Perception Stack for Indy Autonomous Challenge and Reinforcement Learning in Simulation Autonomous Racing

Tianlun Zhang

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-187
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-187.html

May 22, 2023

Acknowledgement

I would like to take this opportunity to thank Professor Sastry and Dr. Allen Yang, who guided the projects and offered invaluable advice and assistance in addressing the challenges and difficulties I encountered along the
way.
I also want to thank every member in the ROAR group, especially to Chris Lai, who made significant contributions in the IAC perception project; and Franco Huang and Yunhao Cao, who made significant contributions in the RL project.
Working in such a collaborative and dedicated team has truly been an enriching experience, and I am sincerely grateful for the opportunity.

Perception Stack for Indy Autonomous Challenge and Reinforcement Learning in Simulation Autonomous Racing

Tianlun Zhang

Abstract

This report presents an exploration of advanced perception stack for Indy Autonomous Racing(IAC) [1] and reinforcement learning for autonomous racing in the simulation.

The first part of the report investigates an efficient perception system that uses the inputs from a variety of sensors, namely cameras, RADAR, LiDAR. The objective of this system is to robustly detect and track other race cars during the race. We have successfully implemented a pure RADAR detection pipeline for long-range detection and a LiDAR-Camera detection pipeline for short-range detection. Both pipelines work properly and efficiently to serve the requirements of perception for our race cars. Merging the detections from both and feeding it into the tracker gives us a stable and promising perception output for the race. We won second place in the Texas Motor Speedway Race and third place in the Las Vegas Motor Speedway Race.

The second part of the paper investigates the implementation and improvement of a Reinforcement Learning (RL) agent for autonomous racing in the simulation. Continuing from the successful RL agent we designed and implemented last year, we furthermore challenge the RL agent to achieve more stable and safer driving in the simulation environment. We redesigned the observation space and action space for the RL environment to reduce the model complexity and provide as much useful information to our agent as possible, hoping to help the agent understand the map better and make the model converge faster. Inspired by the breakthroughs in the Atari game by Agent57[1], we have adapted and enhanced our network structure and replaced the optimization policy to Soft Actor Critic(SAC)[5], hoping the more robust network can better deal with the extracted features and result in a safe RL driving solution. The training is still underway, but the current results demonstrate that the new RL agent has the ability to learn to speed up on the straight and slow down upon turning. Future results are needed to evaluate the performance of our new agent, and we will continue our efforts in further training and optimization of the RL agent in our pursuit of excellence in simulation autonomous racing.

[1] https://www.indyautonomouschallenge.com/

1 Introduction

The advent of autonomous vehicles has undeniably transformed the landscape of automotive technology and transportation systems. The application of this technology, however, extends beyond the realms of public and private transport, reaching into the competitive sphere of racing, specifically in the realm of Indy Autonomous Racing. Autonomous racing presents a unique set of challenges in terms of speed, precision, and environmental complexity, requiring advanced perception and decision-making systems. My study aims to investigate the design and implementation of an efficient perception stack and reinforcement learning for simulation in autonomous racing. The perception stack is a crucial component of an autonomous vehicle's system, playing a pivotal role in interpreting sensory data to understand the surrounding environment. It typically consists of various sensor modalities, including radar for far detection, and a combination of LiDARs and cameras sensors for close detection. The ultimate aim of the perception stack is to provide comprehensive, accurate, and real-time information about the vehicle's environment, enabling the vehicle to make safe and effective decisions. Berkeley Robot Open Autonomous Racing(ROAR) Team[2] participated in the Indy Autonomous Challenge as a part of AI Racing Tech(ART)[3], and got second place in the Texas Motor Speedway Race and third place in the Las Vegas Motor Speedway Race. In addition to the perception stack, my study also explores the application of Reinforcement Learning (RL) for simulation in autonomous racing. RL, a subset of machine learning, offers a promising avenue for developing autonomous racing strategies by allowing the vehicle to learn optimal actions through trial and error in a simulated environment. The simulation provides a safe and efficient platform for training and testing the vehicle's decision-making algorithms without the risks associated with real-world testing. We designed and implemented an AI agent to complete the race in the simulation.

2 Related Work

The concept of autonomous racing has gained significant traction in recent years due to the technological advancements in the field of autonomous vehicles. This research draws on a range of studies and efforts dedicated to the development of high-speed autonomous racing vehicles.

Perception Systems in Autonomous Racing Perception systems are pivotal in enabling autonomous vehicles to learn their own state and understand their surrounding environment. Recently, high-speed LiDAR and RADAR-based perception systems have been increasingly used in autonomous racing scenarios. During the race, the perception stack is required not only to be accurate and stable, but also fast and efficient, because the race car is driving at very high speed and limited computing resources are provided. Teams participating in the Indy Autonomous Challenge have reported advancements in the use of these sensor systems for high-speed object detection and tracking, which is critical for safe and efficient racing. Teams from Technical University of Munich(TUM) [4] and Korea Advanced Institute of Science and Technology(KAIST) [5] have published their study on different methods of perception stack over the recent years.

