robotics MDPI

Article

Deep Reinforcement Learning for Autonomous Dynamic Skid Steer Vehicle Trajectory Tracking

Sandeep Srikonda †, William Robert Norris *,†, Dustin Nottage and Ahmet Soylemezoglu

Autonomous and Unmanned Vehicle Systems Laboratory, Department of Industrial and Enterprise Systems Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
* Correspondence: wrnorris@illinois.edu
† These authors contributed equally to this work.

Abstract: Designing controllers for skid-steered wheeled robots is complex due to the interaction of the tires with the ground and wheel slip due to the skid-steer driving mechanism, leading to nonlinear dynamics. Due to the recent success of reinforcement learning algorithms for mobile robot control, the Deep Deterministic Policy Gradients (DDPG) was successfully implemented and an algorithm was designed for continuous control problems. The complex dynamics of the vehicle model were dealt with and the advantages of deep neural networks were leveraged for their generalizability. Reinforcement learning was used to gather information and train the agent in an unsupervised manner. The performance of the trained policy on the six degrees of freedom dynamic model simulation was demonstrated with ground force interactions. The system met the requirement to stay within the distance of half the vehicle width from reference paths.

Keywords: trajectory tracking; reinforcement learning; skid-steer robots; deep learning; dynamic model

check for updates

Citation: Srikonda, S.; Norris, W.R.; Nottage, D.; Soylemezoglu, A. Deep Reinforcement Learning for Autonomous Dynamic Skid Steer Vehicle Trajectory Tracking. *Robotics* **2022**, *11*, 95. https://doi.org/10.3390/robotics11050095

Academic Editor: Sunan Huang

Received: 24 June 2022
Accepted: 17 August 2022
Published: 9 September 2022

1. Introduction

Mobile robots play an important role in modern daily life and in various industries such as construction, logistics and transportation. Their influence has become even more evident with the advancement in autonomous driving research. On-road vehicles operate in a much more structured environment than off-road vehicles, where the environment is complex, unstructured and mechanically taxing. Skid-steered robots have several advantages while operating in these tough environments by maintaining greater traction on rough terrain. The same design can be used for both tracked and wheeled robots. Since there are no steering wheels, there is no need for a steering mechanism. All the wheels on each side drive in same direction so only two motors are sufficient for driving and steering the robot.

Skid-steered robots, whether they are tracked or wheeled vehicles, have complex dynamics associated with their motion. This arises from interaction with the road or the contact surface and their joints which constrain their motion. This is not considered by models without any slip descriptions or purely kinematic descriptions of their motion. This makes it difficult to model them using simplistic models that have no slip assumptions or purely kinematic descriptions of their motion.

One of the fundamental requirements for an autonomous system is to be able to follow a trajectory provided by a path planner in a reliable manner. Due to limitations of existing control schemes, the objective was to develop a controller to adapt to this complexity while ensuring the vehicle followed a trajectory. Most of the previous approaches developed for skid-steer robot path following use kinematic models of the robot. These models do not account for wheel slip. For example, Ref. [1] used a terrain dependent kinematic model whose parameters were experimentally evaluated. This might not be feasible for vehicles operating in uncertain and dynamic environments. For example, Ref. [2] used a slip aware model predictive component to correct the control signals generated by a pure pursuit path follower. This approach used linearized dynamics for estimating slip, which is not ideal for some of the operating conditions of a field robot. A 6 Degrees of Freedom (DoF) dynamic model of the vehicle with non-linear ground contact forces was employed. This improved the accuracy of the vehicle motion modeling and enabled better training and tuning of the reinforcement learning agent.

The remainder of the paper is organized as follows. Section 2 provides a literature review. Section 3 provides a description of the nonlinear vehicle model. Sections 4 and 5 discuss the structure of the Deep Deterministic Policy Gradient and reinforcement learning neural network setup and training. Section 6 provides the results of the trained policy as evaluated on various trajectories in simulation to assess performance measurements and errors. Lastly, Section 7 concludes the paper.

2. Background

In recent years, deep learning has been successfully used in computer vision, natural language processing and speech recognition. Deep learning was developed from artificial neural networks(ANNs), whose structure is explained in Figure 7. ANNs were originally inspired by biological neuronal networks. Deep neural networks have intermediate layers of neurons called hidden layers and they help a deep network form abstract representations of the problem domain. This is evident within feature representations of various layers of a trained deep neural network and in contrast to the traditional methods where the features are designed by domain experts. Despite their success in many fields, deep neural networks are trained with huge sets of labeled training data. This approach is prohibitively expensive or impractical for problems involving real world robotic control due to safety concerns. Reinforcement learning involves agents that learn to perform a task by interacting with the environment without expert operation and guidance. Feedback to the agent is provided through a reward function from which it learns the decision making policies to perform the task. The advances made in reinforcement learning and deep learning in the past few years to solve classic Atari games [3] inspired the research community to design algorithms to address problems in vehicle control [4], motion planning [5] and navigation [6] related tasks.

In [7], the dynamics of a skid-steered vehicle was used for the path following task. The controller needed to be tuned in a supervised manner. Using the reinforcement learning paradigm allowed the designer to train the controllers in an unsupervised manner through the use of a reward system. This removed the need for large databases or hand tuning of controller parameters. Reinforcement learning has been successfully applied in control problems developing policies for low level motor control such as [8], robot tracking [9] and aircraft control [10]. The success of deep learning and reinforcement learning in control problems led to the exploration of these approaches for path tracking of skid-steered vehicles. Robotics control problems have continuous variables as outputs. Deep Deterministic Policy Gradients [4] (DDPG) is one of the reinforcement learning algorithms designed to accommodate continuous control actions. The DDPG was applied towards designing a controller to perform the trajectory following task for a skid-steered robot. This work is, given similar research, the first time that a reinforcement learning algorithm, as demonstrated in Figure 1, is used to train a Deep Neural Network based controller for the trajectory following of a dynamic skid-steer vehicle system in simulation and in a real world application.

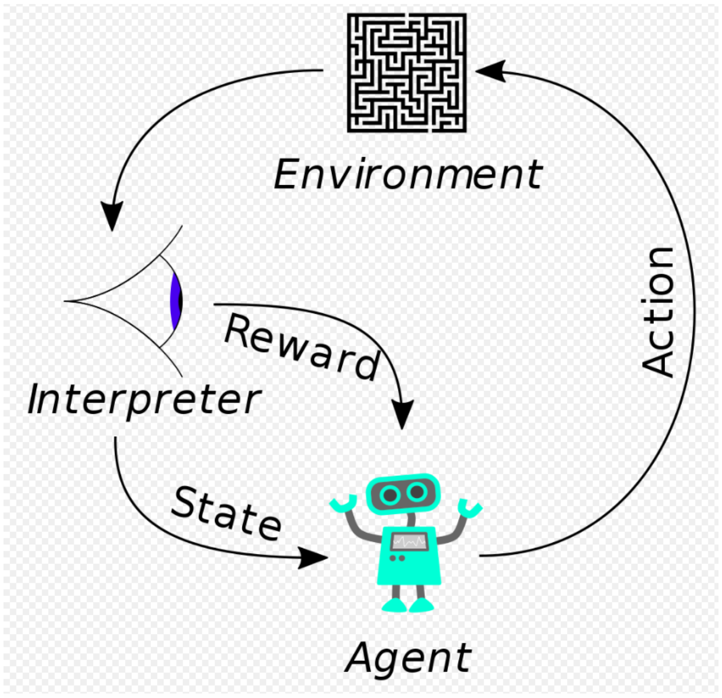

Figure 1. Reinforcement Learning overview. Source: Wikipedia.

3. Vehicle Model Description

Skid steering is a driving mechanism implemented on vehicles with either tracks or wheels, which uses a differential drive concept. Common skid-steered vehicles are tracked tanks and bulldozers. This driving method engages each side of the tracks or wheels separately, and turning is accomplished by generating differential velocity on opposite sides of the vehicle. The wheels or tracks are non-steerable. For example, a right turn can be achieved by turning the left wheels forward and the right wheels backward or making them stationary and vice versa for a left turn. Another advantage to this mechanism is that if the velocities are applied continuously then the robot can theoretically complete a full 360°. turn with zero radius provided there is no wheel slip. Skid-steered robots have complicated dynamics due to the nature of the interaction of wheels with the contact surface leading to nonlinear forces acting on the vehicle. Much of the work performed in this area uses simplified kinematic models or dynamic models that cannot incorporate ground contact forces in a detailed manner. The dynamic model developed in the simulation environment was based on a commercial skid-steer robot called the Jackal and made by Clearpath Robotics. The Jackal, as seen in Figure 2, is a small, entry-level field robotics research platform. It has an onboard computer, GPS and an IMU along with the Robot Operating System (ROS) integration. This facilitates communication, algorithm development and deployment of the final controllers. It has a long battery life and a rugged aluminum chassis which enables all-terrain operation.

