Proceedings of Machine Learning Research vol 120:1–14, 2020

# Practical Reinforcement Learning For MPC: Learning from sparse objectives in under an hour on a real robot

**Napat Karnchanachari** [1 2 *] PATK@ETHZ.CH

**Miguel I. Valls** [1 *] MIGUEL.DELAIGLESIAVALLS@SEVENSENSE.CH

**David Hoeller** [2 3] DHOELLER@ETHZ.CH

**Marco Hutter** [2] MAHUTTER@ETHZ.CH

## Abstract

Model Predictive Control (MPC) is a powerful control technique that handles constraints, takes the system's dynamics into account, and optimizes for a given cost function. In practice, however, it often requires an expert to craft and tune this cost function and find trade-offs between different state penalties to satisfy simple high level objectives. In this paper, we use Reinforcement Learning and in particular value learning to approximate the value function given only high level objectives, which can be sparse and binary. Building upon previous works, we present improvements that allowed us to successfully deploy the method on a real world unmanned ground vehicle. Our experiments show that our method can learn the cost function from scratch and without human intervention, while reaching a performance level similar to that of an expert-tuned MPC. We perform a quantitative comparison of these methods with standard MPC approaches both in simulation and on the real robot.

A demonstration of our method can be seen in the video: https://youtu.be/PJB8XdXBP_M

**Keywords:** Reinforcement Learning, Model Predictive Control, Autonomous Robots

## 1. Introduction

Model Predictive Control (MPC) is a trajectory optimization technique that has gained immense popularity over the last decades due to its ability to tackle inherently hard control problems (Lee (2011)). The theory is well understood and it is proven to be stable and optimal for a large variety of systems (Lee (2011)). MPC has been widely adopted due to algorithmic and technological advances. It can run in real time on a robot's on-board computing unit, allowing for applications such as autonomous racing (Kabzan et al. (2019)), aggressive flight maneuvers with drones (Mueller and D'Andrea (2013)), and legged locomotion (Neunert et al. (2018)).

In practice, however, it is well known that the cost functions for MPC have to be tuned. Experts craft specific costs that are a proxy for the original high level objectives, but also use costs that help the optimization converge, and avoid exploiting unmodelled or uncertain system dynamics. We refer to the latter as regularization cost terms. Finding a trade-off among regularization and proxy costs can be extremely difficult and time consuming (Garriga and Soroush (2010)).

On the other side of the spectrum, Reinforcement Learning (RL) has shown to be a powerful tool, not only capable of handling binary and sparse rewards, but also overcoming the credit assignment problem when dealing with long horizons (Silver et al. (2016); Lillicrap et al. (2015); Schulman et al. (2017)).

* Both authors contributed equally. [1] Sevensense Robotics A.G. [2] Robotic Systems Lab, ETH Zurich. [3] NVIDIA

© 2020 N. Karnchanachari, M.I. Valls, D.H. & M. Hutter.

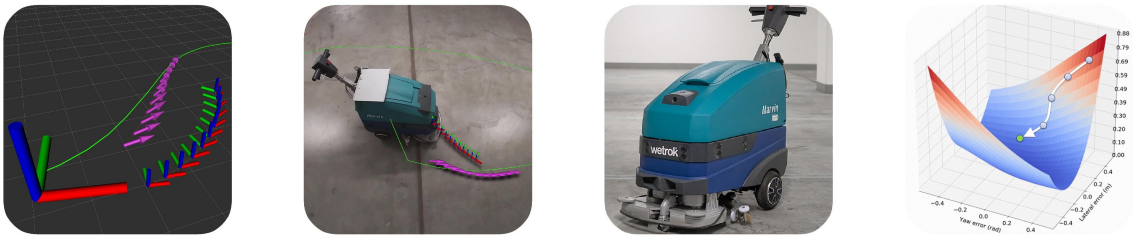


Figure 1: From left to right: simulation of the UGV, the real life UGV platform with visualization overlayed, and lastly an example of a 3D representation of the learned value function.

Recent approaches such as Plan Online, Learn Offline (POLO, Lowrey et al. (2019)) or Deep Value Model Predictive Control (DMPC, Farshidian et al. (2019)) attempt to combine best of both worlds by employing trajectory optimization with value function estimation. In this paper, we extend these works and learn to solve tasks defined by simple and easily interpretable high level objectives on a real unmanned ground vehicle (UGV). This is reputably challenging for most MPC algorithms because such objectives are represented by sparse and binary rewards. With this method, we can exploit our knowledge of the system dynamics and also endow the optimizer with a representation of the value landscape to solve the task in a sample efficient manner. The main contributions of this paper are:

- Presenting a practical extension of deep value function learning that outperforms a baseline MPC and is comparable to an expert-tuned MPC, trained from scratch on the real physical system in under 30 min with on-board CPU only.
- Showing that such learning based methods can be trained from high level binary and sparse rewards only, in under an hour, outperforming hand-tuned dense rewards learning.
- A comparison of two learning based techniques with standard MPC algorithms on a dynamical system with an uncertain model, in simulation and on a real system for trajectory tracking.

In the remainder of this paper, we introduce the background in Section 2, describe the method in Section 3 and then proceed with the experiments in Section 4. We conclude with a review on related work in Section 5 and the conclusion in Section 6.

