DATA-DRIVEN METHODS FOR DESIGN OF MODEL PREDICTIVE CONTROLLERS

BY

WILLIAM EDWARDS

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2022

Urbana, Illinois

Adviser:

Professor Kris Hauser

ABSTRACT

Model predictive control (MPC) is a powerful feedback technique that is often used in data-driven robotics. The performance of data-driven MPC depends on the accuracy of the model, which often requires careful tuning. Furthermore, specifying the task with an objective function and synthesizing a feedback policy are not straightforward and typically lead to suboptimal solutions driven by trial and error. In this work, we seek to address these challenges by investigating data-driven methods for system identification, task specification, and control synthesis of unknown dynamical systems. First, we conduct a case study on the design of a data-driven MPC for performing automatic needle insertion in deep anterior lamellar keratoplasty, a challenging ophthalmic microsurgery task. We propose a data-driven method for controller synthesis and selection and demonstrate that the synthesized controller outperforms a state-of-the-art baseline in *ex vivo* physical experiments. Next, we present AutoMPC, an open-source Python package for automatic synthesis of data-driven MPC. We demonstrate the AutoMPC outperforms a state-of-the-art offline reinforcement learning algorithm on several standard control benchmarks. We further demonstrate that AutoMPC outperforms standard control baselines in physical experiments on an underwater soft robot.

ACKNOWLEDGMENTS

Thank you to my advisor, Dr. Kris Hauser, for guiding me with a firm hand and helping me to see into my blind spots. Thank you to all my collaborators, especially to Gao Tang and Giorgos Mamakoukas, for their early mentorship, and to David Null and Dohun Jeong, for their friendship and support. Thank you to all members of the Intelligent Motion Laboratory, for providing a welcoming and stimulating community. Thank you to my financial support from the NIH (Grant R21-EY029877) and NSF (Grant IIS-2002492). And finally thank you to my family, for supporting my every step along the way.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Model predictive control (MPC) is a powerful framework for designing robot controllers. By leveraging knowledge of the dynamics, it can predict and optimize a robot's behavior over a multi-step time horizon and has been demonstrated to be effective on high-dimensional robots [1]. Moreover, MPC has been used as a component of model-based reinforcement learning (RL) solvers for continuous control problems to improve the sample complexity [2].

Successfully implementing MPC is challenging, as the control performance relies heavily on the accuracy of the model. For soft robots or robots with complex dynamics, i.e., aerodynamic or hydrodynamic interactions, developing a representation from first-principles can often be laborious or even intractable. Alternatively, researchers are increasingly relying on data-driven models using—among others— neural networks [3], Gaussian processes [4], or Koopman operators [5]. On the other hand, system identification (SysID) methods typically suffer from tedious hyperparameter tuning, scalability issues, or limited model capacity [6]. Further, when done manually, hyperparameter tuning is time-consuming and prone to errors.

Besides model accuracy, the control performance of MPC is also sensitive to factors such as the objective function, including regularization terms, the planning horizon, and state or control constraints. These hyperparameters create a large search space that is often left largely unexploited leading to suboptimal solutions. The optimizer must also be carefully chosen to exploit any nonlinearities in the dynamics. This is especially true for nonlinear objectives (e.g., "sparse rewards" in the RL community) and underactuated nonlinear systems.

In this work, we make two key contributions. First, we conduct a case study of data-driven MPC design for an ophthalmic surgical robot (Chapter 3). We study a highly challenging microsurgery procedure known as deep anterior lamellar keratoplasty (DALK). This setting poses unique modelling challenges due to the small-scale (< 30 μm) interactions between the surgical instrument and soft tissue, as well as robust control challenges, due to the sensitivity of the procedure to small errors in tool positioning. We develop a data-driven MPC for this procedure based on a learned autoregressive (ARX) model of the system dynamics. We also propose a cross-validation-like measure for offline evaluation and tuning of the MPC. We demonstrate in physical *ex vivo* experiments that our controller outperforms an established baseline controller, which was shown to have comparable performance to human surgeons.