The dynamic model of the skid-steered vehicle has a wheelbase and four wheels attached to it with 6 DoF. The forward dynamics is calculated using 6 DoF rigid body dynamics, with torques and forces generated by normal forces and friction forces acting on the wheels as well as the internal motor torques.

In this model, the vehicle was treated as a rigid body that interacted with a compliant ground. This compliant ground was modeled as a uniform distribution of an infinite number of non-linear spring-damper pairs. Further, these rigid body-ground interactions were represented as a set of discrete contact points, each of which caused the ground to deflect spherically.

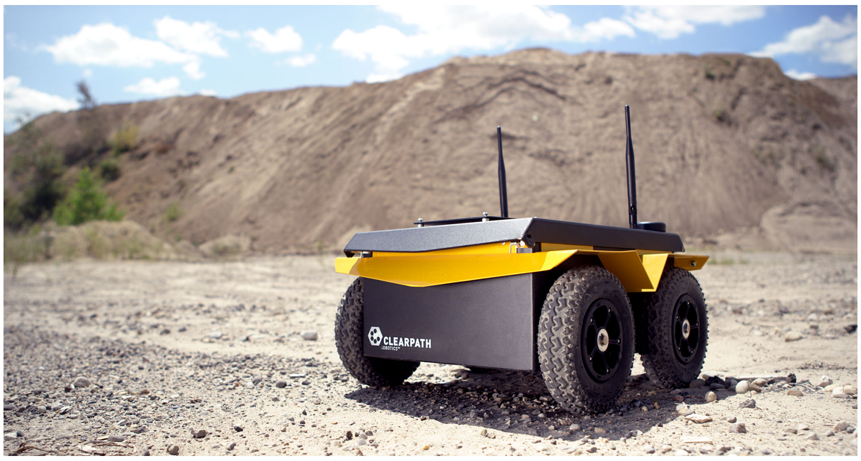

Figure 2. Jackal robot. Source: Clearpath Robotics.

Ground forces were calculated by modeling the tires as pneumatic soft tires. The wheel was approximated with a set of springs of given spring constant under compression from ground forces. Skid-steered vehicles have fixed tire orientations which means that during a turn, each tire has a different turn radius. Assuming a no slip condition would geometrically constrain the vehicle and movement would be impossible. Therefore a skid-steer vehicle relies on slipping to complete turns. A detailed dynamic model was required to calculate the reaction forces and account for the effect of slipping. The tire-terrain model proposed by [11] required slip ratio, slip angle, sinkage and tire velocity to compute at each tire for the reaction forces in the x, y, and z directions as well as the reaction torque. Tire slip ratio was defined as:

$$s = 1 - \frac{\mu_x}{R\omega} \tag{1}$$

where μ_x is the forward velocity of the tire, R is the radius of the tire, and ω is the rotational tire velocity. Slip angle is defined as:

$$\beta = tan^{-1}\left(\frac{\mu_y}{\mu_x}\right) \tag{2}$$

where μ_y is the lateral velocity of the tire. Sinkage is the depth the tire sinks, called z_r. The model detailed in [11] works by integrating the shear and normal stresses caused by the soil displacement over the surface of the tire.

$$z_u = (\frac{p_g}{k_u})^{\frac{1}{n}} \tag{3}$$

$$z_e = (\frac{p_g}{\frac{k_c}{b} + k_\phi})^{\frac{1}{n}} \tag{4}$$

z_u refers to the unloading height and z_e is the sinkage of the flexible tire. p_g is the tire pressure,which is assumed constant and b is the tire width. k_c, k_ϕ, k_u are soil parameters.

$$\theta_f = cos^{-1}(1 - \frac{z_r}{R}) \tag{5}$$

$$\theta_r = -cos^{-1}(1 - \frac{z_u + z_r - z_e}{R}) \tag{6}$$

$$\theta_c = cos^{-1}(1 - \frac{z_r - z_e}{R}) \tag{7}$$

where $\theta_f, \theta_r, \theta_c$ are tire angles corresponding to z_f, z_r, z_c. R is the radius of the tire. Equations from [11] demonstrate how the shear and normal stress can be calculated from the deformation of the soil in these three zones. The result is a nonlinear relationship between the input parameters and the reaction forces which can be observed in Figures 3–6.

The relative modulus of elasticity between the wheel(s) and the ground, E^*, was computed using (8). In (8) the wheel(s) and ground had moduli of elasticity tied to E_w and E_g, and Poisson ratios given by ν_w and ν_g, respectively.

$$\frac{1}{E^*} = \frac{1 - v_w^2}{E_w} + \frac{1 - v_g^2}{E_g} \tag{8}$$

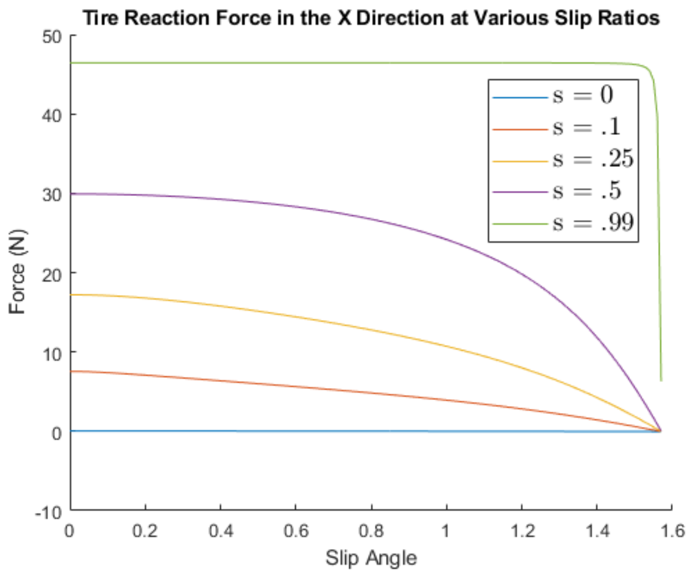

Figure 3. Tire Reaction Force in the X Direction by Slip Angle at Various Slip Ratios.

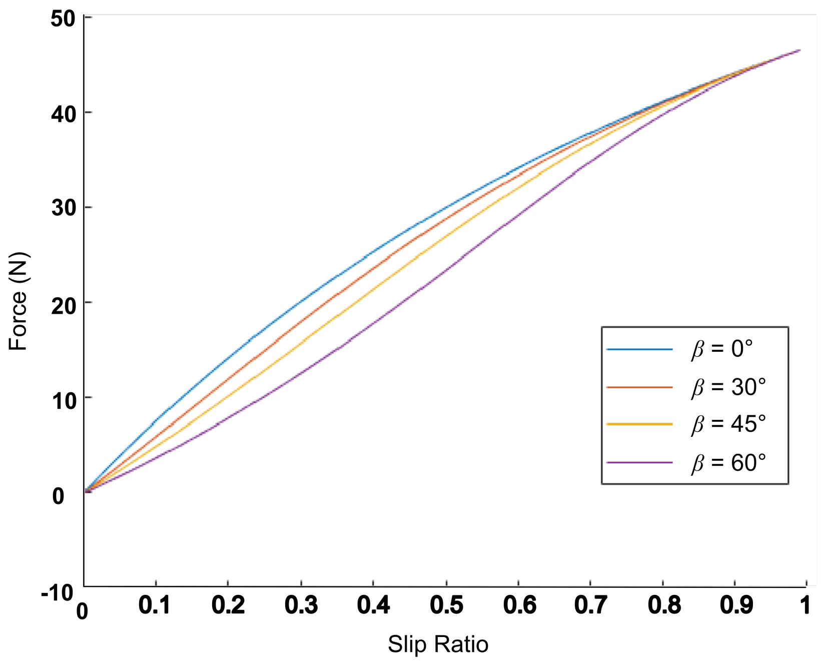

Figure 4. Tire Reaction Force in the X Direction by Slip Ratio at Various Slip Angles.