## 2. Background

### 2.1. Notation and Definitions

In this problem setting, we consider an agent within an environment $E$ whose task is to maximize the discounted expected sum of rewards collected from the current time onwards. This is modelled by a Markov Decision Process (MDP) described by the state $\boldsymbol{s}_t \in \mathbb{R}^{n_s}$ and dynamics $d\boldsymbol{s} = f(\boldsymbol{s}_t, \boldsymbol{a}_t)$. At each time step, the agent observes a state $\boldsymbol{s}_t$ and takes an action $\boldsymbol{a}_t \in \mathbb{R}^{n_a}$ according to the policy $\pi(\boldsymbol{s}_t)$. The environment returns the corresponding evolved observations $\boldsymbol{s}_{t+1}$ , as well as a reward $r_t \in \mathbb{R}$. This forms a transition tuple $\langle \boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1} \rangle$ for one time step. All transitions are stored in a replay buffer $\mathcal{B}$. The return is defined by the discounted future rewards $R = \sum_{t=0}^{\infty} \gamma^t r_t(\boldsymbol{s}_t)$, where the discount factor is $\gamma \in [0, 1)$. The value function $V(s)$ describes the expected return of being in state $s$, i.e. $V(s_0) = \mathbb{E}_\pi[R|\, s = \boldsymbol{s}_0]$. $V(s)$ is assumed to be time independent. The term cost is used as negative value, rendering minimizing cost an maximising value equivalent statements.

### 2.2. Model Predictive Control

MPC is a receding horizon control technique that maximizes a value function with respect to a sequence of control actions $\boldsymbol{a}_0 \dots \boldsymbol{a}_{N-1}$ along a horizon of length $N$. The problem is constrained under state dynamics $\hat{f}(\boldsymbol{x}_t, \boldsymbol{a}_t)$, and state and input constraints. Note that here, $\boldsymbol{x} \in \boldsymbol{S}_k \subseteq \mathrm{R}^{n_s}$ and $\hat{f}(\boldsymbol{x}_t, \boldsymbol{a}_t)$ denote the state and state dynamics of the actor respectively, which are distinguished from the true state and state dynamics of the environment. This results in the optimization

$$\pi_{MPC}(\boldsymbol{x}_0) = \underset{\boldsymbol{a}_0,\dots,\boldsymbol{a}_{N-1}}{\arg\max}\ \gamma^N g_N(\boldsymbol{x}_N) + \sum_{k=0}^{N-1} \gamma^k g(\boldsymbol{x}_k, \boldsymbol{a}_k) \tag{1}$$

$$\text{s.t.} \qquad \partial_t x_k = \hat{f}(\boldsymbol{x}_k, \boldsymbol{a}_k),\ \boldsymbol{a}_k \in \boldsymbol{A}_k,\ \boldsymbol{x}_k \in \boldsymbol{S}_k,\ \text{for } k = 0, \dots, N,$$

where $\boldsymbol{x}_0 \in \mathrm{R}^{n_s}$ denotes the current state feedback and $\boldsymbol{a}_k \in \boldsymbol{A}_k \subseteq \mathrm{R}^{n_a}$ the control action, of which only the first is applied. This is then repeated for every control cycle. Notice that this MPC strategy is approximating the initial problem as a finite horizon optimal control problem, which depending on the horizon choice, is not guaranteed to be stable (Lee (2011)). However, as will be seen in Section 3.2, defining the terminal ($g_N(\cdot)$) and stage ($g(\cdot)$) costs in terms of the value function allows us to alleviate this limitation.

### 2.3. Value Function learning

Value function learning is commonly employed in reinforcement learning problems (Sutton and Barto (2018)). In most cases, the value is represented by a state value function it $V_\theta(\boldsymbol{s}_t)$ or an state action value function $Q_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t)$, parameterized by parameter vector $\theta$. This function is optimized minimizing the mean squared error loss with respect to a learning target, $y_t$ :

$$L = \mathbb{E}_{\pi_\theta}[(y_t - V_\theta(\boldsymbol{s}_t))^2]. \tag{2}$$

## 3. Method

The presented method is based off the actor-critic framework. The critic captures the global value function represented by a neural network. The actor is represented by a non-linear model predictive controller. The focus lays on the contributions that make it possible to run on a real physical system. The learning algorithm can be found in Appendix A.

### 3.1. Critic

The key components to training the critic that are listed next. Note, this does introduce a new set of hyper parameters but were observed to have marginal effect compared to the value function.

**Gradient regularization** In the value function loss (Equation 2), we also regularize the Jacobian of the network. Since the Jacobian, $\boldsymbol{J} = \partial_{\boldsymbol{s}} V_\theta(\boldsymbol{s})$, and the Hessian of the network are required on the actor side (See Section 3.2), adding such a term will favor a smoother class of functions, which are easier to optimize for QP solvers. Note that the Hessian is often approximated as $\boldsymbol{J}^T\boldsymbol{J}$ in the Gauss-Newton Hessian approximation (Björck (1996)). Therefore, regularizing the Jacobian norm indirectly reduces the Hessian norm, and also aids numerical stability while training. Without this regularization, we found that running MPC with the learned value while training often did not converge. This resulted in unstable behavior, making it difficult to run on a real system.

