Improving Model Predictive Control with Value Function Approximation

Sahit Chintalapudi[1], Nolan Wagener[1], Byron Boots[2]

***Abstract*— Existing Model Predictive Control methods rely on finite-horizon trajectories from the environment. Such methods are limited by the length of the samples because the robot cannot plan for scenarios beyond this time horizon. Simply extending the time-horizon of sampled trajectories is not feasible as an increase in the time-horizon requires more sampled trajectories from the environment in order to maintain controller performance. On robots such as the AutoRally platform, which operate in real time with limited computational power, increasing the number of sampled trajectories is computationally intractable. This work improves the long-term planning capabilities of autonomous systems by augmenting cost-estimates of trajectories with a learned value of the terminal state. This learned value approximates the expected cost under the car's current control policy from the terminal state for an arbitrary time-horizon without requiring an increase in the number of samples. We show that this improves the lap times of the AutoRally platform.**

I. INTRODUCTION

Algorithms for autonomous driving have been at the forefront of both machine learning and robotics in recent years. A crucial prerequisite to achieving autonomous vehicles is designing algorithms to solve the controls problem: how do we output controls to our system that minimize some cost function (such as distance travelled, fuel consumed, or collisions encountered)?

At Georgia Tech, work on high speed autonomous driving has lead to the development of model predictive control (MPC) algorithms that demonstrably yield collision-free behavior despite challenging terrain conditions and limited computational power [1]. This work was performed on the AutoRally platform [2], a 1/5 scale battery-powered RC car driving around a dirt track. A motivating factor behind work on this platform was that research on high-speed systems can expose boundary conditions encountered by autonomous vehicles in real-world environments.

Model Predictive Control Algorithms, such as receding horizon Gauss-Newton LQR [3] and Model Predictive Path Integral (MPPI) [1], continuously re-optimize a simple model of the system over a short planning horizon to make reasonable short-term decisions. This continual re-optimization over control sequences allow MPC algorithms to handle modeling errors as well as changes in the environment. For example, MPPI allows the AutoRally platform to achieve speeds of 12 m/s on a dirt track where the wheels of the car are prone to slipping. These results are especially impressive because the dynamics of the vehicle are modeled with a small (2 layers with 32 nodes each) network and all computation is performed on board the vehicle itself.

It has been demonstrated that many MPC algorithms are actually a special case of dynamic mirror descent, an algorithm proposed in the online learning setting [4]. In the online learning framework an agent makes a decision only to have a loss revealed to it by the environment, much like how a controls algorithm outputs a motor control and then incurs a cost in terms of fuel or collision. The authors refer to the family of MPC algorithms that come from dynamic mirror descent as DMD-MPC (Dynamic Mirror Descent Model Predictive Control).

Lowrey et al.[5] show that Model Predictive Control methods can be improved with the help of a Value Function Approximator in their work POLO: Plan Online, Learn Offline. Whereas standard model predictive control methods are limited by their planning horizon, POLO uses information from the value function to better understand the costs of sampled trajectories. However, previous research in this space assumes access to a perfect dynamics model. This is impossible in the case of AutoRally where expensive dynamics models cannot be used in real-time and track conditions introduce uncertainty into how the system behaves.

This work will bridge the gap by extending existing infrastructure around DMD-MPC to learn and exploit a value function as is done in POLO. We will then study how this value function affects the quality of trajectories generated by the controller even when the value function is informed by an approximation of a dynamics model rather than the ground truth dynamics.

II. LITERATURE REVIEW

A. Model Predictive Path Integral

In response to the challenges of controlling a real-time system where the dynamics can only be approximated, researchers have proposed a series of algorithms that take advantage of advancements in computer hardware and machine learning. Put simply, the control theorists are solving the following problem: given the dynamics of a system (i.e a a model of how the robots state changes over time) and some cost function that measures the quality of a trajectory, what controls should the robot execute to minimize the cost function?