In our second contribution, we seek to generalize the approaches used in the surgical robotics case study, by developing `AutoMPC`, an open-source Python package for the automatic synthesis of data-driven MPC (Chapter 4). The goal of this package is to make MPC

accessible to non-experts in the same way that AutoML libraries such as `auto-sklearn` [7] and `AutoKeras` [8] have done for supervised machine learning. The package implements a wide variety of SysID models (autoregression, Gaussian Processes (GP), Koopman operators, SINDy, neural networks) and optimizers (LQR, iLQR, direct transcription, and Model Predictive Path Integral (MPPI) control), and presents an open framework for contributors to add their own algorithms. We demonstrate that `AutoMPC`-generated controllers achieve superior performance to a state-of-the-art offline reinforcement learning algorithm on several standard control benchmarks (Chapter 5). We also present extensions to AutoMPC (Chapter 6) to improve robustness, support multi-task controllers, and improve scalability. We also evaluate the performance of `AutoMPC` on a physical underwater soft robot system, where we demonstrate superior performance compared to several standard control techniques (Sec. 6.4).

This work has been presented at several venues, including

- Preliminary findings of the surgical robotics case study (Chapter 3) were presented at the 2020 IROS Cognitive Robotic Surgery Workshop [9].
- `AutoMPC` package design (Chapter 4) and results (Chapter 5) were presented at the 2021 IEEE Conference on Robotics and Automation (ICRA) [10].
- Full results of the surgical robotics case study (Chapter 3) were published in Robotics and Automation Letters 2022 [11] and were also presented at ICRA 2022 [12].
- A manuscript presenting the physical experiments with `AutoMPC` on the underwater soft robot (Sec. 6.4) has been submitted to ICRA 2023 and is currently under review [13].

CHAPTER 2: BACKGROUND

Model Predictive Control: MPC is a widely used closed-loop control approach in robotics that incorporates state and control constraints [3, 14, 15]. For linear systems, the theory of closed-loop stability has been established and optimization is carried out efficiently with convex optimization solvers or avoided by offline precomputation known as explicit MPC [15]. However, for general nonlinear systems the closed-loop behavior is not yet well understood and successfully implementing MPC requires manual tuning.

Besides the high computational demand, MPC relies on accurately modeling the system dynamics. Although it has has been shown to achieve the desired performance when using models learned from data [2], inaccurate data-driven representations can cause suboptimal performance and even unstable control. Although robust MPC can address model inaccuracy [16] for linear systems, robust nonlinear control is harder to analyze and achieve. In this work, we will address this issue by automatically tuning the hyperparameters with Bayesian optimization where the performance of the SysID, task specification, and control synthesis is tuned end-to-end on a simulated surrogate model.

System Identification and Data-driven Control: System identification has been thoroughly explored [6] and theory of data-driven control for linear systems is relatively well developed, with regret bounds derived for policy gradient methods [17]. However, nonlinear SysID and data-driven control remains an open research question. Recently, there has been significant interest in using deep neural networks [3, 18, 19], GP based methods [4] and Koopman operator theory [5, 20]. Both classical system identification techniques and deep learning-based techniques require careful hyperparameter tuning to obtain useful models.

Our work is also related to model-based RL approaches [21, 22, 23], which explicitly learn a dynamics model and rewards, and run an optimizer to generate the agent's policy. However, to generate experience model-based RL accesses the true system, which is often infeasible or even unsafe. Our setting is also related to offline RL, which does not use online interaction with the robot [24, 25, 26]. In this work, we will present a method which is both offline and model-based, and also auto-tuned.

CHAPTER 3: CASE STUDY: AUTOMATIC NEEDLE INSERTION FOR DEEP ANTERIOR LAMELLAR KERATOPLASTY

Deep anterior lamellar keratoplasty (DALK) is a cornea transplantation technique which has been shown to improve patient outcomes compared to prior methods. Whereas the alternative penetrating keratoplasty (PKP) method transplants the full thickness of the cornea, DALK is a partial thickness transplant, replacing only the anterior layers (around 90% of the total thickness), while leaving the original endothelium and Descemet's membrane intact [27]. This has been shown to significantly reduce the risk of tissue rejection [28].

