Reinforcement Learning Report

AUTONOMOUS DRIFTING RC CAR WITH REINFORCEMENT LEARNING

May 9, 2018

Supervisor: Dr. D. Schnieders
Sourav Bhattacharjee (3035123796)
Kanak Dipak Kabara (3035164221)
Rachit Jain (3035134721)

Written by Sourav Bhattacharjee

Abstract

The advent of self-driving cars has pushed the boundaries on the safety of automobiles, but most modern self-driving car systems ignore the possibility of a car slipping resulting from inclement weather or driver error [1]. Passengers and bystanders would benefit heavily if self-driving cars could handle slipping by learning to drift with the turn rather than against it (by applying the brakes, or turning away, which is the instinctive action), preventing many fatalities [2].

Our project is aimed at studying the drifting of an autonomous remote controlled (RC) car using reinforcement learning (RL) techniques. Specifically, we experimented with a model-free approach with dueling double Deep Q-networks (DQN) and a model-based approach with Probabilistic Inference for Learning COntrol (PILCO) for finding an optimal drift controller. Since robotic systems are prone to wear with use, a simulator is used to model the car dynamics and train a preliminary drift controller which is then transferred to the real car.

Using these techniques, we were successful in obtaining an optimal drift controller on the simulator, which was stable and robust to varying physical conditions. Other than the drift controller, this project makes important contributions in the form of novel approaches like using DQN for obtaining a drift controller and using the policy learned from DQN for PILCO initialization. Additionally, this report presents a metric, D_m, to objectively quantify the quality of a sustained circular drift.

Acknowledgement

We would like to thank our supervisor, Dr. Dirk Schnieders, for guiding us throughout the project. We are also grateful for the help received from a friend, Mr. David Ip, who helped us acquire the hardware needed for this project. Finally, we are thankful for the help we received from Dr. Chris R. Roberts with various hardware issues encountered.

Contents

List of Tables

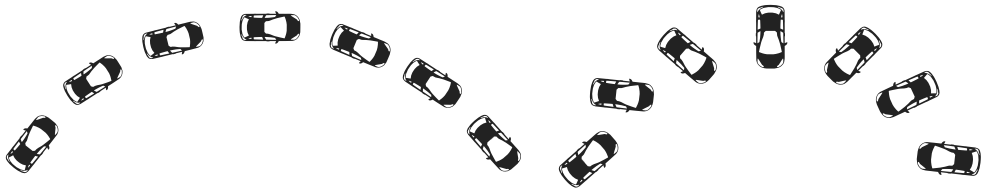
List of Figures

Abbreviations

Abbreviation	Meaning
2WD	Two-wheel drive
4WD	Four-wheel drive
DQN	Deep Q-networks
L-BFGS	Limited-memory Broyden–Fletcher–Goldfarb–Shanno
PILCO	Probabilistic Inference for Learning COntrol
RBF	Radial basis function
RC	Remote Controlled
RL	Reinforcement Learning

1 Introduction

Before discussing the implementation details of the project, it is crucial to understand the background of the problem we are trying to solve, and the actual scope. This section addresses that and highlights the need to study and solve the problem of drifting, and outlines how we plan to do so with an approach based on simulation aided reinforcement learning. For the purposes of this report, drifting is defined as the oversteering of a car which results in the loss of traction of the rear wheels. This results in the front wheels pointing in the opposite direction to the turn and the car appears to be moving sideways as shown in Figure 1.



(a) Steering around bend **(b) Drifting around bend**

Figure 1: Drifting is defined as the oversteering of a car which results in the loss of traction of the rear wheels. This results in the front wheels pointing in the opposite direction to the turn and the car appears to be moving sideways. The diagram illustrates the difference between simply turning around a bend and drifting around a bend.

1.1 Background and Motivation

Passenger vehicles usually implement stability control in a number of ways like differential braking [3], active steering [4] [5] or integrated chassis control [6] [7] [8]. Other methods, based on independent wheel torque, have also been developed to make passenger vehicles more stable. However, these methods function by making sure that the tires avoid slipping. In doing so, these methods essentially restrict the operation of the vehicle. Similarly, control algorithms in current self-driving car systems (Anti-lock brake systems, Electronic stability control etc.) try and mitigate the chances of slipping due to its unpredictable nature [1]. Sufficiently lowering the speed of the car and making turns that are not too tight will mostly

prevent slipping, but this does not consider cases where the system must make evasive moves to avoid crashes or when a car is already in a slipping state due to the driver's fault. For example, hydroplaning, which refers to a situation where a layer of water builds up between the car tires and the road, is a major reason for vehicle accidents. According to the United States' Department of Transportation, 15.97% of all vehicle crash fatalities in the United States [2] are attributed to wet and icy roads. An autonomous car system should be prepared for the scenarios outlined above to ensure the safety of the passenger and bystanders, regardless of the weather conditions or the state of the car. To reduce fatalities and ensure that these car systems are as robust and safe as possible, it is essential to study drifting, and eventually deduce how cars can respond quickly to unintentional slipping states as those encountered due to hydroplaning. Not only can drifting be useful to steer out of these unintentional slipping states, but can also be useful in taking full advantage of the capabilities of a vehicle to avoid accidents in emergencies.

Many of the systems discussed above try to tackle the issue of stability control and slipping by approaching it as an optimal control and open looped problem with explicit dynamics model. Approaches using optimal control are often deterministic and use closed-form expressible equations of motions. The resulting policies depends entirely on the model used to compute them. Sometimes, these restrictions on the model neglect parts of the true system either because they are non-linear or they are just not well-enough understood to be expressed in equations. We thus propose a method that does not rely on explicit equations of motion, but rather on an implicit understanding of the world obtained by trial and error.

Another motivation to study drifting is that Paul Frère [9] points out the usefulness of drifting to turn fast around sharp bends. Since high-speed stability is of greater importance to ordinary touring vehicles and competition cars, they tend to understeer, and for a competition car average circuit time is improved by going fast through fast bends while slowing down through sharp ones [9]. However, a car is able to turn sharp bends faster by drifting because the yaw angle formed by the drift brings the vehicle in line with the straight path following the bend even before the vehicle completes the turn [9].

1.2 Objective

The objective of this project is to get a remote controlled car to maintain a sustained circular drift autonomously. This paper proposes a framework for learning the best way to drift using simulation aided reinforcement learning which is one approach to solving the problem without having to input the dynamics of the system explicitly. Then the project aims to transfer the learned optimal drift policy or strategy from the simulation to a physical RC car for further learning.

1.3 Scope

The area of drifting falls into two categories – sustained drift and transient drift. Due to the wide breadth of the two categories and the time and cost constraints, our project will mainly focus on sustained drift, and more specifically steady state circular drift on an RC car with constant forward throttle. Additionally, despite the wide range of reinforcement learning algorithms available, due to reasons elaborated in the remaining of the report, we investigate two different algorithms to obtain the sustained circular drift controller - DQN and PILCO.

1.4 Deliverables

The complete implementation of the project is available on `https://github.com/kanakkabara/Autonomous-Drifting`. There are a few major deliverables in this project, which are outlined below:

1. Reinforcement Learning (RL) algorithms – Implementation of Double dueling Deep Q-networks for finding an optimal drift controller as well as model based policy search with PILCO.

2. Drift controller - A successful sustained circular drift controller along with tests to prove its robustness and stability.

3. Drift metric - A drift metric to objectively quantify the quality of a drift.

4. Simulator – We trained the RL algorithms on a simulated car that models the RC car in an environment with physics that mimic the real world. The

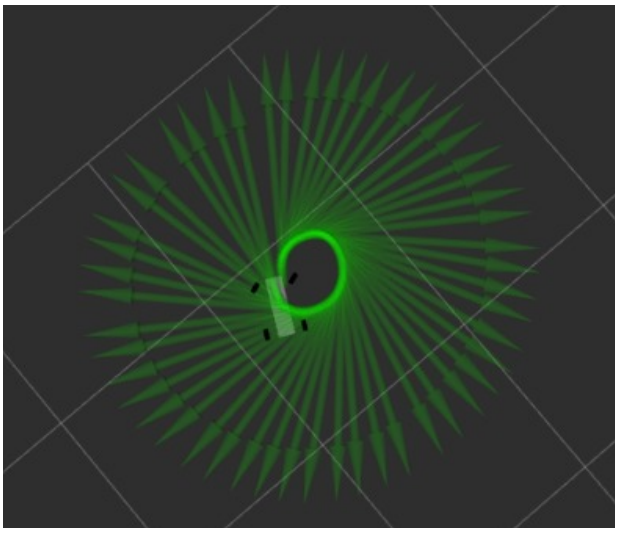


Figure 2: Time-lapsed path traced by the car on the simulator using the successful sustained circular drift controller.

environment was used to test and improve different RL algorithms efficiently and quickly, without causing any wear or damage to the RC car.

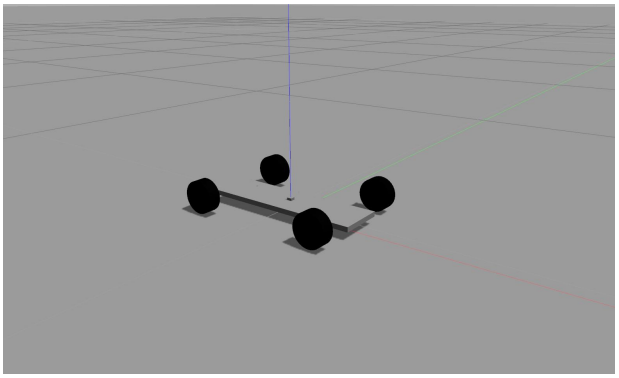


Figure 3: The final simulated car

5. A remote controlled (RC) car – This car is a 1/10th scale model of an actual car, integrated with sensors (Inertial measurement unit combined with magnetometer and optical sensors) for measuring data like the translational and angular velocities of the car to perform steady state circular drifting. The project then aims to transfer the optimal policy learned in simulator onto the RC car for validation.

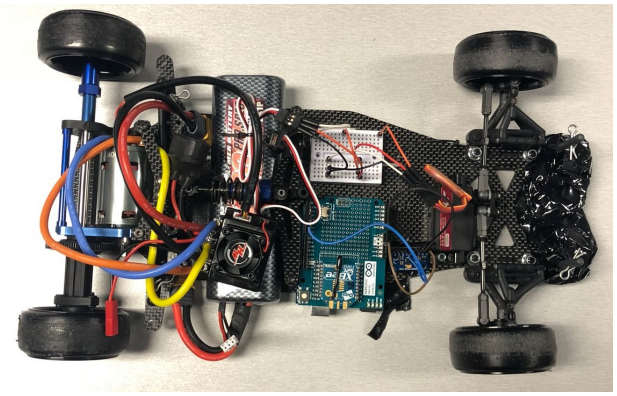


Figure 4: The final RC car

1.5 Contributions

The project introduces the following novel ideas, as elaborated further later in the report:

1. Using double dueling Deep Q-networks (DQN) to find an optimal drift controller.
2. Using policy learned from the DQN model to initialize PILCO learning.
3. A drift metric, D_m, to objectively evaluate a sustained circular drift:

$$D_m = \frac{1}{T}\sum_{t=0}^{T} \exp\left(-\frac{(||\boldsymbol{s}_t - \boldsymbol{s}_{target}||^2}{2\sigma^2}\right) \in [0,1]$$

1.6 Outline of Reports

The documentation for this project is divided into three reports. Although the reports share the same background and motivation behind the project, each emphasizes on the methodology, experiments, results and difficulties encountered for different aspects. A reader is thus suggested to refer to all three individual reports to acquire a complete understanding of the project. The three reports are as follows:

Report outlining the hardware, written by Rachit Jain, highlights the implementation of the RC car and the various challenges faced in indoor localization and velocity estimation.

Report outlining the simulator and communication, written by Kanak Kabara, describes the implementation of an RC car in a simulated environment. It also talks about the communication network connecting the various components of this project.

