Autonomous Racing: Enhancing Performance through Reinforcement Learning and PPO

Manuel Andruccioli, Tommaso Patriti, Giacomo Totaro,
University of Bologna (Italy)
e-mail: {manuel.andruccioli, tommaso.patriti, giacomo.totato2}@studio.unibo.it

Abstract—Autonomous driving represents a vital area of research in the advancement of automotive technology with applications that range from city roads to extreme motor sport environments. In the context of racing cars, there is a unique challenge of demand for excellent performance and timely decisions that prompts the adoption of innovative approaches. In this work, we focus on the application of Reinforcement Learning in developing an adaptive and high-performance autonomous driving system for racing cars with specific emphasis on using Proximal Policy Optimization (PPO) algorithm known for its stability and ability to handle continuous action spaces. This approach seeks to improve the vehicle's ability to follow optimal paths while considering unique features of circuits used in car racing competitions. By analyzing and optimizing waypoint-based trajectories, our goal is to show how our autonomous system overfit different tracks and achieve good score in lap time.

Index Terms—Reinforcement Learning, Deep Learning, Autonomous Racing, ROS

I. Introduction

Autonomous driving represents a vital area of research in the advancement of automotive technology with applications that range from city roads to extreme motor sport environments. In the context of racing cars, there is a unique challenge of demand for excellent performance and timely decisions that prompts the adoption of innovative approaches. In this work, we focus on the application of Reinforcement Learning, which is a machine learning paradigm, in developing an adaptive and high-performance autonomous driving system for racing cars with specific emphasis on using Proximal Policy Optimization (PPO) algorithm.

Autonomous driving in motor sports such as Formula 1 requires a synergy between vehicle control precision and adaptability to changing track conditions. The use of Reinforcement Learning algorithms offers a promising approach because it allows the vehicle to learn optimal strategies through interaction with its surrounding environment based on rewards and penalties. In our study, we aim at enhancing the performance of race cars by using the PPO algorithm known for its stability and ability to handle continuous action spaces [1].

The novelty of this research lies in the model training approach that incorporates specific waypoints of circuits into the training maps. This approach seeks to improve the vehicle's ability to follow optimal paths while considering unique features of circuits used in car racing competitions. By analyzing and optimizing waypoint-based trajectories, our goal is to show how our autonomous driving system can dynamically adjust its driving path to fit lane changes with better lap timing and to deal with adverse conditions.

After training the model using PPO in a simulated environment, it will subsequently be used to predict the trajectory and speed of a vehicle inside a ROS-enabled simulator.

In summary, our work involves training a model in OpenAI's simulator that then can be used in ROS simulator.

II. State of the art

The advent of driverless car research has made great strides with applications ranging from road cars to race cars. In motor racing, the incorporation of autonomous driving systems has become a significant challenge necessitating sophisticated solutions to tackle the peculiarities of the competitive environment. Different approaches and relevant study findings after literature review provide a full picture of the current landscape and the main techniques of autonomous driving approaches.

A. PID

One of the most significant approaches of autonomous driving is the PID control algorithm. PID stands for a proportional, integral, and derivative controller used in automated control systems.

- Proportional (P): The proportional component responds proportionally to the current error, determining the response speed of the system.
- Integrated (I): The integrated component takes into account past errors and operates to eliminate any cumulative discrepancies, guaranteeing that the system reaches and maintains the set point in the long run.
- Derivative (D): The derivative component predicts the future behavior of the system thereby helping to prevent undue oscillations and enhance stability.

A significant variant is the Adaptive-PID [2]. It introduces adaptability into traditional PID, enabling the controller to automatically adjust its proportional, integral, and derivative parameters in response to changes in system dynamics. This adaptation is important when the system is subjected to changes in operational conditions, such as variations in speeds, vehicle masses, or road surface conditions. The key stages of Adaptive-PID are:

1) System Identification: An important aspect of Adaptive-PID is the ability to dynamically identify system parameters in real time. This could be done through parameter identification techniques such as linear regression or adaptive estimation algorithms.