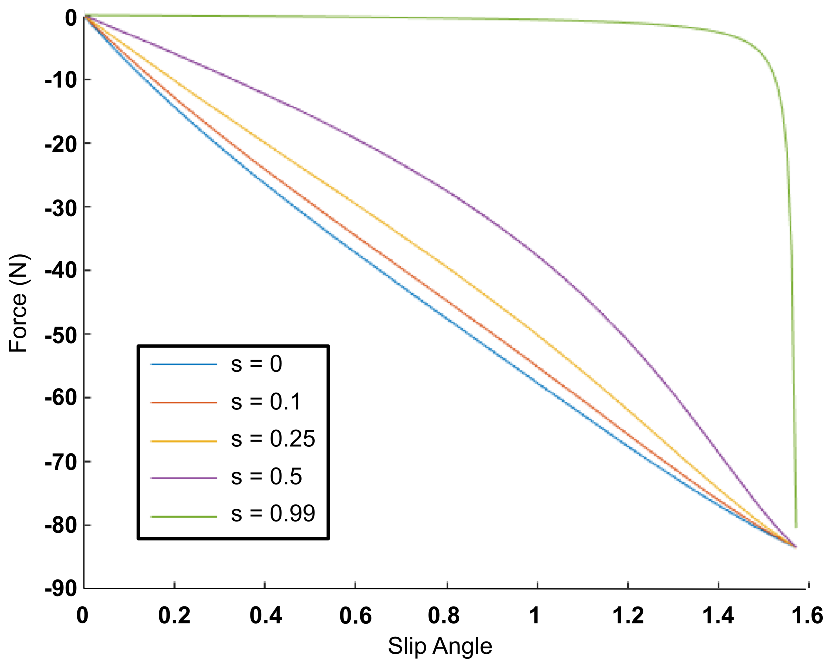

Figure 5. Tire Reaction Force in the Y Direction at various slip ratios.

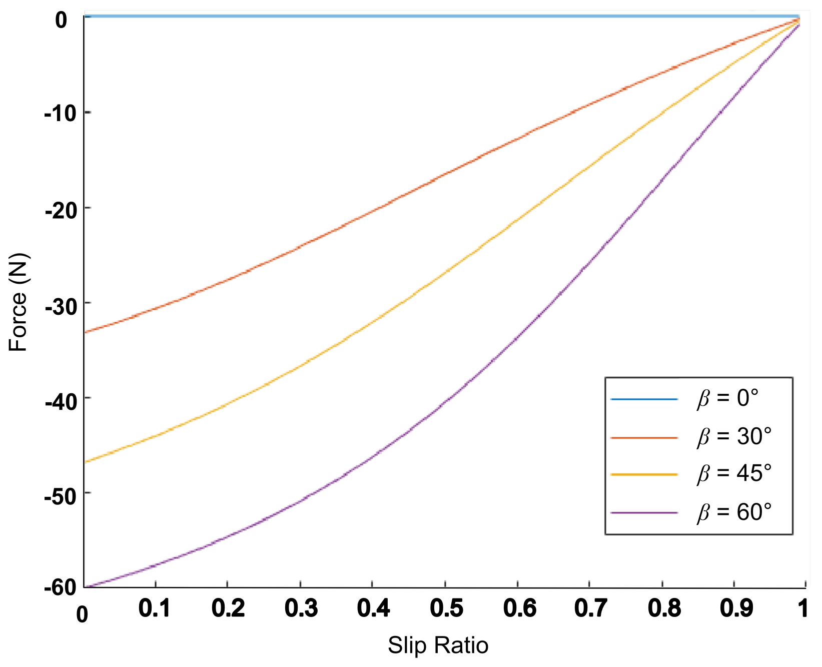

Figure 6. Tire Reaction Force in the Y Direction by Slip Ratio at Various Slip Angles.

The stiffness and damping coefficients were defined in Equation (9) where r was the radius of the sphere and α was a constant.

$$K = -2E^{*}\sqrt{r}, D = 4\pi\alpha \tag{9}$$

The spatial velocity vector for F1, the main body reference frame, was given by Equation (10). This vector was comprised of the angular and translational velocities represented by ω and v respectively.

$$\begin{bmatrix}\omega_{1x} & \omega_{1y} & \omega_{1z} & v_{1x} & v_{1y} & v_{1z}\end{bmatrix} \tag{10}$$

Equation (11) was the velocity of each of the wheels. The motion transformation from F_1 to F_i was given by ${}^i\mathbf{X}_1$, the subspace matrix of each wheel was $\mathbf{S}_i$, and the angular velocity was $\dot{q}_i$.

$$\mathbf{v}_i =^i \mathbf{X}_1\mathbf{v}_1 + \mathbf{S}_i\dot{q}_i \tag{11}$$

4. Deep Deterministic Policy Gradients

The DDPG is an actor-critic reinforcement learning algorithm and has separate neural networks for actor and critic functions. In this section, multi layered neural networks, as shown in Figure 7, are introduced including policy function μ and critic function Q.

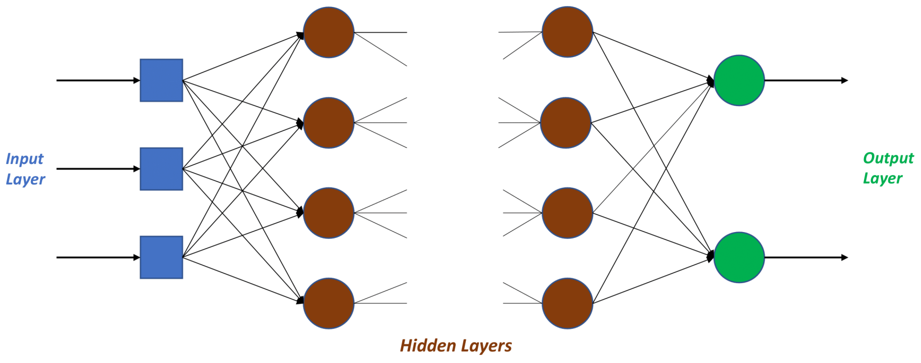

Figure 7. Artificial Neural Network.

4.1. Fully Connected Layer

A fully connected layer can be represented by

$$y = \sigma(w^T x + b) + \varepsilon \tag{12}$$

where w^T is the weights matrix and b is the bias. x is the input to the layer and y is the output. The Rectified Linear Unit (ReLU) was chosen as the activation function and ε was the error function. The entire network was made of multiple fully connected layers and it can be shown from the universal approximation theorem [12] that

$$\|\varepsilon\| \leq b_1, b_1 \in Const \tag{13}$$

4.2. Activation Function

The simplest activation function is referred to as linear activation, where no transform is applied at all. A network comprised of only linear activation functions is quite easy to train but cannot learn complex mapping functions. Nonlinear activation functions are preferred as they allow the nodes to learn more complex structures in the data. Traditionally, two widely used nonlinear activation functions are the sigmoid and hyperbolic tangent activation functions.

The nonlinear activation functions, such as sigmoid and tanh, suffer from the problem of vanishing gradients. Layers that are deep in large networks using nonlinear activation functions do not receive useful gradient information. The error is back propagated through the network and used to update the weights. The amount of error decreases dramatically with each additional layer through which the gradients are propagated, given the derivative of the chosen activation function. This is called the vanishing gradient problem that prevents deep (multi-layered) networks from learning effectively. ReLU activation helps deal with the vanishing gradients problem and enables the development and training of deep neural networks. The ReLU function is a piece wise linear function that will output the input directly if is positive, otherwise, it will output zero, as seen in Figure 8. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

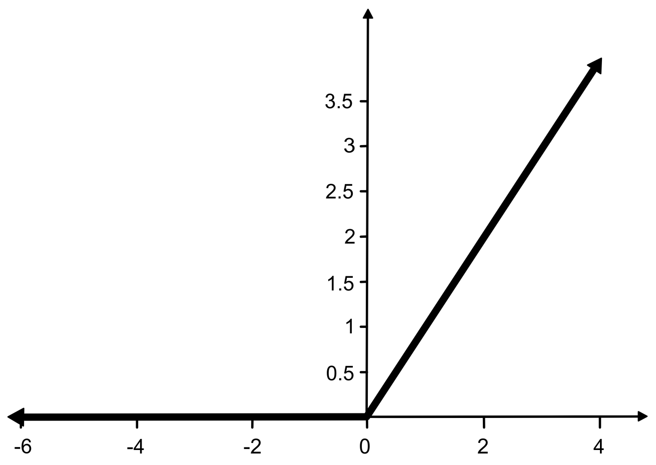

Figure 8. Rectified Linear Unit(ReLU) activation function.

4.3. Control Policy

The control policy is represented by the following equation:

$$\mu(s_t) = a_t \tag{14}$$

where s_t is the state at time t and a_t is the action calculated by the policy function μ at time t.