For localization, TUM has proposed to use Combined LiDARs and Cameras to perform simultaneous localization and mapping (SLAM) [12]. They use LiDARs to get the detection of the track boundary, and apply OpenVSLAM [14] on camera data for segmentation. The environment is then reconstructed based on these findings and a pre-established track layout. However, there are two main challenges for this method: its limited effectiveness in feature-poor environments and its expensive computational requirements. KAIST also reports that SLAM might not be a good choice for the race for the same reasons and they decided to stick with traditional GNSS-based localization.

Assuming that the only objects on the track are race cars, KAIST proposed the detection pipeline with pure LiDAR input [6]. They implemented a non-ground object removal algorithm. The algorithm first projects all the points clouds onto the XY-plane and represents them into grid cells, and then it removes the points clouds based on the number of points inside the grid and their height distribution. They further remove every point about 1m away from the walls to conclude their detection on the track. While this method detects everything on the track as race cars, TUM proposed a method

[2]https://roar.berkeley.edu/

[3]https://www.hawaiiavtech.com/

[4]https://www.indyautonomouschallenge.com/technical-university-of-munich

[5]https://www.indyautonomouschallenge.com/team-kaist

to LiDAR data in two ways[11] to differentiate race cars and other objects. First, they propose a deep learning-based algorithm, PointRCNN [8], directly on the points clouds. This algorithm was trained exclusively on race car data, resulting in an overfit model that exclusively detects race cars and ignores other objects. Second, they proposed a Geometry Clustering on the points clouds to detect arbitrary classes of objects. TUM also suggested processing camera data with YOLOv5 [16] to aid long-range detection and using RADAR data to measure the relative speed of detection, although these processes were intended to supplement the primary LiDAR pipelines.

Autoware[7] proposed a comprehensive autonomous driving pipeline suitable for real-world scenarios. Their perception pipeline primarily relies on point cloud data from LiDAR scanners. The point clouds pre-processed and segmented using the nearest neighbors Algorithm [4]. The point cloud can also be projected onto the image frame to provide depth information of detected bouncing boxes. Although the Autoware pipeline is designed for much more complicated environments and not ideally for racing, its data pre-processing algorithm and object detection concepts greatly inspired the design of our own stack.

Reinforcement Learning for Autonomous Racing One of the seminal works on deep end-to-end reinforcement learning for autonomous driving is the DAVE-2 system presented at NVIDIA[3]. This system pioneered the concept of using convolutional neural networks (CNNs) to map raw images collected by front-facing cameras directly to steering commands. This end-to-end approach bypasses manual feature extraction and simplifies the driving task to a function approximation problem. A similar direction was taken by the Wayve.ai[6] team, which further extended the end-to-end driving approach by integrating reinforcement learning. They utilized a relatively straightforward convolutional architecture and trained the network via model-free reinforcement learning. This allowed the system to learn a policy from high-dimensional sensory inputs to low-dimensional vehicle control commands, thereby enhancing its capability to handle complex traffic scenarios.

The inspiration of our RL design in the simulation autonomous driving came from the successful RL solution to playing Atari game [9]. The algorithm observes the game states via 4-frame, two-dimensional, grayscale images and accurately decides the direction for moving the paddle. Similar to our case, our intent is for the agent to understand a two-dimensional grayscale occupancy map so that it can drive safely. Consequently, adopting the network structures from the Atari solution appears to be a reasonable starting point.

The performance of Atari solution has seen consistent improvements over the years. Agent57[1], published in 2020, obtained a score that is above the human baseline on all 57 Atari 2600 games. The strategy incorporated a variant of Deep Q-Network (DQN)[10] and Actor-Critic methods, and introduced a Long Short-term Memeroy (LSTMs) block after CNN feature extraction. This innovative approach has inspired our use of LSTM and Soft Actor Critic(SAC)[5], potentially serving as the continuous-space version of DQN, in our approach and showed the potential of reinforcement learning in mastering simulation autonomous driving problem, given the similar observation format.

3 Perception Stack for Indy Autonomous Racing (IAC)

The Indy Autonomous Challenge is an international competition focused on advancing autonomous vehicle technology in the realm of high-speed racing. The goal of the challenge is to develop and race fully autonomous race cars capable of completing laps and overtaking other race cars safely without any human intervention. In order to achieve the goal, an efficient, confident and stable pipeline of detecting and tracking other race cars on the track is required.

