MPC with Sensor-Based Online Cost Adaptation

Avadesh Meduri[1], Huaijiang Zhu[1], Armand Jordana[1], Ludovic Righetti[1]

Abstract— **Model predictive control is a powerful tool to generate complex motions for robots. However, it often requires solving non-convex problems online to produce rich behaviors, which is computationally expensive and not always practical in real time. Additionally, direct integration of high dimensional sensor data (e.g. RGB-D images) in the feedback loop is challenging with current state-space methods. This paper aims to address both issues. It introduces a model predictive control scheme, where a neural network constantly updates the cost function of a quadratic program based on sensory inputs, aiming to minimize a general non-convex task loss without solving a non-convex problem online. By updating the cost, the robot is able to adapt to changes in the environment directly from sensor measurement without requiring a new cost design. Furthermore, since the quadratic program can be solved efficiently with hard constraints, a safe deployment on the robot is ensured. Experiments with a wide variety of reaching tasks on an industrial robot manipulator demonstrate that our method can efficiently solve complex non-convex problems with high-dimensional visual sensory inputs, while still being robust to external disturbances.**

I. Introduction

Robots deployed in factories and warehouses are able to perform repetitive tasks accurately. However, they are not able to carry out more complex tasks as they lack the ability to adapt their motions according to high-dimensional sensory feedback such as vision, touch, or sound from the surrounding environment. This inadequacy arises from the fact that these robots typically track pre-defined motions that do not adapt to changes in the environment.

One popular way to adapt motions online is through model predictive control (MPC), which requires solving trajectory optimization problems repeatedly based on state feedback. MPC has become standard in many robotic applications, such as legged locomotion, enabling robots to adapt their behavior to unforeseen events. Many approaches use linear approximations of the non-linear robot dynamics and quadratic cost to describe the desired behaviours such as walking, trotting or bounding [1]–[3]. This then allows the trajectory optimization problem to be solved as a Quadratic Program (QP). Thus, they guarantee real-time convergence to a unique global optimum and hard constraint satisfaction, leading to robustness to external disturbances and easy transfer to robots. However, they can only generate limited types of behaviours due to the use of simplified models and fixed, hand-tuned cost functions that do not change based on sensor feedback. As a result, different behaviours often require different hand-crafted cost functions and simplified models.

[1]Tandon School of Engineering, New York University, USA

This work was in part supported by the National Science Foundation grants 1932187, 1925079, 2026479, 2222815 and Meta Platforms Inc.

Recent advances in MPC have removed the need for reduced order models by directly using the full non-linear robot dynamics [4]–[9]. This enables rapid generation of a variety of behaviours with a unified formulation. However, low computation time is achieved by making sacrifices such as inability to enforce hard constraints [4], [7], [9] or allowing only quadratic costs [6]. Consequently, any increase in the complexity of the cost, such as adding non-convex obstacle avoidance tasks, would significantly increase computation time. To date, only empirical real-time convergence has been shown with these methods.

More importantly, visual sensory data is rarely used in the feedback loop, because modelling system or environment dynamics with images as part of the state is usually intractable. As a result, the few approaches that do use visual inputs extract important features from the images such as obstacles, goal locations or keypoints through image processing [10] or train neural networks that checks for collisions with the environment [11]. This information is then included manually inside the cost function. Consequently, the feature extraction is tailored for a particular task and would require modifications for new situations. Further, image processing based methods are sensitive to lighting, image contrast, etc. which can undermine the quality of trajectories or stability of the numerical solver, especially when the underlying optimizer cannot enforce hard constraints [10].

To address these issues, we introduce a MPC scheme featuring an adaptive QP. The cost function of this QP is constantly updated by a neural network directly from multiple sensors (e.g. joint encoders and vision) and aims to generate trajectories that minimize a non-convex general task loss. This procedure enables solving a convex problem (QP) online while still generating rich behaviours defined by a non-linear general task loss. Consequently, this guarantees safe deployment of sensor-driven MPC on real robots by enforcing hard constraints and real-time convergence. At the same time, direct integration of vision into the MPC feedback loop is simple by leveraging the neural network. Thus eliminating the need for hand-engineered feature extraction.

Figure 1 depicts the MPC scheme deployed on the robot. For every new sensor measurement, a neural network, called QPNet, predicts the cost function. Then, the associated QP is solved and its solution is tracked by a model-based inverse dynamics controller at a higher frequency. Consequently, at each new sensor measurement, the QP varies and adapts to changes in the environment and the desired task goal, in order to minimize the general non-linear task loss. The online adaption of the cost function allows the generation of complex behaviours while still solving a QP. The training of the QPNet is done in a supervised way after generating the

QP costs using techniques of implicit differentiation [12].