The long term reward function was calculated by using the equation

$$R(s_t, a_t) = \int_t^\infty \gamma^{-(\kappa - t)} r(s_t, a_t) d\kappa \tag{15}$$

γ is the discount factor ($0 < \gamma < 1$), which was used to discount the future rewards with respect to current reward. The problem was to find $\arg\min_{s_t, a_t}\{R(s_t, a_t)\}$. To solve the above problem, the critic function was defined as

$$Q(s_t, a_t) = R(s_t, a_t) = \int_t^\infty \gamma^{-(\kappa - t)} r(s_t, a_t) d\kappa \tag{16}$$

An iterative approach was used to calculate the critic function, discretizing the above equation to obtain

$$Q(s_t, a_t) = R(s_t, a_t) = \textstyle\sum_{i=t}^{\infty} \gamma^{(i-t)} r(s_t, a_t) \tag{17}$$

and the optimal critic function was

$$\hat{Q}(s_t, a_t) = \max_{s_t, a_t}(Q(s_t, a_t)) \tag{18}$$

The critic function $Q(s_t, a_t)$ was replaced with a neural network

$$Q(s_t, a_t | \omega) \tag{19}$$

In order to obtain an optimal critic function, the loss function was defined as

$$Loss = \frac{(y_t - Q(s_t, a_t | \omega))^2}{2} \tag{20}$$

$$y_t = r(s_t, a_t) + \gamma Q(s_t, a_t | \omega) \tag{21}$$

The optimal critic function $\hat{Q}(s_t, a_t)$ was found by minimizing the value of the Loss function.

Thus, the policy gradient algorithm [13] was applied by sampling a batch of (s_t, a_t) and computing the average Loss function.

$$Loss_{avg} = \frac{1}{N}\sum_{i=1}^{N}(y_i - Q(s_i, a_i|\omega))^2 \tag{22}$$

The gradient of Loss was given by

$$\nabla_\omega Loss_{avg} = -\frac{2}{N}\sum_{i=1}^{N}(y_i - Q(s_i, a_i|\omega))\frac{\partial Q(s_i, a_i|\omega)}{\partial \omega} \tag{23}$$

The gradient descent method was used to update the weights

$$\omega_{t+1} = \omega_t + \alpha \cdot \nabla_\omega Loss_{avg} \tag{24}$$

where α is the learning rate which can be tuned according to need.

4.4. Actor Function

After obtaining the critic function, it was used to update the actor function by propagating the gradients back to the actor. Then $\mu(s_t)$ was replaced using neural network $\mu(s_t|\theta)$ and substituting $a_t = \mu(s_t|\theta)$ into $Q(s_t, a_t|\omega)$.

$$J = Q(s_t, \mu(s_t|\theta)|\omega) \tag{25}$$

The gradient was given by

$$\nabla_\theta J = \frac{\partial Q(s_t, \mu(s_t|\theta)|\omega)}{\partial a_t} \cdot \frac{\partial \mu(s_t|\theta)}{\partial \theta} \tag{26}$$

Similar to the critic function, the gradient from the above equation can be used to update the actor function weights.

In order to update the actor and critic functions in a smooth fashion, the DDPG [4] algorithm creates a copy of actor and critic networks, $Q'(s_t, a_t|\omega')$ and $\mu'(s_t|\theta')$ and updates them gradually. The stopping criteria can decided by a set reward value or number of episodes during training.

5. Reinforcement Learning Setup and Training

Reinforcement learning consists of an environment and an agent. The agent performs an action in the environment which leads to a change in the state. The environment then supplies the next state and a scalar quantity called the reward.The reward is computed and provided as feedback to the agent. In this case, state information derived from vehicle position was used as an input to the agent, which outputs angular velocity. Both the inputs and outputs were continuous variables.

5.1. DDPG Agent Structure

The DDPG agent consists of two component neural networks called the actor and the critic. The actor network is the controller that maps the current state of the environment to angular velocity. The critic network takes the state of the environment and the corresponding actions predicted by the actor and outputs the Q value, or value of the corresponding state-action pair.

5.1.1. Actor

The actor network, provided in Figure 9, was made of two intermediate layers with 128 and 64 neurons each, and the final output layer had 1 neuron (2 neurons if linear velocity control was enabled). ReLU activation was used for the outputs of intermediate layers and the final layer was a hyperbolic tangent (i.e., tanh activation to limit the action output of the controller).

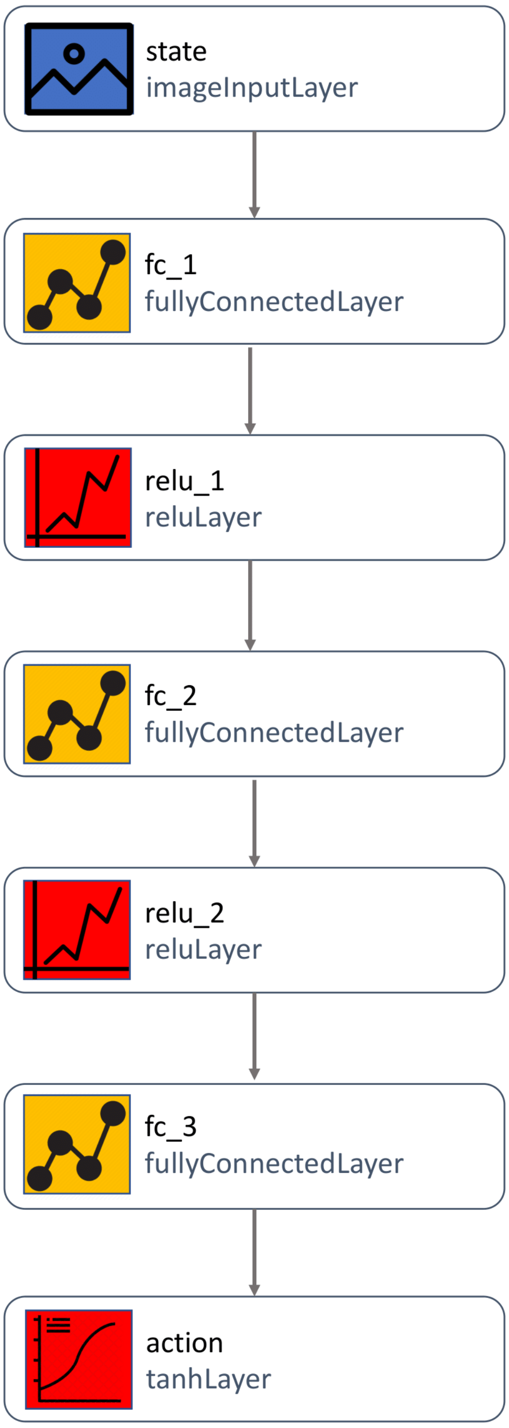

Figure 9. Actor network architecture.

5.1.2. Critic

The critic network, as illustrated in Figure 10, was made of two intermediate layers with 128 and 64 neurons and the final output layer had 1 neuron . ReLU activation was used for the outputs of intermediate layers. The critic network uses the reward values given by the environment and the Hamilton Jacobi Bellman (HJB) equation to calculate target Q values for training.