One approach to DALK is the "big-bubble" technique. As shown in Fig. 3.1, a cannulation needle is carefully inserted into the cornea and its tip is positioned just above the cornea apex. Air is then injected through the needle to perform pneumodissection. Ideally, an air bubble is formed which separates the endothelium and Descemet's membrane from the anterior cornea layers [29]. In practice, the air bubble frequently fails to form properly. Borderie *et al.* [30] found that the bubble failed to achieve pneumodissection in around 59% of cases, though the success rate can vary significantly depending on the surgeon and technique [31].

Successful bubbles can be classified as either type I or type II. Type I bubbles form within the stroma, while type II bubbles form deeper, between the stroma and the Descemet's Membrane. Depending on the type of bubble formed, the surgeon will have to modify the graft preparation [32]. Moreover, the type of bubble impacts the perforation rate and the histological properties of the graft [32, 33]. Thus, it is highly desirable to control the type of bubble formation. Yoo *et al.* [32] found that the relative depth of the needle within the cornea is an important factor for both the success rate and type of bubble formation. They found that type I bubbles were consistently formed when the relative depth of the needle was between 75% and 85%, while type II bubbles dominantly formed for depths greater than 90%. Depths between 85% and 90% yielded a mix of bubble types. Thus, when the needle depth can be controlled to within 5% of intended, it is possible to reliably achieve bubble formation of a particular type. Depending on thickness of the cornea, this corresponds to an accuracy of approximately 30 µm, which is challenging for human surgeons to achieve due to the small scales and difficulty of depth perception [34].

The need for small scale manipulation and visualization make the DALK procedure a promising application for surgical robotics, and there have been several recent works addressing this. Guo *et al.* [35] and Park *et al.* [36] use custom robots to guide the needle insertion, but these devices have limited range of movement and are limited to steeper needle insertion angles, which has been associated with worse pneumodissection outcomes [37].

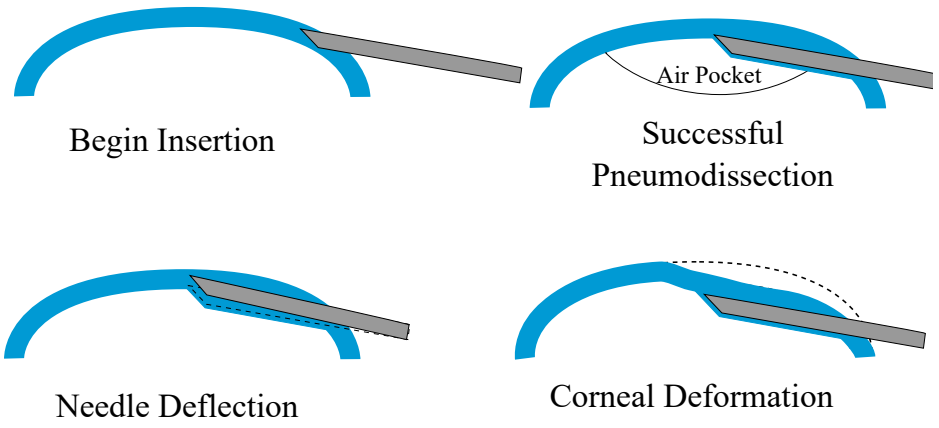


Figure 3.1: Top: In deep anterior lamellar keratoplasty (DALK), a cannulation needle is inserted from the side of the cornea and air is injected to separate the cornea layers for dissection. Bottom: Needle deflection and corneal deformation are sources of error that can prevent the needle from reaching the desired target depth.

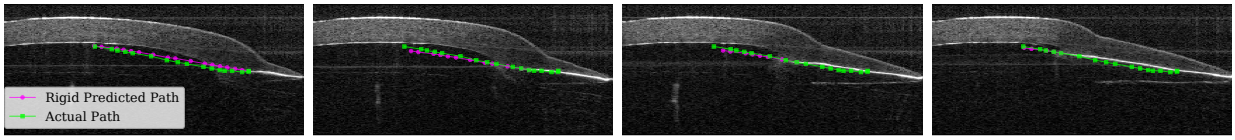


Figure 3.2: Four points in time of the same needle insertion. Notice that the rigid predicted path (magenta) diverges from the actual path (green) and that the cornea surface deforms as the insertion proceeds.