**Experience replay and data augmentation** Experience replay is advantageous in two ways: it increases sample efficiency, and it stabilizes neural network training. This is realized in the form of replay buffer $\mathcal{B}$, see (Silver et al. (2014)). Moreover, we augment our data profiting from the symmetries of the system (e.g. $V_\theta(\boldsymbol{x}) = V_\theta(h(\boldsymbol{x}))$). This not only implies that we can augment our data by the number of symmetries in the system but also that the agent will behave similarly well for symmetric states despite not having visited some of them.

**n-step target** An n-step target is employed, it is the bootstrapped sum of discounted rewards over n consecutive steps (Sutton and Barto (2018)):

$$R^{(n)}(\boldsymbol{s}_t) = \sum_{i=0}^{n-1} \gamma^i r(\boldsymbol{s}_{t+i}) + \gamma^n V_\theta(\boldsymbol{s}_{t+n}). \tag{3}$$

This choice is due to its ability to balance bias and variance which affects Monte-Carlo and TD(0) returns respectively and in practice, it accelerates convergence.

**Target network** We maintain a target network $V_{\theta'}(s)$ along side the critic network $V_\theta(s)$. Effectively, the Polyak-averaged version of the critic's estimated value function is used for bootstrapping. As shown by Lillicrap et al. (2015), this trick greatly improves the actor-critic learning interaction.

### 3.2. Actor

RL and MPC can be combined in several ways further explained in Section 5. In this section we focus on two methods that employ value function learning.

**Terminal Deep Value MPC** The first and most intuitive combination (presented in Lowrey et al. (2019)) uses the value function as terminal cost for the MPC. They show that bootstrapping the trajectory optimizer with the value function enables it to find global optimal solutions. Indeed, the value provides the missing information about the expected return from the end of the optimization horizon onwards. In this formulation, Equation 1 takes the following form:

$$\pi_{TDMPC}(\boldsymbol{x}_0) = \underset{\boldsymbol{a}_0 \dots \boldsymbol{a}_{N-1}}{\arg\max} \gamma^N V_\theta(\boldsymbol{x}_N) + \sum_{k=0}^{N-1} \gamma^k g(\boldsymbol{x}_k, \boldsymbol{a}_k). \tag{4}$$

Lowrey et al. (2019) use MPPI (Williams et al. (2017)) to solve the MPC problem. Here, we use a Sequential Quadratic Programming (SQP) formulation that offers several key advantages, which are detailed later in this section. In the experiments, we refer to this variant as TDMPC.

**Deep Value MPC** The second formulation (presented by Farshidian et al. (2019)) extracts both the stage and terminal cost from the learned value function. Hence, DMPC is handle able to sparse and binary rewards. The intuition behind our modification is that the stage cost represents the value gained between consecutive time steps, and when taken to the limit it can be expressed as as the time integral between two steps of the derivative of the value function with respect to time. Formally, this is referred as the Lie derivative of a function $f(\cdot)$, $\mathcal{L}_f$. The stage cost then takes the form:

$$\begin{aligned} g(\boldsymbol{x}_t, \boldsymbol{a}_t) := V_\theta(\boldsymbol{x}_{\boldsymbol{t+\Delta T}}) - V_\theta(\boldsymbol{x}_{\boldsymbol{t}}) &= \int_{\tau=t}^{\tau=t+\Delta T} \mathcal{L}_f V_\theta(\boldsymbol{x}_t)\, d\tau = \int_{\tau=t}^{\tau=t+\Delta T} \partial_x V_\theta(\boldsymbol{x}_t)^T \partial_t \boldsymbol{x}_t \, d\tau \\ &= \int_{\tau=t}^{\tau=t+\Delta T} \partial_x V_\theta(\boldsymbol{x}_t)^T \hat{f}(\boldsymbol{x}_t, \boldsymbol{a}_t)\, d\tau \approx \Delta T\, \partial_x V_\theta(\boldsymbol{x}_t)^T\, \hat{f}(\boldsymbol{x}_t, \boldsymbol{a}_t). \end{aligned} \tag{5}$$

In Farshidian et al. (2019), the authors solve the MPC problem with an algorithm known as SLQ (Sideris and Bobrow (2005)), which does not handle constraints systematically and is sensitive to initialization (does not globalize well). SQP on the other hand, efficiently solves both issues. The MPC formulation (Equation 1) is expressed as:

$$\pi_{DMPC}(\boldsymbol{x}_0) = \underset{\boldsymbol{a}_0,\ldots,\boldsymbol{a}_{N-1}}{\arg\max}\, \gamma^N V_\theta(\boldsymbol{x}_N) + \Delta T \sum_{k=0}^{N-1} \gamma^k \partial_x V_\theta(\boldsymbol{x}_t)^T \hat{f}(\boldsymbol{x}_k, \boldsymbol{a}_k). \tag{6}$$

**Quadratic Program approximation** A popular approach to solve non-linear optimization problems is SQP, it solves a sequence of QP approximations of the non-linear problem. We chose this approach to solve the MPC problem due to its ability to handle state-input constraints compared to SLQ. SQP run is time efficient if tailored QP solvers are used it is robust to initial guesses. For our implementation we used the ACADO toolkit toolkit (Quirynen et al. (2014)) with the QP solver qpOASES (Ferreau et al. (2008)). Note, that ACADO does use the real-time iteration scheme, which only solves one QP per time ste. This further helps to reduce the computational load, for an in depth analysis of the guarantees of this method see Diehl et al. (2002).

