# Transferable RL for Real-World Navigation Using Semantic Segmentation and Bird's-Eye View Abstraction

**Benedikt Schlereth-Groh**[1], **Şakir Furkan Yöndem**[1], **Ramin Tavakoli Kolagari**[1]

[1]Technische Hochschule Nürnberg, Keßlerplatz 12, 90489 Nürnberg, Germany
benedikt.schlereth-groh@th-nuernberg.de, sakirfurkan.yoendem@th-nuernberg.de, ramin.tavakolikolagari@th-nuernberg.de

## Abstract

Reinforcement Learning (RL) has shown significant promise in developing autonomous navigation algorithms for complex environments. However, the direct application of RL policies trained in simulation to real-world scenarios often faces challenges due to the reality gap. This paper proposes a two-stage system incorporating a segmentation strategy and a bird's-eye-view (BEV) representation to mitigate the domain gap between simulation and reality. In the first stage, the segmentation transforms sensor data into a simplified and interpretable representation of the surrounding area, facilitating transferability across different deployments. In the second stage, the agent navigates through the BEV map, which can be trained using a vectorized simulation environment—a setup that runs multiple parallel instances of the environment to provide a wide range of training scenarios. This vectorization enables rapid exposure to varied environmental conditions, thereby accelerating and diversifying the training of a deep RL agent to achieve optimal navigation behaviours while maintaining high-speed, in-bound trajectories. The segmentation is crucial because it supports generalization of the learned policy across different robotic platforms. The contribution of this paper lies in combining real-time semantic segmentation with a bird's-eye-view navigation policy, resulting in a transferable and scalable framework for real-world deployment of RL-based navigation agents. Experimental results demonstrate that agents trained with this methodology exhibit robust navigation performance and adaptability in both simulated and real-world environments, validating the efficacy of combining vectorized simulation with real-world segmentation for practical robotic navigation.

**Code** — https://github.com/furkanyondemm/DomainIndependentNavigation

## Introduction

The potential of autonomous navigation systems is limited by the significant challenge of ensuring their safety, reliability and interpretability in complex real-world environments. When operating in close proximity to humans, it is crucial that autonomous systems' decision-making processes are safe and trustworthy (Kuznietsov et al. 2024). While traditional navigation methods have been successful in structured settings, they often struggle with the unpredictability and dynamic nature of unstructured human environments, resulting in brittle, non-generalisable solutions. Reinforcement learning (RL) presents an alternative approach, whereby agents learn optimal behaviours through interaction with an environment.

In the field of vision-based robotic grasping, it is well known that there is a significant gap between simulation and reality for reinforcement-trained agents. The costs, time and safety risks involved with training in the real world mean that simulation is a necessary step (Rao et al. 2020). The gap is closing, with simulations and adaptations becoming even more realistic during training, enabling learned agents to be deployed directly into the real world (Zhao, Queralta, and Westerlund 2020). This paper addresses the gap by presenting a feature extraction method that is domain-dependent but provides an interpretable intermediate state. The control policy is trained using a simplified semantic representation of the environment, creating a highly effective and easily transferable policy.

This research addresses the critical need for a scalable, robust method of transferring information from simulation to reality. Our proposed solution uses a real-time segmentation algorithm to transform raw sensor data into a structured bird's-eye view representation, effectively creating a domain-independent state space. This enables an agent, which has been trained using a deep reinforcement learning algorithm in a highly efficient vectorized environment, to be deployed directly onto a physical robot. The resulting system is designed to be efficient in training and transparent in decision-making.

The remainder of this paper is structured as follows. First, we detail the architecture of our system, explaining the vectorized simulation environment, the segmentation pipeline for the mediated state representation and how we train our control network. Next, we provide a thorough evaluation of the performance of the learned navigation policies in a variety of simulated environments and on a 1:10 scale physical vehicle. Finally, we discuss the implications of our findings and outline directions for future work.

