Learning Model Predictive Control with Attention for Real-World Navigation

Jannik Grothusen
jannik.grothusen@berkeley.edu

Stjepan Kusenic
stjepank@berkeley.edu

Kaushik Prakash
kaushikprakash@berkeley.edu

Srinath Rangan
srinath.rangan@berkeley.edu

Abstract: Real-world robot navigation in human-centric environments remains an unsolved problem. Model Predictive Control (MPC) has emerged as a powerful control strategy in the field of robotics, offering the capability to optimize performance while considering constraints and predicting future behavior. The combination of deep learning techniques, particularly Transformer architectures, with MPC, has the potential to significantly improve the efficiency and applicability of control policies in real-world systems. This project presents Transformer-MPC, an approach that integrates Transformer-based attention mechanisms into the MPC framework for a context-aware, learnable control policy. The architecture is trained end-to-end via imitation learning to mimic expert demonstrations.

Code & Dataset: https://github.com/J4nn1K/transformer-mpc
Training: https://wandb.ai/j4nn1k/transformer-mpc/

Keywords: Transformer, Attention, Model Predictive Control

1 Introduction

Model Predictive Control (MPC) has been a popular method for planning and control of robotic systems due to its ability to incorporate constraints and optimize for performance in real-time. It is an optimal control technique in which the calculated control actions minimize a cost function for a constrained dynamical system over a finite, receding, horizon [1].

Designing cost functions for MPC that can adapt to varying environments and contexts is a challenging task. Recently, deep learning techniques have been introduced to enhance the performance of MPC and adapt to various situations more effectively. One such architecture, the Transformer [2], has been successfully applied in various tasks due to its attention mechanism and scalability. In this project, we present an approach that combines MPC with Transformer architectures, named Transformer-MPC.

Our Transformer-MPC leverages the power of deep learning to adapt and improve its performance in real-world navigation scenarios. By using a learnable inverse optimal control framework, we integrate sensory context into the cost function and generate actions for the robot based on the context. Our method ensures a viable trajectory and significantly reduces user-defined constraints in the MPC problem, resulting in decreased computational demands for solving the MPC optimization.

2 Related Work

As we look into Model Predictive Control and try to combine it with Transformer architectures, we need to consider the previous works done in these fields. This paper takes great inspiration from the work done by Xiao et al. [3]. In their paper, they introduce a Performer-MPC approach to tackle the challenges of real-world navigation systems. Thus, they combine MPC with imitation learning

used to learn a cost function with vision inputs by Performers. The Performer architecture was introduced by Choromanski et al. [4] and is in essence an architecture that approximates regular full-rank-attention Transformers but only has linear space and time complexity. Therefore, they solve a bi-level optimization by joint training of the cost function and controller. This leads to significant performance improvements compared to standard MPC policies.

A similar approach is taken in the work of Salzmann et al. [5]. Here, they present a real-time Neural MPC framework that tries to integrate deep neural nets within the MPC pipeline to boost performance. Leveraging this architecture, they reduce the positional tracking error up to 80% compared to state-of-the-art approaches not utilizing deep learning. Remarkably, they are tackling the capacity problem of using larger models in online learning MPC and still being able to use it in real time.

Another approach by Bhardwaj et al. [6] shows how deep learning can be leveraged for trajectory optimization. In their paper, they use it to learn the parameters of a Gaussian Process Motion Planning algorithm. This allows them to benefit from past experience and they gain an end-to-end trained method with competitive performance compared to traditional state-of-the-art motion planning algorithms.

A further possible framework for exploring the integration of deep learning into MPC is presented by Drews et al. [7]. For their purpose, they build a vision-based MPC framework using deep convolutional neural networks to predict cost functions from video input. Furthermore, these cost functions can be immediately used for online trajectory optimization with MPC. Not only did the authors achieve this, but the neural net even learns out-of-plane representations which can be used to generate a map of the drive-able area in front of a vehicle. This gives the controller a performance edge in real-time applications and allows for more aggressive driving styles.