This report outlines the Reinforcement Learning Algorithms, and proceeds as follows. First, I will provide a literature review on the various methods that have been used to implement steady-state drifting. Next, a detailed description of the implementation of the DQN and PILCO algorithms will be provided. Then I will discuss the results obtained from the experiments performed. Finally, a metric is presented to objectively evaluate a sustained circular drift before concluding remarks.

2 Literature review

2.1 Optimal Control Approach to Autonomous Drifting

Sustained drift with various optimal control techniques has been explored through multiple prior research. For instance, Velenis et al. [10] described a simple single-track vehicle model using equations of motion to design a 'sliding' control policy to stabilize steady state conditions using basic acceleration/braking applied to the wheels. Similarly, Hindiyeh and Gerdes [11] developed an open-loop control policy using nested feedback loops to attempt stable drift equilibrium. They too developed a complex model of the vehicle, chassis and wheels to form the basis of their control policy. On the other hand, Wu and Yao [12] created a control algorithm to stabilize an RC drifting car by balancing the tail sliding with counter-steering measures to prevent slipping during circular motion. Their system is based on understanding the dynamics of the car, including the planar force and moment generated by the car's wheels during drifting. These modeled approaches work well in scenarios where the model encapsulates the various dynamics of the real-world, but do not work well when the dynamics of the world are not understood completely to be modeled by equations of motion. The open-loop approach of the optimization cannot be implemented in the presence of uncertainties [13]. Thus, a better approach, which is independent of the underlying models, is needed.

This is the perfect use case for learning-based methods, specifically Reinforcement Learning (RL). Since RL algorithms learn policies by directly interacting with the environment, the policies are dependent on the real-world instead of being reliant on our understanding of the world.

2.2 Reinforcement Learning Approach

Reinforcement learning techniques are employed in this project to learn an agent that maximizes the sum of expected future rewards [14] by interacting with their environment repeatedly. As illustrated in Figure 5, the agent interacts with the environment according to a policy by taking actions and evaluates how good or bad taking a particular action in a particular state (a_t) is by ob-

serving the next state it transitions to (s_{t+1}) and the reward it receives along the way in the next time step (r_t). A state space is all the possible states that an agent can experience in the environment at any particular time while an action space is the set of all possible actions an agent can take [14]. A policy is a function that maps from the state space to an action [14]. More concretely, a policy is a function $\pi : S \rightarrow a$, where S and a are the state space and an action in the action space respectively. If an action $\pi(s)$ is taken by an agent in state s, it is said that the agent is acting according to policy π. The goal of any reinforcement learning problem is to find a policy π that maximizes the expected sum of discounted future rewards (reward at state s is given by $r(s)$),

$$E_{s_0,s_1,\ldots} = \left[\Sigma_{t=0}\gamma^t r(s_t)|\pi\right] \tag{1}$$

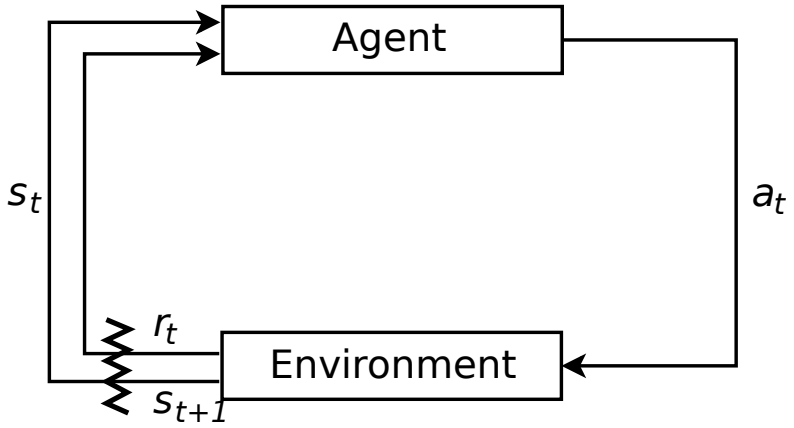


Figure 5: The diagram shows the architecture of the reinforcement learning framework. The agent interacts with the environment according to a policy by taking actions (a_t) and evaluates how good or bad taking a particular action in a particular state (s_t) is by observing the next state it transitions to (s_{t+1}) and the reward it receives along the way(r_t) [14].

2.3 Model-free Learning with Dueling Double DQN

In our project, one of the initial approaches used to find an optimal drift controller is Q-learning [14]. The reason Q-learning is chosen is because it is an off-policy learning algorithm and is model-free, which means the algorithm learns by "looking over someone else's shoulder" without being explicitly told the dynamics of the system. This allows it to explore the state space by using a stochastic behaviour policy, β, while converging to a deterministic optimal policy, π. The algorithm is represented with a Q-learning neural network. Let's say at time step t the state of the car is s, the action chosen according to the current policy is a and the next state the car ends up in after taking the action a is s_{t+1}. According to the online Q-learning algorithm, the target for the (s_t, a_t) pair is given by

$$y_t = R(s_t, a_t) + \gamma max_{a' \in A} Q_\phi(s_{t+1}, a_{t+1}), 0 \leq \gamma < 1 \tag{2}$$

where $0 \leq \gamma < 1$ is the discount factor [14] and ϕ is the parameters of the neural net. Thus, the weights, ϕ, of the neural network are adjusted to account for the error in the target and current value via optimization methods like gradient descent. Concretely,

$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_t, a_t)(Q_\phi(s_t, a_t) - y_t) \tag{3}$$

The basic online Q-learning algorithm has no convergence guarantees. (3) is not strictly a proper gradient descent since the target itself is ever changing and dependent on the parameters of the network. However, many research attempts have been made to increase the chance of convergence, the ideas of which have been incorporated into our implementation. Firstly, in the online Q-learning algorithm, the state-action pairs are correlated (the next state is highly dependent on the current state and action taken). To overcome this problem, we use an approach similar to [15] and draw random batches of experience from an experience buffer, which holds the agent's past experiences. Secondly, we use two networks instead of one - a target network and a primary network, which overcomes the problem of overestimation of action values as described in [16]. The target network is used to get an estimate of the target in (2) while the parameters of the primary network

are updated using optimization methods. The parameters of the target network are updated towards that of the primary network at a rate τ [16].

Finally, inspired by [17], we use a dueling Q network architecture with separate value streams and advantage streams. The reason behind doing so is to allow the network to learn the state value functions independently from the advantage of taking an action and remove the coupling to any specific action.

The most imperative component of the DQN model is defining the rewards. The reward function is like a semantic that controls the policy learned by an RL agent. So, it is essential to come up with a proper reward function that encourages our learning agent to behave in a way we want it to [18]. Andrew describes one of these approaches to defining this reward through potential based shaping in [18]. The fundamental idea behind potential based reward shaping is that the learning agent is rewarded along the path we want it to follow and not just a huge reward at the end of achieving a goal. Another property of potential-based rewards is that it avoids the agent from being stuck in a sub-optimal positive reward loop and does not alter the optimal policy [18].

2.4 Model-based Policy Search Using PILCO

Another algorithm that we experimented with to get an autonomous drift in our project is Probabilistic Inference for Learning COntrol ($PILCO$) [19]. Although we managed to get some form of circular turn using our double DQN model as described above, we consistently fell short of getting a sustained circular drift. This may be partially attributed to the fact that the double DQN model works with a discrete action space, which greatly constrains the possible range of steering actions the car can take to execute a proper drift. On the other hand, $PILCO$ not only deals with continuous action spaces for finer control for our car, but also requires much less data to find an optimal policy for executing a circular drift. This is because PILCO is a model-based algorithm. More concretely, it uses Gaussian processes to model the forward dynamics of the car and does not require any explicit parameterization of the complicated dynamics of a drift. The added

advantage of using Gaussian processes for predicting the forward dynamics over other function approximators like neural nets is that their probabilistic properties help to reduce the effect of model errors [20].

3 Methodology

With the relevant decisions made after a comprehensive literature review, we now talk about the implementations of the RL algorithms. In our project, we took two separate approaches to obtain an optimal drift controller. We initially started with a DQN model, and learning from the shortcomings of this approach as discussed earlier, we implemented a PILCO agent. This section summarizes both the approaches taken.

3.1 Autonomous Drifting with DQN Model

After discussing the generic RL methodology in the literature review, this section further elaborates on it in context to the DQN model. More concretely, for the DQN model in our project, the reward for the agent is computed based on the position of the car in the two-dimensional x-y space. Position space is restricted to the 2 dimensions because it will result in less computation done by the agent to converge to an optimal drift policy. The reward is maximized if the car manages to maintain a fixed radius, r, from the centre of the circular drift trajectory, and the further it deviates from this circular track the larger the penalty it receives (negative reward).

The input to the DQN model is the global Markovian state given by

$$s_t = [x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}] \tag{4}$$

where s_t is the state of the car at time step t while x, y, θ, $\dot{x}$, $\dot{y}$, $\dot{\theta}$ are the x-coordinate, y-coordinate, angular orientation, x-velocity, y-velocity and angular velocity with respect to the world reference frame respectively. Since the DQN model works in a discrete action space, the output action a is an integer where $a \in A$ and $A = [65, 75, 85, 90, 95, 105, 115]$, which represents the steering angles (throttle of the car is kept constant to reduce the action space to optimize over). The exact values for the set A were obtained based on our physical RC car and the constraints in its steering angle.

3.1.1 Value Function Approximation

RL algorithms for small number of discrete states are relatively simple because the action values (the total expected discounted reward when an agent is at state s and takes action a [14]) for the individual states can be stored in a simple look-up table. What makes this approach difficult in our project is the fact that our RC car has a continuous state space and infinitely many states. For example, the x and y coordinates of the car's position and the linear velocities $\dot{x}$ and $\dot{y}$ are all real and continuous, contributing to the infinite state space. Thus, a different approach is needed to generalize to the state space for our RC car. That is why a function approximator will be used, a neural network to be precise, to approximate the Q-values and generalize over the entire state space for the RC car. More concretely, given a state the car is in, s, as the input, our function approximator will output $\hat{q}(s, a, \mathbf{w})$, where $\mathbf{w}$ are the parameters of the function approximator (weights in the neural net) for all $a \in A$. $\hat{q}(s, a, \mathbf{w})$ will give us the approximate Q-value for the state-action pair and we do not need to store the action values in a table for each pair.

3.1.2 Reward Definition

For our project, we use potential based reward shaping to make sure the car follows a circular pattern [18]. In addition, a negative reward is added that penalizes for deviation of the car's path from the target trajectory. Since the target trajectory is circle, an equation for the circle can be obtained and the reward for the deviation is the negative of the squared error between the car's actual position and the target trajectory. Our final reward is a summation of these individual rewards. More concretely, the final reward, R is related to the potential reward, R_p and the deviation reward, R_d as

$$R = R_p(\Delta\theta) - R_d(d, r), \text{where} \tag{5}$$

$$R_p(\Delta\theta) = \begin{cases} 1, \text{if } \Delta\theta > 0 \\ -1, \text{otherwise} \end{cases} \tag{6}$$

$$R_d(d, r) = (d - r)^2 \tag{7}$$

3.1.3 Double Dueling DQN Architecture

As previously discussed in the literature review section of this report, we settled with using double dueling Deep Q-networks for Q-learning of the action values due to the fact that it is an off policy and model-free algorithm. The dueling architecture of our network is illustrated in Figure 6. Our network comprises of a few fully connected layers and dropout layers initially, which then branches into two separate advantage and state value streams to combine again into a final fully connected layer to output the final action values. Although we experimented with the number of the hidden layers and their sizes to get an optimal network, our current implementation comprises of 3 fully connected layers initially of hidden size 500 and dropout 0.5. The 3 fully connected layers in the separate streams have a size of 300 each.

