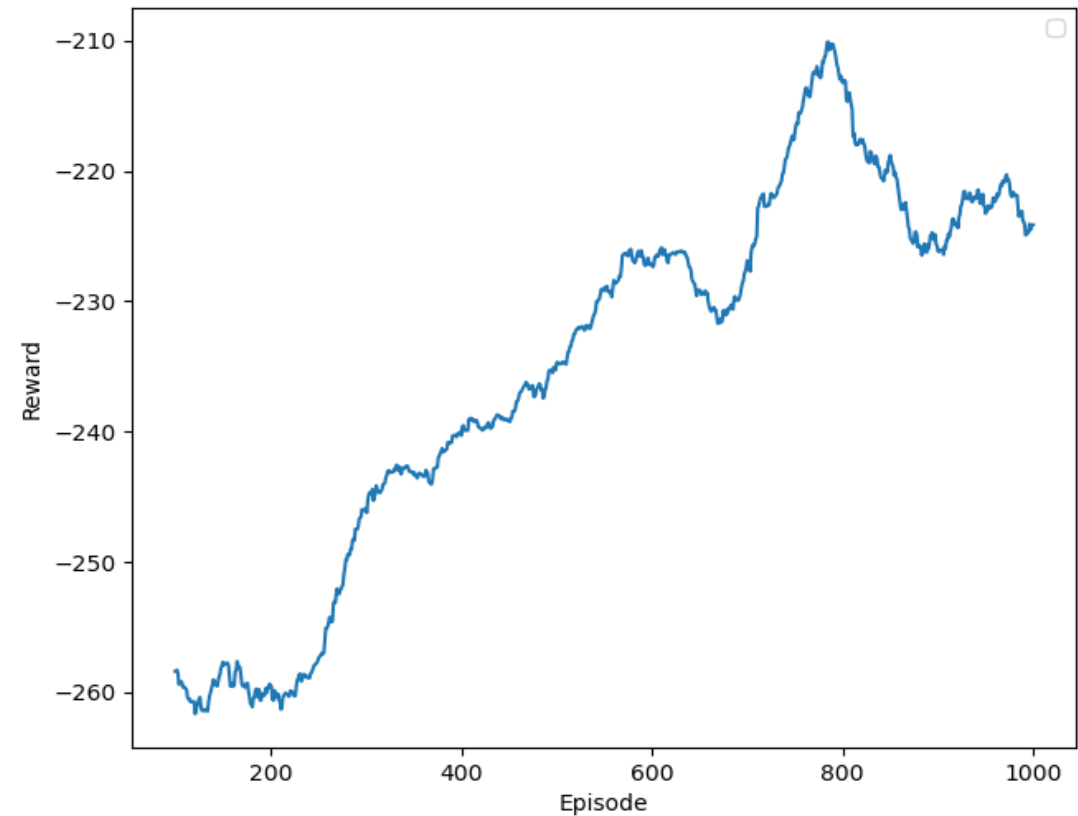
- There are 20 reward gates along the total length of the track.
- The car obtains a reward of 200 as it passes through the reward gate.
- The training convergence occurs at 150 episodes
- This can be stated, as on completion of the track along with collision gives us a collected reward of 3950

Better convergence with reward gates

With reward gates



Without reward gates



On Hardware – using Reward Gates Method

Torque = 0.1 (43s)



Torque = 0.2 (17s)



Outline

Introduction & Background

Trajectory Tracking

Reinforcement Learning

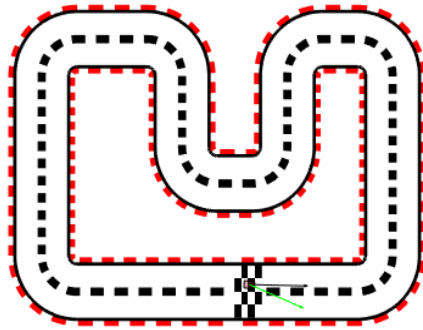
RL in Matlab - Python

Analysis

Conclusion

Analysis

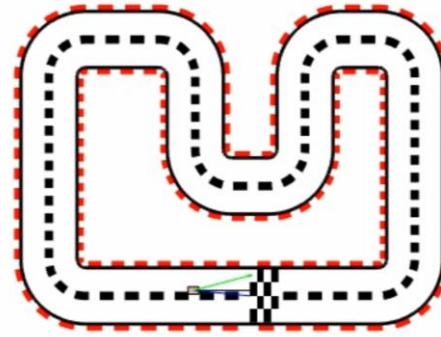
DDPG (Matlab)



→ Yaw
→ Desired Steering Angle

Torque: 0.13
Lap Time: 35 [sec]

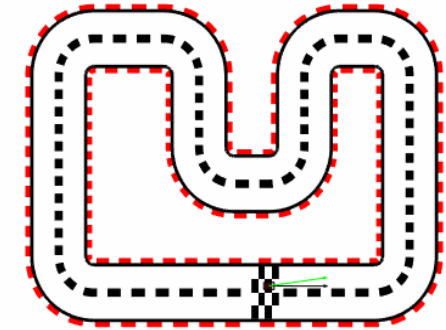
DDPG (Python)



→ Yaw
→ Desired Steering Angle

Torque: 0.1
Lap Time: 68.4 [sec]

PPO (Matlab)