On a similar task, Nubert et al. [8] show how MPC methods and deep learning can be combined to achieve safer and faster performance on tasks in robotics. They propose a robust MPC method with combined planning and control layers. Furthermore, they boost the performance of the MPC by approximating the cost function with a deep neural network controller. Therefore, they outsourced the common online optimization by simply evaluating a neural network that was trained on a robust MPC in an offline setting. This gives them speed and performance improvements for real-time applications while the robust MPC ensures that safety measures are met.

With this brief look over the field, we see that the combination of MPC methods with deep learning techniques can bring significant efficiency bumps in robotics systems while complying with strict constraints. Furthermore, it also allows for economic computations which enable real-time use cases.

3 Transformer-MPC: Learnable MPC with Attention

The principal challenges of synthesis of model predictive controllers are: (i) to construct cost functions that remain suitable across a wide variety of robot-environment situations, and (ii) to generate reliable solutions to the underlying trajectory optimization problems. This project focuses on the first challenge, specifically by moving away from the traditional method of hand-engineered cost functions and embracing a learning-based inverse optimal control framework. We use sensory context to induce what MPC problem to solve in order to generate actions. We first outline the training and inference procedures involved in the learnable MPC framework, and then discuss the details of the learnable components. For a comprehensive visual overview, refer to Fig. 1.

Let the context be an image frame, the occupancy grid in the robot frame. As in Vision Transformer architectures [9], each frame is flattened to a sequence. Each element (token) of the sequence corresponds to a different patch of the original frame which is then enriched with positional encodings. The preprocessed input is then fed to self-attention and MLP layers. The final embedding is chosen as a latent representation of the entire context to parameterize the learnable cost.

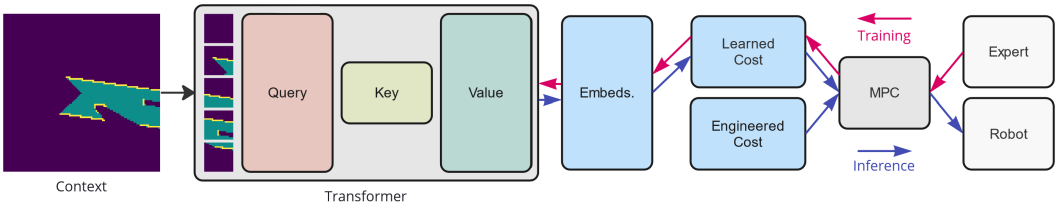

Figure 1: Overview of the Transformer-MPC. The latent embeddings are used to construct a context-dependent learnable cost. The backpropagation is shown by red arrows.

3.1 Learnable Model Predictive Control

Let C_0 denote the current "context". Consider a feedback policy implicitly defined by solving the following parametric optimal control problem at each time step:

$$\begin{aligned} \underset{\mathbf{u}_0 \ldots \mathbf{u}_{T-1}}{\arg\min} \quad & J_c\left(\mathbf{u}, \theta \mid C_0\right) = \sum_{t=0}^{T-1} c\left(\mathbf{x}_t, \mathbf{u}_t, t; C_0, \theta\right) + c_T\left(\mathbf{x}_T; C_0, \theta\right), \\ \text{s.t.} \quad & \mathbf{x}_{t+1} = f\left(\mathbf{x}_t, \mathbf{u}_t; \theta\right), \quad \mathbf{x}_0 = g\left(C_0, \theta\right) \text{ given.} \end{aligned} \tag{1}$$

Denote the optimizer as $\mathbf{u}^*\left(\mathbf{x}_0; C_0, \theta\right)$, and the corresponding optimal state sequence as $\mathbf{x}^*\left(\mathbf{x}_0; C_0, \theta\right)$. Here, θ are learnable parameters for stagewise and terminal cost neural networks $\{c, c_T\}$, f the dynamics function and g current state estimator. The dynamics function here corresponds to the drive dynamics of our robot.