The proposed method is a general reactive MPC framework that has three main advantages:

1) Allows a simple and clean way to incorporate high-dimensional sensory inputs into the feedback loop while still using established optimization techniques for MPC.
2) Enforces hard constraints during deployment even while using neural networks in the feedback loop to integrate different sensor modalities.
3) Guaranteed real-time solve rates of the MPC loop that remain constant irrespective of the task complexity.

We demonstrate these advantages by choosing one application - a set of reaching tasks (3D locations and 6D poses) on a 7-DoF KUKA LBR iiwa robot under different settings 1) when the target location is provided by a Motion Capture (MC) system, 2) when only a RGB-D camera is available, and 3) when the robot is required to avoid an obstacle. We also demonstrate stable push recovery in all the above mentioned situations. These experiments show that by simply modifying the high-level task loss, our approach can handle more complex settings without any change to the general method while retaining all the above mentioned advantages. This is in contrast to existing methods for adaptive reaching tasks that only show a subset of these advantages and require modification in the cost design to extract suitable features from images [10], [11].

II. Related Work

a) Learning-based MPC: It is not a new idea to approximate MPC with neural networks. For example, Parisini and Zoppoli have proposed in [13] to approximate a receding-horizon controller (MPC) for a nonlinear system using neural networks almost three decades ago. Since then, advances have been made by exploiting techniques from explicit MPC to establish constraint satisfaction [14] and robust MPC to improve robustness by design [15]. However, they do not address the problem of fusing high-dimensional sensory data. More recently, this methodology has been further investigated through the lens of imitation learning (IL). In particular, the MPC policy is viewed as an algorithmic expert to be imitated by a student policy that can incorporate high-dimensional sensory inputs [16]; however, in this work, the underlying trajectory optimization problem is solved by dynamic differential programming (DDP), thus does not enforce hard constraints.

b) Implicit differentiation through convex optimization: A key ingredient of our approach is the ability to differentiate through convex optimization problems via the implicit function theorem [12], [17]. Indeed, such techniques have been successfully applied to learning parameters of optimization-based control policies. For instance, Amos et al. learns the cost function and the dynamics jointly for a MPC policy in a model-free setting [18]; Agrawal et al. tunes parameters for various forms of control policies represented as convex optimization [19]. However, neither of these methods address the problem of fusing high-dimensional sensory data and they have only been demonstrated on simple problems without deploying the learned policies on real hardware.

c) Inverse optimal control: Learning cost functions to generate desired behaviours is studied in inverse optimal control (IOC). In particular, cost parameters are inferred to recreate desired behaviour from expert demonstrations with policy optimization [20]–[22]. Visual demonstrations have also been used to learn cost functions via bi-level optimization [23]. However, in this line of work, the learned cost cannot be adapted by additional sensory inputs to generalize to new situations. Also, due to the use of trajectory optimization, our approach does not require human demonstrations, which can be tedious to obtain.

III. Method

In this section, we present our approach that is able to perform MPC to achieve a non-convex reaching task [4] with high-dimensional sensory inputs in real time on a robot.

A. Problem Formulation

We consider a typical optimal control problem, where the goal is to find an optimal trajectory $x^\star$ of states and controls that minimizes a non-convex task loss. Instead of directly solving this problem with a nonlinear optimization method, we would like to re-parameterize it as a QP so that it can be solved efficiently at run time. To achieve this, we formulate the trajectory optimization problem as bi-level optimization, with an upper-level non-convex cost that describes the complex motion needed to be performed and a lower-level QP-based trajectory optimizer. Mathematically, the problem is formulated as follows

$$\min_{G,c} \qquad \phi_{\text{task}}(x^\star, g)\,, \tag{1}$$

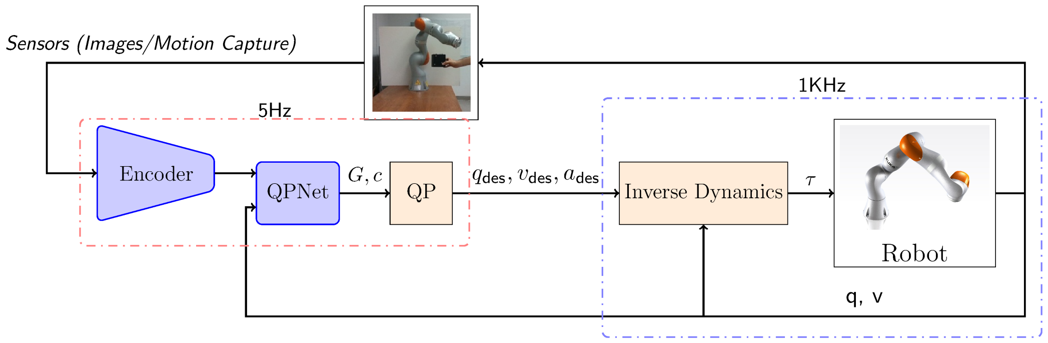
$$\text{where} \quad x^\star = \quad \underset{x}{\operatorname{argmin}} \qquad x^T G x + c^T x \tag{2a}$$

$$\text{s.t.} \qquad Ax = b,\ Kx \le h\,. \tag{2b}$$

This bi-level optimization problem aims to minimize the high-level non-convex task loss (1) parameterized by a task goal g with respect to the cost parameters, the matrix G and the vector c, of the lower-level QP (2). The QP has a formulation of a standard optimal control problem, where the constraints (2b) enforce the system dynamics and the feasibility of the states and controls.