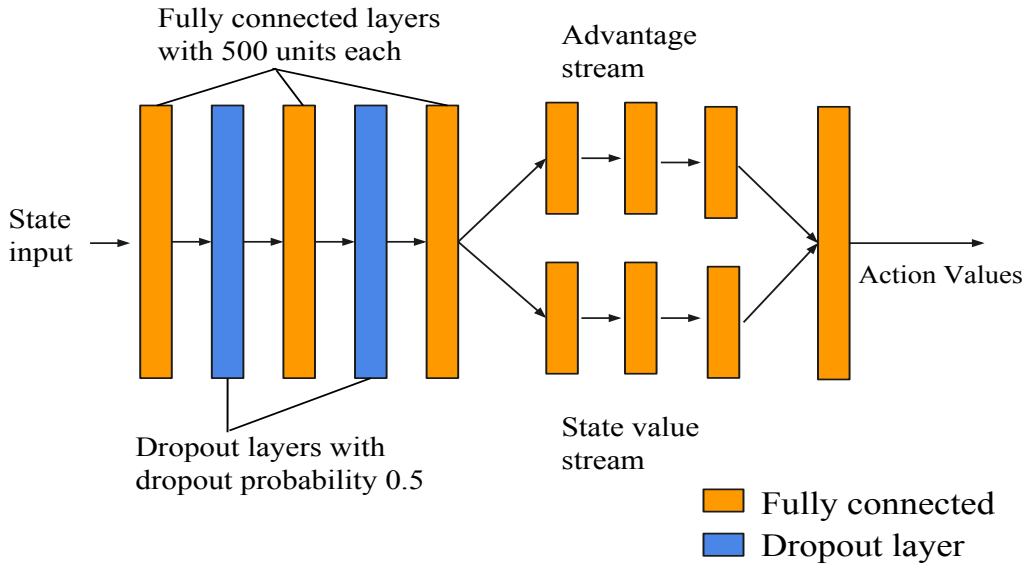


Figure 6: The diagram shows the double dueling architecture of the Deep Q-network. The blocks coloured orange represent the fully connected layers while the blue blocks represent dropout layers for regularization. The network then branches into two separate advantage and state value streams to combine again into a final fully connected layer to output the final action values.

3.2 PILCO Design

In addition to our DQN model, we also implemented a PILCO agent in MATLAB to learn the optimal drift controller. On a high level, the algorithm has three essential steps - model learning, policy learning and policy application, which are elaborated further in the subsequent paragraphs.

3.2.1 Model Learning

The first step of the algorithm is to build a probabilistic model of the how the RC car interacts with the world (dynamics of the car), which is achieved using Gaussian processes [20]. The input to this model is a state-action pair (s_t, a_t) and the output is the successor state, s_{t+1}, where t denotes the time step. In our project, a full Gaussian process model is trained by maximum likelihood estimation on the states encountered (evidence maximization).

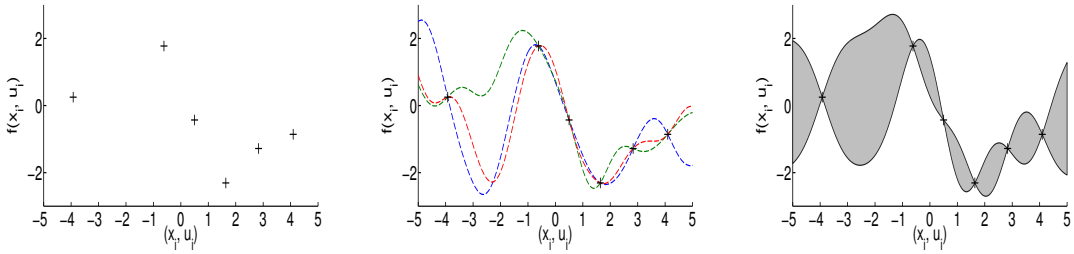


Figure 7: In the diagrams, (x_i, u_i) represents the state-action pair at ith time step and $f(x_i, u_i)$ is a function for the forward dynamics that is to be predicted. The left diagram represents the state-action pairs encountered, while the middle diagram represents a few functions that could have satisfied the distribution of the state-action pairs. The rightmost diagram represents how Gaussian processes build a probability distribution over functions by maximum likelihood estimation on the state-action pairs encountered [19].

To initialize the Gaussian process, various values of (s_t, a_t) and s_{t+1} were collected by driving the simulated (and real) car. In this project, various methods were considered for collecting the data needed to initialize the Gaussian process model:

- Random Actions: The simplest way to collect the required data is to have the agent take random steering actions on the car, and collect the resultant state. This is a simple, yet effective approach to initialize the model, but can extend the length of the learning process.
- DQN Model: Another approach that has never been used before, is to use the preliminary DQN model. The DQN model is in no way perfect, but can be used as a good starting point for the PILCO algorithm. The DQN model is used to control the car as before, but the values of (s_t, a_t) and s_{t+1} are also forwarded to the PILCO controller to initialize the model.
- Demonstration: Finally, another way is to make use of a demonstration, as used in various inverse reinforcement learning problems [21]. A demonstration of driving the car around can be used as an effective alternative to initialize the Gaussian process model.

Once the values are collected, we use a Radial basis network as a functional approximator to calculate the parameters needed to initialize the Gaussian process. The centres for the RBF networks are found using the k-means algorithm [22] and the length scales are also learned by evidence maximization similar to the Gaussian processes. The learned parameters from the RBF network are then used to initialize the policy network instead.

3.2.2 Policy Learning

Using the model of the forward dynamics, the algorithm approximates long term evolution of the state of the car $p(s_1|\pi)$, $p(s_2|\pi)$ ... $p(s_T|\pi)$, given the current policy π. Using these long term predictions, the algorithm estimates the cost for using the policy π analytically. This constitutes the policy evaluation of policy learning.

This is followed by policy improvement, with iterative applications of optimization algorithms such as L-BFGS [19].

Since the PILCO algorithm requires a cost design that can be used to analytically compute the total cost of an episode [19], our new cost function had the following form

$$c_t = 1 - \exp\left(-\frac{(\boldsymbol{s}_t - \boldsymbol{s}_{target})^T \boldsymbol{W} (\boldsymbol{s}_t - \boldsymbol{s}_{target})}{2\sigma^2}\right) \tag{8}$$

where σ (width of the cost function which controls the sensitivity of the cost to deviation from target) [19] was set to 5. c_t is the cost incurred, s_t is the simplified state of the car at time step t and $\boldsymbol{W}$ is a matrix that represents the weights assigned to each component of the state in calculating c_t. $\boldsymbol{s}_t$, $\boldsymbol{s}_{target} \in R^d$ and $\boldsymbol{W} \in R^{dXd}$, where d is the number of elements in s_t.

$\boldsymbol{s}_{target}$ was set to a constant reasonable target state for our model, which was obtained using equations of motion governing an object moving in a circle. According to [23], the centripetal force on the car and frictional force acting on it are related by the equation

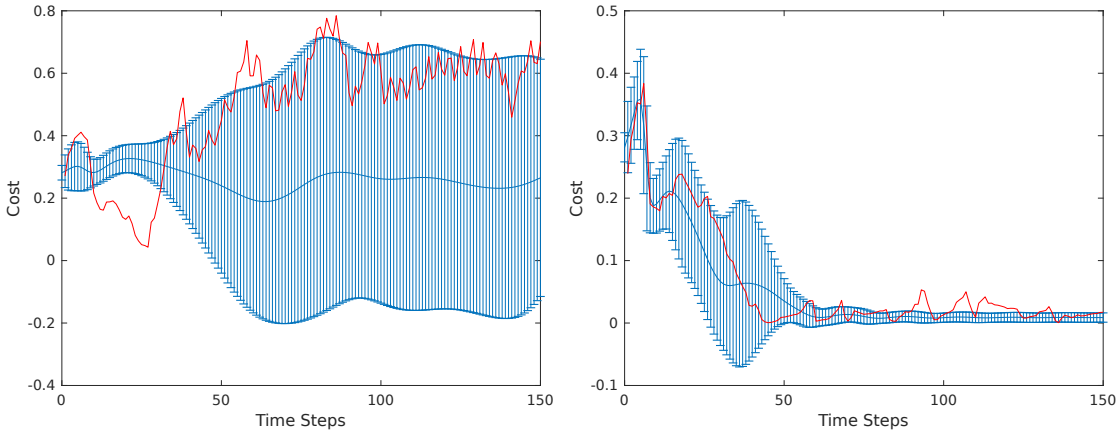
$$\frac{mv^2}{R} = \mu mg \tag{9}$$

where m and v are the mass and velocity of the car respectively while R and g are the radius of the circular path and the gravitational constant respectively. μ is the coefficient of friction between the tire and the ground. Moreover, the angular velocity, $\dot{\theta}$, and the speed of the car, v, are related by

$$v = R\dot{\theta} \tag{10}$$

3.2.3 Policy Application

Using the policy learned in the previous step, the policy is applied in our simulator of the RC car. The car state that is obtained from the simulator is mapped through the policy to get an action to execute at every time step.



(a) Episode 1 **(b) Episode 15**

Figure 8: Cost predictions during policy application at different stages of the training of the PILCO algorithm. The blue vertical lines represent the uncertainty error bars in the algorithm's estimate of the cost during an episode while the blue curve in the middle represents the average estimate of the cost. The red line represents the actual cost incurred by the car during the episode. The diagrams illustrate that the algorithm performs much better at predicting the cost of an episode in only 15 trials.

In addition, Figure 8 shows the cost incurred during policy application in the first episode and the 15th episode of the PILCO training with the final state representation on a four-wheel drive car. The blue vertical lines represent the uncertainty error bars in the algorithm's estimate of the cost during an episode while the blue curve in the middle represents the average estimate of the cost. The red line represents the actual cost incurred by the car during the episode. As can be seen from the figures, the actual cost incurred by the car closely mirrors the algorithm's average estimate by the end of just 15 episodes. Moreover, since the algorithm has seen more of the dynamics of the car by the 15th episode, the actual cost incurred also lies more within its estimate uncertainty by the 15th episode.

3.2.4 State Representation

Although we experimented with different state spaces for the car and various other aspects of the model in the simulator (these are discussed in the results section of this report), the final PILCO implementation has a few key differences to the DQN model. The implementation for PILCO that converged in the best controller for a sustained circular drift used the state $s_t = [\dot{x}_{car}, \dot{y}_{car}, \dot{\theta}]$, where $\dot{x}_{car}$ and $\dot{y}_{car}$ are the velocity components of the car along the x and y axes from the car's reference frame respectively (Figure 9) while $\dot{\theta}$ is the angular velocity, instead of the global Markovian state as in equation 4.

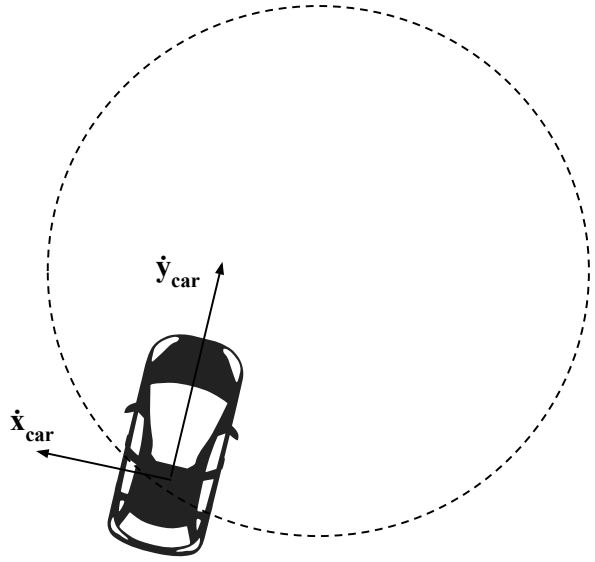


Figure 9: The diagram shows the velocity components of the car along the x axis, $\dot{x}_{car}$, and along the y axis, $\dot{y}_{car}$, from the car's reference frame used in our final state representation. R represents the radius of the circular drift path.