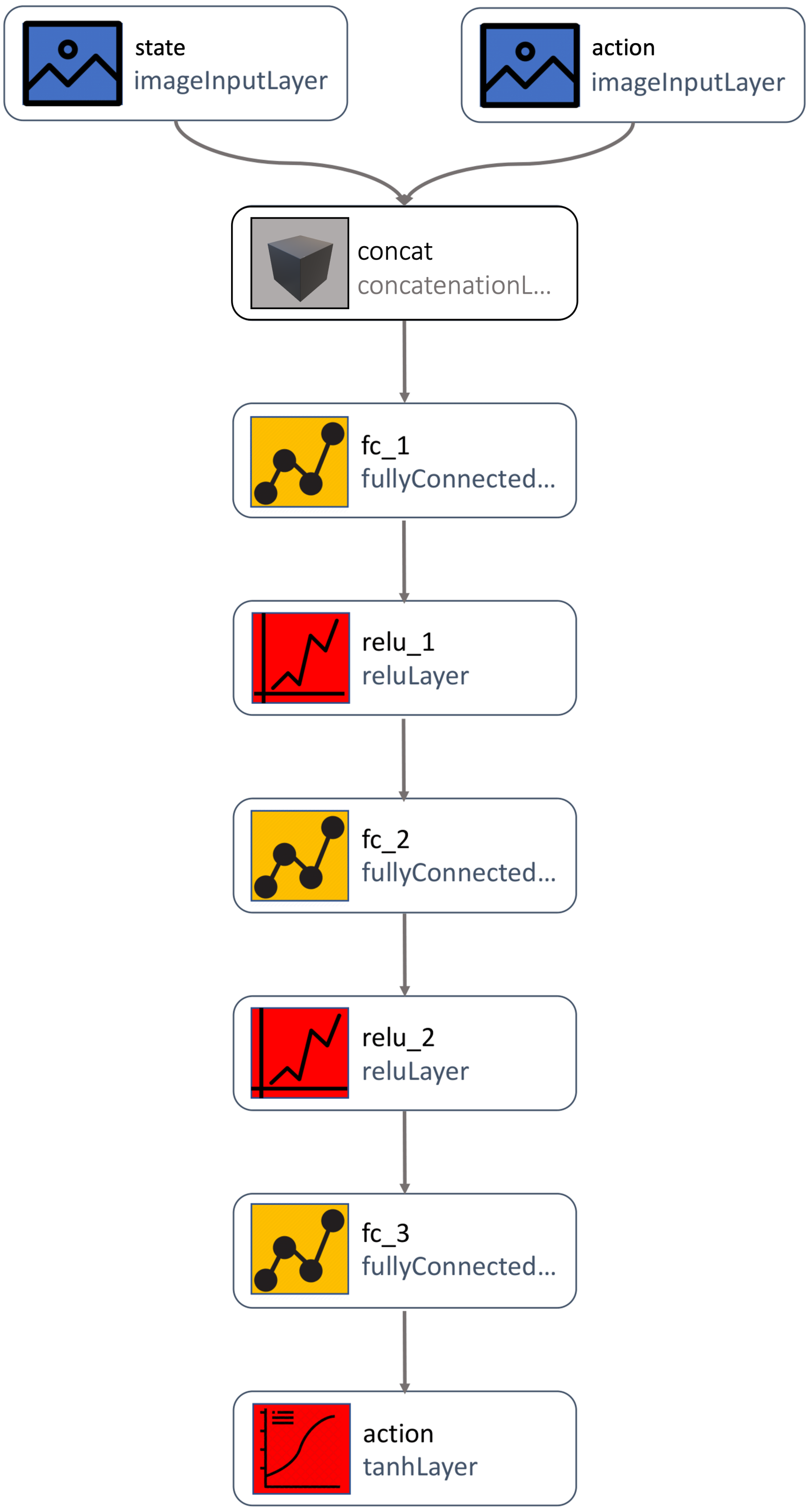

Figure 10. Critic network architecture.

5.2. State

Information regarding the state of the vehicle with respect to the trajectory was supplied to the actor and critic neural networks to enable them to act upon the state. State observations consisted of a 10-element array.The states provided were as follows:

1. Distance Error (DE);
2. Derivative of distance Error;

3. Heading Error (HE);
4. Distance to Look-Ahead Point 1 (D_{lp1});
5. Angle to Look-Ahead Point 1 (θ_{lp1});
6. Distance to Look-Ahead Point 2 (D_{lp2});
7. Angle to Look-Ahead Point 2 (θ_{lp2});
8. Distance to Look-Ahead Point 3 (D_{lp3});
9. Angle to Look-Ahead Point 3 (θ_{lp3});
10. Angular Velocity of the Vehicle.

State information was calculated from the vehicle position, heading and waypoint positions. The information was acquired from the reference trajectory and sensors on-board the vehicle. State information provided to the agent was broken down into three categories: error information, predictive action information, and angular velocity. Figure 11 provides a visual description of the state variables supplied to the agent.

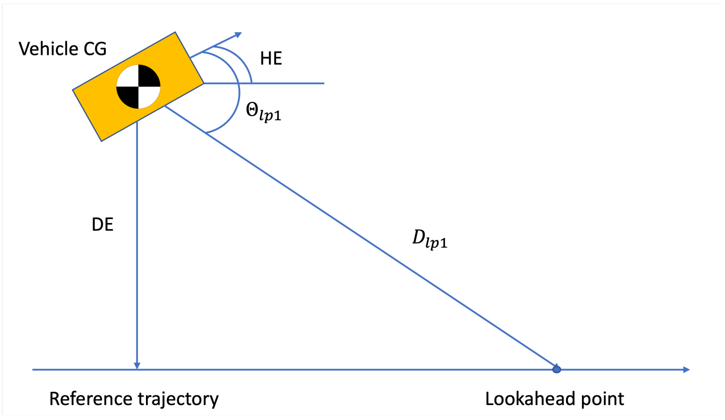

Figure 11. State variable description.

5.2.1. State Information—States 1–3

The error information consisted of distance error, derivative of distance error and heading error values, so that the agent could learn to associate these errors with action outputs. The derivative of heading error was not supplied to the agent because of the discontinuities in the derivative near the waypoints. These discontinuities introduced instability in the training process, so the agent was supplied with angular velocity. This provided similar information without the spikes and stabilized the training.

5.2.2. State Information—States 4–9

The predictive action information was supplied to the agent to encourage predictive turning behaviors near a sharp turn to reduce overshoot and hence minimize overall tracking error. A look-ahead point was defined as a point located at a specified constant distance down the path from the point on the path perpendicular to the vehicle's current position. The angle to the look-ahead point was the angle between the vehicle's current heading and the line drawn between the vehicle and the look-ahead point. The intention of the look-ahead point was to provide the controller with information describing the test course that lay ahead of the vehicle's current position. Three look-ahead points were used with specified distances of 1, 2 and 3 m.

5.2.3. State Information—State 10

There was a difference between commanded angular velocity (ω_c) and actual angular velocity (ω_a). Even though the agent had the information about predicted output angular velocity, the internal motor controllers took a finite amount of time to converge to the new angular velocity. This introduced a time delay between the predicted and actual output velocities leading to a difference in predictions about vehicle motion. The goal of supplying this state information was to create a smarter controller that learned to map its target angular velocity to its actual angular velocity which was then mapped to desired actions in the environment.

5.3. Rewards

The reward/penalty functions were used to evaluate performance and provide feedback during training. The outputs of these functions were summed to a scalar reward value and supplied to the agent (penalty functions are simply negative reward functions). This reward value, in addition to state information, enabled the DDPG algorithm to evaluate the effectiveness of actions and train the neural networks.

5.3.1. Distance Error Penalty

The Distance Error (DE) Penalty function was very important for the training of agents because it directly corresponded to the primary objective of reducing distance error. As the agent strayed further away from the current trajectory, it accrued more DE Penalty which forced the agent to begin creating a policy that would avoid this behavior.

5.3.2. Heading Error Penalty

As heading error (HE) increased, so did the HE Penalty, which allowed the agent to create a policy that minimized heading error. In addition, there was a relationship between the magnitudes of the HE Penalty and distance error. The HE penalty was scaled down when distance error increased. This relationship was helpful in the scenario where the vehicle found itself off track. The vehicle needed to accommodate some heading error as it finds its way back to the track and continues its mission.

5.3.3. Steering Penalty

The steering penalty was implemented to reduce unnecessary turning. A small penalty was added that was proportional to the absolute value of angular velocity. An added benefit of the steering penalty was that it helped combat circular behavior. The steering penalty dominated distance and heading error penalties when the vehicle was close to waypoints. Large distance and heading error penalties overpowered the smaller steering penalty causing the vehicle to follow the next path segment. Like the HE Penalty, the Steering Penalty was related to distance error and it was scaled down as distance error increased.

$$DE\ Penalty = -K_d(|E_d| + |E_d|^{s_d}) \tag{27}$$

$$HE\ Penalty = -K_h \frac{(\frac{|E_h|}{\pi})^{s_h}}{e^{R_h|E_d|}} \tag{28}$$

$$Steering\ Penalty = -K_\omega \frac{|\omega|}{e^{R_\omega|E_d|}} \tag{29}$$

The reward scalar supplied to the agent was the sum of the values produced by Equations (27)–(29). Modifying the values of constants changed what the controller perceived as good and bad behavior. This affected the behaviors acquired during the training process. Penalties were scaled by the gains (K_d, K_h, K_ω). E_d and E_h were the distance error and the heading error, respectively. The magnitudes of the gains in proportion to one another enabled the Deep Neural Network (DNN) Controller to prioritize the behavior. Penalty shapes (S_d, S_h) determined the rate of increase in penalties as their corresponding error increased. Figures 12 and 13 demonstrate how varying penalty values affect the unscaled distributions according to the corresponding error.

As simulations were terminated when distance error surpassed 1 m, the distribution of the distance error penalty was bounded (from 0 to 1). The distribution of heading error penalty was bounded between 0 and π because the absolute value of heading error could not exceed $\pi(-\pi < HE < \pi)$. The distance error relationship coefficients were used to prioritize the vehicle to minimize the distance error rather than directly chase after the next waypoint in a pure-pursuit-like manner. If the vehicle had an overshoot near a turn, it would prioritize getting back to the track and then continuing its mission.