We solve this bi-level problem with Adam [24]. The gradients through the lower-level QP ($\frac{\partial x^\star}{\partial G}$ and $\frac{\partial x^\star}{\partial c}$), are obtained using the implicit differentiation technique proposed in [12]. The cost function matrix G is internally parameterized as $G = \mathit{ReLu}(L)\mathit{ReLu}(L)^T$ where L is the actual parameter optimized in the bi-level problem. This ensures that all constructed cost parameters G in the QP are positive semidefinite. Since the dynamics is known and the gradients are exact, our method does not require sampling to estimate the gradient as opposed to model-free methods [23], [25] and takes very few iterations to converge.

Remark 1: It is important to note that we design the lower-level problem as a QP due to the availability of efficient off-the-shelf solver, bounded computation time for reliable deployment, and guaranteed convergence to global optimum (which is important for implicit differentiation).



Sensors (Images/Motion Capture)
5Hz
1KHz
Encoder
QPNet
G, c
QP
$q_{\text{des}}, v_{\text{des}}, a_{\text{des}}$
Inverse Dynamics
τ
Robot
q, v

Fig. 1: Pipeline in the case of image measurement. Note that in the case of measurement from motion capture, the goal position is directly provided to the QPNet (encoder is an identity map).

However, the lower-level problem can also be represented as cone programming or geometric programming [17], which are differentiable as well.

B. Trajectory Optimization for a Reaching Task

We now use the above optimization problem to perform various reaching tasks with a robot manipulator. In this scenario, the goal g is a 3D vector describing the target reaching position of the end-effector. For this, we would like to generate a trajectory x of joint positions q, velocities v and accelerations a that minimize the high-level non-convex task loss (1). Consequently, we set the QP-based trajectory generator to

$$\min_{x=[q_0,v_0,a_0,\ldots q_N,v_N]} x^T G x + c^T x \tag{3a}$$

$$\text{s.t. } v_{t+1} = v_t + a_t \delta t, \quad q_{t+1} = q_t + v_{t+1}\delta t \tag{3b}$$

$$a_{\text{min}} \leq a_t \leq a_{\text{max}}, \quad q_0, v_0 = q_{\text{init}}, v_{\text{init}} \tag{3c}$$

where $a_{\text{min}}, a_{\text{max}}$ are the acceleration limits (which indirectly enforce torque limits) and $q_{\text{init}}, v_{\text{init}}$ is the initial state for the MPC optimization problem. This QP generates a feasible joint trajectory $x^\star = [q_0^\star, v_0^\star, a_0^\star, \ldots, q_N^\star, v_N^\star]$ that minimizes the quadratic cost defined by G, c.

Remark 2: Note that while the QP only enforces double integrator dynamics, the complex robot model is implicitly incorporated in the convex costs G, c. This is because the upper-level cost in the bi-level optimization contains the non-linear robot kinematics. Additionally, an inverse dynamics controller takes into account the robot dynamics to realize the desired joint commands (cf. Sec. III-D).

C. Online Cost Adaptation using Measurement Data

With the bi-level optimization method discussed in Sec. III-A, it is now possible to find cost parameters $y^\star = [G^\star, c^\star]$ that minimize the non-convex cost (1) via QP given an initial state $s_{\text{init}} = [q_{\text{init}}, v_{\text{init}}]$ and a goal position g. While the QP can be solved efficiently, the bi-level problem (1) is still non-convex; hence, it does not have any real-time guarantees and might not be solved fast enough if the task loss is very complex. Furthermore, at run time, the goal position g might not be directly available and one might only have access to sensor data describing g. Therefore, we learn a policy network $\hat{\pi}$ which we refer to as QPNet, mapping s_{init} and sensor information to the optimal QP cost parameters $G^\star, c^\star$. This ensures that at run time, given sensor data, the cost parameters are available in a bounded amount of time from a neural network forward pass. In this work, we use two different types of measurements of the goal (reaching position): MC and RGB-D images.