We arrived at this simplified state representation by iterative improvements based on the results obtained from experiments discussed later. This was also greatly motivated by our realization that this representation is sufficiently Markovian for the problem of drifting. This is because based solely on this representation, it is possible to predict the future evolution of the state of the car. Since the cost

encoding a circular drift is an explicit definition involving just these components of the state, it is hence possible to predict the future rewards the RL agent receives for executing a proper drift entirely from the new state representation, proving that the state is Markovian. However, doing so comes at the expense of not having a fixed centre of the circular drift.

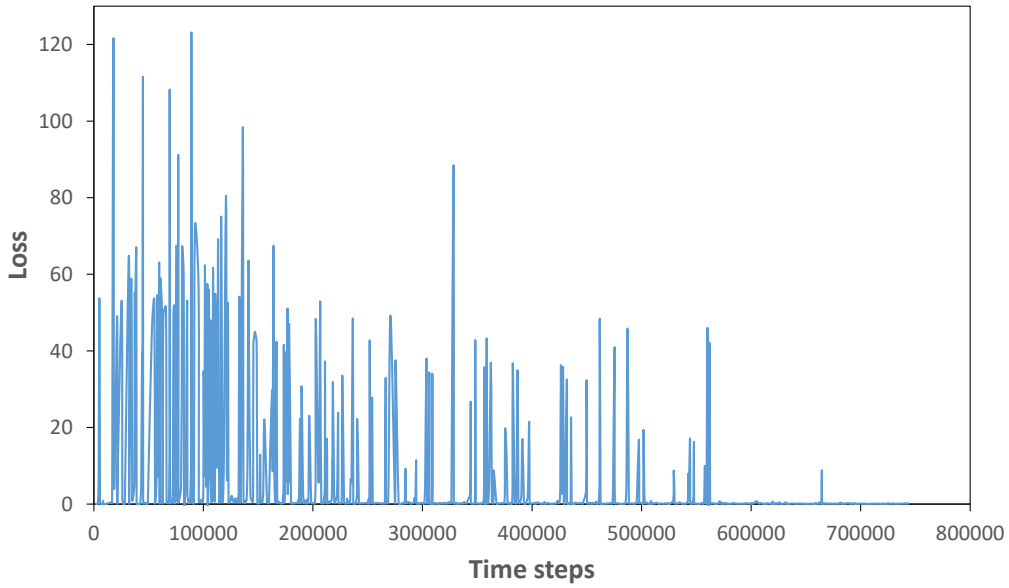
Following the discussion of the PILCO design in the project, the next section presents the results of experiments performed with the DQN model and progressive iterations of the PILCO algorithm.

4 Results and Experiments

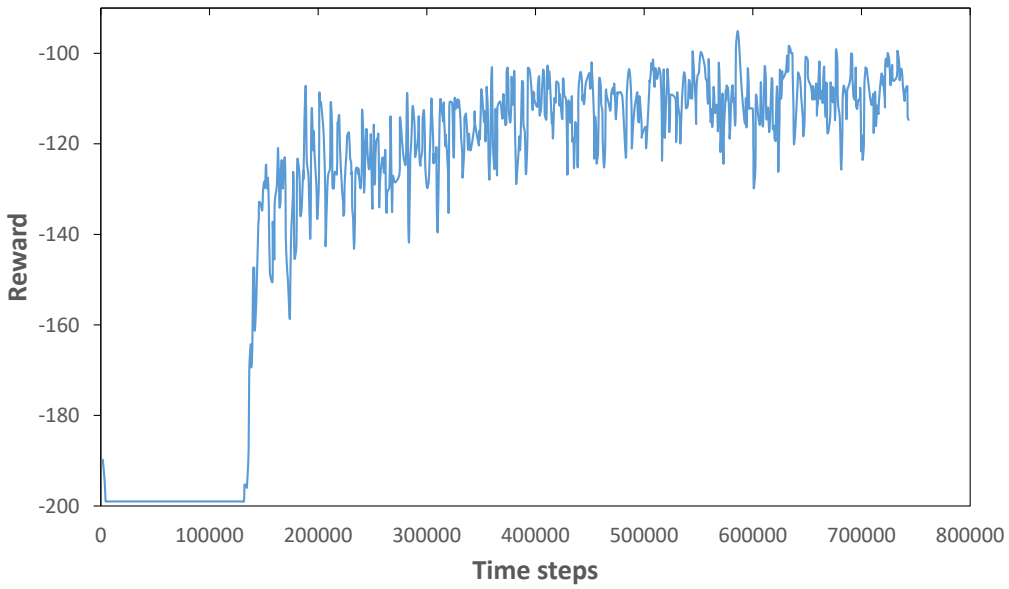
4.1 DQN Results

To ensure that the implementation of the algorithm was correct, it was used to solve two baseline problems in the RL realm - Mountain Car and Cart Pole. To quantify the performance of the algorithm, we use two metrics - the mean loss and mean reward of the agent over the number of steps of the algorithm. The loss is formulated as the squared difference between the expected action value and the target action value. As the agent learns the expected action values over number of steps, the value converges with the target action value, and hence the loss should decrease over number of steps. The reward is simply the total reward the agent earns over an episode of the algorithm, which should be increasing over the number of steps as the agent learns the appropriate behavior.

Both these trends can be observed for the Mountain Car problem in Figure 10.



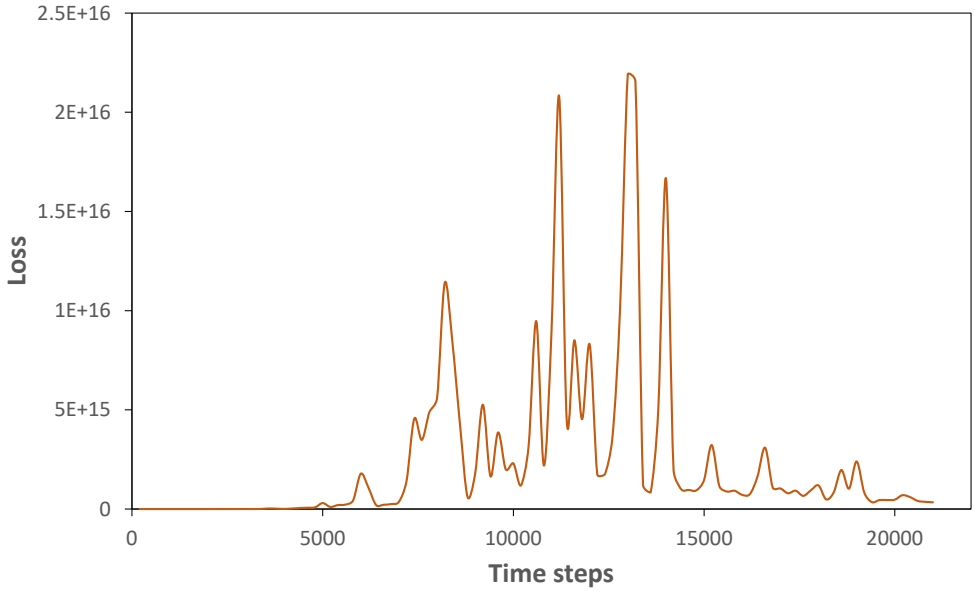
(a) Mean loss



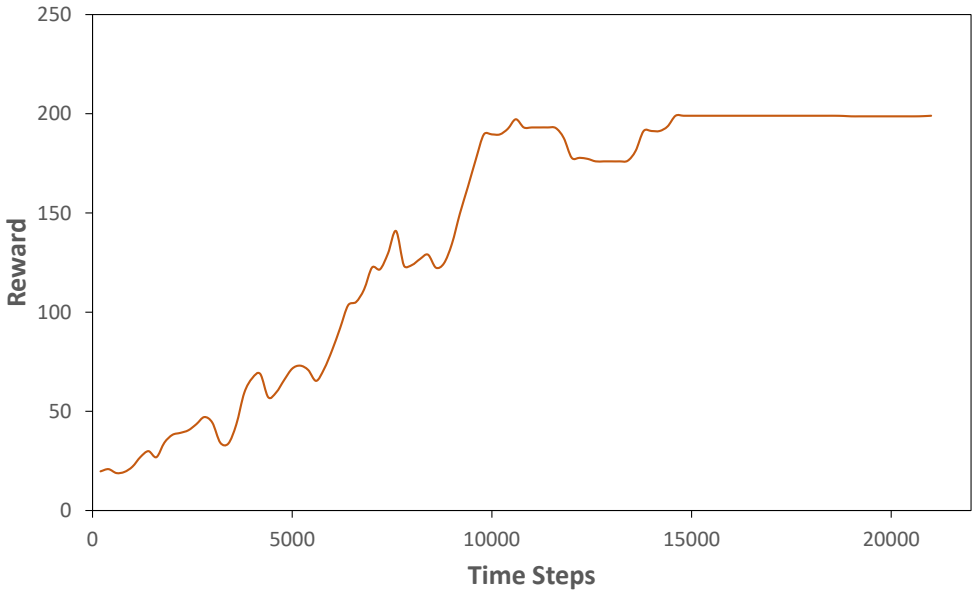
(b) Mean reward

Figure 10: As can be observed on the diagrams the mean loss reduces and the mean reward increases with the number of training iterations, which is as expected on the Mountain Car problem.

Similarly, we can observe these trends for the Cart Pole problem in Figure 11.



(a) Mean loss



(b) Mean reward

Figure 11: The loss and reward trends for the DQN model on the Cart Pole RL problem follows a similar eventual downward and upward trend respectively as expected.

Finally, once the required behavior was observed in the baseline problems and we were confident our implementation is correct, the first iteration of the double dueling DQN algorithm was executed on the simulated car. Figures 12 and 13 illustrates the mean loss and mean reward of the agent after 17 hours of training respectively. Once again, the results are congruent with what we had expected.

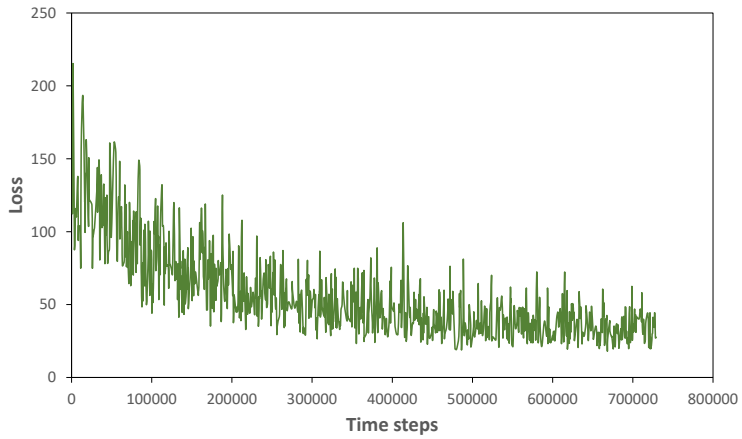


Figure 12: The mean loss over the number of training steps during the training of the DQN model on the simulated car (trend as expected).

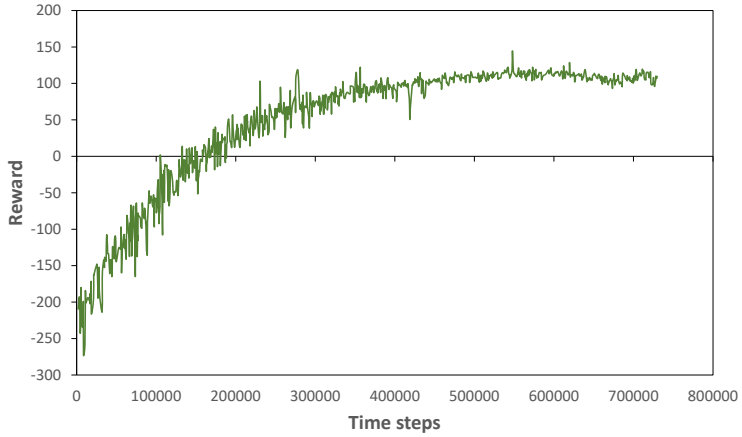


Figure 13: The mean reward over the number of training steps during the training of DQN model on simulated car. The reward increases over time, which proves the algorithm implementation optimized for the reward defined.