We take an Imitation Learning approach where the robot has access to N expert demonstrations. The MPC structure provides a form of a strong inductive bias for Imitation Learning and can lead to improved data efficiency, robustness, and generalization. Denote $\bar{\mathbf{u}}^i = \left(\bar{\mathbf{u}}_0^i, \ldots, \bar{\mathbf{u}}_{T_i-1}^i\right)$ and $\bar{\mathbf{x}}^i = \left(\bar{\mathbf{x}}_0^i, \ldots, \bar{\mathbf{x}}_{T_i}^i\right)$ as the control and state sequence for the i^{th} demonstration snippet, with associated sensor context C_0^i, which can be extracted from offline planning or human teleoperation. We optimize θ as follows:

$$\theta^\star = \arg\min_\theta \sum_{i=1}^{N} J_l\left(\mathbf{u}^{i*}(\theta) \mid \bar{\mathbf{u}}^i\right), \tag{2}$$

where J_l denotes the total imitation loss that measures the discrepancy between MPC-generated and expert state-control trajectories. We use the squared L^2 norm as our loss criterion.

3.2 Training via Bi-level Optimization

The training optimization problem in Eq. 2 also contains the MPC optimization, Eq. 1. This can be seen as bi-level optimization as the higher level finds the optimal θ and the lower level uses this to get the optimal control sequence. In order to perform stochastic gradient descent for the higher-level problem we need the gradient of J_l with respect to θ. This gradient should be evaluated at a control sequence $\mathbf{u}^*\left(\theta_k\right)$ where θ_k denotes the parameters of the current iteration k. As shown in [3], this quantity decomposes as a vector-Jacobian product,

$$\nabla_\theta J_l\left(\mathbf{u}^*\left(\theta_k\right) \mid \bar{\mathbf{u}}^i, \bar{\mathbf{x}}^i\right) = \nabla_{\mathbf{u}} J_l\left(\mathbf{u}^*\left(\theta_k\right) \mid \bar{\mathbf{u}}^i\right)^T \partial_\theta \mathbf{u}^*\left(\theta_k\right). \tag{3}$$

The first term in the product on the right-hand side is the gradient of the total imitation loss. The second term is the sensitivity of the MPC solution with respect to parameters, which may be efficiently computed using the Implicit Function Theorem:

$$\partial_\theta \mathbf{u}^\star\left(\theta_k\right) = -\left[\nabla_{\mathbf{u}}^2 J_c\left(\mathbf{u}^\star, \theta_k\right)\right]^{-1} \nabla_{\theta, \mathbf{u}}^2 J_c\left(\mathbf{u}^\star, \theta_k\right). \tag{4}$$

Figure 2: Experimental Setup (left). Example scenario for data collection (right): an obstacle on the right side of the hallway.

By chaining, the vector-Jacobian product from Eq. 3 can be efficiently calculated as

$$\nabla_\theta J_l\left(\mathbf{u}^*\left(\theta_k\right)\right) = -\left[\left[\nabla^2_{\mathbf{u}} J_c\left(\mathbf{u}^\star, \theta_k\right)\right]^{-1} \nabla_{\mathbf{u}} J_l\left(\mathbf{u}^*\left(\theta_k\right)\right)\right]^T \nabla^2_{\theta,\mathbf{u}} J_c\left(\mathbf{u}^\star, \theta_k\right). \tag{5}$$

Please refer to [3] for details.

3.3 Attentive Cost Functions for Learnable MPC

In our inverse optimal control framework, only the cost function contains learnable parameters. Following [3], we structure this cost as a context-dependent quadratic function, parameterized by an embedding matrix $\mathbf{P}$ and vector $\mathbf{q}$:

$$c\left(\mathbf{x}, \mathbf{u}, t; C_0, \theta\right) = \begin{bmatrix} \mathbf{x} \\ \mathbf{u} \end{bmatrix}^T \mathbf{P}^T\left(C_0, \theta\right) \mathbf{P}\left(C_0, \theta\right) \begin{bmatrix} \mathbf{x} \\ \mathbf{u} \end{bmatrix} + \mathbf{q}\left(C_0, \theta\right)^T \begin{bmatrix} \mathbf{x} \\ \mathbf{u} \end{bmatrix}. \tag{6}$$

The Transformer-backed cost model attends to the current context C_0 to generate residual quadratic cost terms for MPC to optimize. Since the residual cost is convex and well-conditioned, the MPC solutions can be easily obtained. The backend maps the current contexts C_0 into a latent embedding which can be reshaped into the matrices $\mathbf{P}$ and $\mathbf{q}$ to support the quadratic parameterization of Eq. 6.