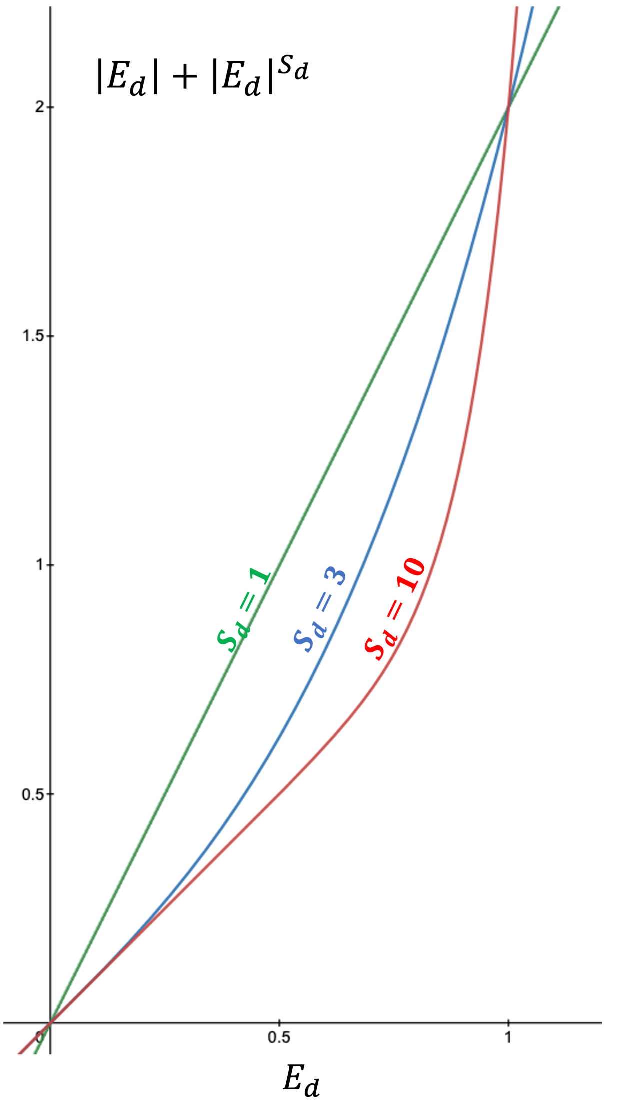

Figure 12. Distance Error penalty shapes.

These relationships were intended to discourage turning when the DE was small but to allow the vehicle to return to the track when the DE was large by not penalizing HE when the vehicle was far from the track. An exponential function was used in Equations (28) and (29) to determine the relationship between DE and other reward/penalty functions. When DE = 0, the exponential function was equal to 1 and the reward/penalty function was unaffected. However, as the absolute value of DE strayed from 0, the exponential function increased, with the corresponding decrease in reward/penalty. The DE Relationship Coefficients (R_h, R_ω) dictated the magnitude of decrease in reward/penalty with respect to DE.

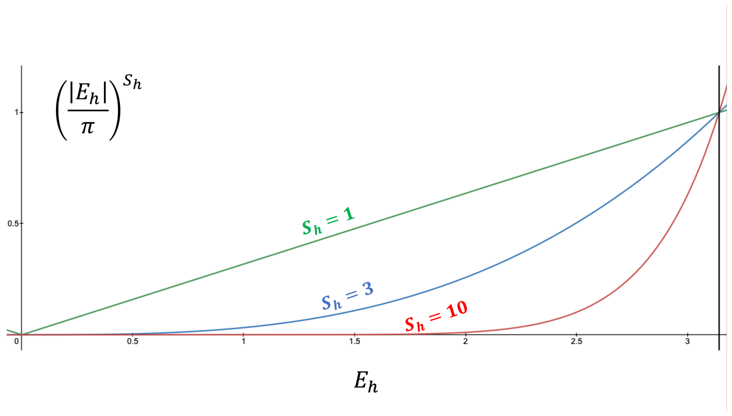

Figure 13. Heading Error penalty shapes.

5.4. Training and Reinforcement Learning Setup

The MATLAB Reinforcement Learning Toolbox [14] was used to implement the DDPG algorithm. Once the dynamic model of the robot was built in Simulink, the models were converted into an environment compatible with the toolbox. The network architecture for actor and critic were specified using the Deep learning Toolbox as described in Section 4.1. The DDPG agent can be constructed from actor and critic networks after specifying the learning rates and losses for individual networks. A mean squared error loss function was implemented.

The Reinforcement Learning (RL) toolbox was used to specify and monitor the training process with built in functionality to save and load trained agents. The training was conducted for 1000 episodes. Each episode was terminated when the agent's distance error from the reference trajectory was above the threshold value 1 m or if the agent had successfully completed the waypoint trajectory. Learning rates of 0.02 and 0.01 were chosen for the critic and the actor, respectively, based on experimentation. Various values were experimented with to improve the training time and maintain the stability of the learning process. A discount factor of 0.95 was used to discount future rewards as described in Section 3. The replay buffer size was set to 10^5 and the minibatch size used for training was set to 32.

The trained agent was saved in the workspace and the policy was extracted from the agent for evaluation. In order to deploy the trained policy on the Jackal, a feed forward function was implemented to take the inputs coming from the robot and compute the control outputs using the weights and biases of the policy. The communication with the vehicle hardware was performed with the Robot Operating System(ROS) using various ROS messages and topics.

6. Results

The trained policy was evaluated on various trajectories in simulation to identify the performance measurements and errors. The test trajectories were based on the system level test plan described in [15]. The objective of the controller was to stay within a distance of half the vehicle width (0.22 m) from the trajectory.

In Figure 14, the vehicle started with an initial distance error of 1 m from the reference trajectory and zero heading error. It was able to quickly converge back to the reference trajectory within a distance of 1 m along the path.

Figures 15–17, demonstrate that the vehicle was able to track trajectories with sharp turns of varying angles while maintaining the original objective of staying within the distance of 0.22 m. In Figure 17, it can observed that it takes a distance of two vehicle lengths to reacquire the trajectory due to a large turning angle of 120°.

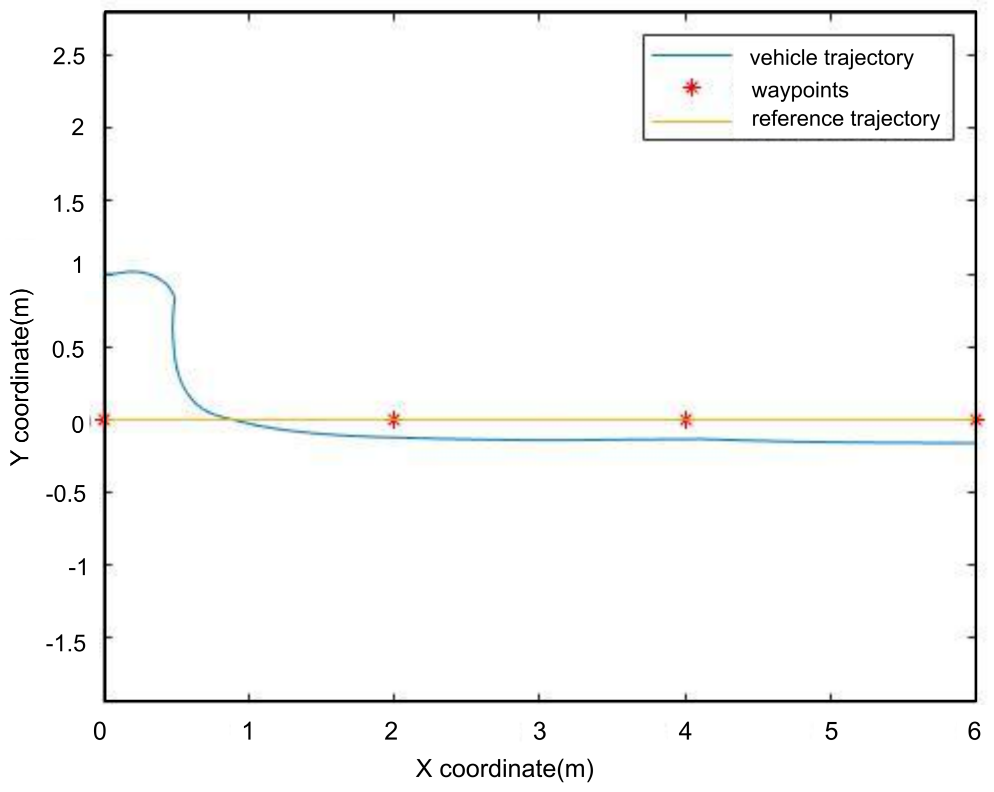

Figure 14. Convergence from offtrack starting position.