3.1 Hardware Configurations

The race car model is Dallara AV-21 and the computer on the car is an ADLINK AVA-3501, which is responsible for spawning sensors and running all of the perception, planning and control stack. Therefore, the perception stack needs to be able to output its detected and tracked objects at least 20 frames per second with less than 20% of CPU usage in real time. Sensors involved in the perception stack are:

- 3 Aptiv ESR RADAR sensors

[6]https://wayve.ai/thinking/evaluating-driving-performance-in-diverse-simulated-worlds/

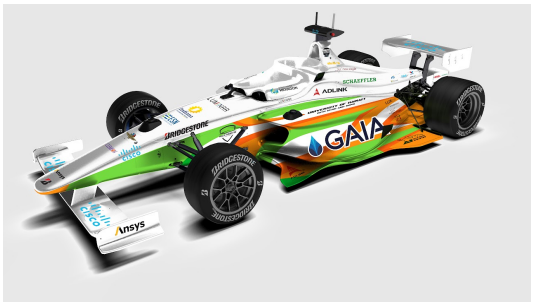


Figure 1: Indy Autonomous Challenge vehicle platform

- 3 Luminar H3 LiDAR sensors
- 6 Allied Vision Mako G319C camera sensors
- 2 Novatel GPS Receivers

RADARs are directly connected to the computer. All LiDARs, cameras and GPS sensors are connected to a Cisco IE5000 switch for Precision Time Protocol(PTP), and the switch is connected to the computer. Three RADAR sensors are placed at the front of the car, one directly facing forward and the other two are oriented towards the sides. They offer objects detection of a range of more than 200 meters, with the ability to measure their relative speed. Three LiDAR sensors are orientated in alignment to the vehicle heading and rotated around a vertical axis to ±120 degree such that LiDAR setup can cover in total 360-field-of-view. Two of the six cameras are also oriented in alignment with the vehicle heading as prime front cameras, while other four cameras are installed around the vehicle, which also provide 360-degree coverage.

3.2 Precision Time Protocol(PTP)

For high-confidence detection, leveraging all available LiDAR, camera, and GPS data necessitates precise sensor synchronization. These sensors are all interconnected via Ethernet and configured in Precision Time Protocol (PTP) mode to align with GPS time. This setup ensures superior time coordination with sub-microsecond precision, crucial for synchronizing the data flow from the multiple sensors accurately and efficiently.

3.3 Object Detection Algorithms

3.3.1 RADAR Detection – Long-Range Detection

Utilizing RADAR for object detection in autonomous racing is a cost-effective and straightforward solution due to its robustness in adverse weather conditions and long-range detection capabilities. The RADAR driver directly outputs a list of both moving and stationary objects, along with their relative speed, while consuming minimal computational and power resources - a significant advantage in a racing scenario. However, RADAR detection often includes noise and false detections. Therefore, the primary task of the detection algorithm is to eliminate all objects that aren't the race cars on the track.

There are four main sources of noise and false detections. Firstly, stationary or slow-moving objects may be misclassified as moving ones. Secondly, objects outside the track are also detected. Thirdly, the fast-moving and direction-changing nature of race cars can cause detection signals to bounce off the surrounding walls, creating 'ghost' cars in RADAR frame. Lastly, when other race cars are close, they might be detected as two separate objects.

False detections due to misclassification of stationary or slow-moving objects are filtered out based on vehicle speed. Any low-speed objects are usually false detections, while fast-moving objects are the race cars during the race. By reading the ego vehicle's speed from the GPS and the relative speed of detected objects from the RADAR, we can calculate their absolute speed. Setting speed thresholds for both the ego vehicle to activate the RADAR detection pipeline and the detected objects to filter out low-speed detections effectively removes this type of false detection.

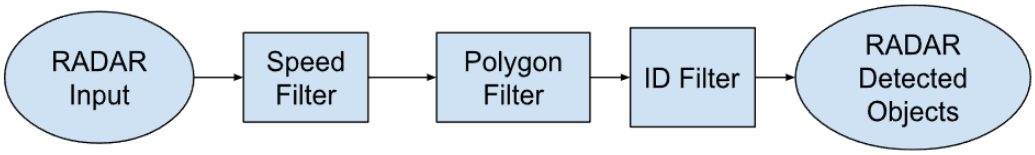


Figure 2: RADAR Detection Pipeline

Noise from detections outside the track and 'ghost' cars are filtered out based on location. We pre-record GPS locations of the race track's boundaries, and construct inner and outer loop polygons. During runtime, we establish the ego vehicle's GPS location as the origin of the baselink coordinate and transform the polygons into this coordinate. Any detection inside the inner polygon or outside the outer polygon is discarded. Running a point-in-polygon algorithm twice successfully eliminates these types of noise.