## Related Work

Robots can calculate the best route through an environment using a general forward search like the Dijkstra's algorithm

or A* algorithm (LaValle 2006, p. 36). During the exploration of unknown environments, the simultaneous localization and mapping (SLAM) algorithm helps us to understand our surroundings(LaValle 2006, p. 656). Galvis et al. propose a navigation framework that segments camera input into cost maps. These cost maps can then be used to calculate the best path for a robot (Galvis et al. 2023). In 2016, a team from NVIDIA demonstrated how a car could be controlled using raw sensor input. The trained network uses a convolutional neural network (CNN) to extract important information from cameras and fully connected layers to provide vehicle control. It can therefore provide vehicle control from three camera feeds (Bojarski et al. 2016).

### Reinforcement Learning for Autonomous Navigation

It has been demonstrated that reinforcement learning can be used to train an agent using sensor data as the input and steering and throttle control as the output. For example, Toromanoff et al. demonstrate the navigation of an urban environment with RL, achieving successful lane keeping and vehicle avoidance (Toromanoff, Wirbel, and Moutarde 2019). End-to-end driving capabilities have also been successfully implemented in racing on different tracks (Jaritz et al. 2018). Recent studies have shown that control algorithms trained using reinforcement learning outperform both model predictive control and human performance (Czechmanowski et al. 2025).

### Simulations With Real-World Examples

A simulator for researching autonomous driving in urban environments is available with *CARLA* (Dosovitskiy et al. 2017). Thanks to the Unreal Engine, sensor renderings, realistic physics and basic non-player characters (NPCs) are possible (Dosovitskiy et al. 2017). The Duckietown platform provides a cost-sensitive alternative for transferring researched algorithms for urban navigation, which is also available as a simulation (Paull et al. 2017; Chevalier-Boisvert et al. 2018).Other simulation environments allow drivers to push the limits of driving dynamics and simulate racing cars (Espié et al. 2005). However, in order to transfer algorithms from simulation to reality, the simulations must already account for real-world sensor setups, dimensions and control behaviours which is possible with the DonkeyCar simulation and car (Kramer et al. 2017). The F1Tenth platform currently allows the development of algorithms for scaled-down racing cars that use Light Detection and Ranging (LiDAR) technology to detect the boundaries of the track (O'Kelly et al. 2020).

## Setup

Figure 1 depicts the developed method, consisting of two neural networks that can be trained independently of each other. The first neural network performs segmentation to identify the drivable track and generate a map showing where the robot can safely drive. This segmentation can be converted into a bird's-eye view, since the intrinsic parameters of the front-facing camera are known. The control neural network uses this map to calculate the best steering and gas commands for the robot.

### Feature Extraction Using Segmentation

Supervised machine learning techniques perform well when it comes to detecting the track. In order to adapt to a new environment, a few sensor readings showing different aspects of the road markings are required. These readings must be annotated before they can be used to train a deep neural network using a supervised learning approach. To enable inference on the edge in the future (e.g. on a *Raspberry Pi* with an AI Kit/*Hailo* device), the YOLO segmentation algorithm has been selected. The dataset consists of 250 images captured directly from the DonkeyCar and NeuroWheel simulation environments. Due to the limited environmental complexity — roads with consistent lane markings and minimal distracting objects — only a small number of images were required to achieve satisfactory segmentation accuracy. The images were annotated in YOLO segmentation format using *Roboflow*, producing polygon masks for drivable areas. The dataset was split into an 80% training set and a 20% validation set. Training was performed for 30 epochs with a batch size of eight and an effective learning rate of 0.00125, obtained by linearly scaling the default rate of 0.01 according to the batch size. This adjustment follows the widely used *linear scaling rule*, which increases or decreases the learning rate in direct proportion to the change in batch size, thereby maintaining similar gradient update magnitudes and stable convergence (Goyal et al. 2017). Visualising the marking of the track as a permitted driving route and maintaining this recognition within the navigation pipeline enables the approach to be visualised in an interpretable way.