4 Experiments

Our Transformer-MPC is designed to be deployed on an omnidirectional wheeled robot, which has a 2D LiDAR and a RGB-D camera in the front. We choose the ClearPath Ridgeback, a midsize indoor robot platform that uses an omni-drive to enable exact position control within 3 degrees of maneuverability. The robot can move at a linear speed between $[-1.1, 1.1]\,\mathrm{m/s}$. The robot is controlled using the Robot Operating System (ROS). Based on the LiDAR measurements we generate at each time step an occupancy grid with 0.1×0.1 m cells (occupied, free, or unknown) around the robot within $[-10, 10]$ m of range for both x and y axis. The RGB-D camera records images with a resolution of 640×480 pixel.

In the chosen experimental scenario, the task involves maneuvering through a narrow hallway. The MPC formulation is designed to maintain a forward speed of 0.4 m/s, with the expectation that the learned cost function will adhere to a safe distance from surrounding obstacles.

4.1 Data Collection

To learn to avoid obstacles in the hallway navigation scenario, we teleoperate the robot and record more than 50 episodes of hallway navigation. Each episode is generated in a different environment. Stationary and moving obstacles are used to simulate real-world situations. Fig. 2 shows an example scenario.

We use the `rosbag` package to record the sensor measurements and expert control inputs from ROS topics. The control inputs $\mathbf{u} = [u_x, u_y, u_\omega]$ are the two translational and one rotational velocity

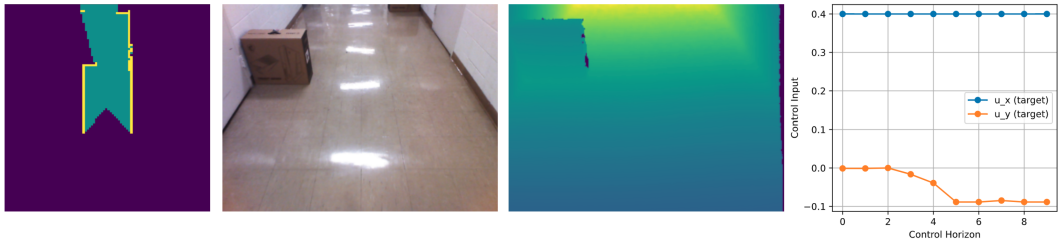

Figure 3: Sample of collected data. The occupancy grid, RGB image, and depth image show an obstacle on the left side of the hallway. The expert control inputs over the next T timesteps (right) take the obstacle into consideration by moving slightly to the right (small u_y).

on the ground plane. Fig. 3 shows a data sample. The final dataset consists of more than 100.000 datapoints with a size of roughly 100 gigabytes.

4.2 Implementation

Based on the results in [3] our Transformer architecture consists of $l = 3$ layers and $h = 1$ head with mlpdim $= 64$. We apply GeLU nonlinearity in the MLP layers. At its core, the Transformer encoder is implemented similar to the ViT encoder in [9] using JAX [10]. The optimal control solver of the MPC is implemented using the differentiable Iterative LQR implementation of trajax [11].

Since we created our own training data, we implemented custom dataset and dataloader classes using `Dataset` and `Dataloader` from `torch.utils.data` from PyTorch [12]. Our policies are trained on a TPU from Google's TPU Research Cloud.

4.3 Results

The goal of this chapter was to compare Transformer-MPC to regular MPC. The training performance of Transformer-MPC is illustrated in Figure 4, and additional insights can be found via the link provided in the abstract. Despite the efforts, the lowest training loss achieved by our model is in the order of 10^{-3}, which is two orders of magnitude higher than the (heuristically determined) desired 10^{-5} for deployment.

There are two potential reasons for the insufficient training performance of our Transformer-MPC model. These are (i) inadequate tuning of model and MPC parameters: The cost function weights, horizon length, and transformer hyperparameters might not have been optimally tuned, resulting in suboptimal performance. A more extensive exploration of these parameters may be required to identify the ideal configuration that allows the model to reach the desired training loss. And (ii) insufficient size of the custom dataset: The current dataset might not be large enough to allow the model to generalize well to new or unseen data. Consequently, the model may exhibit high error rates when encountering unfamiliar situations. An expansion of the dataset with diverse and representative samples could potentially improve the model's performance.