The method for determining whether a point lies inside a polygon is grounded in Cauchy's Integral Theorem from Complex Analysis. This theorem states that the integration of a function $f(x) = \frac{1}{x-p}$ over a closed, positively oriented curve is zero if and only if the point p is located outside the curve. By applying this theorem to our scenario, where p is the detected point and the polygon defines the curve $C = [c_1, c_2, \ldots, c_n]$, we can ascertain whether p is situated within the polygon through the calculation of the integral. Then the integration over curve is

$$\int_C f(x)\,dx = \sum_{i=1}^{n} \int_{c_i c_{i+1}} f(x)\,dx = \sum_{i=1}^{n} \log \frac{c_{i+1} - p}{c_i - p}$$

and it equals to 0 if and only if p is inside the curve C.

Algorithm 1 point-inside-polygon test

procedure
  $integration \leftarrow 0$
  $\epsilon \leftarrow small\_positive\_number$
  for $i = 1$ to n **do**
    $integration \leftarrow integration + \log \frac{c_{i+1}-p}{c_i-p}$
  end for
  return $integration > \epsilon$
end procedure

The fourth type of noise is typically filtered using the object ID supplied by the RADAR's detection output. Usually, a consistent object retains a fixed ID, thus maintaining a record of previously true detections' ID can assist in noise filtration.

However, there are two primary concerns regarding this approach. Firstly, even if the RADAR operates at a high frame rate, the true detection rate only comes in around 8 to 15 FPS, which is not sufficient for updating the locations of other race cars during overtaking. Secondly, as RADAR outputs detection as a single point, it fails to accurately reconstruct the physical shape of other race cars, which is important when the distance is close. Furthermore, the ID filtering is not always stable, with detection potentially fluctuating intermittently. Therefore, while RADAR provides long-range detections with low computational resources, it fails to provide reliable close-range detection. This necessitates the implementation of an additional LiDAR-Camera Detection system.

3.3.2 LiDAR-Camera Detection – Short-Range Detection

In order to accomplish high-accuracy, short-range detection, we designed and implemented a LiDAR-Camera detection algorithm. This approach utilizes inputs from six cameras and three LiDAR sensors, leveraging the You Only Look Once Version 7 (YOLOv7) model to obtain precise bounding boxes from the camera inputs. We then project the corresponding LiDAR point clouds to reconstruct 3D vehicle models for short-distance detection.

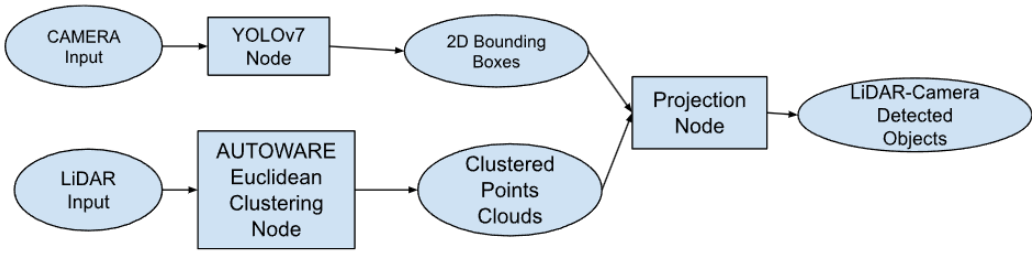


Figure 3: LiDAR-Cemera Detection Pipeline

YOLOv7 Node on Cameras YOLOv7 offers real-time object detection and localization in images and videos, improving upon previous YOLO versions for enhanced accuracy and efficiency. We gathered data from all six cameras during several multi-car runs, extracted frames with other race cars, and manually labeled bounding boxes using an online tool. Using these datasets, we applied transfer learning to a pre-trained YOLOv7 model, which confidently predicted bounding boxes with over 90% accuracy. To facilitate parallel computing, we merged synchronized images from all six cameras and converted the YOLOv7 Python prediction code into TensorRT, which significantly accelerated computation.

LiDAR Euclidean Clustering Our perception stack also leverages Euclidean Clustering from Autoware [7] for points cloud data segregation based on spatial proximity. This technique is a crucial element in object detection and tracking systems for autonomous vehicles. The initial step involves converting raw points clouds data from the ROS PointCloud2 message into a format compatible with the Point Cloud Library (PCL). The algorithm then groups proximate points in the 3D point cloud into clusters based on a specified distance threshold. This process also allows for configuration of parameters such as $min\_cluster\_size$, $max\_cluster\_size$, and use_height. Points clouds from all three LiDAR sensors are transformed into the baselink frame and concatenated for parallel computing.

Lidar-Camera Projection The bounding boxes derived from the YOLO nodes define the pyramid regions that may contain point clouds corresponding to race cars. We chose six projection planes, each positioned one meter away from its corresponding camera and parallel to the camera's image plane. Bounding boxes from the corresponding camera and all point clouds are projected onto these planes. Point clouds that fall within the projected boxes contain the detected race cars. To further eliminate noise, we exclude points that exceed the average distance, as these often represent background walls and the ground, which are typically much farther away. The remaining point clouds are assembled into 3D bounding boxes and broadcast as $autoware\_detected\_objects$ to the tracker.