### Bird-Eye-View Transformation

In order to enable the effective transfer of reinforcement learning (RL) policies from simulation to the real world, it is essential to standardise the agent's spatial perception. First-person visual input captured from onboard cameras is often distorted by projective effects and scene-specific features, hindering the generalisation of trained policies. To mitigate this issue, we use a bird's-eye-view (BEV) transformation to project the scene into a consistent top-down view. This serves as an intermediate representation for policy learning.

In our setup, an RC-car is equipped with a **PiCamera 3** mounted at the front as the primary vision sensor. The camera captures images at a resolution of $640 \times 480$ pixels with a horizontal field of view (FoV) of approximately $72°$. These parameters are used to construct the camera's intrinsic matrix and estimate the focal length required for accurate BEV projection.

The BEV representation allows the RL agent to interpret the environment in a domain-invariant way, making it less sensitive to variations in camera position, orientation and lighting. To obtain this consistent spatial abstraction, the BEV transformation is constructed through the following steps (Szeliski 2011, p. 31). This process also aligns well with vectorized simulation environments, where multiple agent instances are trained in parallel under diverse conditions.

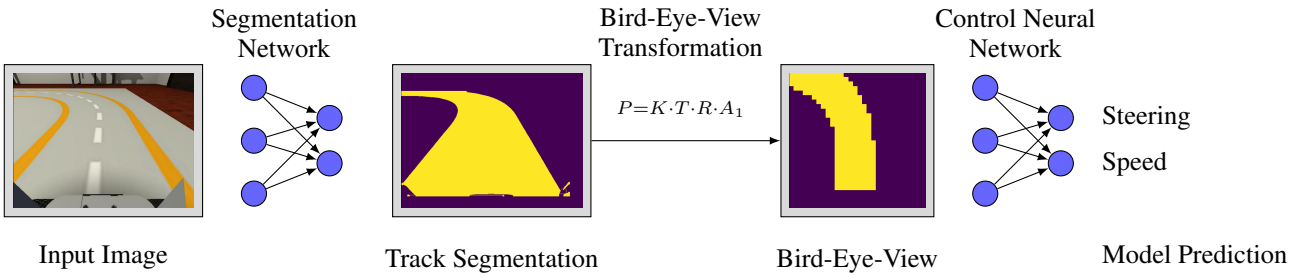


Figure 1: Overview

**1) Centering the Image Coordinate System:**
The original image coordinate system is shifted such that the origin is located at the centre of the frame, simplifying subsequent 3D transformations:

$$A_1 = \begin{bmatrix} 1 & 0 & -\frac{w}{2} \\ 0 & 1 & -\frac{h}{2} \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

**2) Rotation Matrices:**
To simulate a virtual top-down view, we apply rotations around the X, Y, and Z axes at angles $\alpha = 35°$, $\beta = 90°$, and $\gamma = 90°$. These are defined as follows:

$$R_X(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$R_Y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$R_Z(\gamma) = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

The combined rotation matrix is computed as:

$$R = R_X(\alpha) \cdot R_Y(\beta) \cdot R_Z(\gamma) \tag{5}$$

**3) Translation Along the Z-axis:**
To simulate the physical height of the camera above the ground plane, a translation is applied along the Z-axis using a fixed distance of $d = 115\,\text{mm}$:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

**4) Camera Intrinsic and Focal Length Estimation:**
The focal length $f$ is computed from the image width $w$ and the horizontal field of view angle $\theta_H$, assuming a pinhole camera model. The relationship is given by:

$$f = \frac{w}{2} \left[ \tan\left( \frac{\theta_H}{2} \right) \right]^{-1} \tag{7}$$

This value is then used to construct the intrinsic matrix:

$$K = \begin{bmatrix} f & 0 & \frac{w}{2} & 0 \\ 0 & f & \frac{h}{2} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{8}$$

**5) Final Projection Matrix:**
The full perspective transformation matrix is computed as:

$$P = K \cdot T \cdot R \cdot A_1 \tag{9}$$

This matrix is applied to the input image using a homographic warping operation (e.g., OpenCV's `warpPerspective`) to generate the final bird's-eye-view output.