a) MC measurement: In this setting, the QPNet takes as an input the MC measurement. It is assumed that the MC system directly provides the ground truth position of the desired reaching location g (the center of the cube in Fig. 1). This means that to train the QPNet, no real sensor data is needed. To create the dataset, an appropriate task loss is chosen based on the desired task (e.g. reaching, reaching with obstacle etc.). We then randomly sample initial robot states s_{init} and desired reaching positions g. Then, the bi-level problem (1) is solved in a MPC fashion until the goal is reached. That is, at each MPC step, the last state q, v of the computed trajectory from the QP is used as the initial state for the next iteration. All the pairs $\{(s_{\text{init}}, g), y^\star\}$ collected along this simulated trajectory are then added to the dataset. This gives us a dataset containing approximately $10,000$ samples that is used to train the QPNet in a supervised manner with a l_1-loss: $\|y^\star - \hat{\pi}(s_{\text{init}}, g)\|_1$. This can be seen as behavioral cloning (BC), a naive form of IL. However, the choice of the specific IL algorithm is not restricted by our method; it is entirely possible to use direct policy learning approaches such as DAgger [26] since we can always query the trajectory optimizer for more data.

b) Image measurement: Ultimately, it is expensive to use a MC system for each of the many robots during deployment. Instead it is preferable to use the MC system once, to train a vision system and then use the trained system on several cameras for several robots. Integrating vision can be achieved in our method by encoding RGB-D images to a lower-dimensional embedding e, which is then given as an input to the QPNet. We present here a procedure to train the encoder alongside the QPNet using MC measurement. To obtain a suitable encoder for the reaching task, we first train a Convolutional Neural Network (CNN) to predict the location of the cube given an RGB-D image. The CNN is trained on $150,000$ pairs of RGB-D image and ground truth cube location obtained from a MC system. After training, the convolution layers are frozen and used as an encoder that outputs the encoding e given an image of the cube. To train the QPNet that can directly take the encoding e as an input, we use the previous dataset and generate $10,000$ pairs of the form $\{(s_{\text{init}}, e), y^\star\}$. Here, e is obtained by inputting the images from the previous dataset into the trained encoder; $y^\star = [G^\star, c^\star]$ is obtained by solving problem (1) using the goal g that is already obtained by MC when collecting the image dataset; and lastly, s_{init} is sampled randomly. This results in a vision-based policy $\hat{\pi}_{\text{vision}}(s_{\text{init}}, e)$ that can fuse the RGB-D sensor data into the feedback loop, whereas traditional trajectory optimization methods would require processing the image into a amenable form [10], [11].

Remark 3: We train the encoder from scratch because collecting 150k image at 60 FPS took us ≈ 40 minutes, while training the CNN took an hour (the QPNet still requires only 10k samples). To further reduce the encoder training time and improve robustness, a pre-trained encoder [27], [28] could be used and fine-tuned on our data. Another option is to train the encoder in simulation where privilege information is readily available. In this case, the sim2real transfer can be considered as a separate problem where promising progress is being made. We have not explored these directions because our main focus in this work is to propose a stable MPC algorithm that can include vision feedback.

D. Multi-Modal Sensor-Based MPC Loop on the robot

At run time, the sensor measurement (either the MC measurement or the image encoding) and the robot state q, v are provided to the QPNet which predicts the QP costs. With this prediction, a QP is then constructed and solved in order to obtain the desired joint trajectory $x_{\text{des}} = [q^0_{\text{des}}, v^0_{\text{des}}, a^0_{\text{des}}, \dots, q^N_{\text{des}}, v^N_{\text{des}}]$, a segment of which is interpolated and tracked by an inverse dynamics controller. The torques τ needed to realise this trajectory are computed using $\tau = RNEA(q, v, a_{\text{des}}) + k_p(q - q_{\text{des}}) + k_d(v - v_{\text{des}})$, where $RNEA$ is the Recursive Newton Euler Algorithm (inverse dynamics), k_p, k_d are joint gains which are added for minor corrections and q_{des}, v_{des}, a_{des} are the desired joint position, velocity, and acceleration at the respective time step in the interpolated trajectory. Since the QP enforces joint constraints, it is guaranteed that feasible joint torques can be computed for the given joint trajectory. Further, the use of a model-based inverse dynamics controller allows easy transfer to the robot and maintains compliance during deployment.

The inverse dynamics controller runs at $1\,\text{kHz}$, while the sensory feedback loop runs at $5\,\text{Hz}$. The compute time of one pass through the QPNet and the QP is under $2\,\text{ms}$ and with the encoder it is under $10\,\text{ms}$, which means that if needed our approach could be run at $500\,\text{Hz}$ or $100\,\text{Hz}$. Figure 1 shows the entire control loop deployed on the robot in the case of using RGB-D image measurement.

Remark 4: When using a fixed cost, the QP can only generate limited behaviour. However, in our setup, by updating the cost function at each control cycle using direct sensor measurements, we are able to produce arbitrarily complex motions online.