The intuitive approach to this problem is to sample a series of possible trajectories the robot could take, compute the cost of each trajectory, and then choose the the trajectory with the lowest cost. In Model Predictive Control (MPC), this is done iteratively at each timestep. The robot's understanding

---

[1] Georgia Institute of Technology
{`schintalapudi,nwagener`}`@gatech.edu`
[2] University of Washington `bboots@cs.washington.edu`

of its dynamics is not necessarily accurate, but by sacrificing accuracy this algorithm can be run in real-time. Furthermore, the evaluation of trajectories at each timestep allows the robot to self-correct in light of its inaccurate dynamics model.

One algorithm in the family of MPC techniques is the Model Predictive Path Integral Controller [1]. This controller, referred to as MPPI, solves the controls problem by formulating cost in terms of ideas from information theory, namely free energy and relative entropy. This formulation allows the authors to show that the resulting trajectory is actually an optimal solution with respect to minimizing the cost function. MPPI advanced the state of controls algorithms by allowing the autonomous agents to determine their next step online (in real-time), with just the hardware that could fit on a scale car. Previous approaches required knowledge of the environment beforehand to precompute optimal controls. However, MPPI makes large assumptions about our knowledge of the dynamics of the robot to make this guarantee. Not only does it assume the dynamics of the robot are known, this algorithm also assumes that these dynamics are control affine.

To mitigate this problem, the next move forward in sampling based methods was an Information Theoretic Approach to Model-Predictive Controls (MPC) [6]. This approach improves MPPI by removing the control affine assumption. This is an important step forward because there is no longer a need to derive a dynamics system from the physics of an environment and existing knowledge of the robot. Instead, the model comes from a data-driven approach to modeling the system; given data on how the robot drives under different controls a modeled is learned using a function approximator. In this case, a deep neural network with 2 hidden layers containing 32 neurons each was sufficient to capture the dynamics of a model RC-car accurately enough to consistently drive the car at 12 m/s. While this work overcomes one problem pertaining to its predecessor, namely the assumption that the system is controls-affine, it is still limited by the fact that the sampled trajectories must be finite. These trajectories then may not account for long-term changes in behavior that would be helpful to the vehicle. Du et al. find that as the length of the trajectories (the time-horizon) is expanded, an exponential number of samples is needed in order to maintain controller optimality [7]. However, this exponential increase is impractical on the AutoRally system which has limited computational ability onboard.

B. Value Functions over Markov Decision Processes

Reinforcement Learning researchers, such as [8] have been working on similar prescriptive analysis but approaching the problem with a slightly different framework called Markov Decision Processes (MDPs).

A Markov Decision Process is a 5-Tuple $\mathcal{M} = \{S, T, A, R, \gamma\}$ that describes the environment of an autonomous agent. Here, S is the state space and A is the set of actions. From this we define $T : S \times A \times S \rightarrow [0, 1]$ as a transition function that gives the probability of being in state s' given that action a was taken from state s, and $R : S \times A \rightarrow \mathbb{R}$ as a reward function. γ is a discount factor that controls how long-term the planning of the robot is. The problem of Reinforcement Learning (RL) is to learn a policy $\pi : S \rightarrow A$ that maximizes the reward encountered by an agent. Notice that the problem formulated by control theory, minimizing system cost given dynamics by outputting the appropriate control at a given state is equivalent to that of RL. Now system dynamics are given by the transition function, the cost function is simply an inverted reward function, and controls are referred to as outputs. As a result, RL algorithms can also provide promizing approaches to vehicle control.

A fundamental technique in Reinforcement Learning is *Value Iteration*. We define the Value function (parameterized by a policy π) as a mapping between a state in the MDP and the real numbers as follows:

$$V^{\pi}(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t, \pi(s_t)))|s_0 = s]$$

In value iteration, the agent repeatedly explores the environment until a predefined termination condition is met. Each rollout allows the agent to further refine its definition of the value function and propogate information about a given state to its neighboring statement. Value iteration can generate good policies in MDPs with a small, discrete state space. However in environments with large, continuous state spaces and high-entropy transition functions Value Iteration fails to converge. High speed driving with the AutoRally platform is one such example of an environment where Value Iteration will fail to converge.