Draelos *et al.* [38, 39] develop a surgical robot system which mounts the needle on a 6-DoF arm, granting a wider range of movement. The needle insertion is guided using feedback from an optical coherence tomography (OCT) sensor, which images the location of the needle with respect to the cornea surfaces. The system can be operated either in cooperative mode, wherein the needle is guided by a human surgeon, or automatic mode, wherein the needle insertion is done autonomously.

Planning for automatic needle insertion in Draelos *et al.* is based on the assumption that both the cornea and needle remain rigid [39]. However, they observe that as the needle is inserted into the cornea, the relationship between the robot end-effector and the needle tip deviates from this rigid assumption, suggesting that either the needle or some component of the attachment assembly is deforming. Moreover, the cornea itself can deform significantly as the needle is inserted, effectively creating a moving target. Fig. 3.1 illustrates both of these effects and Fig. 3.2 shows examples in OCT data. To address these issues, the authors' approach reacts to deformation via feedback control rather than constructing an accurate plan. Once embedded, the needle cannot be translated laterally, which it makes difficult to

correct course once the needle has drifted off track. This can cause the needle to miss its target position by more than 100 µm.

This paper presents an insertion controller which accounts for the predicted needle and cornea deformation. A data-driven model is created to predict the motion of both the needle and the cornea in response to the robot's motion. A model predictive controller (MPC) then uses this model to plan an insertion path consistent with the learned dynamics. We use a *cross-simulation controller selection* procedure, similar to cross-validation, to compare candidate models for MPC without using physical experiments.

An autoregressive linear model (ARX) is chosen for the MPC controller based on this scoring procedure. We evaluate the method on *ex vivo* corneas and find that it outperforms the state-of-the-art [39] in terms of both final error in cornea depth achieved by the needle (3.96 ± 1.48 % vs 7.80 ± 3.37 % for a target depth of 90%) and in terms of vertical error (43.29 ± 15.78 µm vs 75.71 ± 33.90 µm). Furthermore, the method achieves less than 5% error in relative depth (the threshold found by Yoo *et al.* [32] to be associated with reliable control of bubble type) in 61% of trials, as opposed to 42% for the baseline.

3.1 RELATED WORK

Robots have been widely studied in microsurgery applications to overcome the limits of human perception and dexterity. Tasks to which surgical robots have been applied include cochlear implantation [40], vascular anastomosis [41], and variocolectomy [42]. In particular, there has been considerable effort to develop robotic surgical tools for ophthalmic surgery [43, 44, 45, 46], including systems which use magnetic fields to guide the tool [47, 48]. Many of these systems are designed for teleoperation by human surgeons, but there has also been work on automatic motion planning for surgical robotics. In particular, automatic steering of flexible needles has been done using sampling-based planners [49] and inverse kinematics [50], though these techniques require a model of needle-tissue interaction. Learning from demonstration (LfD) has also been used for surgical tasks [51, 52], and Keller *et al.* [53] uses LfD and reinforcement learning (RL) for DALK. LfD and RL can pose safety concerns in a surgical setting and typically require large amounts of data to be effective, which can be expensive to obtain.

A significant challenge in surgical robotics is modelling the interaction of surgical instruments and tissues [54]. Finite Element Modelling (FEM) has been used to model such interactions [55], but such FEM models are often difficult to develop and computationally intensive to simulate. They are, therefore, not well-suited for the real-time demands of surgical robots. In other robotics domains, data-driven model predictive control (MPC) has