Although the DQN algorithm performed particularly well on baseline reinforcement learning problems, the algorithm didn't succeed in finding an optimal drift controller. As mentioned previously this could partially be attributed to the fact that the DQN algorithm works on a discrete action space, which greatly constrains the possible steering actions the car can execute. This could have also resulted from the use of potential based rewards; potential based reward shaping likely was not enough to differentiate a car from moving in a circular motion to that drifting in a circle. This led us to explore PILCO as discussed previously and the subsequent paragraphs summarize some of the experiments performed and results obtained by progressively iterating on the PILCO algorithm, culminating in our final successful implementation.

4.2 PILCO Results

We started exploring PILCO using a the same global Markovian state as we had used for the DQN model (equation 4). We used a different objective function as defined by equation 8 earlier with $\boldsymbol{W}$ being an identity matrix. Both the $\boldsymbol{s}_t$ and $\boldsymbol{s}_{target}$ contained all components of the global Markovian state ([x, y, θ, $\dot{x}$, $\dot{y}$, $\dot{\theta}$]). We noticed significant improvements in doing so. In particular, using the controller learned from the PILCO algorithm, the car in simulator managed to slip and turn. However, it did not manage a sustained circular drift.

Although we started with a full Markovian state including the x and y coordinates for the position of the car, owing to difficulties encountered with indoor localization of the RC car as discussed in Section 3.2.1 of *Hardware Report*, we experimented with removing the position coordinates from the state. As already mentioned, this was greatly motivated by our realization that the state of the car excluding the x and y coordinates is sufficiently Markovian for the problem of drifting. In addition, we added a speed component to the total state (making use of the separate velocity components), which played a role in calculating the cost. It was used to penalize non-uniform circular motion since an object moving in a uniform circular motion maintains a constant speed although it is accelerating by virtue of changing direction of its velocity [23]. This resulted in the new state representation

$$s_t = [\theta, \dot{x}, \dot{y}, \dot{\theta}, S] \tag{11}$$

where S is added to represent the speed of the car. We used the same definition of c_t as given by equation 8. The diagonal matrix $\boldsymbol{W}$ then had the form

$$\boldsymbol{W} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \alpha_1 & 0 \\ 0 & 0 & 0 & 0 & \alpha_2 \end{bmatrix}$$

α_1 and α_2 were integer weights for the $\dot{\theta}$ and S components of the state respec-

tively accounted into the cost c_t. The other diagonal entries were set to 0 since the corresponding state components were not taken into account in calculating c_t. However, θ, $\dot{x}$ and $\dot{y}$ were still left in the state because they were used by the Gaussian process model to predict the forward dynamics of the car. Initially, both α_1 and α_2 were set to unity. The results we obtained by taking these steps were slightly better than the ones with a full Markovian state, which supports the argument that the coordinates are not necessary for finding an optimal drift controller. By a similar reasoning, we additionally experimented with removing the angular orientation of the car from the complete state and once again, the results were convincing to strengthen the argument. By adding a speed component to the state, our car in simulator was also trying to turn more often, as seen in Figure 14, which proved it played a crucial role in trying to obtain a circular drift controller.

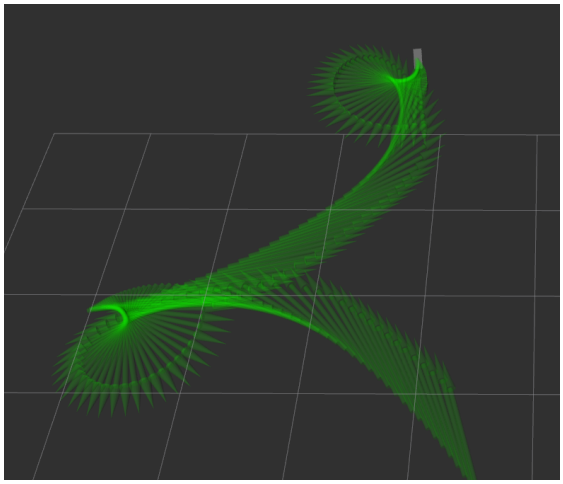
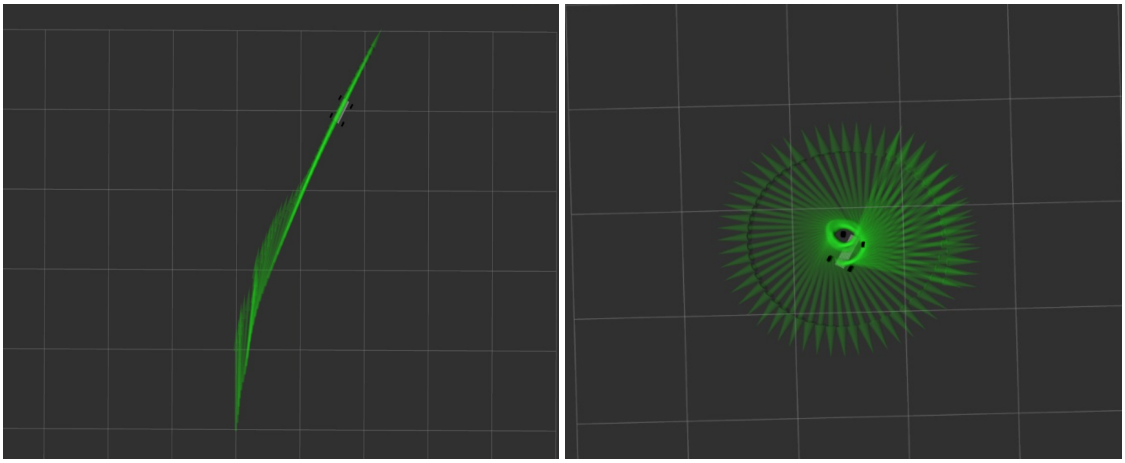


Figure 14: The time-lapsed path followed by the car with a speed component in the cost definition. As can be observed, the car turns and drifts but fails to sustain a circular drift and lacks stability.

Despite some improvements over the previous full Markovian state, we observed that the algorithm sometimes converged to a suboptimal controller where the car only followed a straight path. We realized that this could have resulted from the specific values assigned to α_1 and α_2 in $\boldsymbol{W}$. Since the two components of the cost had equal weights, the controller initially tries to optimize for both them. However, after a few episodes, the controller is unable to optimize for both components of the

cost simultaneously and hence chooses to prioritize the speed component, which is easier to achieve, leading to a suboptimal controller. Furthermore, it was also evident that the controller PILCO converged to was also influenced by the values set for α_1 and α_2. To validate our reasoning, we experimented with different weights for the components of the cost, and the results are illustrated in the path diagrams below.



(a) $\alpha_1 = 0$ and $\alpha_2 = 1$ **(b) $\alpha_1 = 1$ and $\alpha_2 = 0$**

Figure 15: Path followed with different α_1 and α_2 values in the W matrix. Different values for α_1 and α_2 caused the algorithm to converge to suboptimal controllers where the car either followed a straight path or spun in place instead of drifting.

Figure 15 illustrates the time-lapsed path of the car with various value settings of α_1 and α_2. The arrows in the diagrams represent the direction in which the car was facing. As shown by the arrows in Figure 15a, with a value setting of $\alpha_1 = 0$ and $\alpha_2 = 1$ **(Experiment 1)**, the policy converged to a controller where the car always followed a somewhat straight path. If the car went completely straight, it would surpass the target speed of 4m/s in s_{target}, based on equations 9 and 10. Thus, the car turns slightly in order to reduce speed and achieve the target. On the other hand, with a setting of $\alpha_1 = 1$ and $\alpha_2 = 0$ **(Experiment 2)**, the algorithm found a controller where the car just spun on the spot to minimize cost, as illustrated by the arrows in Figure 15b.

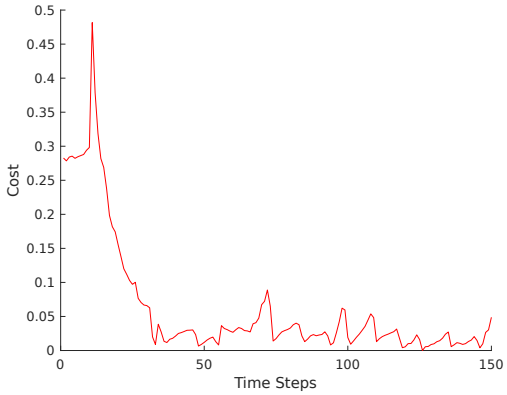
Following our experiments as illustrated, it was evident that settings for the values of α_1 and α_2 played a role in deciding the type of controller the algorithm converged to. After several attempts at manually tweaking the values for the weights **(Experiments 3 and 4)**, we managed at best to obtain a controller where the car would drift but would appear to lose momentum and fail to maintain the drift.

Considering the difficulties associated with finding the optimal values for α_1 and α_2, we decided to shift to an objective cost that was not only much easier to deal with, but also encoded the problem of drifting much more precisely. To do so, the state of the car was modified further to have two different components, which were used in the objective function as in equation 8. Unlike the velocity components with respect to the world reference which are constantly changing during a circular motion, both x and y components of the body frame velocities will remain constant during a perfect circular drift. We arrived at the final simplified state representation given by

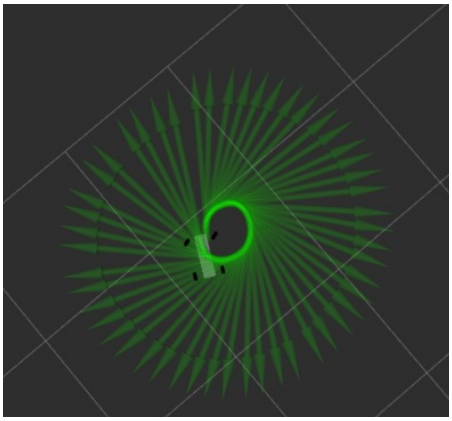
$$s_t = [\dot{x}_{car}, \dot{y}_{car}, \dot{\theta}] \tag{12}$$

where x_{car} and y_{car} are the velocity components of the car along the two perpendicular axes from the car's reference frame as in figure 9 while $\dot{\theta}$ is the angular velocity. With this representation, $\boldsymbol{W}$ was again set to a 3x3 identity matrix so that the algorithm weighed each component of the new state into the cost function in equation 8 equally. We experimented with this final state and cost representation with a two-wheel drive car **(Experiment 5)**, and got some success in getting a circular drift due to the fact that the car was now actively trying to counter steer in order to achieve and maintain the body frame velocities. However, we noticed that even though the rear wheels of the car slipped out to initiate a sustained circular drift, the front wheels did not have enough torque on them to drive the car sideways (using the steering angle) and maintain sideways momentum. In order to alleviate this problem, we experimented with a four-wheel drive car **(Experiment 6)**. This allowed the front wheels to independently add to the sideways velocity component, allowing the car to optimize for both velocity components and

maintain the sideways velocity without losing momentum. Doing so achieved the successful sustained circular drift we have been aiming for.



(a) Cost incurred



(b) Path followed by car

Figure 16: Cost incurred by the converged controller and the path followed by the car. In (a), the cost incurred during the first few time steps is slightly higher due the fact that the car initiates a circular drift from a standstill, after which, the average cost stays fairly low and constant. In (b), the time-lapsed diagram shows the car drifting in a circular path.

Figure 16a represents the cost incurred at every time step for the converged successful sustained circular drift controller on the four-wheel drive car. As illustrated, the cost incurred during the first few steps is slightly higher due the fact that the car initiates a circular drift from a standstill, after which, the average cost stays fairly low and constant.