Since the presented value function is differentiable, one could directly find the first and second order derivatives. However, these approximations were found to be numerically unstable, which is due to the fact that the stage cost Hessian uses the third order derivative of the original value function. To mitigate this problem further, a Gauss-Newton approximation of the Hessian has been chosen with a Levenberg-Marquadrdt style damping factor (Marquardt (1963)), which also guarantees that the Hessian is positive semi-definite.

## 4. Experiments

### 4.1. Experimental setup

**Platform** The differential drive UGV used is shown in Figure 1. The on-board visual-inertial SLAM system from Sevensense Robotics[1] provides an estimate of the pose and velocities of this system that are used as feedback to control the robot in real time and to report the results. Training is run on an on-board quadcore Intel i7-6600U CPU at 2.60GHz.

**System model** The model used by the MPC controller is a unicycle model with limited velocities and accelerations presented in the following equation:

$$\hat{f}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \left[\dot{x},\, \dot{y},\, \dot{\psi},\, \dot{v},\, \dot{\omega}\right]^T = [v\cos(\psi),\, v\sin(\psi),\, \omega,\, a,\, \alpha]^T \tag{7}$$

where $x$ and $y$ are the position in cartesian coordinates, $\psi$ is the 2D rotation, and $v$ and $\omega$ are the linear and angular velocities in body frame. $a$ and $\alpha$ are the linear and angular accelerations in body frame. Velocities are the input to the UGV, accelerations are only included into the optimization to achieve a smooth behaviour with bounded velocity rates.

**Error coordinates** In order to formulate the trajectory tracking objective, we perform a change of coordinates to trajectory error coordinates, where the coordinates are expressed with respect to the reference. Further details can be found in Appendix C.

1. www.sevensense.ch

**Experiment formulation** The objective of all experiments is to accurately track a given reference trajectory. We compare 4 controllers, two learning based , TDMPC, DMPC (see Section 3) and two classical MPC controllers. The first one is referred to as Naive MPC, which uses the reward as stage cost and terminal cost. Naive MPC serves as the baseline controller for comparision. The second one is referred to as Expert MPC, it was the controller deployed by Sevensense Robotics, where the diagonal weights of the cost function had been hand tuned by their control team.

### 4.2. Model mismatch experiment

First, we study how the learned value function improves tracking performance in presence of a perturbed system model, without modifying or learning such a model. Consider imperfectly modelled turning dynamics that are represented by a first-order system, where the turning delay is parameterized by a time constant $\tau$. The learning based methods are trained with $\tau = 0.6s$. In Figure 2, it can be seen that learning based methods perform well even in scenarios far from their training regime; thus learning to be cautious due to the unmodelled dynamics. The Naive and Expert MPC perform well when the model is accurate but quickly deteriorate and even become unstable with $\tau = 0.8s$.

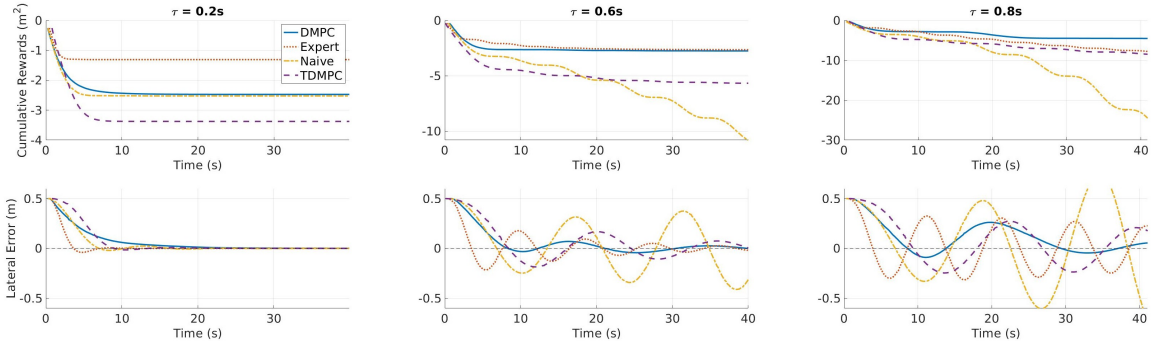


Figure 2: Model mismatch experiment. From left to right, the plots show the performance of the methods for increasing time constants $\tau = \{0.2, 0.6, 0.8\}s$. Top row: accumulated rewards over an episode given a dense reward equal to the tracking error squared. Bottom row: tracking response. DMPC and TDMPC are trained in simulation with $\tau = 0.6s$ but the model still assumes perfect turning dynamics for all methods.

### 4.3. Training on a real world UGV

TDMPC is capable of learning in the real world on an UGV trained from scratch. This is mostly due to the sample efficiency of the actor i.e TDMPC. It is able to make the learning process converge in within 10000 training iterations, collecting around 3000 samples. In addition, due to the basic knowledge of the model, TDMPC can drive the UGV around the track even if the model is poorly known and the deviations are large at the beginning. The tracking error can be seen in Figure 3 over the training period. Once trained, the learned value function can also be used to run DMPC.