Recent work in Deep Reinforcement Learning still aims to learn a Value Function, but uses the representational power of machine learning techniques to capture the complexities of a value function corresponding to a challenging environment. For example in [9], the authors formulate neural networks as a valid way to perform fitted value iteration. In discrete MDPs, a more common technique is to train a neural network to learn a Q-function, where a neural network learns to approximate the function

$$Q^{\pi}(s, a) = \mathbb{E}[R(s, a) + \sum_{t=1}^{\infty} \gamma^t R(s_t, \pi(s_t))|s_0 = s]$$

and then setting

$$V^{\pi}(s) = \max_{a} Q^{\pi}(s, a)$$

C. Plan Offline, Learn Online

Using Deep Reinforcement Learning techniques to learn a policy function yields better policies than more classical value iteration methods. However, a value function doesn't necessarily exploit knowledge of system dyanamics and more importantly, it does still suffer from approximation errors. Conversely, Model-Predictive Control methods such as DMD-MPC do optimize the cost function while considering system dynamics. DMD-MPC suffers from only being able to plan over a finite-time horizon, meaning it cannot formulate long-term plans in the same way value-function based policies do. Lowrey et al. [5], developed POLO: Plan

Offline, Learn Online to leverage the strength of both of these approaches.

In POLO, finite length trajectories are sampled and have their cost computed by summing over each state in the trajectory, as is done in MPPI. However, POLO then augments the cost with the result of evaluating the Value function at the last state of the rollout. Formally, we are now defining the reward of a sample trajectory as

$$\max_a \mathbb{E}[\sum_{t=0}^{N-1} \gamma^t r(s_t, a_t) + \gamma^N V(s_N) | s_0 = s]$$

This combines both the precise, local-optimization based nature of MPPI as well as the global information embedded in the Value function. The authors of this work evaluate POLO on a system where the dyanamics are known and so the Value function can be approximated with high precision. The goal of this work is to show that the insights of POLO generalize to systems where this assumption is removed. That is, even with a noisy Value Function generated from an approximated dynamics function, POLO outperforms control algorithms that do not leverage information from the Value Function.

III. METHODS

This work will be conducted by simulating the AutoRally Platform by using its dynamics model (a learned neural network) as a model of how the car really moves. The software to execute controls on the robot in the simulation is written in Python leveraging the ROS (Robot Operating System) framework. The implementation of the DMD-MPC (Dynamic Mirror Descent Model Predictive Control) algorithm is backed by PyTorch. This work extends the previous implementation of DMD-MPC by integrating a bridge between the ROS framework governing sensing and motor output of the vehicle and the PyTorch implementation generating controls for the system. While previous iteration of the DMD-MPC software either focused on simpler control systems or had limited capability of working with the AutoRally system, our approach allows any special case of DMD-MPC to be run on the robot. This work will then provide a formulation of POLO (Plan Online, Learn Offline) as an unexplored extension of DMD-MPC.

Our hypothesis is that exploiting the POLO framework, i.e taking advantage of an approximated value function, will enable augmented MPC methods to outperform those that are purely reactive in nature. The Value Function will be modeled with an ensemble of neural networks, each of which will only consist of 2 layers and a hyperbolic tangent activation function. The network will be trained via Adaptive Moment Estimation. It is assumed that the system has access to the costmap beforehand, which is the same assumption made in previous work. To train the value function, the robot will simulate rollouts across the costmap offline, using its learned dynamics model to train offline. This is to ensure that we can state the improvements from using POLO come from the existence of an approximated value function, not because the value function captured additional dynamics information.