Figure 16b illustrates the time-lapsed circular path followed by the car with our successful drift controller as it drifts sideways. In contrast to Figure 15b, the circle outlined in Figure 16b has a larger radius, which also indicates that the car was not simply spinning in place.

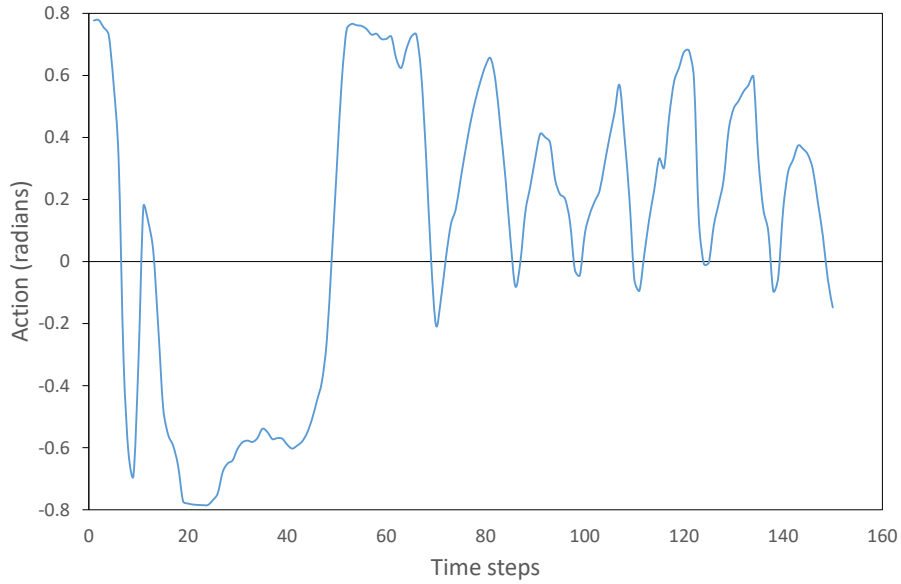
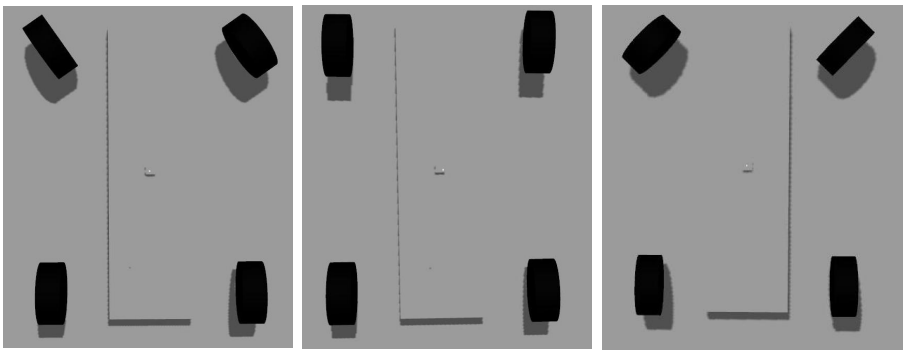


Figure 17: Steering actions (radians from centre) issued by controller at each time step. The diagram helps to appreciate the complexity of the problem of sustaining a circular drift.

Figure 17 illustrates the steering action (radians from the centre) issued over the same 150 time steps by the successful controller to maintain the sustained circular drift in Figure 16b . This figure helps to appreciate the complexity of the problem of sustaining a circular drift, which is difficult for a human to recreate.



(a) -0.8 **(b) 0** **(c) 0.8**

Figure 18: Steering of the simulated car at different values of the steering action in radians from centre; -0.8 (steer left completely), 0 (steer centre) and 0.8 (steer right completely).

Upon obtaining a successful drift controller, we further experimented with the effects of initialization methods on the PILCO algorithm. As discussed previously in the methodology, we investigated with random initialization, initialization with the DQN model and initialization from a demonstration. Table 1 summarizes the number of episodes taken by the algorithm to converge to a controller that receives an average cost of 0.1 over the length of the episode (150 time steps). As can be observed, the algorithm converges quicker with both a DQN model initialization and initialization from demonstration than a random initialization. However, there is not much difference between a DQN model initialization and initialization from demonstration since the demonstration was done by one of us and not experts at drifting RC cars.

Initialization method	**Episodes taken**
Random initialization	12
DQN initialization	7
Demonstration initialization	8

Table 1: Number of episodes taken by the PILCO algorithm to converge to a controller that receives an average cost of 0.1 over the length of the episode (150 time steps). As can be observed, the algorithm converges quicker with both a DQN model initialization and initialization from demonstration than a random initialization. However, there is not much difference between a DQN model initialization and initialization from demonstration since the demonstration was done by one of us and not experts at drifting RC cars.

4.3 Improvements to DQN Model

Following our success in obtaining a sustained circular drift controller with PILCO, we performed a few experiments to compare the effectiveness of the two algorithmic approaches used during the course of the project. We were interested in comparing the results obtained from PILCO with those from the same double dueling Deep Q-network we previously discussed. However, for a fairer comparison, the state input into the DQN model was altered from a equation 4 to 12. Moreover, instead of using potential based rewards like we had previously, the reward, r_t, was altered to be inverse of the cost(8),

$$r_t = -c_t \tag{13}$$

Figure 19 shows the mean reward of an episode of the DQN model at regular intervals during the training with the changes discussed. Similarly, Figure 20 illustrates the mean loss incurred over time steps during the training.

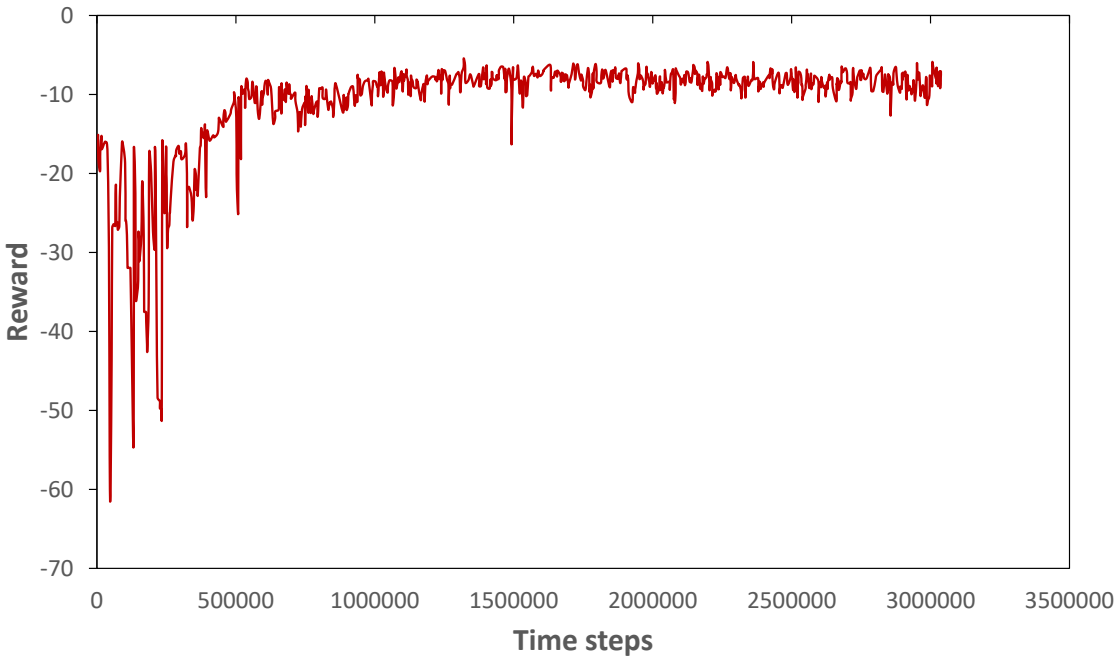


Figure 19: Mean episodic reward during the training of improved DQN model with the final state representation (equation 12) and the new reward function. The mean reward increases over time which is expected.

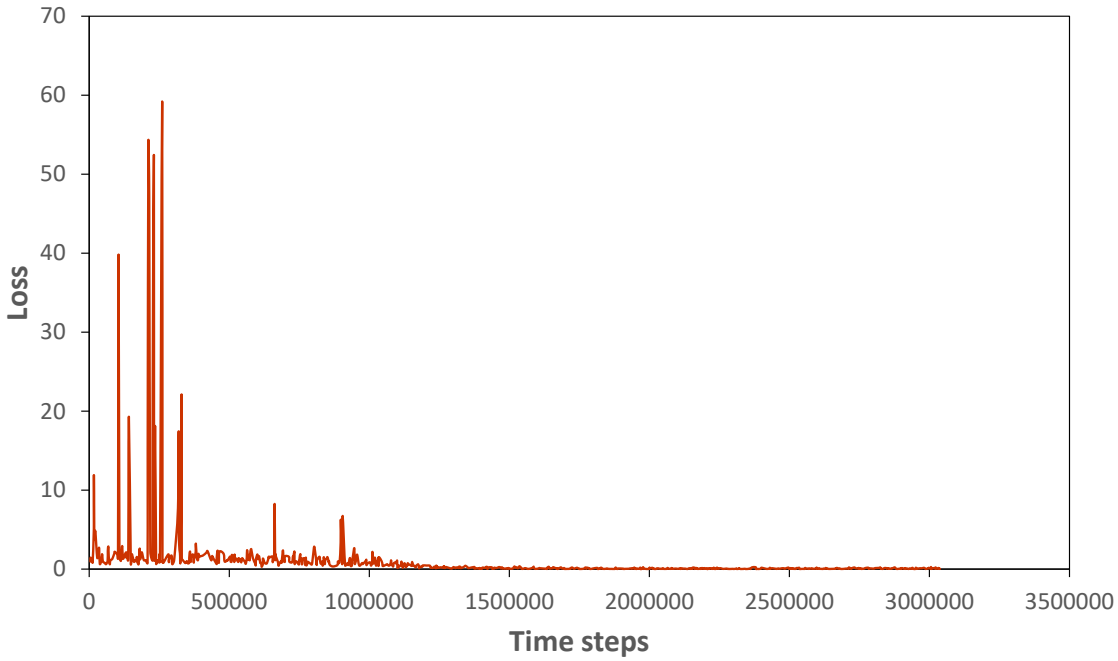


Figure 20: Mean loss during the training of the improved DQN model with the final state representation (equation 12) and the new reward function. The mean loss increases over time as expected.

The controller learned from the DQN model after the changes described was applied to the car in the simulator. The results were remarkably better than what we had previously obtained from the DQN model. Although the DQN controller incurs a slightly higher average cost over an episode than with the PILCO controller (due to the model's restricted steering action space), the car was trying to maintain a circular drift as closely as possible.