## Control Neural Network

The Control Neural Network is trained using the *Gymnasium* framework, leveraging the *CarRacing-v3* environment as a simulation platform (Towers et al. 2024). This environment emulates a continuous control task in which an agent must navigate a generated track using visual input from a top-down perspective.

To more closely approximate the perception system of real-world autonomous vehicles, a custom wrapper has been implemented to convert the camera observations (originally 96×96×3) into simplified bird's-eye view segmentation. This is achieved by applying a preprocessing pipeline that includes greyscale conversion, an intensity threshold and an approximation of the possible field of view, in order to extract a binary mask of the drivable road area. The result is a compact binary image that highlights the track boundaries, providing a top-down view that is similar to the output of a front-facing camera after semantic segmentation and bird's-eye view transformation.

Training the network using *Gymnasium* in combination with the *Stable Baseline 3* environment significantly speeds up the training process (Raffin et al. 2021). Generating 1,000,000 runs and training the control network takes around five hours with an *RTX 2080 Ti*. In order to train a stable neural network, policies from the Deep Q-Network

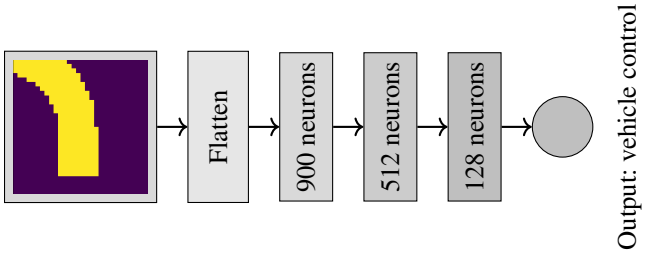


Figure 2: Control Network Layout

(DQN) are used alongside a custom network. Mnih et al. first presented this method to learn how to play Atari 2600 games and successfully applied the same network to different games (Mnih et al. 2013).

Figure 2 illustrates the structure of the most effective control neural network..

This network is then trained end-to-end to learn control policies (steering and acceleration) directly from the processed input. Using this segmentation-based representation enables the network to focus on the spatial structure of the road, filtering out textures and colours. This improves generalisation and the potential for transferring the control algorithm to other domains.

## Experiments

In order to evaluate the effectiveness and generalisability of our proposed system, we conducted experiments in three distinct simulation environments: DonkeyCar, NeuroWheel, and CARLA. In all settings, the system follows a consistent pipeline: first, sensor data is processed through a segmentation-based perception module to extract drivable areas. These segmented outputs are then transformed into interpretable bird's-eye-view (BEV) masks, which serve as the observation space for a deep reinforcement learning agent.

Figure 3 illustrates the full perception-to-control loop used in our experiments. The modular nature of this architecture enables transferability to different simulation platforms and robot configurations, as demonstrated in our deployments.

Although LiDAR-centric approaches, such as those employed in the *F1TENTH* framework, are less susceptible to domain discrepancies between simulation and reality, our research focuses specifically on vision-based control.In this context, visual domain shifts (e.g., changes in illumination or texture) present a significant challenge. This design choice enables us to study robustness under realistic camera-based perception constraints. Alongside our proposed segmentation-based RL pipeline, we also compare with a classical HSV-based PD/PID lane-following baseline. This baseline processes HSV-thresholded camera images to detect lane markings and compute steering commands directly, bypassing segmentation and RL entirely. Furthermore, we incorporate human driving performance into the evaluation to provide a more comprehensive comparison, enabling the quantification of the performance gap between automated control policies and skilled human operation. This comparison sheds light on the trade-offs between learning-based policies, traditional control strategies, and human expertise in terms of accuracy, robustness, and latency.