Recall that DMD-MPC defined a class of algorithms, not one control strategy in general. The closest algorithm to POLO in the family of algorithms defined by DMD-MPC is MPPI (Model Predictive Path Integral), and as a result MPPI will be the specific baseline we compare against. We will conduct a number of trials of the vehicle running MPPI in a number of different environments. Note that MPPI can be considered a special instance of POLO where the expected utility of every terminal state is 0. We will then augment MPPI with the approximated value function and have the car drive an equal number of laps in simulation. We will compare overall lap times between POLO and baseline approaches.

A. State Space

The standard state space of the AutoRally platform is given as

$$[x, y, yaw, roll, \dot{x}, \dot{y}, -y\dot{a}w]$$

The purpose of a value function is to provide a mapping between the state and expected reward over time. We make a number of modifications to the state in order to facilitate learning this maping.

First, rather than tracking the position of the car as its x, y coordinates, we instead consider the car in terms of the progress around the track. We call this longitudinal measurement θ and it takes the value 0 at the beginning of the track and 2π at the end of the track. Because θ is discontinuous (it jumps from 2π to 0 at the track start line), we keep track of $\sin\theta$ and $\cos\theta$ instead.

For a given amount of progress around the track, we also need to keep track of whether the car is closer to the inside of the track or the outside of the track. This latitudinal value ranges from 0 to 1 and is also kept as part of the state.

Finally, we us a value ψ to to look at the orientation of the car rather than the yaw value. Specifically, we set ψ equal to the difference between the car's yaw from the original state and the yaw the car woul face if it were parallel with the track boundaries at that point. In the interest of tracking continuous values as we did with θ, we track $\cos\psi$ and $\sin\psi$ in the state. Thus, our new state definition is:

$$[\sin\theta, \cos\theta, \text{latitude}, \cos\psi, \sin\psi, \text{roll}, \dot{x}, \dot{y}, -y\dot{a}w]$$

B. Cost Function

The original cost function for MPPI for a state X on AutoRally was given as:

$$c(X) = w_1 \cdot (s_{des} - s)^2 + w_2 \cdot M_{xy} + w_3 \cdot \text{slip} + w_4 \cdot \phi(X)$$

Where s_{des} is the desired speed of the system, s is the actual speed of the system, M_{xy} is the costmap cost, "slip" penalizes the slippage of the car and $\phi(X)$ is an indicator of whether the car leaves the track. $w_1 ... w_4$ are the weights of each term. The slip angle is given as

$$-\arctan\frac{\dot{y}}{\dot{x}}$$

and is only nonzero when the slip angle exceeds a certain threshold. We modify this cost function as well to better capture the racing task.

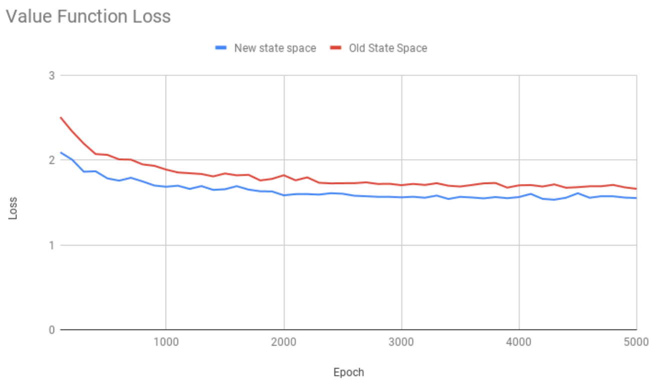

Fig. 1: Red, convergence with old state space. Blue, convergence with new state space

The first modification we make is adding a new term, $\dot{\theta}$, to the cost function. This measures the change in track progress (which we already computing in the state as discussed in subsection A). Computing this term requires knowledge of the next step in the trajectory, but MPPI computes costs with every state of the trajectory already rolled out. We also modify the first term of the cost function so that it is only nonzero when $s > s_{des}$. This incentivizes the car to not accelerate if it risks crashing into the track boundaries. Finally, we modify the behavior of ϕ so that if any state in a rollout leaves the trajectory, then all states incur the collision cost. This further incentivizes against crashing.