2) Parameter Adaptation: Based on the identified information, the controller dynamically adapts PID parameters for optimal performance. For instance, if a vehicle experiences a change in mass due to variation in the load, the Adaptive-PID can automatically adapt parameters to ensure stable and responsive control response.

3) Tolerance to Changes: The adaptive approach ensures that control remains robust and effective even when significant changes are made to operating conditions, thus improving dynamic handling capabilities.

B. MPC

Many studies have focused on some traditional control techniques such as model predictive control (MPC) by Schwenzer et al. [3].

MPC is an advanced control technique that relies on iterative prediction of the evolution of the system over time, enabling the generation of optimal control commands. In more detail, Model Predictive Control can be divided into several key phases:

1) Dynamic model of the system: MPC demands an accurate and dynamic model of the system to be controlled. In the context of self-driving, this model includes parameters like vehicle dynamics, road geometry and other factors influencing its dynamics.

2) Future prediction: By using the dynamic model and prediction horizon, MPC iteratively predicts the future behavior of the system. This means that in each step the system foresees how it will evolve, and hence different control input possibilities are taken into account.

3) Control optimization: A cost function is defined to measure the quality of possible trajectories. MPC solves an optimization problem in order to identify a sequence of control commands that minimize this cost function while taking into consideration binding dynamics and kinematics of the system.

4) Control implementation: The implementation of the identified optimal control law for the system is carried out. The prediction and optimization process is then repeated cyclically, adapting to the system conditions in real time.

What distinguishes the MPC algorithm is its ability to handle complex constraints and non-linear dynamics of the system. Thus, it provides an adaptive, optimal control solution. However, implementing it may involve significant computational effort and forecast accuracy highly depends on the precision of a dynamic model.

The MPC algorithm uses a complete dynamic model of the system unlike the Adaptive-PID which is often less efficient from a computational point of view. However, it may pose challenges in dealing with more complicated dynamics or in scenarios where variations are extreme and not readily modeled by a standard PID approach.

On the downside, these approaches are often restricted in how well they handle dynamic complexities of race circuits, and machine learning overfitting could be a resource in this particular use case.

One of the most important milestones is the increasing adoption of machine learning algorithms focusing on reinforcement learning to achieve a driving style as similar as possible as a human driver, but free of distractions and emotions that can have a negative impact on performance [4]. The use of reward and penalty based techniques along with dynamic interaction between agent and environment have been shown to be effective in enhancing performance in autonomous driving. Researches such as Silver et al. (2016) [5] have made notable successes in training deep neural networks through Reinforcement Learning for human game contexts.

In the specific framework of car races, the optimal handling of vehicles calls for a combination of accuracy, speed and adaptability to the change of the track.

Proximal Policy Optimization (PPO) is one of the algorithms that has become popular for Reinforcement Learning algorithm due to its ability to handle continuous action spaces and stability during training (Schulman et al., 2017) [1]. This makes PPO particularly useful in applications where precision and dynamic management of the car is important such as automobile racing.

Our approach is different from the existing literature in introducing a specific use of race track waypoints in training maps. This decision aims to improve the model's ability to follow optimal trajectories on particular circuits, taking into account the unique characteristics of each track. After the training step, the model will be tested in the a another kind of environment, supported by ROS, in order to achieve a bit more realistic use case.

In summary, our work lies at the intersection between Reinforcement Learning research for autonomous driving and specific needs of auto racing by training a PPO model using waypoints on tracks given the fact that circuits will not change over the time and could be optimized. Next section provides detailed methodology, illustrating how we implemented and trained our model to achieve best results on selected circuits.

III. The proposed system

The project aims to provide a two-part integrated architecture. The first part employs use of the Simulator Gym (F1tenthGym) [6], based on OpenAI Gym [7], is a toolkit for reinforcement learning. Then, the model is based on PPO [1], a policy gradient methods for reinforcement learning, which alternate between sampling data through interaction with the environment, and optimizing a "surrogate" objective function using stochastic gradient ascent. Afterwards, the model is trained using a waypoint-follow approach, in order to complete the circuits.