3.4 Tracker

Our perception stack employs Autoware's multi-object tracker [8], which blends the detections from RADAR and LiDAR-camera systems. Confidence in detections varies with the detected object's distance; RADAR detections gain confidence with long distance, while LiDAR-camera detections are more reliable at closer range. The Interacting Multiple Model Unscented Kalman Filter (IMM-UKF) algorithm is employed for tracking, capable of handling the real-world dynamic driving environment. When a new object is detected, the system triggers a set of Kalman filters, each representing a potential motion model. Each filter predicts the object's next state, influenced by its motion model and the previous state estimate. The IMM component merges these predictions into a single 'mixed' prediction, considering the object's observed behavior. Upon receipt of new sensor data, each filter updates its state estimate based on the new data and its prediction. The Unscented Transformation is used to approximate the non-linear motion and measurement model application results. The likelihood of each model is updated based on the prediction's alignment with the observed data. Finally, the state estimates from all filters are merged into a single state estimate, considering each motion model's

[7] https://autowarefoundation.gitlab.io/autoware.auto/AutowareAuto/euclidean-cluster-design.html
[8] https://autowarefoundation.github.io/autoware.universe/main/perception/multi_object_tracker/

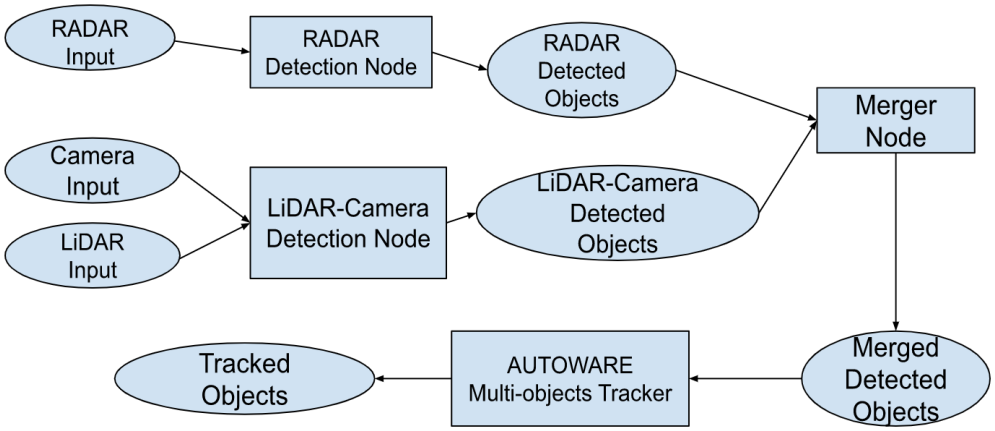
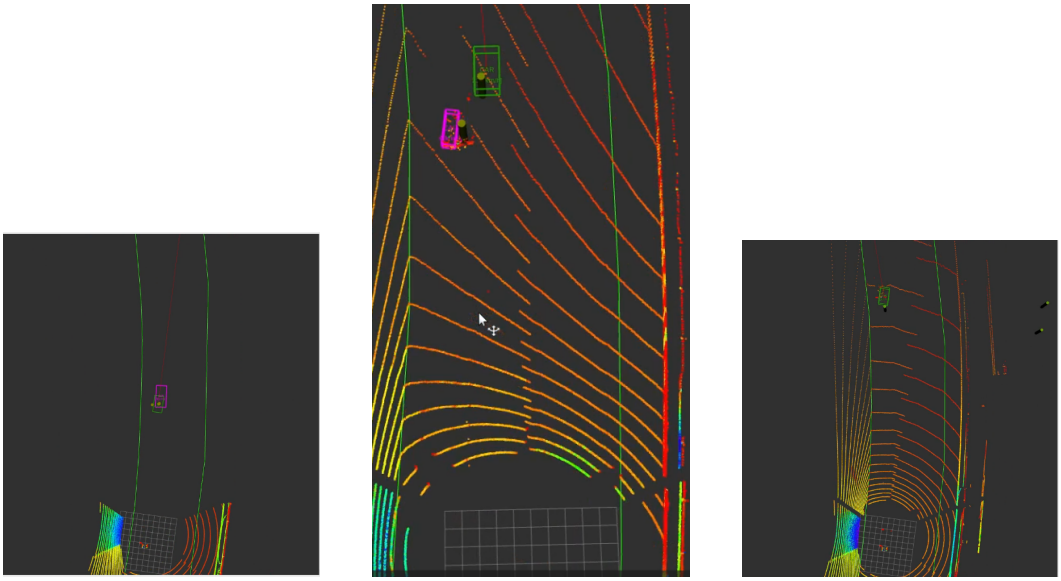


Figure 4: Complete Perception Pipeline



(a) Both pipelines detected the objects, and the detection matches.