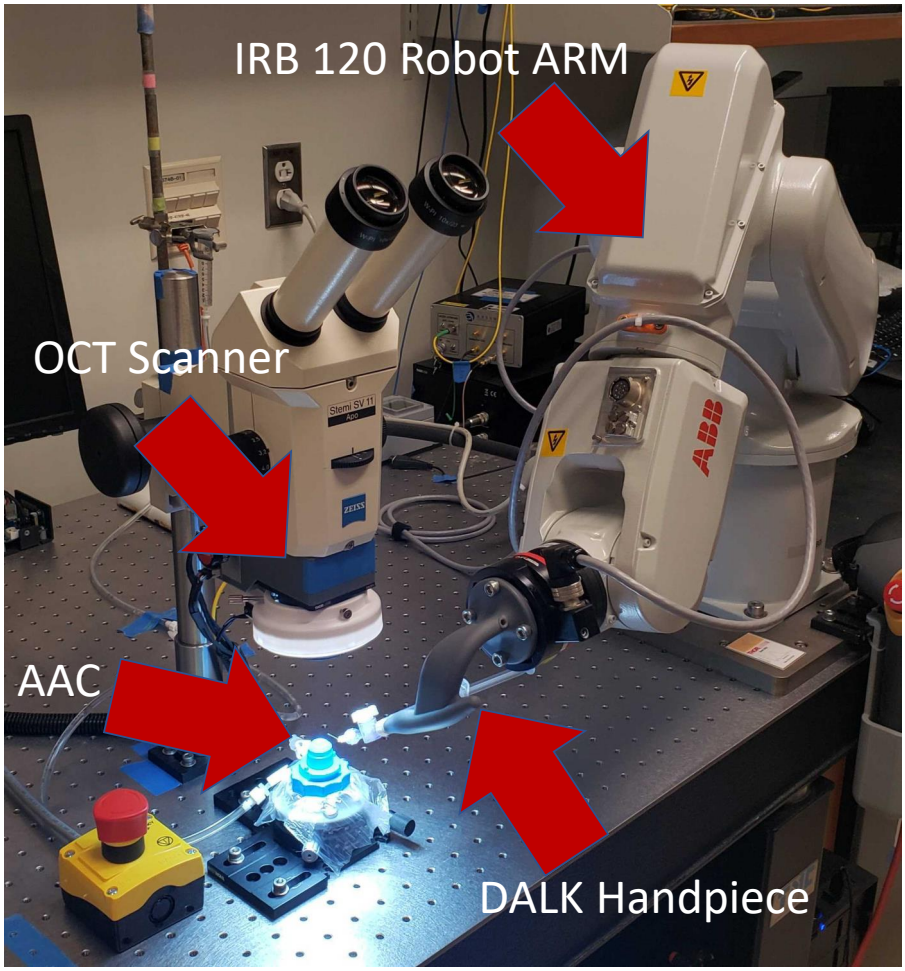


Figure 3.3: The DALK Workstation uses an IRB 120 robot arm which holds the needle via the DALK Handpiece. The artificial anterior chamber (AAC) holds the cornea sample while the OCT scanner provides feedback.

been used to control challenging systems such as cutting food [3], aggressive driving [18], and robotic fish [56]. Data-driven MPC is less computationally intensive than techniques like FEM and more data-efficient than LfD and RL, making it a more suitable choice for surgical robotics. In this work, we apply data-driven MPC to the DALK task.

3.2 DALK WORKSTATION

Our experimental platform is the DALK workstation (shown in Fig. 3.3) previously described in [39]. It includes a manipulation subsystem, which guides the needle, and a perception subsystem, which tracks the position of the needle and cornea. The manipulation subsystem consists of an IRB 120 robot arm (ABB robotics; Shanghai, China), and a

custom-designed "DALK handpiece," which attaches to the robot's end-effector and holds the 27-gauge cannulation needle. The arm and handpiece are designed so that they can guide the needle into the cornea without colliding with either the patient's anatomy or other equipment in the surgical theater. The perception subsystem consists of a custom optical coherence tomography (OCT) scanner mounted beneath a stereo microscope. The OCT scanner captures volumetric images of the cornea and needle during the DALK procedure at a rate of ∼1.5 Hz. Fig. 3.2 shows sample cross-sections of the OCT volumetric image, capturing the needle at several points through the insertion.

The perception subsystem extracts features of interest from the OCT-acquired volumetric image in real-time. The anterior and posterior cornea surfaces are segmented using Dijkstra's algorithm using the method described in [57]. The cornea apex is then identified by fitting a parabola to the posterior cornea surface. The needle is tracked by matching a 3D model of the needle to the OCT voxels using the iterative closest point algorithm [58]. The positions of both the needle and the posterior cornea surface are corrected for the index of refraction imposed by the curved anterior cornea surface. Refer to [39] for more details.