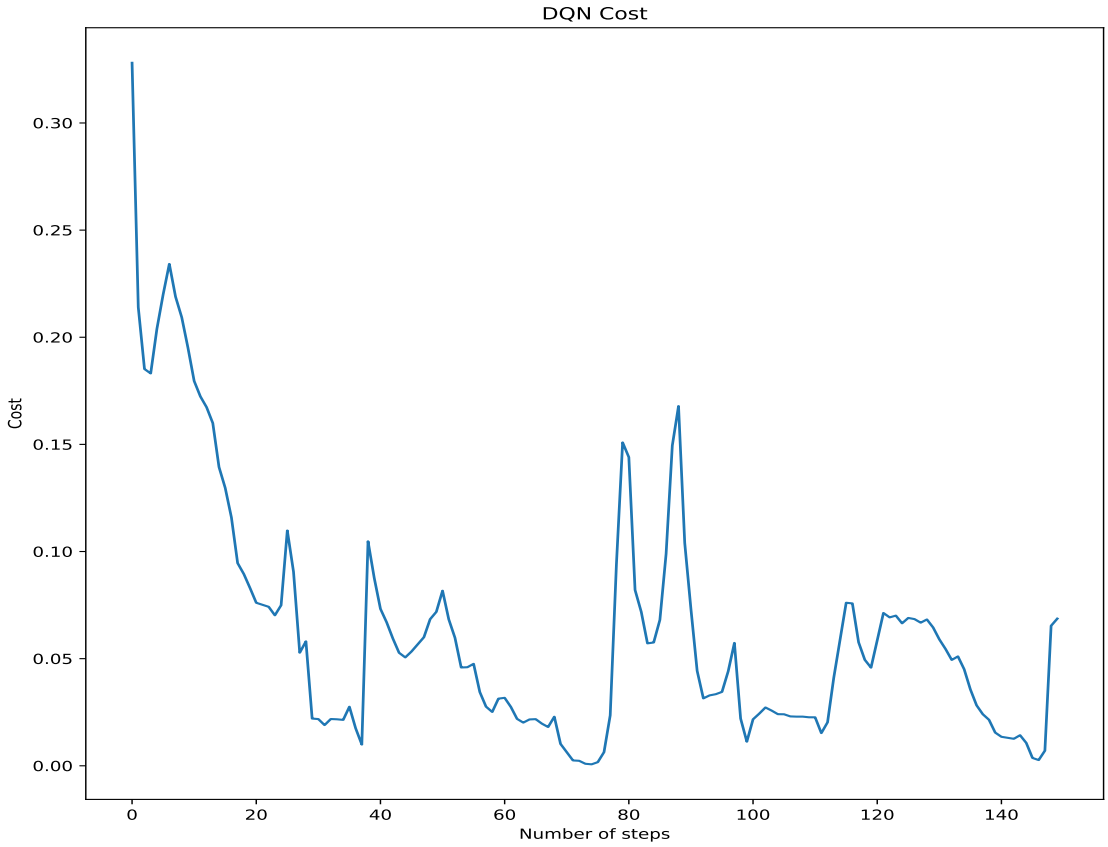
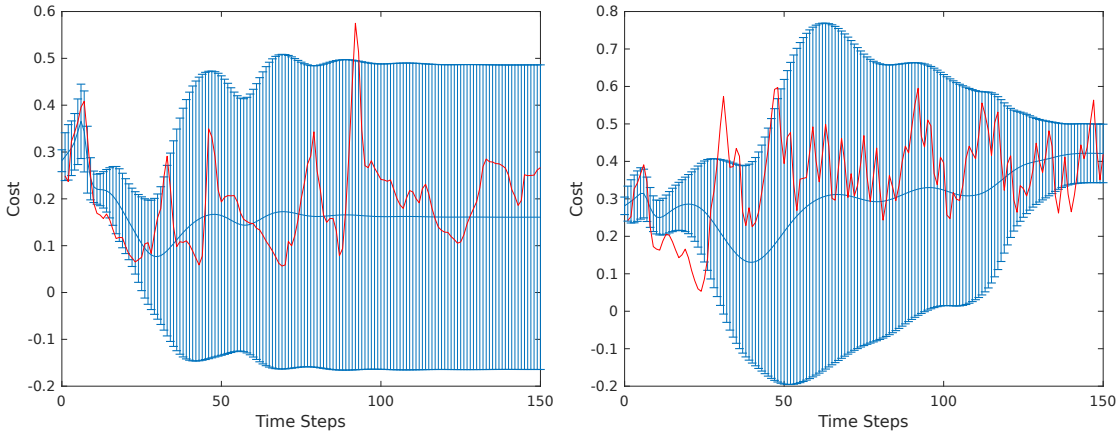


Figure 21: The cost incurred during an episode with the improved DQN controller. Although the average cost is slightly higher than that with the PILCO controller, the cost is consistently low, which proves it is reasonably good despite the constrained action space.

The effectiveness of the controller learned from the DQN model in maintaining a sustained circular drift with the changes discussed can be observed in Figure 21 **(Experiment 7)**. Although the average cost is slightly higher than that with the PILCO controller, it is consistently low, which proves it is reasonably good despite the constrained action space.

4.4 Transfer to Physical RC Car

Using the final controller obtained from the PILCO algorithm as discussed previously, we transferred it the physical RC car to initialize PILCO learning. Figure 22 illustrates the cost incurred over an episode of 150 time steps by the physical RC car with the converged controller after 15 episodes of training **(Experiment 8)**. As can be observed, the cost incurred is higher on the physical RC car than it was on the simulator. This can be attributed to the noisy state data obtained from the car and the inaccurate state information. Due to this, the physical RC car did not manage to obtain a substantial sustained circular drift. Issues encountered with obtaining accurate state information and our attempts at resolving them are discussed further in the *Hardware* and *Software* reports.



(a) 2WD car in simulation **(b) Physical RC car**

Figure 22: Cost incurred with the converged PILCO controller on the physical RC car (2WD) after initializing the learning for 15 episodes compared to the cost incurred in the simulator with a 2WD car. As can be observed, the cost incurred is higher on the physical RC car than it was on the simulator. This can be attributed to the noisy state data obtained from the car. Additionally, the physical RC car did not manage to obtain a substantial sustained circular drift due to this noise in state and also because it is a 2WD car and not a 4WD car. As discussed earlier, the spikes in the diagram 22a is due to the loss in sideways momentum during the drift in a 2WD.

4.5 Quality Evaluation with Drift metric

Since we conducted various experiments with state spaces, action spaces and even the algorithm itself, it was important to be able to objectively evaluate the quality of the drift resulting from the converged controller. So a metric was designed, with some resemblance to the cost definition in (8), which encodes the parameters relevant to a sustained circular drift. D_m scores the quality of a drift in the range [0, 1], with 1 being a perfect circular drift. This metric, D_m is defined as

$$D_m = \frac{1}{T}\sum_{t=0}^{T} \exp\left(-\frac{(||\boldsymbol{s}_t - \boldsymbol{s}_{target}||^2}{2\sigma^2}\right) \in [0, 1] \tag{14}$$

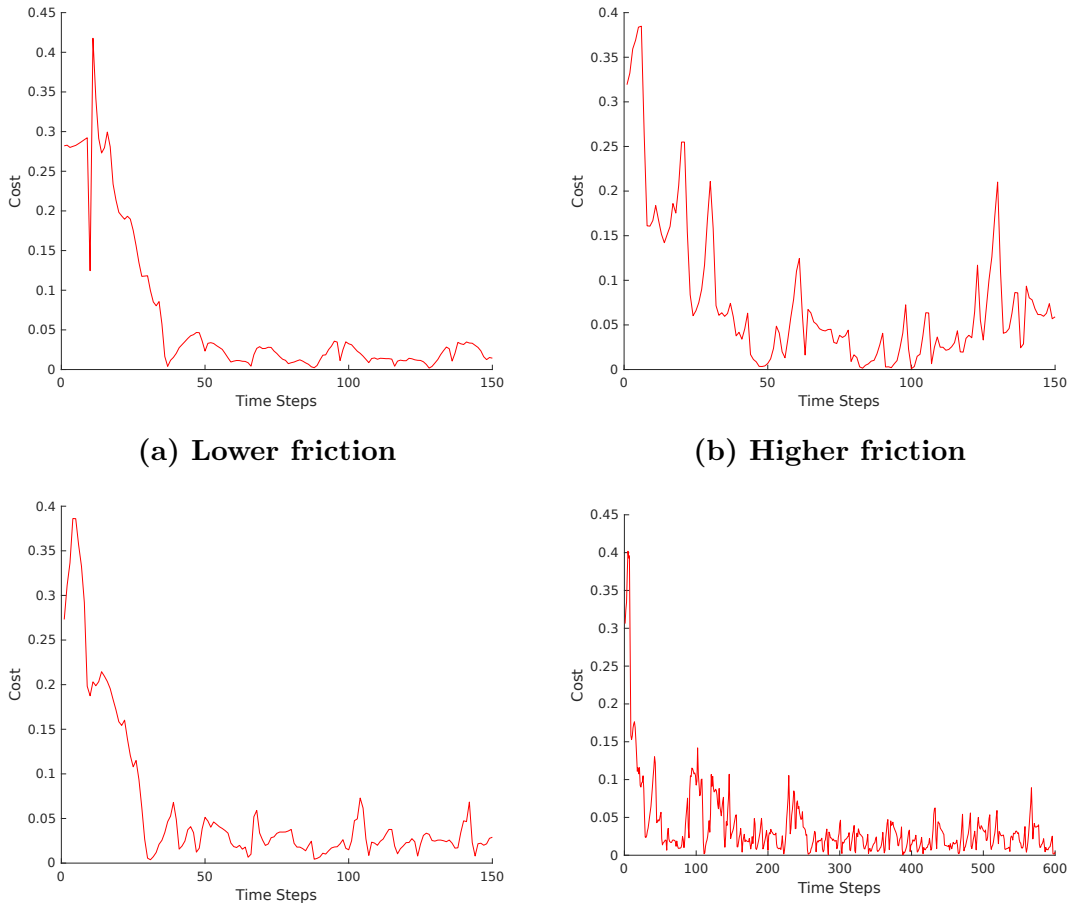
where s_t is the state of the car at time step t given by equation 12 and s_{target} is a reasonable constant vector chosen as [3.5, 0.5, 2] (by reasoning with equations 9 and 10) and σ set to 5 [19]. T is the episode length. Table 2 summarizes the drift metric, D_m, which was evaluated for some of the experiments highlighted and numbered in the previous sections.

Number	**Experiment parameters**	$\boldsymbol{D_m}$
1	$s_t = [\dot{x}, \dot{y}, \dot{\theta}, S]$ with $\alpha_1 = 1$ and $\alpha_2 = 0$	0.402
2	$s_t = [\dot{x}, \dot{y}, \dot{\theta}, S]$ with $\alpha_1 = 0$ and $\alpha_2 = 1$	0.273
3	$s_t = [\dot{x}, \dot{y}, \dot{\theta}, S]$ with $\alpha_1 = 1$ and $\alpha_2 = 1$	0.582
4	$s_t = [\dot{x}, \dot{y}, \dot{\theta}, S]$ with $\alpha_1 = 3$ and $\alpha_2 = 1$	0.615
5	$s_t = [\dot{x}_{car}, \dot{y}_{car}, \dot{\theta}]$ with 2WD	0.763
6	$s_t = [\dot{x}_{car}, \dot{y}_{car}, \dot{\theta}]$ with 4WD	0.948
7	$s_t = [\dot{x}_{car}, \dot{y}_{car}, \dot{\theta}]$ with 4WD (DQN)	0.918
8	$s_t = [\dot{x}_{car}, \dot{y}_{car}, \dot{\theta}]$ with 2WD (physical RC)	0.633

Table 2: Summarized values for D_m for various experiments. The numbers are referenced in the previous sections. As observed, the highest D_m values were achieved with final state representation (12) with both a two-wheel and four-wheel drive on the simulator. The D_m obtained on the physical RC car is lower due to the noisy state information received from the car and also because it is a 2WD and not a 4WD RC car.

4.6 Testing Robustness and Stability

We tested the robustness of our converged sustained drift controller by exposing the car to physical conditions that were different to which the algorithm was trained on. More concretely, the controller was tested with different surface friction and with a lower car mass. Our successful drift controller was obtained by running the algorithm on a surface that had 0.5 as the coefficient of friction (μ) with each episode spanning 150 time steps.