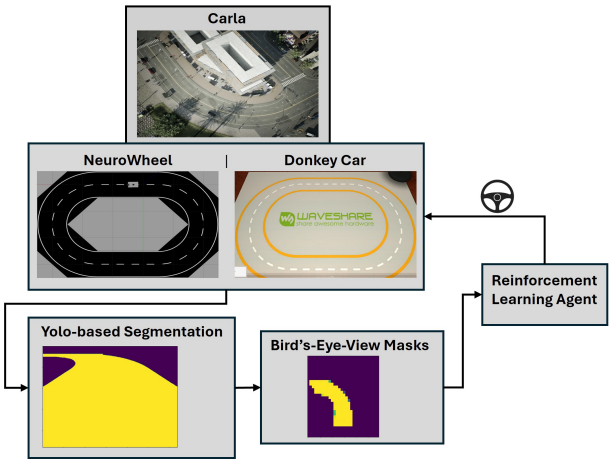


Figure 3: Modular Perception-to-Control Architecture for Sim-to-Real Transfer

### Training of Control Neural Network

Our control neural network is trained using the top-down view of the *CarRacing-v3* environment provided by *Gymnasium* (Towers et al. 2024). Vectorisation enables many epochs to be simulated and evaluated within a short period of time. This enables a wide range of network architectures and command structures to be tested for the robot. Figure 4 shows the progress, illustrating how the reward increases over time and how the best model is found within the total simulation time.

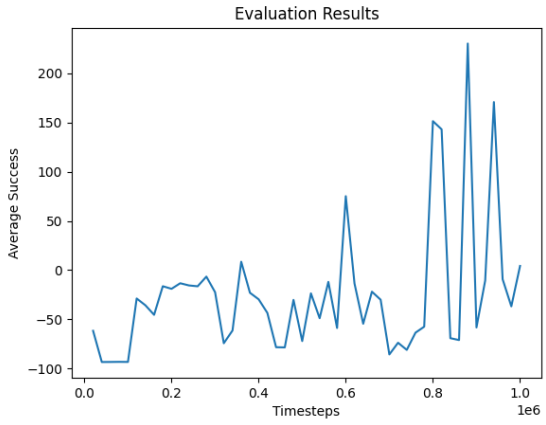


Figure 4: Training Progress of the Control Neural Network

### Donkey Car

To ensure reproducibility and enable comparison with other approaches, we integrated our system into the widely used

| Model | Lap time [s] |
|---|---|
| Donkey-Car Simulator | 6.7 |
| Human Performance | 12.95 |
| Our | 11.66 |

Table 1: Donkey Car Results

| Model | Lap time [s] |
|---|---|
| Lane Following with PID | 22.13 |
| Human Performance | 23.16 |
| Our | 27.68 |

Table 2: NeuroWheel results

DonkeyCar simulation framework for autonomous driving research with model vehicles. We selected the Donkey-Waveshare-v0 map because of its structural similarity to the indoor test track in our laboratory. The camera parameters within the simulation were calibrated to closely match those of our real RC car platform, enabling consistent visual perception across domains.

We exported images from the simulation in order to train a YOLO-based segmentation model that generates lane masks from raw camera input. These masks were then converted into top-down, bird's-eye-view representations and used as observations for a reinforcement learning agent that was trained in a vectorised Gym environment. As illustrated in Figure 5, the agent performs real-time steering actions based on segmentation inputs, maintaining a consistent trajectory within lane boundaries.