The second part uses the previously trained model to predict actions that need to be taken by a car inside the ros-based simulator employing sensor feedback.

Through a containerized environment, we aim to give you insight into our approach to Reinforcement Learning-based Autonomous Driving, especially when using the PPO algorithm.

A. Model Training

1) Reward: The provided code implements a reward assignment system for an autonomous driving agent in a simulation environment. The main components of the system are as follows.

- Acceleration Reward: The reward depends on the acceleration action of the agent. If the acceleration exceeds 2, the reward increases accordingly; otherwise, a fixed reward of 0.02 is added.
- Reward for Proximity to the Race Line Point: The reward depends on the distance between the agent's position and the next point on the race line. A small reward is added if the distance is less than 2. If the distance is less than 2.5, a growing reward is added within the first 100 steps, followed by a decreasing penalty.
- Penalty for Deviation from the Race Line Point: A penalty is subtracted if the distance from the next point on the race line exceeds 3.
- Reward for Lap Completion: A reward proportional to the deviation from the goal is added after completing each lap.
- Collision and Time Limit Handling: A significant penalty is imposed in case of collision. If the number of steps exceeds a limit, an end-of-episode event is handled with a penalty.

The acceleration reward and lap completion reward are assigned only when the model has matured enough to complete multiple laps without collisions; otherwise, it tends to converge to a solution where the speed increases constantly, without finish any lap.

2) General model: The model was trained on 19 different tracks, which were randomly selected with a randomized spawn of the car on the track. As a result, it completes eight tracks with a success rate of 100%, never completes six tracks, and occasionally completes the remaining five tracks, with variations in success and failure.

3) General Model + YasMarina Model: At this point, a track was selected that the model consistently failed to complete, specifically YasMarina. A training cycle was then conducted on this track with the speed optimization active. The resulting model not only mastered YasMarina perfectly, achieving a significant improvement in lap time, but also exhibited positive transfer effects on other tracks. This led to an overall enhancement in the model's performance, indicating that it did not overfit the YasMarina track but instead generalized the knowledge gained from YasMarina to improve its performance on other tracks.

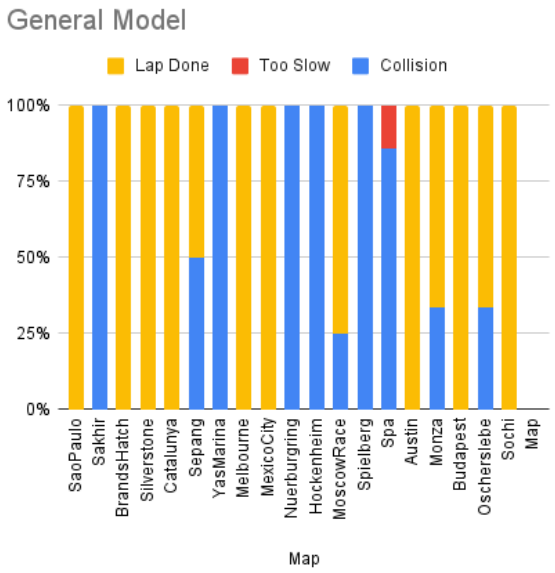


Fig. 1. General model, trained using all the map at the same time

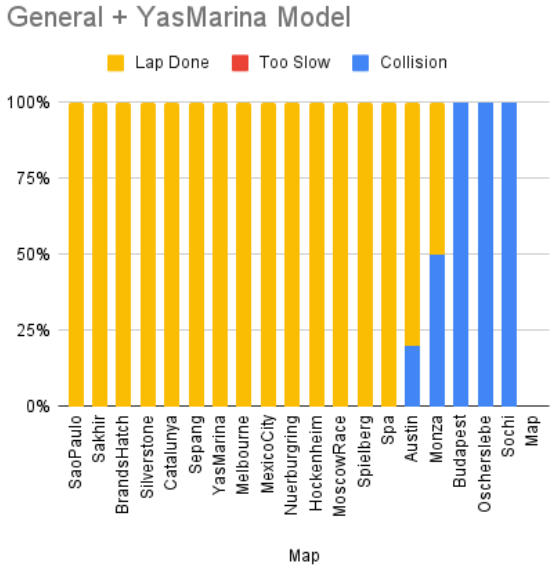


Fig. 2. General model + YasMarina trained model

B. Model Usage

During this phase, the reinforcement learning model trained in the first part of the project becomes the operational core of the car, taking decision on what to do. The application environment is extent to a more complex scenario, based on ROS.