IV. Experimental Results

In this section, we demonstrate the benefits of the proposed method on a 7-DoF KUKA LBR iiwa robot for various

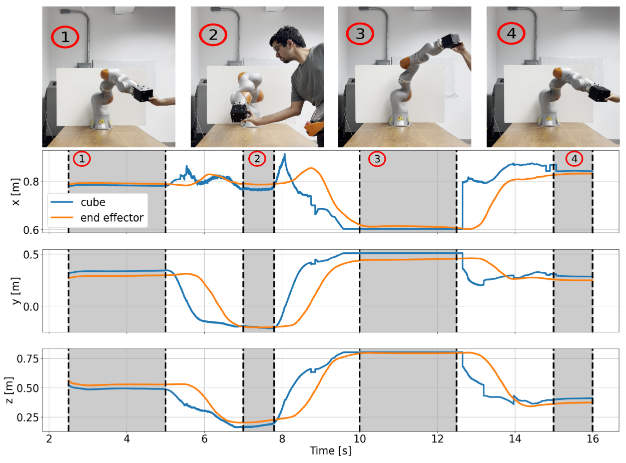


Fig. 2: KUKA reaching different cube locations. Each numbered image shows the corresponding situation at that instant.

reaching tasks. First, we show the performance of the QPNet that is trained to predict the QP cost from low-dimensional sensor data: robot states q, v and desired reaching position g provided by MC. Then, we show the ability of the approach to avoid obstacles while reaching different positions and 6D poses without any change to the computation time. Finally, we present results obtained when raw images of the cube (the desired reaching target) are used as an input to the pipeline along with the robot state q, v.

The neural networks are implemented using PyTorch [29]. Quadprog is used as the QP solver for the lower-level problem. The RGB-D images are obtained using the Intel RealSense D435 camera at 60 FPS and ground truth reaching locations are measured by a MC system tracking the cube. The camera was placed $1.35\,\text{m}$ in front of the robot's base so that the entire workspace is covered in each frame. The codebase is available with additional information on all the parameters in this link - https://github.com/machines-in-motion/ioc_qp.

A. Reaching Task with Motion Capture

In this experiment, the goal for the KUKA robot is to reach a cube located in its workspace. The location of the cube is obtained using a MC system. Therefore, we define a non-convex task loss $\phi_{\text{reaching}}(x, g)$ that aims to minimize the distance between the end-effector position and the desired goal position. In addition, the loss regularizes joint positions, velocities, accelerations and penalizes rapid changes in joint accelerations. The task loss is

$$\begin{aligned}\phi_{\text{reaching}}(x, g) &= w_6 \left\| f_{\text{fk}}(q_N) - g \right\| + w_7 \left\| f_{\text{v}}(q_N, v_n) \right\| \\ &+ \sum_{t=1}^{N-1} w_1 \left\| f_{\text{fk}}(q_t) - g \right\| + w_2 \left\| q_t \right\| + w_3 \left\| v_t \right\| + w_4 \left\| a_t \right\| \\ &\qquad + w_5 \left\| a_t - a_{t-1} \right\| \qquad (4)\end{aligned}$$

where $f_{\text{fk}}(q_t)$ is the forward kinematics function that returns the position of the end-effector (a highly nonlinear function of the states), $f_{\text{v}}(q_t, v_t)$ computes the end-effector velocity using the end-effector Jacobian and $w_{1,2,\dots,7}$ are cost weights that are tuned by the user. We use this cost in the bi-level

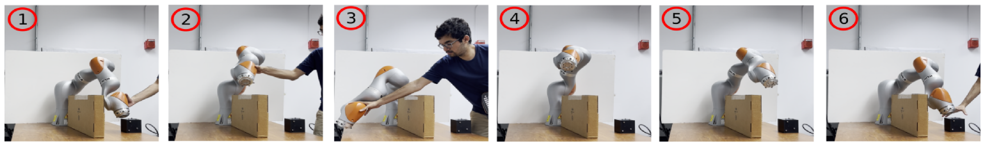


Fig. 3: Obstacle avoidance with external pushes. The image numbers correspond to the time sequence in the key frames.

optimization to train the QPNet as described in Sec. III-C. This experiment is chosen to demonstrate that the predictions of the QPNet are satisfactory to be applied to a real robot when using sensor measurement that is normally used with trajectory optimization. For this experiment, the framework run on the robot is identical to the scheme presented in Fig. 1 with the only difference that the target reaching location is directly provided by MC—there is no encoder network.