### 4.4. Dense reward on real world UGV

In this experiment, we analyze the performance of all controllers for several paths that cover most typical maneuvers (see Appendix D). The reward given to the learning based methods is the squared deviation from the path. It can be seen how the Expert MPC behaves best on the real UGV while

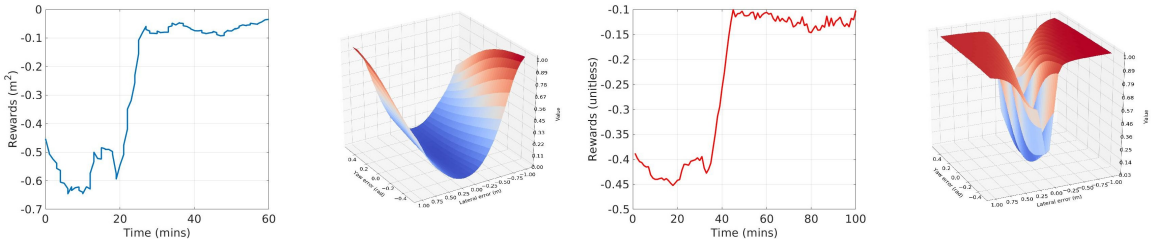


Figure 3: Average episode reward over training time with dense rewards (left) along with a 3D representation of the value function and similarly with sparse rewards (right). It can be seen that training with sparse rewards takes longer to converge compared to dense rewards, but still takes less than 45 minutes to converge. Moreover, the shape of the value functions correspond to the rewards given; sparse rewards being much steeper and slimmer compared to dense rewards.

DMPC consistently outperforms TDMPC. Both learning methods perform consistently better than Naive MPC. It is worth mentioning that DMPC has a damped behavior when approaching the reference in the straight path with no oscillation (see Appendix D). For the curves and the tight turn scenario, DMPC starts as good as the Expert MPC but eventually accumulates more error.

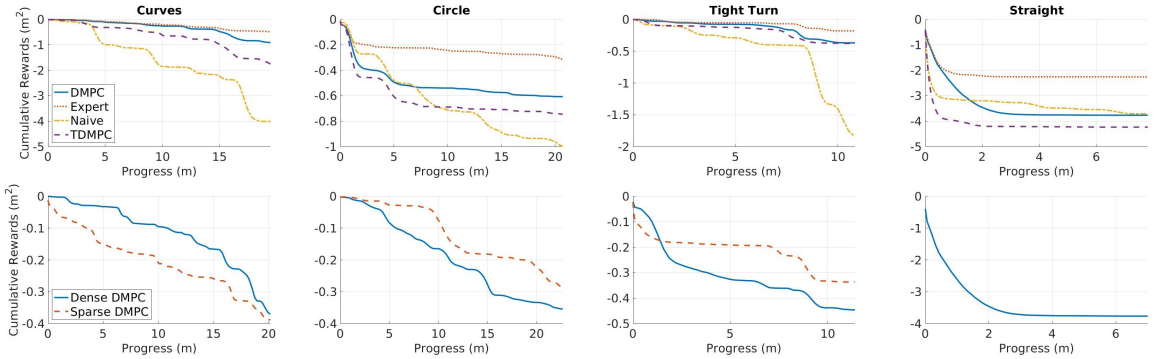


Figure 4: Top row: Cumulative rewards over an episode on different trajectories given a dense reward (tracking error squared). Progress means the projected distance travel along the reference path. Bottom row: Cumulative rewards of the DMPC for dense and sparse reward set-up respectively.

### 4.5. Sparse rewards on real world UGV

Neither Naive, Expert nor TDMPC can handle binary rewards since the cost function has to be differentiable for most efficient solvers. However, DMPC can handle this and we give a binary reward only when close to the reference path. We refer to this as sparse DMPC. This intuitive requirement leads to a successful tracking performance without the need to tune the cost on tracking error. Results can be seen in Figure 4. It can be seen how dense DMPC tracks comparatively worse than sparse DMPC. In this experiment the value function has been trained solely on sparse binary rewards; receiving rewards when the UGV was closer than 10cm to the path (see Appendix B). It has to be noted that DMPC trained on sparse rewards was not able to complete the straight path experiment starting 0.5m away, as it just stood still. This is probably due to the value function loosing gradient information when very far from the reward. In theory, the reward should be propagated without problems due to the bootstrapping of the target with the value function but in practice the value function flattens out, see Figure 3.

## 5. Related Work

Closely related to our work, one can find POLO (Lowrey et al. (2019)) and DMPC (Farshidian et al. (2019)), which use the learned value function in MPC for the terminal cost, and for the stage and terminal cost, respectively. Our pipeline builds on these works and extends them with practical elements that make it possible to run them on real physical systems. In contrast to this work, Lowrey et al. (2019) maintains several value functions to encourage exploration. Moreover, Lowrey et al. (2019) used MPPI (Williams et al. (2017)) as the optimizer. It has the advantage that rewards do not need to be differentiable, as opposed to SQP and SLQ based methods, suffers from the curse of dimensionality. In addition, there are no guarantees that local maxima of the value function will be found. In Farshidian et al. (2019), the authors employ a SLQ (Sideris and Bobrow (2005)), which is known to scale well with system dimensions, but is very sensitive to the initial guess and it is not trivial to include state constraints.