(b) RADAR fails at short range. The green box is a false detection.

(c) LiDAR fails to detect objects at long range.

Figure 5: The detection results of RADAR and LiDAR-camera pipelines. The green boxes represent the detection from RADAR and the purple boxes represent the detection from LiDAR-camera.

updated likelihood. This iterative process lends robustness to the IMM-UKF algorithm when dealing with changes in an object's motion.

3.5 Conclusion

The complete perception pipeline is shown in the figure4 below for a visual representation of the complete perception pipeline. Capable of detecting race cars up to approximately 150 meters away and updating detections at around 20 FPS, this pipeline ensures accurate and consistent perception for our autonomous race car.

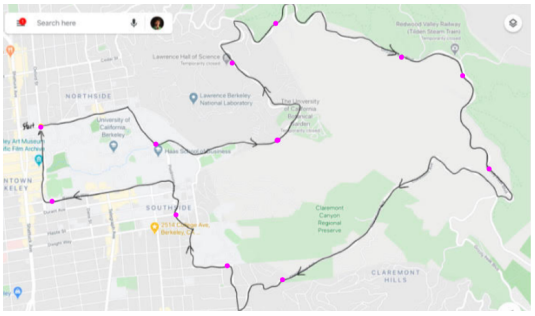


Figure 6: Berkeley Major Map Loop

4 Reinforcement Learning for Simulation Autonomous Racing

Last year, we devised and implemented a Proximal Policy Optimization (PPO) Reinforcement Learning (RL) solution [9] for the ROAR Simulation Autonomous Race. This enabled us to complete a single lap on the Berkeley Major map[10] in a simulation timeframe of 997.5 seconds. However, this version of the agent had notable shortcomings, particularly its inability to prevent driving onto pavement. In fact, it was considered crashed and would be reset while it left the road during trianing, so the training process never completed.

To address this, modifications were made to the racing environment in the subsequent year. We elevated the pavement level, which successfully stopped the vehicle from driving onto the pavement and grass areas, thus maintaining track adherence throughout the race. Nevertheless, these alterations heightened the challenge for the RL solution, necessitating significant revisions and enhancements to effectively manage this more complex scenario.

4.1 Improvement on Action Space

The vehicle control parameters - steering, throttle, and brake, initially inspired a three-output channel design. However, this configuration faced convergence difficulties, primarily due to the interplay between throttle and brake. In real-world scenarios, drivers seldom press the throttle and brake simultaneously, which was a distinct possibility in our original design, complicating the simulation results.

To overcome this, we redesigned the action space, reducing it to two channels: the first channel, ranging from 0 to 1, controls steering; the second channel combines throttle and brake, with a range of -1 to 1. Positive values on the second channel apply throttle, while negative values apply brake. This strategic modification encourages the agent to speed up during straight paths and decelerate when encountering turns, thereby enabling longer driving distances and faster convergence.

4.2 Improvement on Observation Space

Our original observation space design drew inspiration from a successful RL solution, "A Graphic Guide to Implementing PPO for Atari Games" [11]. It comprised three components: the Occupancy Grid Map (OGM), reward lines, and vehicle state. The OGM delineated the road boundaries, while the reward lines fragmented the entire loop into hundreds of segments, providing incremental rewards for crossing each line. The vehicle state encapsulated the car's location and rotation. We stacked the last four observations to encode velocity information, resulting in twelve 84x84 grayscale images serving as the model's input.

However, we discovered the initial design insufficient for the agent to fully understand the map. The binary nature of the OGM failed to convey any height information, impeding the agent's ability to adjust its speed accordingly during uphill or downhill sections, leading to frequent crashes. Moreover, the uniform representation of the areas beyond the road boundaries did not help distinguish drivable

[9] https://roar.berkeley.edu/roar-end-to-end-reinforcement-learning/

[10] https://roar.berkeley.edu/berkeley-major-map/

[11] https://towardsdatascience.com/a-graphic-guide-to-implementing-ppo-for-atari-games-5740ccbe3fbc

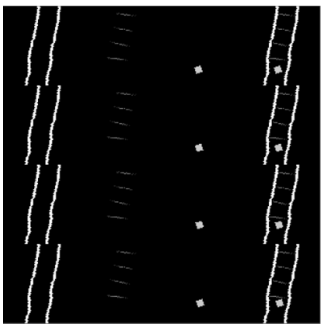


Figure 7: Previous Observation Input. The first three columns are the input. The last column are just for visualization

Figure 8: New Observation Input. The last two images are the input. The first two images are just for visualization.

areas from the out-of-bound regions. To rectify these issues, we revamped the input map to include height information. We assigned height values to each map position rather than using binary values, and allocated a zero value to all areas beyond the track boundaries, restricting the car to positive value areas.