IV. RESULTS

A. *Learning a Value Function*

We train a neural network with two hidden layers consisting of 32 nodes each and a $\tanh$ nonlinearity to learn the value function. The loss curve for a network learning a value function from the original state space as opposed to our own state space is shown below.

From the curve we can see that learning a value function from the original state space takes does not converge to as well as learning a value function from our modified state space. This implies that our transformation of the state space facilitates learning the value function. Because the accuracy of our estimated cost per rollout is dependent on the accuracy of the value function, a good representation of the value function is a must.

B. *Quality of Trajectories*

Using this value function approximator, we race the car around the track with a target speed of 11 m/s to see if its racing performance is improved. The results over 10 laps or until collision are given in the table below:

	Lap Times
Old Cost Function	7.46 seconds
No Approximate Value Function	6.446 seconds
Approximate Value Function	6.28 seconds

First, we see that the changes made to the state and action space better encode the racing task than the original cost function as defined in the MPPI paper. Our new cost function does not force the car to stay at the center of the track as heavily, allowing for the car to make tighter turns. Comparing the new cost function with and without value function approximation, the results show that value function approximation does indeed improve the lap time of the vehicle.

Qualitatively, we can also observe the car being able to stay closer to the track boundary, enabling tighter turns. This is closer to human expert driving behavior. where the expert is fully aware of the importance of planning for turns in advance. It is important to note that we keep the number of samples fixed across the no-value-function scenario and the driving algorithm with the value function enabled.

V. CONCLUSION

Model Predictive Control algorithms have demonstrated the ability to control autonomous systems despite inaccuracies in the dynamics model and the limitation to computational power that is onboard the vehicle. However, these algorithms are limited by the need for exponentially more trajectories to be sampled as the planning horizon is increased (which leads to a lower controller frequency). This limitation means that the planning horizon cannot be extended to lengths comparable to how humans plan into the future, leading to lower quality trajectories.

Value function approximation addresses this gap by approximating the cost of the rollout over an infinitely long time horizon without having to sum over each state independently. Thus, the car exhibits behavior that would imply it is planning further into the future without suffering from exponentially growing computational costs. Qualitatively, we see this in how the car is preparing for turns by moving into the inner boundary of the track earlier on the straight part of the track. Quantitatively, we see that the lap times for the car are improved.

Future work will involve experimenting with alternative network architectures to learn the value function such as what is done by Tamar et al. [10]. It could also be fruitful to train the value function off of data obtained from the real world as well as the dynamics model. Richards and How [11] show that MPC is a valid control tool on unmanned aerial vehicles and Lowrey et al [5] show that is useful on humanoids. Therefore it stands to reason that this work on value function approximation for MPC (based on a noisy dynamics model) can extend to systems beyond racing cars as explored in this work. This can improve the control capability of a large number of already deployed autonomous systems.

A. *Real world experiments*

Part of this work has included extending an existing implementation of POLO so that it can deploy controls to the AutoRally platform, a physical robot. An image of the car running in the Gazebo simulator is shown in Figure. Here, the system's state comes from the car's state estimator, not