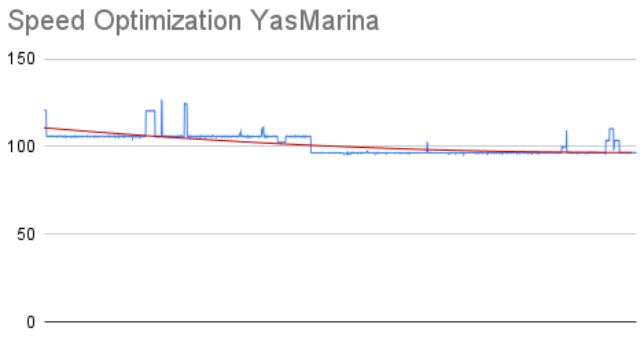


Fig. 3. YasMarina Speed Optimization

The project is based on F1tenth Gym Ros project [1], a containerized ROS communication bridge for the environment that allow the use of ROS2 API [8].

The starting point of building a ros node is defining a class that will declare the communication needs, both sending and receiving messages.

```
class PPOModelEvaluator(Node):
  def __init__(self):
    super().__init__('car_node')

    #Topics for Pubs & Subs
    lidarscan_topic = '/scan'
    drive_topic = '/drive'

    # Model loading
    self.model = PPO.load(path)

    # ROS
    self.lidar_sub =
      self.create_sub(callback)
    self.drive_pub =
      self.create_pub(...)
```

After that we can let the control flow spin to handle the specified callback. This node is designed to continuously receive data from the lidar scanner on the vehicle and use the model to predict the optimal actions to be taken based on these inputs. The input data of the model is a vector of 1×1080 linear distances between the actual position of the car and the first obstacle the light will find over the ray path. It spans an angle of 360 degrees around the car. When new data are ready to be processed, the model starts the prediction and the output is the action that the car will take. In this case we can control the steering angle and the speed of the car.

[1]https://github.com/f1tenth/f1tenth_gym_ros

$$\begin{bmatrix} d_1 \\ d_2 \\ \dots \\ d_{1080} \end{bmatrix} \xrightarrow[\text{Prediction}]{\text{Model}} \begin{pmatrix} \text{angle} \\ \text{velocity} \end{pmatrix}$$

```
def callback(self, data):
  distances = normalize(data)

  action, _ = self.model.predict(d)
  action = denormalize(action)

  msg = ...
  msg.angle = action[0]
  msg.speed = action[1]
  self.drive_pub.publish(msg)
```

Docker & multiplatform: Docker [9] is a common and efficient practice, particularly for achieving multiplatform compatibility and ensuring consistency across different environments.

The framework it's been use in order to achieve a multiplatform run on ROS2 environment side. This it's been possible thank to a build that target a multi platform architecture (arm64, amd64). The result is published at https://hub.docker.com/r/manuandru/f1tenth-gym-ros-model-env

IV. Usage

A. Installation

You can find everything about the project at the following link: https://github.com/zucchero-sintattico/svs-f1tenth_gym.

1) Requirements: Python 3.8, Docker.
2) (Optional) | Create a python environment.
3) Install the dependencies:

   $ (svs-f1tenth_gym/) pip install -e .