The previous input was also overly complex. The OGM and reward lines across four frames appeared almost identical, making the extra frames redundant.They only encodes some kind of velocity and acceleration information together with 4 frames of vehicle state input, and their large-size but low-dimenstionality also made feature extraction by convolutional network challenging. To address this, instead of appending ten additional 84x84 images to encode velocity data, we read those information directly from the vehicle and IMU sensors at each frame as six-element vectors. These were then merged with the feature vectors output from the convolutional neural network, considerably reducing the input dimension.

Consequently, the revised observation space comprised two 84x84 images (as shown in figure 8)and a six-element vector. This not only provided a more comprehensive understanding of the map for the agents but also significantly reduced the complexity, thereby facilitating model convergence.

4.3 Improvement on Rewards and Termination conditions

We have tried different combinations of reward functions over the last year, and realized minimizing the incorporation of human bias in the reward might be effective for training. Previously, we had tried to encourage the car to maintain high speed by assigning rewards for high throttle and speed, and to discourage wobbling by penalizing sudden changes in steering. However, these strategies inadvertently suppressed the vehicle's ability to learn to brake. Moreover, we noticed that the vehicle could self-learn to drive faster and smoother once we removed these human-biased rewards. Therefore, we have shifted towards a more minimalistic reward function design, focusing solely on crucial objectives like completing laps and avoiding crashes, without prescribing specific strategies or behaviors to achieve these objectives. The reward functions are as follows:

- Termination for Non-Forward Movement: If the vehicle remains stationary for a duration of 5 seconds or drives backward, the training will be terminated, imposing a penalty of 100 points.
- Termination on completion: If the vehicle finishes the lap, the training will be terminated.
- Forwarding Reward: The entire lap is uniformly partitioned into 3372 sectors by distance. There is a reward line at the end of each sector. When the vehicle crosses a reward line, it will gain a reward of 100 points.
- Crashing Penalty: The vehicle is equipped with a collision sensor which detects and records each collision event along with its timestamp and intensity. For every recorded collision, the agent incurs a penalty proportional to the intensity of the collision: $\frac{intensity\_of\_the\_collision}{100000}$.Notably, collisions with an intensity exceeding 10000 are considered as major collisions, imposing an additional penalty of 100 points.

4.4 Improvement on model

The previous network structures used in our work were inspired by the successful Atari solution and maintained similar network structures. However, the complexity and dynamic nature of autonomous racing environments require an approach that can handle a higher level of intricacy and abstraction.

Recent advancements in reinforcement learning, such as the DeepMind's Agent57, have demonstrated superior performance in the Atari benchmark by proposing a more sophisticated model structure. Drawing inspiration from this, we introduced substantial modifications to our network structure to enhance its performance and adaptability for the autonomous racing task.

The images input are fed into a Convolutional Neural Network with exactly the same structures as Agent57 uses. There is no normalization or pooling layers in the CNN structures.

A critical component of our new network structure is the inclusion of Long Short-Term Memory (LSTM) [12] blocks. LSTMs are a type of recurrent neural network that can capture temporal dependencies in data, making them suitable for tasks where past observations can influence the understanding of current and future states, as is often the case in dynamic environments like autonomous racing. We use an LSTM block to process the feature vectors derived from the observations, enabling our agent to maintain and leverage a form of memory about the temporal sequence of events in the environment. Following the LSTM block, we incorporated two fully connected layers into our network. These layers allow the model to learn non-linear relationships and interactions among the features provided by the LSTM, which contributes to the agent's decision-making capabilities.

Given the effectiveness of off-policy optimization demonstrated in recent research, we decided to adopt the Soft Actor-Critic (SAC) algorithm for our agent. SAC is a model-free reinforcement learning algorithm that has been shown to outperform other methods in a variety of continuous action space tasks. It provides a balance between exploration and exploitation, which is particularly important in complex environments like autonomous racing.

Finally, we replaced the traditional loss function used in SAC with the Huber loss function. This choice is in alignment with the loss function used in Agent57, and it offers robustness to outliers in the data, which can be particularly beneficial when dealing with real-world data in autonomous racing environments.

By incorporating these modifications into our network architecture, we aim to enhance the performance and robustness of our reinforcement learning agent in the challenging and dynamic domain of autonomous racing.

4.5 Conclusion and Future Work

In its current state of training, our new agent has shown remarkable improvements compared to the previous solutions. The new agent has quickly learned accelerating in straightaways and decelerating during turns. The training of the agent is still underway, but the continuous improvement in the distance traveled at each run and the increasing overall reward signal a promising trend.