5 Limitations

The presented architecture currently only leverages spatial attention, but in principle, it can leverage the temporal axis as well. For example, in an approach with a walking human in a hallway, motion history may shed light on how the human intends to move in the future, e.g., yielding left or right. Furthermore, exploring richer modalities than the occupancy grid and RGB-D images, for example, language contexts, may enable robot-human interactions beyond simple geometry. Additionally, the quadratic form of our cost function limits its expressiveness and complexity. Incorporating non-linear terms into the cost function, such as higher-order polynomials or even non-polynomial functions can capture more intricate relationships and dependencies among the variables in our optimization problem.

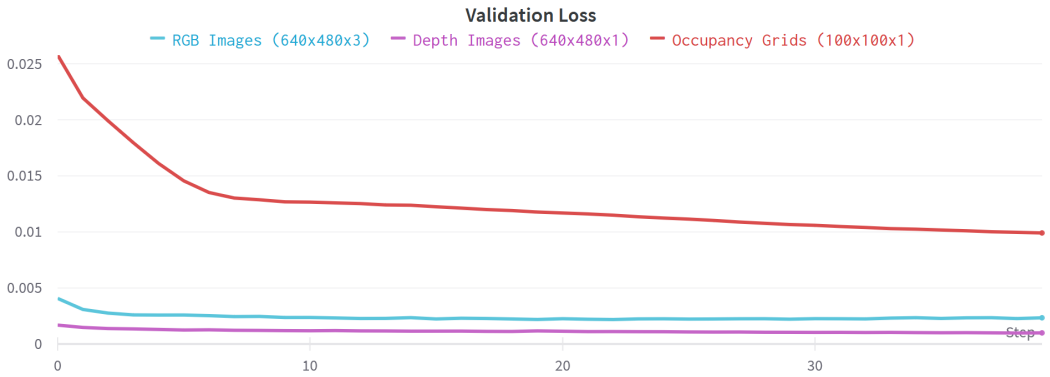

Figure 4: Validation loss while training with three different input data types.

6 Conclusion

In summary, our paper presents Transformer-MPC, a learnable model predictive control system that leverages Transformers to learn context representations and parameterize trainable cost functions. We collected a custom dataset of expert demonstrations and trained our model. We expected our model to learn to avoid local minima, maneuver through constrained spaces, and adhere to social norms. Instead, the training performance of our model was insufficient, and we were not able to validate its real-world compatibility yet.

References

[1] C. E. Garcia, D. M. Prett, and M. Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[3] X. Xiao, T. Zhang, K. Choromanski, E. Lee, A. Francis, J. Varley, S. Tu, S. Singh, P. Xu, F. Xia, S. M. Persson, D. Kalashnikov, L. Takayama, R. Frostig, J. Tan, C. Parada, and V. Sindhwani. Learning model predictive controllers with real-time attention for real-world navigation, 2022.

[4] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, D. Belanger, L. Colwell, and A. Weller. Rethinking attention with performers, 2022.

[5] T. Salzmann, E. Kaufmann, J. Arrizabalaga, M. Pavone, D. Scaramuzza, and M. Ryll. Real-time neural mpc: Deep learning model predictive control for quadrotors and agile robotic platforms. *IEEE Robotics and Automation Letters*, 8(4):2397–2404, 2023.

[6] M. Bhardwaj, B. Boots, and M. Mukadam. Differentiable gaussian process motion planning. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 10598–10604. IEEE, 2020.

[7] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg. Aggressive deep driving: Model predictive control with a cnn cost model. *arXiv preprint arXiv:1707.05303*, 2017.

[8] J. Nubert, J. Köhler, V. Berenz, F. Allgöwer, and S. Trimpe. Safe and fast tracking on a robot manipulator: Robust mpc and neural network control. *IEEE Robotics and Automation Letters*, 5(2):3050–3057, 2020.

[9] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[10] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

[11] R. Frostig. trajax: differentiable optimal control on accelerators, 2021. URL http://github.com/google/trajax.

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.