B. Model training API

The model training API that uses F1tenth Gym environment has a CLI in order to:

- Train the model

  $ python src/main.py train
- Run the trained model

  $ python src/main.py run
- Evaluate the model

  $ python src/main.py evaluate

For specific parameters for each program, you can use the CLI documentation:

$ python src/main.py --help

C. Model ROS usage API

1) (Optional) build the image from Dockerfile (modify the docker-compose file accordingly.):
   $ docker build -f Dockerfile-gym-ros .
2) Run the docker compose file:
   $ docker compose \
   -f docker-compose-ros-env.yml up
3) Attach to the ROS2 running container:
   $ docker exec \
   -it svs-f1tenth_gym-sim-1 /bin/bash
4) Run the simulator:
   $ ros2 launch f1tenth_gym_ros \
   gym_bridge_launch.py
5) Open http://localhost:8080/vnc.html.
6) Run the node for autonomous driving the car:
   $ ros2 run car_node car

V. Conclusions

Our study's outcomes reflect the successful training of the Proximal Policy Optimization (PPO) model on a diverse set of 19 car racing tracks. Randomly choosing and spawning the car randomly in each track helped create a heterogeneous and realistic training environment, challenging the model with a wide variety of driving scenarios. The 100% success rate on eight tracks demonstrates that this is a robust model which can adapt to different situations of driving. On the other hand, Six tracks were not fully completed, while intermittent success was achieved in five remaining ones indicate inherent challenges in autonomous driving where vehicle dynamics and environmental complexities may result into varied results. Additionally, regarding the next step it involved choosing one specific track called YasMarina that proved consistently to be problematic for our model. The introduction of an active speed optimization cycle during training at YasMarina has yielded surprising results. Not only did the model achieve perfect mastering of this track with significant improvement in lap time but also showed positive transfer effects to other tracks. Given that the model has generalized the knowledge acquired from YasMarina, thereby improving its performance on other tracks, implies a robust learning ability and capacity to extract driving principles that can be applied in different contexts. This occurrence indicates that rather than adapting itself to just one track, the model grasped a wider understanding of driving dynamics. The final results show remarkable overall progress of the model. From an inability to complete YasMarina, the model is now able to 100% complete as many as 14 courses without failing any three tracks and occasionally completing any two remaining courses. This enhancement highlights the power of targeted training strategies and how well it enables models to transfer learned skills from specific contexts into a wider range of driving scenarios. Despite these achievements, our study has some limitations. The representativeness of the selected tracks may not fully reflect the diversity of challenges that an autonomous vehicle might face in reality. Furthermore, further research is required to fully understand the impact of active speed optimization and refine the training strategies more. In conclusion, PPO model training on diversified set of race tracks followed by a targeted phase in YasMarina has demonstrated model's robustness, adaptability and transferability. This work contributes to the growing understanding of the challenges and opportunities in autonomous vehicle training, paving way for future practical implementations and advanced research on autonomous driving.

References

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," Arxiv, 2017.

[2] P. Zhao, J. Chen, Y. Song, X. Tao, T. Xu, and T. Mei, "Design of a control system for an autonomous vehicle based on adaptive-pid," International Journal of Advanced Robotic Systems, vol. 9, no. 2, p. 44, 2012. [Online]. Available: https://doi.org/10.5772/51314

[3] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, "Review on model predictive control: an engineering perspective," The International Journal of Advanced Manufacturing Technology, vol. 117, pp. 1–23, 11 2021.

[4] M. Andruccioli, M. Mengozzi, R. Presta, S. Mirri, and R. Girau, "Arousal effects on fitness-to-drive assessment: algorithms and experiments," in 2023 IEEE 20th Consumer Communications & Networking Conference (CCNC), 2023, pp. 366–371.

[5] D. Silver, A. Huang, and C. J. Maddison, "Mastering the game of go with deep neural networks and tree search," Nature, vol. 529, pp. 484–489, 01 2016.

[6] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam, "F1tenth: An open-source evaluation environment for continuous control and reinforcement learning," in NeurIPS 2019 Competition and Demonstration Track. PMLR, 2020, pp. 77–89.

[7] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[8] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," Science Robotics, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074

[9] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," Linux journal, vol. 2014, no. 239, p. 2, 2014.