The performance of the QPNet is shown in Fig. 2 and the accompanying video. The images aligned with the shaded areas depicts the state of the robot at the respective times. As it can be seen, our method is able to generate motions to reach the cube accurately—the end-effector of the robot is in contact with the cube at key frames. The small error between desired and actual reaching position arises because the robot tries to reach the center of the cube which creates an offset. The generated trajectories are also smooth at run time, which is important for safe use on real robots. The QPNet is robust and stable to unforeseen pushes on the robot and brings back the robot to the location of the cube after the disturbance, even though the QPNet was never specifically trained for it. The robot is also very compliant while moving; this makes it safe to interact with, an advantage gained from constantly re-planning (MPC).

To systematically analyze the reaching performance of the QPNet, we run the reaching experiment in simulation for 1000 randomly generated points in the entire workspace. This includes a significant number of points outside the region in which training targets were created. The robot reached all the targets with a mean and standard deviation in reaching error of $8\,\mathrm{cm}$ and $3\,\mathrm{cm}$, which is rather low given the size of the iiwa robot ($1.5\,\mathrm{m}$) and the small training set. Importantly, the method always generated stable trajectories that reached the target. The errors are rather small compared to other reported results for learning approaches [30], even while considering targets outside of the training region.

B. Reaching Task with Obstacle Avoidance

We now increase the complexity of the task from just reaching a desired location in free space to also avoiding an obstacle while reaching. This is to demonstrate that the proposed method can achieve more complicated tasks while keeping the computation time unchanged. To achieve this, we now use a task loss for both obstacle avoidance and reaching. The obstacle avoiding task is encoded in the cost using a repulsive exponential potential, $\phi_{\text{obstacle}}(x, o)$, defined as

$$\sum_{t=1}^{N} w_8 \exp\left(-\Big(\big(f_{\text{fk}}(q_t)-o\big)^T A\big(f_{\text{fk}}(q_t)-o\big)-1\Big)/w_9\right) \tag{5}$$

where o is the location of the obstacle center, w_8, w_9 are tuning parameters that determine how strictly the obstacle should be avoided, and A is a diagonal scaling matrix that determines the convex hull circumscribing the obstacle to be avoided. By changing the diagonal elements, the obstacle can take the shape of a sphere to a narrow capsule. The QPNet is now trained by solving the problem (1) using the combined task loss for avoiding and reaching $\phi_{\text{ar}} = \phi_{\text{reaching}}(x, g) + \phi_{\text{obstacle}}(x, o)$. In addition, we also add an intermediate goal to guide the end-effector around the obstacle as the potential field may result in the optimizer being stuck in local minima. This intermediate goal can be generated automatically by a RRT-based planner and is only needed to compute $G^{\star}, c^{\star}$. The trained QPNet is then used in the same pipeline as discussed in Sec. IV-A. Note that during execution, the via-point is not needed and trajectories avoiding the obstacle are automatically generated by the QPNet. During execution, our approach successfully achieve this task without any increase in the computation time, since it still only requires one forward pass of the QPNet and solving one QP at each MPC time step. The robot reaches different cube locations that are changed online while avoiding the obstacle. The robot is robust to external perturbations, compliant and moves smoothly as presented in Sec. IV-A. A typical behaviour of the robot is shown in Fig. 3: the robot is first pushed away from the desired target location (the black cube) and then released. The robot then moves back towards the cube while avoiding the obstacle (the brown box). More examples can also be found in the supplementary video.

C. 6D Pose Reaching Task

For tasks such as object grasping or insertion, it is desirable to control both the reaching position and orientation of the end effector. In this experiment, we show that by just adapting the nonlinear cost ϕ and retraining the QPNet we can control the 6D pose without any change to the computation time during deployment. The modified cost function is

$$\begin{aligned}\phi_{\text{pose}}(x, g) &= w_6 \left\|\log_6(T(q_N) \ominus T_{\text{des}})\right\| + w_7 \left\|f_{\text{v}}(q_N, v_n)\right\| \\ &+ \sum_{t=1}^{N-1} w_1 \left\|\log_6(T(q_t) \ominus T_{\text{des}})\right\| + w_2 \left\|q_t\right\| + w_3 \left\|v_t\right\| + w_4 \left\|a_t\right\| \\ &+ w_5 \left\|a_t - a_{t-1}\right\| \end{aligned} \tag{6}$$

where $T(q)$ is the current 6D pose of the end effector, T_{des} is the desired reaching pose and $\ominus$ is the difference operator in SE(3). The $\log_6$ function gets the twist error between the two 6D poses. We run this experiment in simulation, since it is difficult to show that the desired 6D pose is actually reached without a gripper on the robot. Figure 4, shows the KUKA robot reaching a desired 6D pose where the visual coordinate axis attached to the end-effector aligns with the desired one in space. More results are available in the attached video.