Another common learning approach in presence of uncertainty is to learn the system's model from data. In this category, Gaussian-Processes (GP) have been successfully used and demonstrated in real and miniature race cars (Kabzan et al. (2019); Hewing and Zeilinger (2017)), but also using deep neural networks (Chua et al. (2018)). GPs have also been used to model disturbances in order to improve tracking performance (Ostafew et al. (2014)). Linking value learning and model learning, Gros and Zanon (2019) pose MPC as a parameterized function approximator, allowing for the entire MPC problem to be subjected to the learning problem. This approach requires for the MPC parameters to be carefully selected for learning.

There are paradigms where the agent learns from expert demonstration. Notable ones are imitation learning (IM) (Hussein et al. (2017)) and inverse reinforcement learning (IRL) (Ng et al. (2000)). It has been shown that combining trajectory optimizion with IM can shorten training time of neural networks (Mordatch and Todorov (2014); Levine and Koltun (2013)). A shortcoming of IM and IRL is that the performance of the agent is often bounded by the performance of the expert.

Last but not least, learning based methods are often susceptible to exploring unsafe areas of the state space and it is difficult to guarantee that such methods will not find itself these unsafe areas. Works of Berkenkamp et al. (2017) estimate the region of attraction in order to provide guarantees during the learning process. Applying safe learning to MPC, Koller et al. (2018) and Rosolia and Borrelli (2017) construct safe terminal sets based on collected experiences. This could be a natural extension of our work as we currently do not provide any guarantees during training.

## 6. Conclusion and Future Work

We presented a sample efficient practical RL scheme capable of training from scratch on a real world physical platform. The algorithm performs value function learning, which is used as the MPC cost function. We use an SQP to solve the corresponding MPC problem. The algorithm is able to learn from given high level objectives including sparse requirements. We demonstrate learned cautious behaviour of the agent in simulation. In practice, it is able to run in real time and matches the performance of expert tuned controllers. The numerical extension presented has shown to be well suited for trajectory tracking. It would prove insightful to explore more challenging scenarios e.g. obstacle avoidance, in future work. Additionally, one could employ a more sophisticated sampling techniques such as prioritized sampling to take advantage of the lesser seen interesting samples.

## Acknowledgments

The authors thank Farbod Farshidian, Renaud Dubé, Gianluca Cesari and Alex Liniger for their support.

## References

Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 908–918. Curran Associates, Inc., 2017.

Åke Björck. *Numerical methods for least squares problems*. SIAM, 1996.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 4754–4765. Curran Associates, Inc., 2018.

Moritz Diehl, H Georg Bock, Johannes P Schlöder, Rolf Findeisen, Zoltan Nagy, and Frank Allgöwer. Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations. *Journal of Process Control*, 12(4):577–585, 2002.

Farbod Farshidian, David Hoeller, and Marco Hutter, editors. *Deep Value Model Predictive Control*, 2019. doi: 10.3929/ethz-b-000368961. 3rd Conference on Robot Learning (CoRL 2019); Conference Location: Osaka, Japan; Conference Date: October 30 - November 2, 2019.

H.J. Ferreau, H.G. Bock, and M. Diehl. An online active set strategy to overcome the limitations of explicit mpc. *International Journal of Robust and Nonlinear Control*, 18(8):816–830, 2008.

Jorge L Garriga and Masoud Soroush. Model predictive control tuning methods: A review. *Industrial & Engineering Chemistry Research*, 49(8):3505–3515, 2010.

Sébastien Gros and Mario Zanon. Data-driven economic nmpc using reinforcement learning. *IEEE Transactions on Automatic Control*, 2019.

Lukas Hewing and Melanie N. Zeilinger. Cautious model predictive control using gaussian process regression. *CoRR*, abs/1705.10702, 2017. URL http://arxiv.org/abs/1705.10702.

Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):21, 2017.

J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger. Learning-based model predictive control for autonomous racing. *IEEE Robotics and Automation Letters*, 4(4):3363–3370, Oct 2019. ISSN 2377-3774. doi: 10.1109/LRA.2019.2926677.

Juraj Kabzan, Miguel de la Iglesia Valls, Victor Reijgwart, Hubertus Franciscus Cornelis Hendrikx, Claas Ehmke, Manish Prajapat, Andreas Bühler, Nikhil Gosala, Mehak Gupta, Ramya Sivanesan, et al. Amz driverless: The full autonomous racing system. *arXiv preprint arXiv:1905.05150*, 2019.

Torsten Koller, Felix Berkenkamp, Matteo Turchetta, and Andreas Krause. Learning-based model predictive control for safe exploration. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 6059–6066. IEEE, 2018.

Jay H. Lee. Model predictive control: Review of the three decades of development. *International Journal of Control, Automation and Systems*, 9(3):415, Jun 2011. ISSN 2005-4092. doi: 10.1007/s12555-011-0300-6. URL https://doi.org/10.1007/s12555-011-0300-6.

Sergey Levine and Vladlen Koltun. Guided policy search. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1–9, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL http://proceedings.mlr.press/v28/levine13.html.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Kendall Lowrey, Aravind Rajeswaran, Sham Kakade, Emanuel Todorov, and Igor Mordatch. Plan online, learn offline: Efficient learning and exploration via model-based control. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Byey7n05FQ.

Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

Igor Mordatch and Emo Todorov. Combining the benefits of function approximation and trajectory optimization. In *Robotics: Science and Systems*, volume 4, 2014.

M. W. Mueller and R. D'Andrea. A model predictive controller for quadrocopter state interception. In *2013 European Control Conference (ECC)*, pages 1383–1389, July 2013. doi: 10.23919/ECC.2013.6669415.

M. Neunert, M. Stäuble, M. Giftthaler, C. D. Bellicoso, J. Carius, C. Gehring, M. Hutter, and J. Buchli. Whole-body nonlinear model predictive control through contacts for quadrupeds. *IEEE Robotics and Automation Letters*, 3(3):1458–1465, July 2018. ISSN 2377-3774. doi: 10.1109/LRA.2018.2800124.

Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

Chris J Ostafew, Angela P Schoellig, and Timothy D Barfoot. Learning-based nonlinear model predictive control to improve vision-based mobile robot path-tracking in challenging outdoor environments. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4029–4036. IEEE, 2014.

R. Quirynen, B. Houska, M. Vallerio, D. Telen, F. Logist, J. Van Impe, and M. Diehl. Symmetric Algorithmic Differentiation Based Exact Hessian SQP Method and Software for Economic MPC. In *Conference on Decision and Control*, 2014.

Ugo Rosolia and Francesco Borrelli. Learning model predictive control for iterative tasks. a data-driven control framework. *IEEE Transactions on Automatic Control*, 63(7):1883–1896, 2017.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Athanasios Sideris and James E Bobrow. An efficient sequential linear quadratic algorithm for solving nonlinear optimal control problems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 2275–2280. IEEE, 2005.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China, 22–24 Jun 2014. PMLR. URL http://proceedings.mlr.press/v32/silver14.html.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

Grady Williams, Andrew Aldrich, and Evangelos A. Theodorou. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40(2):344–357, 2017. doi: 10.2514/1.G001921. URL https://doi.org/10.2514/1.G001921.

## Appendix A. Deep Value MPC Training Algorithm

The Deep Value MPC training process is split into two alternating phases: the roll-out phase and the training phase.

**Roll-out phase** The roll-out phase generates transition samples$\langle \boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1} \rangle$, which are stored in the replay buffer $\mathcal{B}$. The samples are generated by taking a control action $\boldsymbol{a}_t$ according to $\pi_{TDMPC}(\boldsymbol{s}_t)$ with the latest learned value function.

**Training phase** Using the samples stored in the replay buffer, the target is calculated by using n-step return where the bootstrapping is evaluated using the target network $V_{\theta'}(\boldsymbol{s}_t)$ to stabilize the learning process, specified at equation 3. Once the training iteration is complete the value function for MPC and the target network are updated. This process then repeats until convergence.

Initialize $\theta, \theta', n, \boldsymbol{s}_0$
**for** $t = 0...T$: **do**
  Take action $\boldsymbol{a}_t = \pi_{TDMPC} s(_t)$ with cost function given by equation 4
  Observe new state $\boldsymbol{s}_{t+1} \sim P(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$ and reward $r_t$
  Store transition tuple in replay buffer $\mathcal{B}$
  **if** *if mod(t,Z) = 0* **then**
    Sample mini batch $M$ from replay buffer $\mathcal{B}$
    Update critic by minimising loss:
    $L(\boldsymbol{s}) = \frac{1}{M}\sum_i^M (R^n(\boldsymbol{s}_i) - V_\theta(\boldsymbol{s}_i))^2 + \beta||\partial_s V_\theta(\boldsymbol{s}_i)||_2^2 + \lambda||\theta||_2^2$
    Update target critic:
    $\theta' \leftarrow \theta'(1-\tau) + \theta\tau$
  **end**
**end**

**Algorithm 1:** Deep Value MPC Training

## Appendix B. Reward Engineering

### B.1. Dense Rewards

$$r_{dense}(s) = -\left[\left(x^F_{error}\right)^2 + \left(y^F_{error}\right)^2\right]$$

### B.2. Sparse Rewards

$$r_{sparse}(s) = \begin{cases} -0.5 & \text{if } |x^F_{error}| > 0.1 \text{ or } |\, y^F_{error}| > 0.1 \\ 0 & otherwise \end{cases}$$

## Appendix C. Error Coordinates

The state of the MDP, $\boldsymbol{s}$, is expressed in terms of frenet or error frame with respect to the reference trajectory and the control point $P$. The high level objective is to track the reference trajectory with the control point.