[12] https://colah.github.io/posts/2015-08-Understanding-LSTMs/

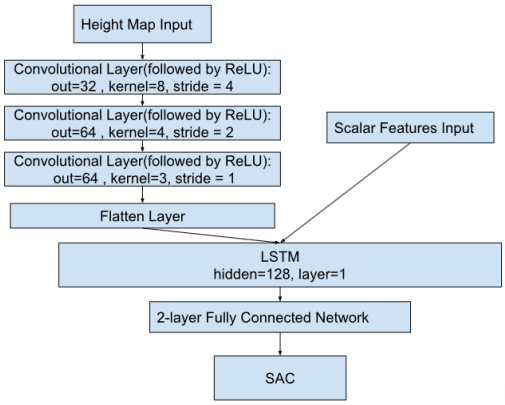


Figure 9: New Network Structures

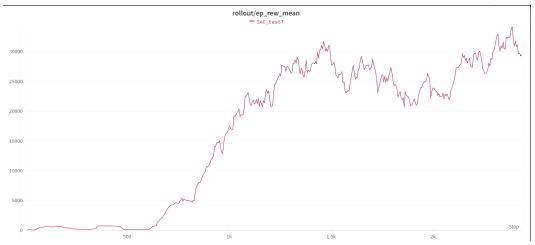


Figure 10: Rewards vs step

However, despite these improvements, there are still aspects of the agent's behavior that require further investigation and fine-tuning. A key observation from the current training phase is that the agent tends to favor the right side of the track and drive closely to the right boundary. It is not yet clear what prompts this behavior. In future work, we will need explore potential causes for this behavior.

We are training the agent over a longer period, which will allow us to evaluate its performance identify any potential issues that may only become apparent over time, and there will be a lot of future work about modifying reward functions and fine tuning.

5 Acknowledgements

I would like to take this opportunity to thank Dr. Allen Yang, who guided the projects and offered invaluable advice and assistance in addressing the challenges and difficulties I encountered along the way.

I also want to thank every member in the ROAR group, especially to Chris Lai, who made significant contributions in the IAC perception project; and Franco Huang and Yunhao Cao, who made significant contributions in the RL project.

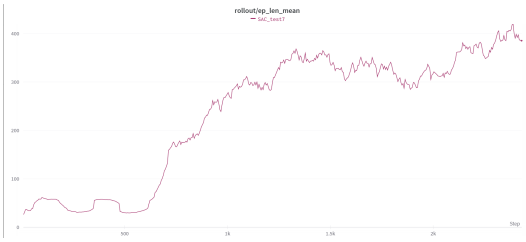


Figure 11: Roll out Length vs step

Working in such a collaborative and dedicated team has truly been an enriching experience, and I am sincerely grateful for the opportunity.

References

[1] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark, Mar 2020.

[2] Johannes Betz, Tobias Betz, Felix Fent, Maximilian Geisslinger, Alexander Heilmeier, Leonhard Hermansdorfer, Thomas Herrmann, Sebastian Huch, Phillip Karle, Markus Lienkamp, and et al. Tum autonomous motorsport: An autonomous racing software for the indy autonomous challenge, Jan 2023.

[3] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, and et al. End to end learning for self-driving cars, Apr 2016.

[4] Padraig Cunningham and Sarah Jane Delany. K-nearest neighbour classifiers: 2nd edition (with python examples), Apr 2020.

[5] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, Aug 2018.

[6] Chanyoung Jung, Andrea Finazzi, Hyunki Seong, Daegyu Lee, Seungwook Lee, Bosung Kim, Gyuri Gang, Seungil Han, and David Hyunchul Shim. An autonomous system for head-to-head race: Design, implementation and analysis; team kaist at the indy autonomous challenge, Mar 2023.

[7] Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. Autoware on board: Enabling autonomous vehicles with embedded systems.

[8] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on §-transformed points, Nov 2018.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, Dec 2013.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, and et al. Human-level control through deep reinforcement learning, Feb 2015.

[11] FLORIAN SAUERBECK, JOHANNES BETZ, DOMINIK KULMER, PHILLIP KARLE, FELIX FENT, and SEBASTIAN HUCH. Multi-modal sensor fusion and object tracking for ... - ieee xplore.

[12] Florian Sauerbeck, Johannes Betz, Markus Lienkamp, and Lucas Baierlein. A combined lidar-camera localization for autonomous race cars.

[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, Aug 2017.

[14] Shinya Sumikura, Mikiya Shibuya, and Ken Sakurada. Openvslam: A versatile visual slam framework, Apr 2023.

[15] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, Jul 2022.

[16] Xingkui Zhu, Shuchang Lyu, Xu Wang, and Qi Zhao. Tph-yolov5: Improved yolov5 based on transformer prediction head for object detection on drone-captured scenarios, Aug 2021.