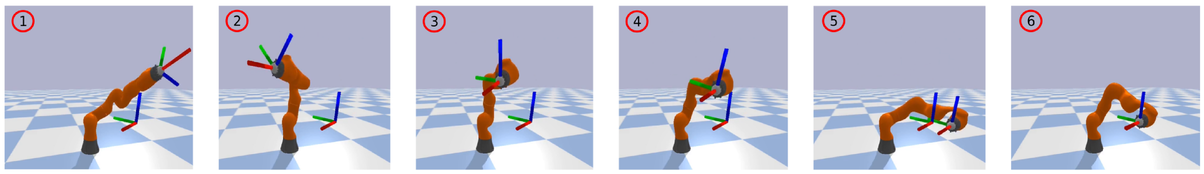


Fig. 4: KUKA reaching a desired 6D pose in simulation. The numbers on the images correspond to the sequence.

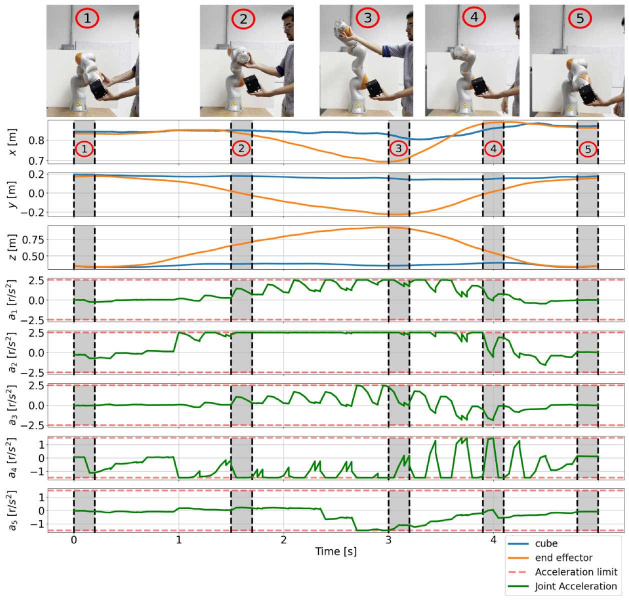


Fig. 5: KUKA recovering from pushes while using vision measurements in the MPC loop. The numbers on the images correspond to the shaded regions on the end-effector plot.

D. Reaching Task with Vision

We now present results when using a RGB-D camera in place of the MC system as discussed in Sec. III-D. The encoder contains 9 convolutional layers of width 64, 64, 128, 128, 256, 256, 256, 512, 512 respectively, followed by 2 fully connected layers with 512 neurons each. All layers except the last one are activated with ReLUs and a max pooling layer is added after the $2^{nd}, 4^{th}, 5^{th}, 6^{th}$ and 7^{th} layer. The goal is again to reach the cube at different locations while using RGB-D images as sensor measurements.

Tracking performance is similar to that presented in Sec. IV-A. During execution, the robot remains very compliant yet precise thanks to online re-planning. The controller is able to reject unexpected pushes and remains stable. Figure 5 shows an example of the robot being pushed away from the target location (the cube) and the robot finds its way back to the cube when the external force is removed. The system remains stable during the push as shown by the smooth trajectory of the end-effector position and the desired joint acceleration obtained from the QP. We further note that the joint accelerations remain bounded during the push, due to the constraints enforced by the QP (the acceleration of joint 2 between $1.5\,\mathrm{s}$ to $3\,\mathrm{s}$ stays on the limit depicted by the horizontal dotted line). This demonstrates the advantage of using an optimizer that enforces hard constraints at all times, thus ensuring a certain level of safety.

V. Discussion

1) Optimal Control Problem (OCP) formulation: The common practice with MPC techniques is to design a non-linear cost function that best describes the desired task and solve it online [4], [10]. This is often achieved by using Differential Dynamic Programming (DDP) to solve the optimization problem. These solvers are custom implemented to reduce computation time. However, this limits the generality of the MPC scheme since if the complexity of the task changes, the computation time may become too high for real-time planning. Here, we propose an alternate paradigm where we reformulate the OCP as a bi-level optimization problem to generate and learn convex costs. This enables online computation regardless of the task complexity due to the fixed computation time of the neural network and QP. In principle, our approach could then also be extended to other applications where the original OCP is numerically challenging to solve such as in legged locomotion.

2) QPNet performance: In all our experiments (MC, vision), the QPNet was able to reach any location in the workspace. It also generalized to targets that lied outside the region of the training data. The MPC was compliant and safe even when pushed. The same framework could handle increased complexity in tasks by simply modifying the high-level task loss, without any change to compute times. Our approach can be easily extended to more difficult tasks such as grasping or pick-and-place by adapting the higher level loss accordingly. However, one downside is that precision of the reaching was of the order of a few centimeters, which is not suitable for high-precision tasks. Future work will include improving precision for manipulation tasks.