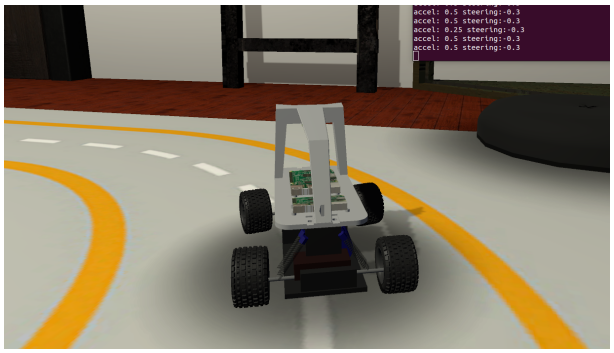

Figure 5: RL Agent Driving in DonkeyCar with BEV from YOLO Segmentation

## NeuroWheel

In order to test our system under conditions that more closely mirror our physical setup, we developed a custom Gazebo-based simulation environment called NeuroWheel. This includes a 1:1 scale model of the RC car in our laboratory, replicating its physical dimensions, camera intrinsic parameters and sensor setup. We also reconstructed the geometry and layout of our real-world indoor track within the simulation. As with the DonkeyCar setup, we captured images from the simulation and trained a YOLO-based segmentation model to identify drivable areas. These segmented images were transformed into bird's-eye-view masks and used as inputs for the reinforcement learning framework. The trained agents demonstrated reliable and efficient navigation on the NeuroWheel simulation track, thus validating the consistency of our approach across environments. Figure 6 shows the RL agent successfully navigating the NeuroWheel simulation track.

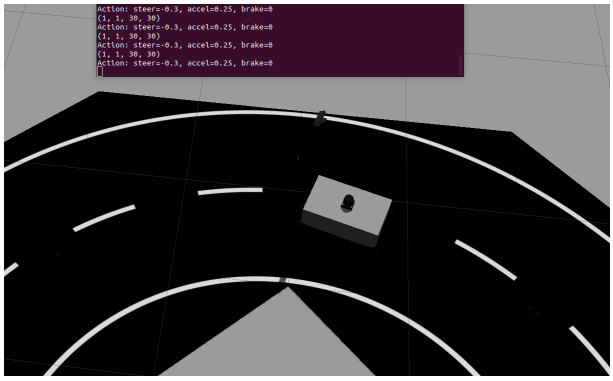

Figure 6: NeuroWheel Navigation Using YOLO Segmentation and BEV

## Carla Simulator Evaluation

In addition to the DonkeyCar and NeuroWheel environments, we evaluated our approach in the CARLA simulator on a predefined route, from point 1 to point 2, as illustrated in Figure 7. The test scenario included a road with varying illumination conditions to better reflect real-world challenges.

Figure 7: Carla Road

As shown in Table 3, variations in lighting conditions in CARLA introduce a significant domain gap for classical lane detection methods, which limits their ability to operate at high speeds. In contrast, DonkeyCar and NeuroWheel simulations exhibit minimal domain gap for such methods, as no lighting variations occur, allowing the vehicle to drive faster. For road segmentation, unlike DonkeyCar and NeuroWheel, we leveraged CARLA's built-in semantic segmen-

| Model | Lap time [s] |
|---|---|
| Lane Following with PD | 31.45 |
| Human Performance | 22.25 |
| Our | 24.75 |

Table 3: Carla Results

tation camera. The generated road mask was transformed into a bird's-eye view, resized, and fed into our RL agent as input. In the intersection-driving scenario, the RL agent successfully navigated the vehicle despite environmental variations. Our RL policy was speed-limited to 75% of the maximum speed (i.e., max_speed = `1.0` in CARLA, but `0.75` for the RL agent), which accounts for the manual driving being slightly faster.

### Real World

The physical testing environment is designed to replicate a simplified road system at a scale of 1:10 for safe and controlled experimentation. The laboratory setup consists of modular black tiles with white line markings that can be arranged to form customisable track layouts mimicking lanes, curves and intersections. These high-contrast markings are ideal for visual perception algorithms, especially those used for line detection and road segmentation with bird's-eye view cameras.