Draelos *et al.* [39] also proposed methods for automatic needle insertion. The insertion is divided into an *embedding phase* and an *advancement phase*. During the embedding phase, the robot executes an open-loop motion to embed the needle tip in the peripheral cornea. During the advancement phase, the needle is steadily advanced centripetally. Ideally, the needle tip moves steadily deeper as it is advanced, reaching the target depth at the cornea apex. A target depth of 90% is used since this depth has been associated with successful pneumodissection [59]. The advancement phase uses feedback from the perception system to correct the needle if it drifts off path. Draelos *et al.* describe multiple advancement planners; however, we take as a baseline the *line planner*, which attempts to follow a straight line path between the needle tip and the target position under the assumption of no needle or cornea deformation. In this work, we design a data-driven model predictive controller for the advancement phase which predicts and plans for the mechanical effects of needle-cornea interaction.

3.3 SOURCES OF ERROR

The DALK procedure requires great precision, with less than 50 µm on average separating a successful pneumodissection from a failure [59]. There are several sources of error which affect needle insertion performance. As the needle is inserted along its axial direction, both the cutting force at the needle tip required to separate the cornea tissue and the friction along the needle shaft are difficult to model. When the needle is pitched or translated along

its lateral direction, it will pull against the cornea tissue, both deforming the cornea and applying torque to the needle. These forces can cause the needle shaft and holder to flex independently from the robot end-effector, accentuating the non-rigid relationship between the end-effector and needle. The deformation induced in the cornea also complicates planning by effectively creating a moving target. We have also observed that even after the robot has stopped moving, the needle and cornea may continue to move due to residual tension in the needle shaft and cornea.

Limitations of the perception system can also create error. The RMS tracking error for the needle tip position and orientation is typically around 12 µm and 0.5° respectively, but failures of the ICP algorithm can occasionally lead to much larger errors or cause the needle tracking to fail entirely. Error in calibration and actuation give the robot a total repeatability of 25.4 µm [39].

As shown in Fig. 3.2, the opaque needle shaft creates a shadow which makes segmentation of the posterior cornea surface beneath the shaft difficult and prone to error. This is complicated by the fact that as the needle shaft is inserted, the cornea tissue is displaced, causing the posterior cornea surface to deform downward. This deformation occurs within the needle's shadow and makes it difficult to judge the needle's depth in real-time.

Finally, there are also anatomical differences between individual corneas, due to factors such as the patient's age, medical history, and in the case of *ex vivo* corneas, tissue preservation time. The cornea thickness varies, and we have observed that some corneas are more resistant to cutting than others. There is also variation in the cornea optical properties, and as a result we have observed that some corneas produce needle tracking failures and surface segmentation failures more frequently than others.

3.4 DATA-DRIVEN MODELLING OF NEEDLE-TISSUE INTERACTION

We train a data-driven model to predict the cornea-needle interaction, based on the features extracted by the perception system and the commanded robot arm movements. Formally, we consider a discrete-time sequence of features and control inputs, where $\mathbf{x}_t$ and $\mathbf{u}_t$ denote the features and control inputs at time t respectively and $\mathbf{x}_{l:h}$ and $\mathbf{u}_{l:h}$ denote the sequence of features and control inputs respectively between times l and h inclusive. The learning task is to predict $\mathbf{x}_{t+1:t+H}$ given $\mathbf{x}_{1:t}$ and $\mathbf{u}_{1:t+H-1}$, for a prediction horizon H.