VI. Conclusion

In this paper, we present an approach that can generate rich behaviours akin to non-convex trajectory optimization-based MPC while only solving QPs online. This is achieved by training a neural network that constantly adapts the quadratic cost using sensor measurements. Due to this, we ensure real-time convergence and strict constraint satisfaction. Further, the use of a neural network to encode the cost function allows easy integration of high-dimensional sensory data into traditional trajectory optimization schemes while keeping the computation time invariant to the complexity of the task. We demonstrate the capability of this approach with diverse reaching tasks on a KUKA manipulator while fusing different sensor measurements.

References

[1] A. Herdt, H. Diedam, P.-B. Wieber, D. Dimitrov, K. Mombaur, and M. Diehl, "Online walking motion generation with automatic footstep placement," *Advanced Robotics*, vol. 24, no. 5-6, pp. 719–737, 2010.

[2] D. Kim, J. Di Carlo, B. Katz, G. Bledt, and S. Kim, "Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control," *arXiv preprint arXiv:1909.06586*, 2019.

[3] M. Khadiv, A. Herzog, S. A. A. Moosavian, and L. Righetti, "Walking control based on step timing adaptation," *IEEE Transactions on Robotics*, 2020.

[4] S. Kleff, A. Meduri, R. Budhiraja, N. Mansard, and L. Righetti, "High-frequency nonlinear model predictive control of a manipulator," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 7330–7336.

[5] J. Koenemann, A. Del Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard, "Whole-body model-predictive control applied to the hrp-2 humanoid," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 3346–3351.

[6] A. Meduri, P. Shah, J. Viereck, M. Khadiv, I. Havoutis, and L. Righetti, "Biconmp: A nonlinear model predictive control framework for whole body motion planning," *arXiv preprint arXiv:2201.07601*, 2022.

[7] C. Mastalli, W. Merkt, G. Xin, J. Shim, M. Mistry, I. Havoutis, and S. Vijayakumar, "Agile maneuvers in legged robots: a predictive control approach," *arXiv preprint arXiv:2203.07554*, 2022.

[8] T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov, "An integrated system for real-time model predictive control of humanoid robots," in *2013 13th IEEE-RAS International conference on humanoid robots (Humanoids)*. IEEE, 2013, pp. 292–299.

[9] 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*, vol. 3, no. 3, pp. 1458–1465, 2018.

[10] J. Pankert and M. Hutter, "Perceptive model predictive control for continuous mobile manipulation," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6177–6184, 2020.

[11] M. Bhardwaj, B. Sundaralingam, A. Mousavian, N. D. Ratliff, D. Fox, F. Ramos, and B. Boots, "Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation," in *Conference on Robot Learning*. PMLR, 2022, pp. 750–759.

[12] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 136–145.

[13] T. Parisini and R. Zoppoli, "A receding-horizon regulator for nonlinear systems and a neural approximation," *Automatica*, vol. 31, no. 10, pp. 1443–1451, 1995.

[14] S. Chen, K. Saulnier, N. Atanasov, D. D. Lee, V. Kumar, G. J. Pappas, and M. Morari, "Approximating explicit model predictive control using constrained neural networks," in *2018 Annual American control conference (ACC)*. IEEE, 2018, pp. 1520–1527.

[15] 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*, vol. 5, no. 2, pp. 3050–3057, 2020.

[16] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, "Agile autonomous driving using end-to-end deep imitation learning," *arXiv preprint arXiv:1709.07174*, 2017.

[17] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable convex optimization layers," *Advances in neural information processing systems*, vol. 32, 2019.

[18] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable mpc for end-to-end planning and control," *Advances in neural information processing systems*, vol. 31, 2018.

[19] A. Agrawal, S. Barratt, S. Boyd, and B. Stellato, "Learning convex optimization control policies," in *Learning for Dynamics and Control*. PMLR, 2020, pp. 361–373.

[20] M. Kalakrishnan, P. Pastor, L. Righetti, and S. Schaal, "Learning objective functions for manipulation," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 1331–1336.

[21] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 729–736.

[22] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 1.

[23] N. Das, S. Bechtle, T. Davchev, D. Jayaraman, A. Rai, and F. Meier, "Model-based inverse reinforcement learning from visual demonstrations," *arXiv preprint arXiv:2010.09034*, 2020.

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[25] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[26] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[27] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 16 000–16 009.

[28] S. Nair, A. Rajeswaran, V. Kumar, C. Finn, and A. Gupta, "R3m: A universal visual representation for robot manipulation," *arXiv preprint arXiv:2203.12601*, 2022.

[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in neural information processing systems*, 2019, pp. 8026–8037.

[30] S. Bechtle, Y. Lin, A. Rai, L. Righetti, and F. Meier, "Curious ilqr: Resolving uncertainty in model-based rl," in *Conference on Robot Learning*. PMLR, 2020, pp. 162–171.