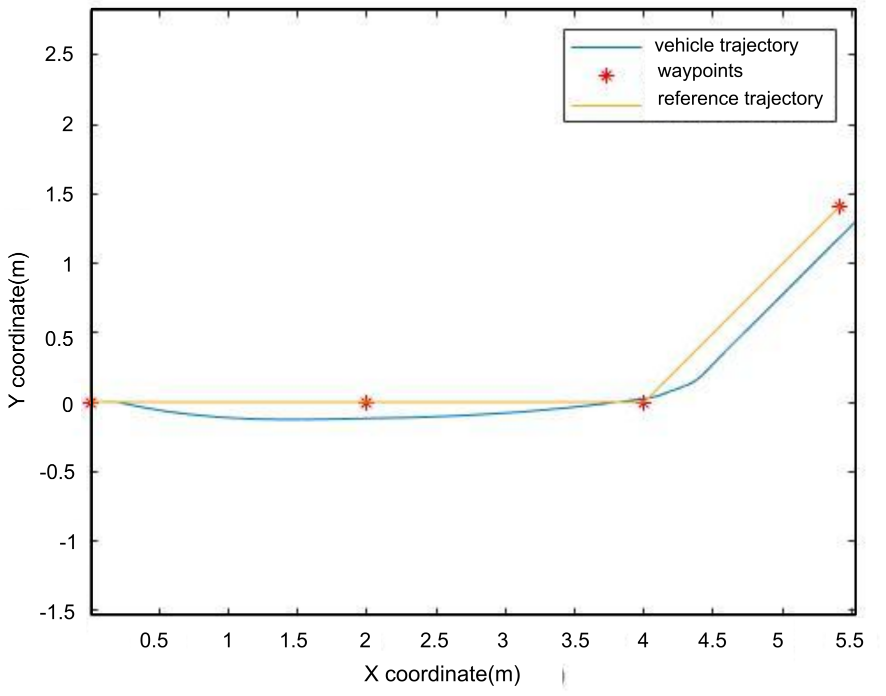

Figure 15. Controller following a 45 degree turn.

Figures 18–20 demonstrate that the vehicle was able to complete relatively complex trajectories with straight line segments combined with consecutive left and right turns while staying within the error bounds specified for the task. A finite phase lag was observed in Figures 18 and 20. There was a consistent undershoot that can observed in the second half of the Figure 8 path in Figure 20. These behaviors were due to the vehicle turning early near the end of a path segment. This was to account for future path segments data provided through the lookahead points. Table 1 provides a summary of RMS distance errors for various trajectories discussed in this section.

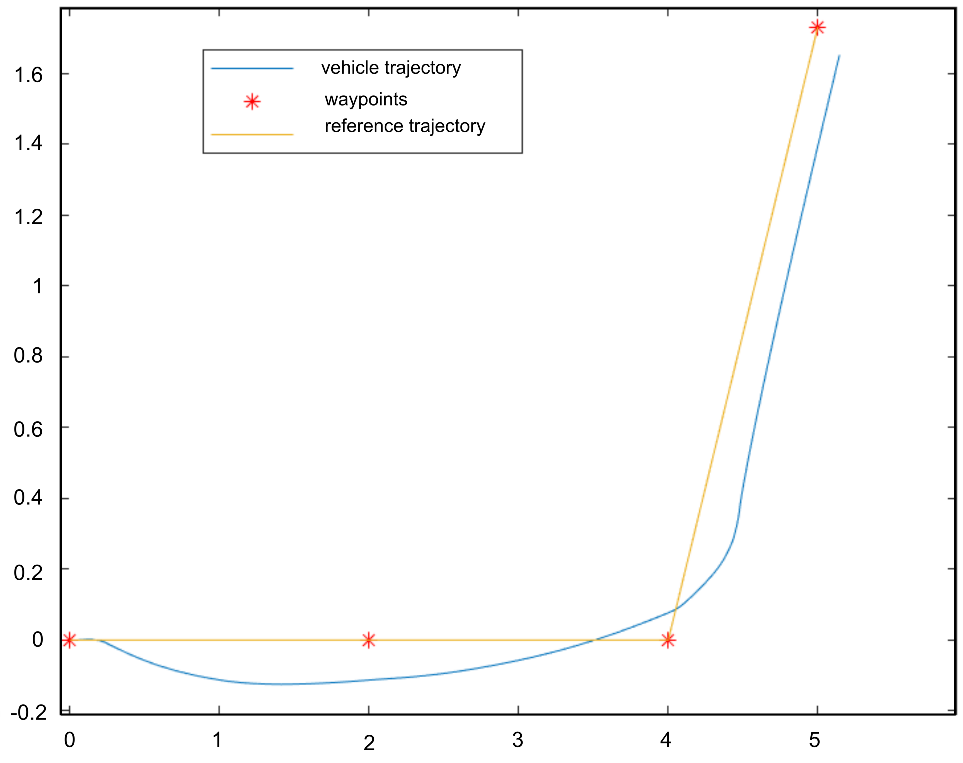

Figure 16. Controller following a 60 degree turn.

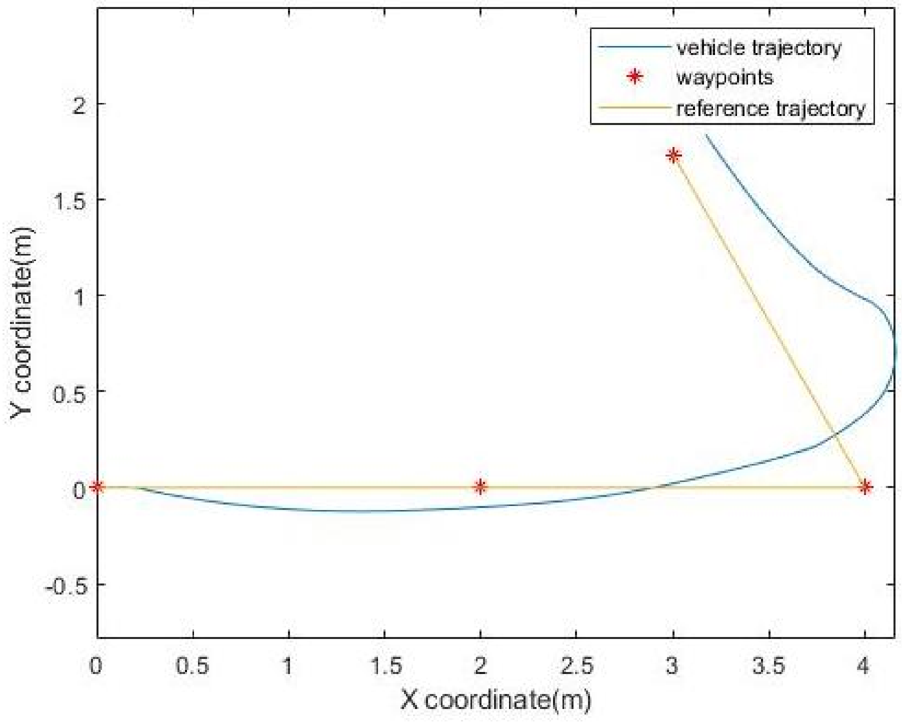

Figure 17. Controller following a 120 degree turn.

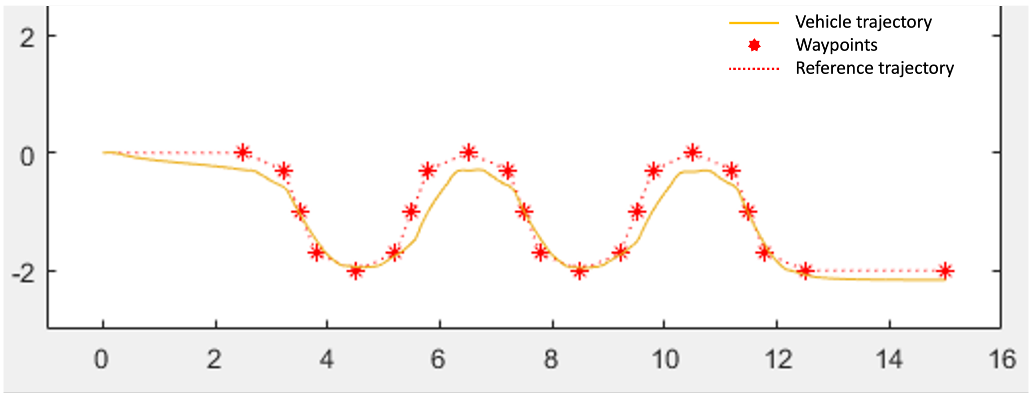

Figure 18. Controller following testcourse 3.