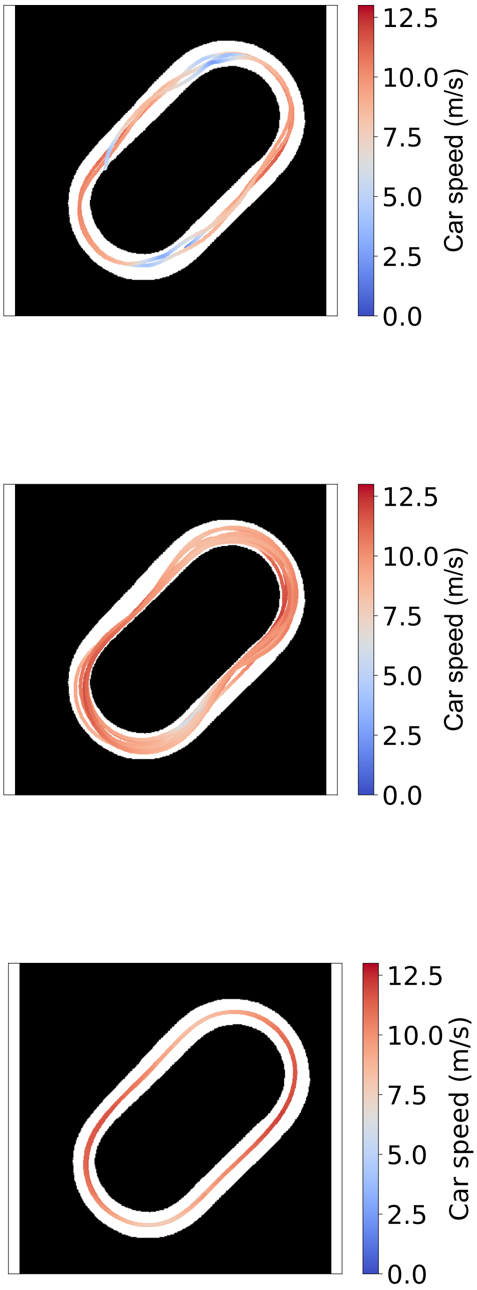

Fig. 2: Top: trajectories with old cost function, Middle: trajectories with new cost function, Bottom: trajectories with new cost function and value function approximation

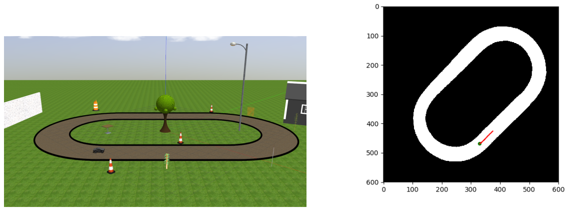

Fig. 3: AutoRally platform in Gazebo simulator along side visualization of the car's rollout

its dynamics model. In future work, we will take the car to outdoor venues to see how the car does subject to challenging conditions in nature.

VI. ACKNOWLEDGEMENTS

This work was done under the supervision of Dr. Byron Boots at the University of Washington and Nolan Wagener at Georgia Tech.

REFERENCES

[1] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on.* IEEE, 2016, pp. 1433–1440.

[2] B. Goldfain, P. Drews, C. You, M. Barulic, O. Velev, P. Tsiotras, and J. M. Rehg, "Autorally an open platform for aggressive autonomous driving," *arXiv preprint arXiv:1806.00678*, 2018.

[3] P. Abbeel, A. Coates, and A. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *IJRR*, 2010.

[4] N. Wagener, C.-A. Cheng, J. Sacks, and B. Boots, "An online approach to model predictive control," *arXiv preprint arXiv: 1902.08967*, 2019.

[5] K. Lowrey, A. Rajeswaran, S. Kakade, E. Todorov, and I. Mordatch, "Plan online, learn offline: Efficient learning and exploration via model-based control," *arXiv preprint arXiv:1811.01848*, 2019.

[6] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. Rehg, B. Boots, and E. Theodorou, "Information theoretic MPC for model-based reinforcement learning." in *Proceedings of the 2017 IEEE Conference on Robotics and Automation (ICRA)*, 2017.

[7] S. S. Du, S. M. Kakade, R. Wang, and L. F. Yang, "Is a good representation sufficient for sample efficient reinforcement learning?" *ArXiv*, vol. abs/1910.03016, 2019.

[8] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[9] R. Munos and C. Szepesvári, "Finite-time bounds for fitted value iteration," *J. Mach. Learn. Res.*, vol. 9, pp. 815–857, Jun. 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1390681.1390708

[10] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *NIPS*, 2016.

[11] A. Richards and J. How, "Decentralized model predictive control of cooperating uavs," in *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, vol. 4, Dec 2004, pp. 4286–4291 Vol.4.