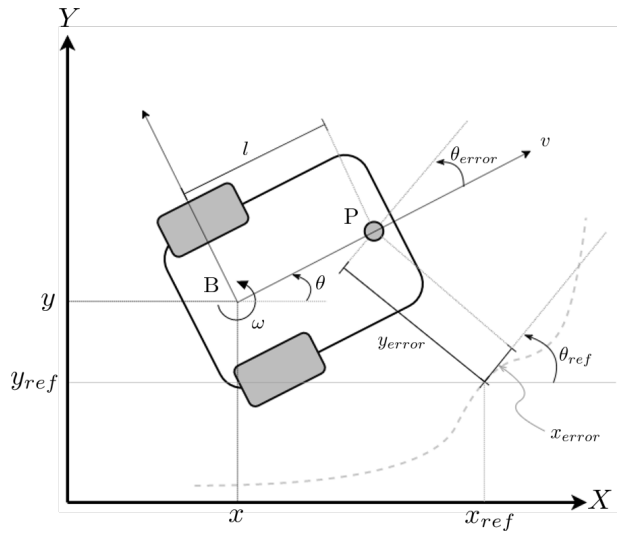


Figure 5: Kinematic Differential Drive Model and error coordinates

$$\begin{bmatrix} x^F_{error} \\ y^F_{error} \end{bmatrix} = \begin{bmatrix} x^M_{error}\cos(\psi_{ref}) + y^M_{error}\sin(\psi_{ref}) \\ -x^M_{error}\sin(\psi_{ref}) + y^M_{error}\cos(\psi_{ref}) \end{bmatrix}, \begin{bmatrix} x^M_{error} \\ y^M_{error} \end{bmatrix} = \begin{bmatrix} x + l\cos(\psi) - x_{ref} \\ y + l\sin(\psi) - y_{ref} \end{bmatrix}$$

where $.^F$ refers to the frenet frame, $.^M$ refers to the map frame. The full frenet states is

$$\boldsymbol{s}^F_{error} = \left[x^F_{error},\ y^F_{error},\ \psi^F_{error},\ v,\ \omega\right]^T$$

where $\psi^F_{error} = \psi - \psi_{ref}$. Throughout the paper the subscript are dropped for clarity.

## Appendix D. Experimental Details

In Figure 6 we show complimentary data on the comparisons of the learning based methods and the classical methods. The first row shows the tracking response of the each controller for each scenario. Second row shows the tracking response but comparing only the dense DMPC with sparse DMPC. It is important to note that we are plotting the performance against the progress along the path rather than time. This way, we are able to compare directly how each controller behave at the same point of the path. Last row shows the reference trajectory for each scenario. The UGV starts at the green dot and finishes at the red dot.

### D.1. Model mismatch experiment

The following first order system parameterized by the time constant $\tau$ is used to model the turning delay:

$$\dot{\omega} = \frac{1}{\tau}(\omega_{cmd} - \omega). \tag{8}$$

### D.2. Dense reward on a real UGV

Figure 6 shows the tracking response on the straight path (top right). Here, the response of the DMPC method is the most damped, approaching the reference in controlled manner with less oscillation. One could argue that this is a more desirable behaviour. This explains why the cumulative rewards of DMPC in Figure 4 on the straight path seems worse compared to Naive MPC and Expert MPC.

### D.3. Sparse reward on a real UGV

Reiterating the point from Section 4.5, the response for sparse reward is missing for the straight scenario because the UGV simply stood still when initialized far from the reference. This is most likely due to the reward not propagating properly resulting in the value function flattening out when far away from reference (see Figure 2). This in turn causes the gradient information to vanish, which makes it difficult for MPC to correctly navigate back to the reference.

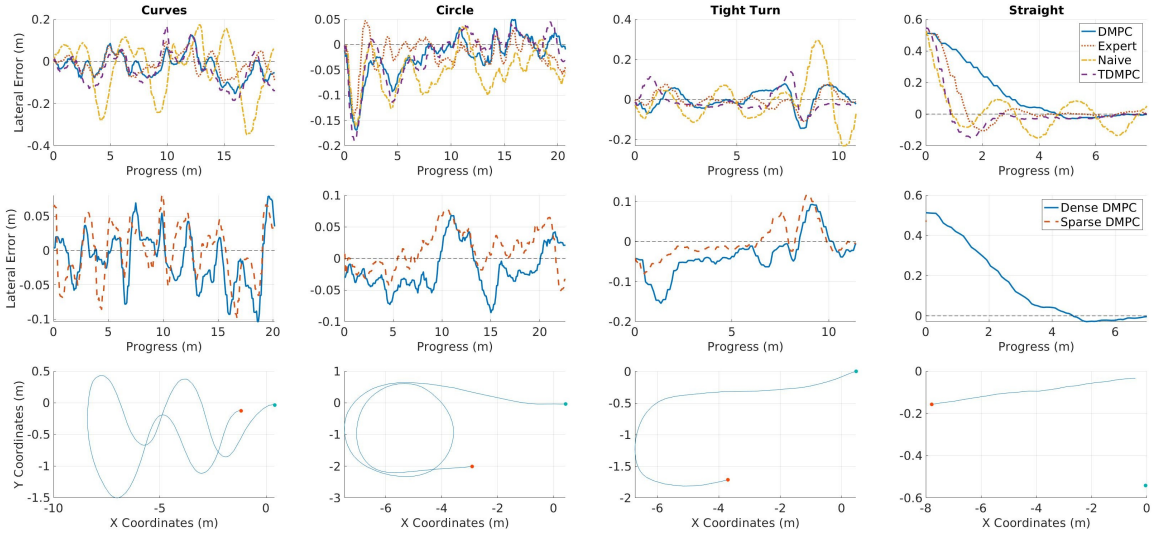


Figure 6: Top row: The lateral tracking error on different trajectories given a dense reward equal to the tracking error squared. Middle row: The lateral tracking error of the DMPC method for dense and sparse reward set-up respectively. Bottom row: The reference trajectories corresponding to each scenario. The UGV starts at the green dot and ends at the red dot.