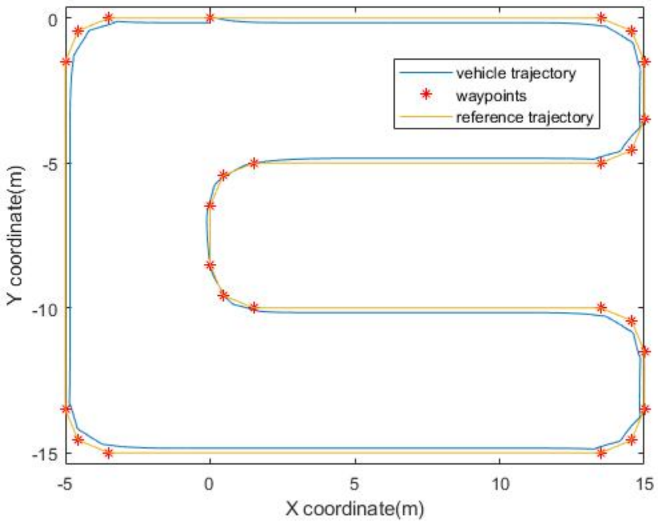

Figure 19. Controllerfollowing testcourse 2.

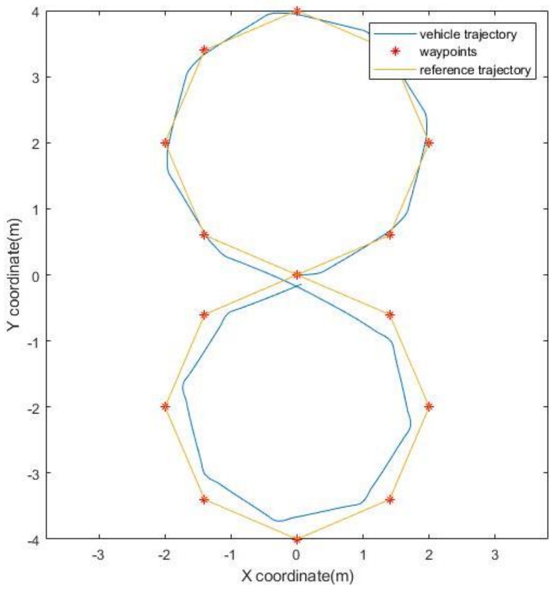

Figure 20. Controller following Figure 8 path.

The performance of the controller deteriorated with larger turn angles as it needed to deviate from the trajectory to accommodate for the next segment as can be seen in Figure 17 with a 120 degree turn which is expected for a steering controller that does not control linear velocity. Even though the vehicle is able to track the trajectory, the performance could be further improved with more tuning and training.

Table 1. RMS errors for various test courses.

Trajectory	RMS Error (m)
45° turn	0.08063
90° turn	0.1183
120° turn	0.2
testcourse 3	0.1674
testcourse 2	0.1629
Figure 8 path	0.159

7. Conclusions and Future Work

A continuous control reinforcement learning algorithm was implemented to train a path tracking controller for the dynamic model of a skid-steered vehicle. The controller was able to adapt effectively to the complex nonlinear forces and torques acting at the wheel-ground interface. The trained controller's performance was demonstrated on a series of complex trajectories while maintaining the objective of keeping the vehicle within half the vehicle width of the desired trajectory. The convergence of vehicle motion towards the desired trajectory from a variety of initial conditions was also demonstrated.

The model based development approach enables the extension of this work to other vehicles of varying size and complexity without major changes to the development tools and framework. Using the code generation capabilities of MATLAB, the trained agents can be deployed directly onto the real vehicle without spending significant amounts of time coding the controllers in other programming languages.

The performance of the trajectory tracking controller can be improved by adding linear velocity control in addition to angular velocity. This would help the system better navigate sharper turns and ensure the convergence of the vehicle towards the trajectory in case of initial off-track errors.

This approach was successful but it has some limitations. The limitations of this approach included the need for an accurate simulation model for training and long training times for the reinforcement learning algorithm to explore and learn useful behaviors. Even though the model takes into account the effect of nonlinear forces and torques, some aspects of the model were lacking when compared to the real-world robot. More importantly, a neural network path tracking controller does not guarantee asymptotic or globally asymptotic stability over the operational range. This makes it difficult to deploy the controllers in safety critical applications where the autonomous robots are operating along with or near human operators.

There are other control schemes designed to satisfy stability conditions such as human operator models which are nonlinear and classical state space optimal controllers. Many of the state space nonlinear controllers suffer from instability outside their linearized operating region. A logical step for future work would be to combine some of these techniques to bound the exploration process of the RL agent with constraints or leverage the advantages of deep neural networks for tuning controller models that have stability guarantees.

Author Contributions: S.S. implemented the reinforcement learning algorithm on the skid-steer vehicle and tested the algorithm. W.R.N. conceptualized the test plan and research that would be implemented while providing supervision and direction for the research. D.N. and A.S. provided project supervision, direction, and funding. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the U.S. Army Engineer Research and Development Center (ERDC) through their Construction Engineering Research Laboratory (CERL).

Institutional Review Board Statement: Not applicable

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Sample Availability: Samples of the compounds are available from the authors.

References

1. Huskic, G.; Buck, S.; Zell, A. Path following control of skid-steered wheeled mobile robots at higher speeds on different terrain types. In Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 29 May–3 June 2017; pp. 3734–3739.
2. Rajagopalan, V.; Meriçli, Ç.; Kelly, A. Slip-aware Model Predictive optimal control for Path following. In Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 16–21 May 2016; pp. 4585–4590.
3. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Joel, V.; Marc, G.B.; Alex, G.; Martin, R.; Andreas, K.F.; Ostrovski, G.; et al. Human Level Control through Deep Reinforcement Learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
4. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *ICLR* **2015**. [CrossRef]
5. Everett, M.; Chen, Y.F.; How, J.P. Motion Planning among Dynamic, Decision-Making Agents with Deep Reinforcement Learning. In Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 1–5 October 2018; pp. 3052–3059.
6. Mirowski, P.; Grimes, M.; Malinowski, M.; Hermann, K.M.; Anderson, K.; Teplyashin, D.; Simonyan, K.; Zisserman, A.; Hadsell, R. Learning to navigate in cities without a Map. *Adv. Neural Inf. Process. Syst.* **2018**, 2419–2430. [CrossRef]
7. Nazari, V.; Naraghi, M. Sliding mode fuzzy control of a skid steer mobile robot for path following. In Proceedings of the 2008 10th International Conference on Control, Automation, Robotics and Vision, Madeira, Portugal, 17–20 December 2008; pp. 549–554.
8. Sheikhlar, A.; Fakharian, A. Adaptive optimal control via reinforcement learning for omni-directional wheeled robots. In Proceedings of the 2016 4th International Conference on Control, Instrumentation, and Automation (ICCIA), Qazvin, Iran, 27–28 January 2016; pp. 208–213.
9. Fan-Cheng, M.; Ya-Ping, D. Reinforcement learning adaptive control for upper limb rehabilitation robot based on fuzzy neural network. In Proceedings of the 31st Chinese Control Conference, Heifei, China, 25–27 July 2012; pp. 5157–5161.
10. Lee, D.; Choi, M.; Bang, H. Model-free linear quadratic tracking control for unmanned helicopters using reinforcement learning. In Proceedings of the 5th International Conference on Automation, Robotics and Applications, Wellington, New Zealand, 6–8 December 2011; pp. 19–22.
11. Tang, S.; Yuan, S.; Li, X.; Zhou, J. Dynamic modeling and experimental validation of skid-steered wheeled vehicles with low-pressure pneumatic tires on soft terrain. *Proc. Inst. Mech. Eng. Part D J. Automob. Eng.* **2020**, *234*, 840–856. [CrossRef]
12. Kurt, H.; Stinchcombe, M.; White, H. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Netw.* **1990**, *3*, 551–560.
13. Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; Riedmiller, M. Deterministic policy gradient algorithms. In Proceedings of the 31st International Conference on Machine Learning, Bejing, China, 22–24 June 2014.
14. The MathWorks, Inc. *MATLAB and Reinforcement Learning Toolbox Release 2019b*; The MathWorks, Inc.: Natick, MA, USA, 2019.
15. Norris, W.; Patterson, A. System-Level Testing and Evaluation Plan for Field Robots: A Tutorial with Test Course Layouts. *Robotics* **2019**, *8*, 83. [CrossRef]