→ Yaw
→ Desired Steering Angle

Torque: 0.12
Lap Time: 46 [sec]

Outline

Introduction & Background

Trajectory Tracking

Reinforcement Learning

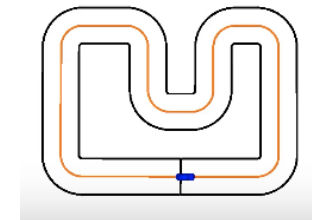
RL in Matlab - Python

Analysis

Conclusion

Insights Gained

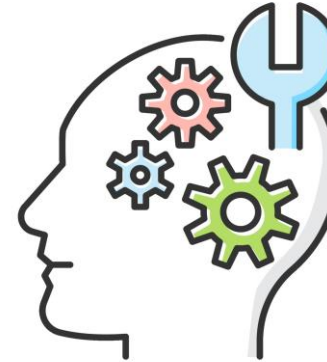
- Practical Implementation of theoretical knowledge (RL)
- Deeper understanding about different parameters and hyperparameters of RL
- Learning fundamental skills like Git, Docker, ROS
- Implementation of different Algorithms and techniques according to the needs





Feedback

- New concepts learned
- Group work
- Idea sharing
- Well-Structured way of working
- Simulation-Hardware Fusion

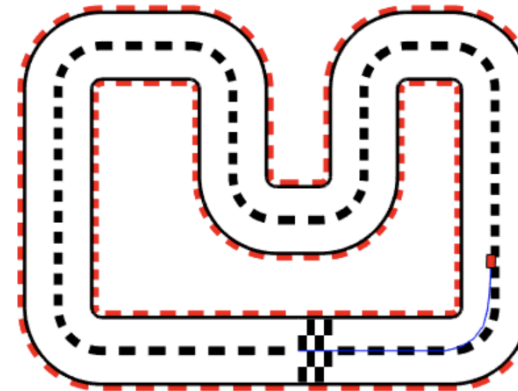


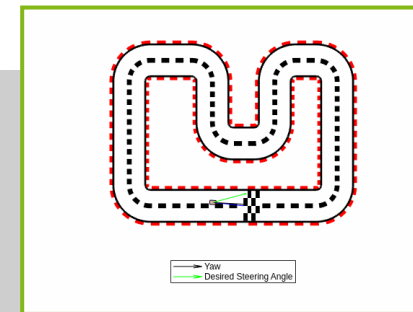
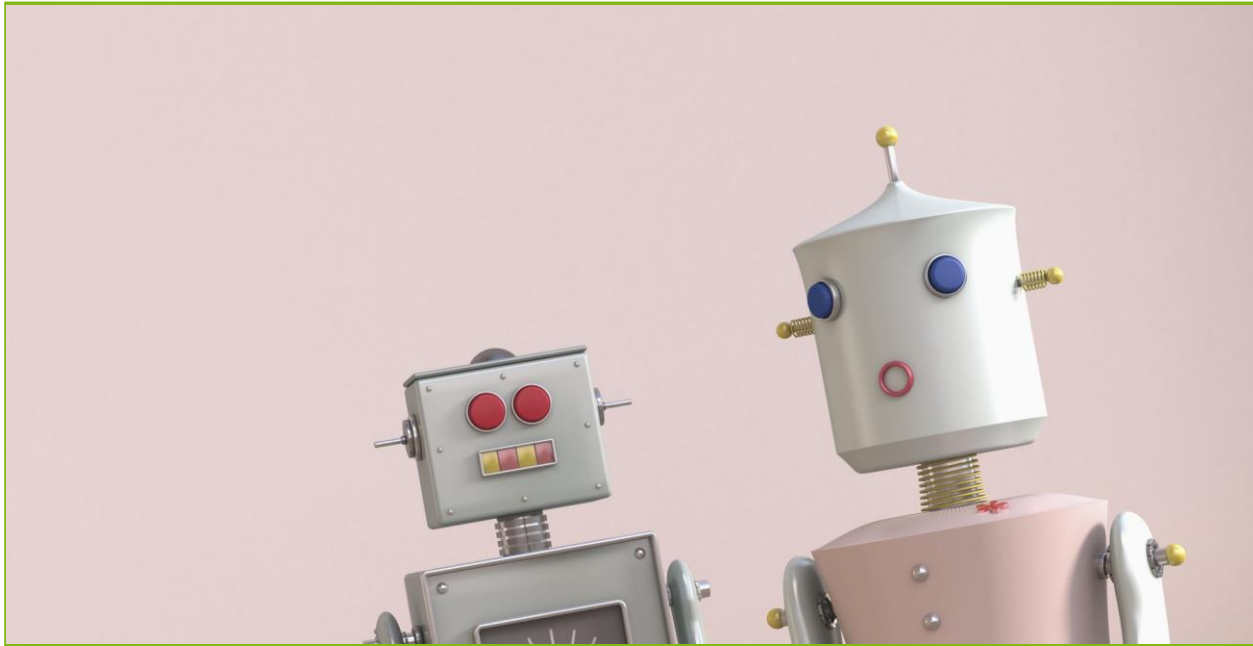
Can be improved

- Battery management

Future work

- Tests with different tracks
- RL to also learn best torques
- Consider more dynamic characteristics of car for learning





Thank you for your kind attention!