The autonomous platform is a 1/10 scale RC car equipped with onboard computation and sensing capabilities, shown in Figure 8. The main onboard components include:

- Raspberry Pi 5: Serves as the primary computing unit, responsible for data processing, control inference, and communication.
- Raspberry Pi Camera Module v3: Mounted at the front of the vehicle to capture high-resolution video, which is used for real-time perception and road-following tasks.
- AI Kit: A hardware extension providing access to additional compute resources (Hailo Device) to run the segmentation algorithm efficiently.

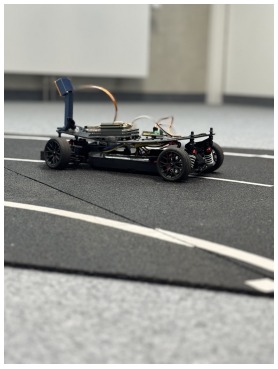

Figure 8: RC Car

To ensure flexibility and portability of the software, the entire system is orchestrated using Docker containers, enabling the seamless deployment and updating of various software components. Sensor drivers, perception modules and control logic are implemented using ROS 2 (Robot Operating System 2), which facilitates inter-process communication and offers real-time capabilities that are well-suited to robotics applications. Each component — camera stream, segmentation node, control inference and actuator commands — runs in its own container, enabling easy debugging, monitoring and upgrading without affecting the entire system.

This lab setup enables rapid prototyping, reproducible experiments and integration with simulation environments such as *Gazebo*, facilitating various training and deployment strategies for autonomous driving.

## Conclusion

In this work, we present a modular and transferable reinforcement learning framework for autonomous navigation. This framework is built upon a two-stage architecture that combines real-time semantic segmentation with bird's-eye-view abstraction. Our approach explicitly addresses the domain gap between simulation and reality by transforming raw sensor input into a structured, domain-invariant representation. This design allows an RL policy trained entirely in a vectorised simulation environment to be deployed in simulated and physical scenarios with minimal adaptation.

One of the key contributions of this work is the development of a pipeline for navigating different environments. This supports adaptation to new challenges, and the intermediate visualisation helps to understand how decisions are made. DonkeyCar, Carla and F1TENTH allow different algorithms to be developed quickly, but there is a particular challenge in closing the sim-to-real gap when deploying to a real robot. Alongside a flexible Docker-based setup for our robot, we demonstrate effective adaptation to three different simulations. However, deployment to our own robot failed due to communication issues with the motors.

Experiments conducted in public and custom-built simulation environments demonstrate that the trained agents exhibit reliable and consistent navigation behaviour. Deploying the system in different simulations confirms its viability for navigation and validates our objective of minimising the effort required to transfer AI-based control systems. This is achieved by reducing the need for retraining, making pipeline components reusable and enabling platform-agnostic deployment.

Although our results emphasise the advantages of using segmentation-based input and domain abstraction, we also found that reinforcement learning agents trained in this manner currently have slower response times than traditional lane-following controllers. Future work will focus on improving the inference speed and decision-making efficiency of the RL agent, particularly under real-time constraints.

Ultimately, our work contributes to the development of scalable, interpretable and transferable AI systems for autonomous navigation, bringing us closer to deploying robust real-world robot controllers that have been trained in fully virtual environments.

## References


Bojarski, M.; del Testa, D. W.; Dworakowski, D.; Firner, B.; Flepp, B.; Goyal, P.; Jackel, L. D.; Monfort, M.; Muller, U.; Zhang, J.; Zhang, X.; Zhao, J.; and Zieba, K. 2016. End to End Learning for Self-Driving Cars. *ArXiv*, abs/1604.07316.

Chevalier-Boisvert, M.; Golemo, F.; Cao, Y.; Mehta, B.; and Paull, L. 2018. Duckietown Environments for OpenAI Gym. https://github.com/duckietown/gym-duckietown.

Czechmanowski, G.; Wegrzynowski, J.; Kicki, P.; and Walas, K. 2025. On learning racing policies with reinforcement learning. *ArXiv*, abs/2504.02420.

Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; and Koltun, V. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, 1–16.

Espié, E.; Guionneau, C.; Wymann, B.; Dimitrakakis, C.; Coulom, R.; and Sumner, A. 2005. TORCS, The Open Racing Car Simulator.

Galvis, J.; Pediaditis, D.; Almazrouei, K. S.; and Aspragathos, N. 2023. An Autonomous Navigation Approach based on Bird's-Eye View Semantic Maps. *2023 27th International Conference on Methods and Models in Automation and Robotics (MMAR)*, 81–86.

Goyal, P.; Dollár, P.; Girshick, R.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; and He, K. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*.

Jaritz, M.; de Charette, R.; Toromanoff, M.; Perot, E.; and Nashashibi, F. 2018. End-to-End Race Driving with Deep Reinforcement Learning. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2070–2075.

Kramer, T.; et al. 2017. DonkeyCar: An Open Source DIY Self Driving Platform. https://github.com/autorope/donkeycar. Version 5.1.0, accessed 2025-07-31.

Kuznietsov, A.; Gyevnar, B.; Wang, C.; Peters, S.; and Albrecht, S. V. 2024. Explainable AI for Safe and Trustworthy Autonomous Driving: A Systematic Review. *IEEE Transactions on Intelligent Transportation Systems*, 25: 19342–19364.

LaValle, S. M. 2006. *Planning Algorithms*. Cambridge University Press.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. A. 2013. Playing Atari with Deep Reinforcement Learning. *ArXiv*, abs/1312.5602.

O'Kelly, M.; Zheng, H.; Karthik, D.; and Mangharam, R. 2020. F1TENTH: An Open-source Evaluation Environment for Continuous Control and Reinforcement Learning. In Escalante, H. J.; and Hadsell, R., eds., *Proceedings of the NeurIPS 2019 Competition and Demonstration Track*, volume 123 of *Proceedings of Machine Learning Research*, 77–89. PMLR.

Paull, L.; Tani, J.; Ahn, H.; Alonso-Mora, J.; Carlone, L.; Cáp, M.; Chen, Y. F.; Choi, C.; Dusek, J.; Fang, Y.; Hoehener, D.; Liu, S.; Novitzky, M. M.; Okuyama, I. F.; Pazis, J.; Rosman, G.; Varricchio, V.; Wang, H.-C.; Yershov, D. S.; Zhao, H.; Benjamin, M. R.; Carr, C.; Zuber, M. T.; Karaman, S.; Frazzoli, E.; Vecchio, D. D.; Rus, D.; How, J. P.; Leonard, J. J.; and Censi, A. 2017. Duckietown: An open, inexpensive and flexible platform for autonomy education and research. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 1497–1504.

Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M.; and Dormann, N. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268): 1–8.

Rao, K.; Harris, C.; Irpan, A.; Levine, S.; Ibarz, J.; and Khansari, M. 2020. RL-CycleGAN: Reinforcement Learning Aware Simulation-to-Real. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 11154–11163.

Szeliski, R. 2011. Computer Vision - Algorithms and Applications. In *Texts in Computer Science*.

Toromanoff, M.; Wirbel, É.; and Moutarde, F. 2019. End-to-End Model-Free Reinforcement Learning for Urban Driving Using Implicit Affordances. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 7151–7160.

Towers, M.; Kwiatkowski, A.; Terry, J. K.; Balis, J. U.; Cola, G. D.; Deleu, T.; Goulão, M.; Kallinteris, A.; Krimmel, M.; Arjun, K.; Perez-Vicente, R.; Pierr'e, A.; Schulhoff, S.; Tai, J. J.; Tan, H.; and Younis, O. G. 2024. Gymnasium: A Standard Interface for Reinforcement Learning Environments. *ArXiv*, abs/2407.17032.

Zhao, W.; Queralta, J. P.; and Westerlund, T. 2020. Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey. *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 737–744.