(a) Lower friction

(b) Higher friction

(c) Lighter car

(d) Longer time horizon

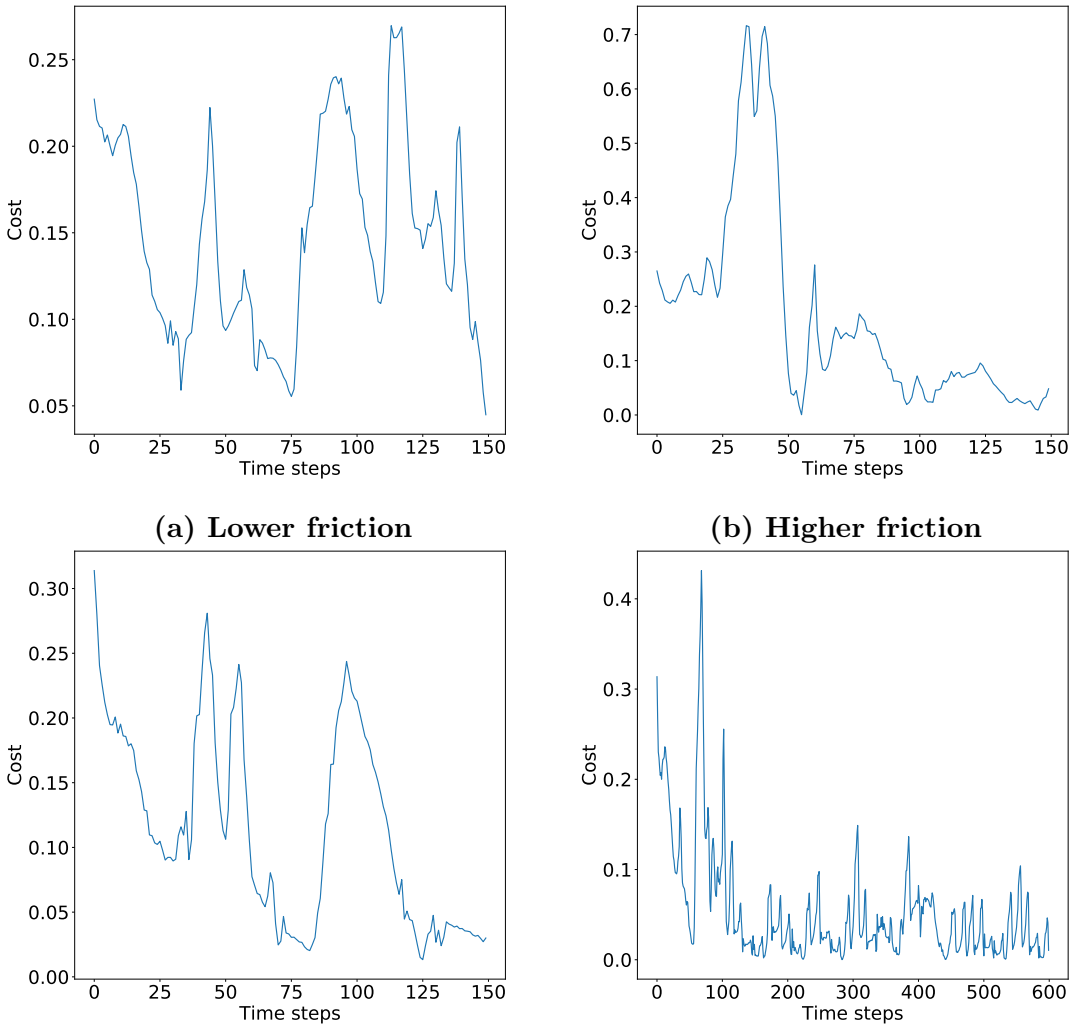
Figure 23: The car was independently subjected to lower friction (a), higher friction (b), lower chassis mass (c) and longer time horizon (d). The consistently low cost incurred during the episode in all conditions prove the stability and robustness of the converged PILCO controller.

Figure 23a illustrates the cost incurred when the learned controller was applied on the car with a lower tire surface friction coefficient ($\mu = 0.4$).

On the other hand, Figure 23b is the cost incurred when the car was exposed to higher surface friction ($\mu = 0.6$) . In both case, the results obtained show a consistently low cost, which is very promising and prove that our controller is robust to varying surface friction.

In one of our experiments, the mass of the car was reduced by a quarter and Figure 23c illustrates the cost incurred. In addition, we also applied the controller for longer time horizon (600 steps) for each episode and Figure 23d illustrates the cost incurred.

The results presented prove that our controller obtained from PILCO is not only robust but is also stable and can adapt to longer time horizons.



(a) Lower friction **(b) Higher friction**

(c) Lighter car **(d) Longer time horizon**

Figure 24: Cost incurred by subjecting the learned DQN controller to different scenarios. As can be observed, the cost incurred during the episode with lower friction (a), higher friction (b) and lower mass (c) is higher and more sporadic. This proves that the controller is not robust. However, the cost incurred over a longer time horizon (d) is consistently low, which shows the controller is stable.

Following our tests with the PILCO controller, we tested the robustness and stability of the controller obtained from the DQN model with the new state representation and reward definition (equation 13). Although the general trend in the

different physical conditions appear to be the same as those with the PILCO controller, the magnitude of the cost incurred during the episode with lower friction (a), higher friction (b) and lower mass (c) is higher and more sporadic. However, the controller appears to be stable since it incurs a consistent low cost with higher time horizons for each episode.

Compared to the DQN model, PILCO was much more data efficient in converging to a controller. Successful results for the PILCO controller discussed previously were obtained only after 30 episodes of training, each lasting 150 time steps, for a total of under 5000 time steps. On the other hand, Figures 19 and 20 show the DQN model converging to a controller after 1.5 million time steps. Such dramatic differences in convergence times is due to the fact that PILCO is a model-based learning algorithm, which uses probabilistic properties of the Gaussian processes to take actions while accounting for uncertainties in dynamics model.

Table 3 summarizes the D_m values observed for the robustness and stability tests performed as described. As mentioned previously, the PILCO controller is both robust to changes in physical conditions of the environment and stable to longer time horizons of episodes. However, the DQN model is not robust to changing physical conditions but stable to longer episode length.

Robustness and stability tests	***DQN***	***PILCO***
Lower friction ($\mu = 0.4$)	0.827	0.921
Higher friction ($\mu = 0.6$)	0.832	0.903
Reduced chassis mass	0.873	0.892
Longer time horizon	0.893	0.938

Table 3: Summary of the D_m values, comparing the robustness and stability of controllers learned from the improved DQN model and PILCO. The PILCO controller is both stable and robust while the DQN controller is stable to longer time horizon but not robust to changes in environment.

5 Conclusion

To summarize, we justified why autonomous drifting cars are important and how drifting can be useful in emergencies to avoid accidents. As we have already discussed, current self driving cars and stability control techniques try to avoid slipping tires and in doing so, restrict the capability of the car. However, we need to exploit the full capability of a car during emergencies. So clearly, having an autonomous drifting car that learns an optimal drift control policy using our methods can help reduce the number of accidents caused by hydroplaning and make roads safer.

Motivated with this intention, we first proposed a framework that uses state of the art model-free reinforcement learning algorithms like double dueling Deep Q-networks to learn an optimal controller for drifting an RC car to maintain a state of steady circular drift. Following the results obtained from our DQN model, we investigated the reason behind not being able to find a sustained circular drift controller and explored PILCO as a model-based approach to alleviate some of the shortcomings of the DQN model - it uses a continuous action space and Gaussian processes for modeling the forward dynamics. Discussion of the methodology was followed by the results obtained from experiments conducted during the course of the project with the DQN model and PILCO. Additionally, results from our successful drift PILCO controller were also presented, and tests were conducted to prove that it is more stable and robust to different physical conditions than DQN controller. Furthermore, one of the important contributions of the project is the drift metric,D_m, which objectively quantifies the quality of a sustained circular drift.

There are a few extensions that can be made to the current state of the project. Firstly, the robust and stable PILCO drift controller that was obtained on the simulator can be extended to be as effective on a physical RC car. Secondly, possible methods of exiting a drift rather than sustaining one can be explored as well. This can then be transferred to an autonomous car that needs to exit a slipping state to avoid accidents.

Although the initial aim of the project was to implement autonomous sustained circular drift in a physical RC car, we did not manage to achieve it completely, owing mostly to hardware challenges associated with indoor localization as discussed in Section 3.2.1 of *Hardware Report* report and cost constraints in acquiring a 4WD RC car. Nevertheless, much effort was put into closely modelling the physical properties of an RC car in the simulator as discussed further in Section 2.3.1 of the *Software Report*. Thus, given our success in finding a robust and stable sustained circular drift controller with the simulator, we firmly believe the results can be easily replicated on a physical RC car once the hardware is acquired.

References

[1] F. Zhang, J. Gonzales, K. Li, and F. Borrelli, "Autonomous drift cornering with mixed open-loop and closed-loop control," in *Proceedings IFAC World Congress*, 2017.

[2] S. Saha, P. Schramm, A. Nolan, and J. Hess, "Adverse weather conditions and fatal motor vehicle crashes in the united states, 1994-2012," *Environmental Health*, vol. 15, 2016.

[3] A. T. van Zanten, R. Erhardt, G. Landesfeind, and K. Pfaff, "Vehicle stabilization by the vehicle dynamics control system ESP," *IFAC Mechatronic Systems, Darmstadt, Germany*, pp. 95–102, 2000.

[4] J. Ackermann, "Robust control prevents car skidding," *IEEE Control Systems Magazine*, vol. 17, pp. 23–31, 1997.

[5] K. Yoshimoto, H. Tanaka, and S. Kawakami, "Proposal of driver assistance system for recovering vehicle stability from unstable states by automatic steering," in *Proceedings of the IEEE International Vehicle Electronics Conference*, 1999.

[6] A. Hac and M. Bodie, "Improvements in vehicle handling through integrated control of chassis systems," *International Journal of Vehicle Design*, vol. 29, no. 1, 2002.

[7] J. Wei, Y. Zhuoping, and Z. Lijun, "Integrated chassis control system for improving vehicle stability," in *Proceedings of the IEEE International Conference on Vehicular Electronics and Safety*, 2006.

[8] A. Trachtler, "Integrated vehicle dynamics control using active brake steering and suspension systems," *International Journal of Vehicle Design*, vol. 36, no. 1, pp. 1–12, 2004.

[9] P. Frère, *Sports Car and Competition Driving*. Bentley, 1969.

[10] E. Velenis, D. Katzourakis, E.Frazzoli, P.Tsiotras, and R.Happee, "Steady-state drifting stabilization of RWD vehicles," *Control Engineering Practice*, vol. 19, 2011.

[11] R. Hindiyeh and J. Gerdes, "A controller framework for autonomous drifting: Design, stability, and experimental validation," *Journal of Dynamic Systems, Measurement, and Control*, vol. 136, 2014.

[12] S.-T. Wu and W.-S. Yao, "Design of a drift assist control system applied to remote control car," *International Journal of Mechanical, Aerospace, Industrial, Mechatronic and Manufacturing Engineering*, vol. 10(8), 2016.

[13] E. Velenis, E. Frazzoli, and P. Tsiotras, "On steady-state cornering equilibria for wheeled vehicles with drift," *Institute of Electrical and Electronics Engineers*, 2009.

[14] R. S. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *CoRR*, vol. abs/1312.5602, 2013. arXiv: `1312.5602`. [Online]. Available: `http://arxiv.org/abs/1312.5602`.

[16] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," *CoRR*, vol. abs/1509.06461, 2015. arXiv: `1509.06461`. [Online]. Available: `http://arxiv.org/abs/1509.06461`.

[17] Z. Wang, N. de Freitas, and M. Lanctot, "Dueling network architectures for deep reinforcement learning," *CoRR*, vol. abs/1511.06581, 2015. arXiv: `1511.06581`. [Online]. Available: `http://arxiv.org/abs/1511.06581`.

[18] A. Y. Ng, "Shaping and policy search in reinforcement learning.," *PhD thesis, EECS, University of California, Berkeley*, 2003.

[19] M.Deisenroth, D.Fox, and C. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *Pattern Analysis and Machine Intelligence, IEEE Transactions*, vol. 99, 2014.

[20] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA, 2006.

[21] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," *Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning and Data Mining*,

[22] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A k-means clustering algorithm," *Applied statistics*, pp. 100–108, 1979.

[23] J. W. David Halliday Robert Resnick, *Fundamentals of Physics*. Wiley, 1960.