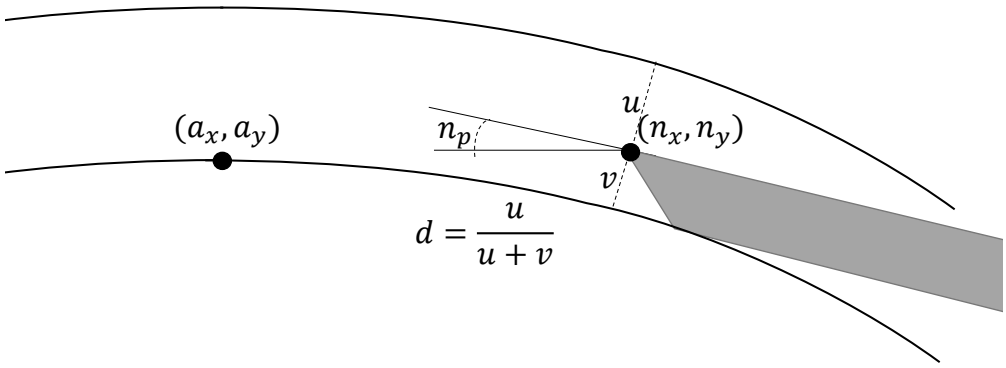


Figure 3.4: Selected state features include the 2D needle position (n_x, n_y), 2D apex position (a_x, a_y), needle pitch n_p, and needle depth d.

3.4.1 Dynamic model

We select the dynamic model and features of the system state based on their relevance to planning (see Fig. 3.4) and model selection experiments in Sec. 3.4.2. The needle state is represented by the 2D needle tip position in the OCT coordinate frame, denoted (n_x, n_y), and the needle pitch n_p. Since the needle predominantly moves within a plane and the OCT volume is re-sampled such that the z-axis is perpendicular to the needle shaft, it suffices to consider only two dimensions. We also include the 2D apex position in the OCT coordinate frame, denoted (a_x, a_y). Modelling apex movement allows the planner to account for cornea deformation as it guides the needle to its target position just above the cornea apex. Finally, we consider the needle depth ratio d. As shown in Fig. 3.4, the depth ratio is computed by finding the index-corrected ray normal to the anterior cornea surface which intersects the needle tip and then intersecting the ray with the posterior cornea surface. Modelling the depth ratio allows the planner to ensure that the needle is embedded in the cornea with sufficient depth and to accommodate variations in corneal thickness. The perception system also provides a segmentation of the cornea surface, but experiments below indicate that including cornea shape features reduces performance due to overfitting.

Control inputs are also modeled in the OCT coordinate frame. Prior to insertion, the system is calibrated using the method described in [39] to obtain a rigid transform between the robot end-effector and the needle tip. At each time step, forward kinematics calculates the *rigid needle tip position*, denoted (r_x, r_y), and the *rigid needle pitch*, denoted r_p. As previously noted, the rigid needle tip position will differ from the actual needle tip position due to the mechanical interaction of the needle and the cornea. We then choose our control inputs to be $[u_x]_t = [r_x]_{t+1} - [r_x]_t$, $[u_y]_t = [r_y]_{t+1} - [r_y]_t$, $[u_p]_t = [r_p]_{t+1} - [r_p]_t$. Since

movement is largely limited to the plane it suffices to consider only 2 dimensions. We define the control inputs in the OCT coordinate frame rather than the robot workspace and use the system calibration to convert to Cartesian robot movements. This approach makes the model invariant to variability in needle mounting.

Let N denote the number of insertions in the training set and let $\mathbf{x}_t^{(i)} = [n_x\, n_y\, n_p\, a_x\, a_y\, d]^T$ and $\mathbf{u}_t^{(i)} = [u_x\; u_y\; u_p]^T$ denote the features and controls respectively at the t-th timestep of the i-th insertion. Let T_i denote the i-th insertion length. We model the cornea-needle interaction using a linear autoregressive model with an exogenous variable (ARX), which models the dynamics as a linear function of a fixed-size window of the feature and control history. ARX has a long history of use in time-series prediction and control problems [60] and is straightforward to use in MPC. In contrast to nonlinear models such as neural networks, trajectory optimization with an ARX model can be formulated as a quadratic program (QP), which can be optimized quickly and reliably. Specifically, ARX predicts $\hat{\mathbf{x}}_{t+1} = \mathbf{A}_k\mathbf{\Theta}_t$, where

$$\mathbf{\Theta}_t = [\mathbf{x}_t\, \mathbf{x}_{t-1} \ldots \mathbf{x}_{t-k+1}\, \mathbf{u}_t\, \mathbf{u}_{t-1} \ldots \mathbf{u}_{t-k+1}\, 1]^T, \tag{3.1}$$

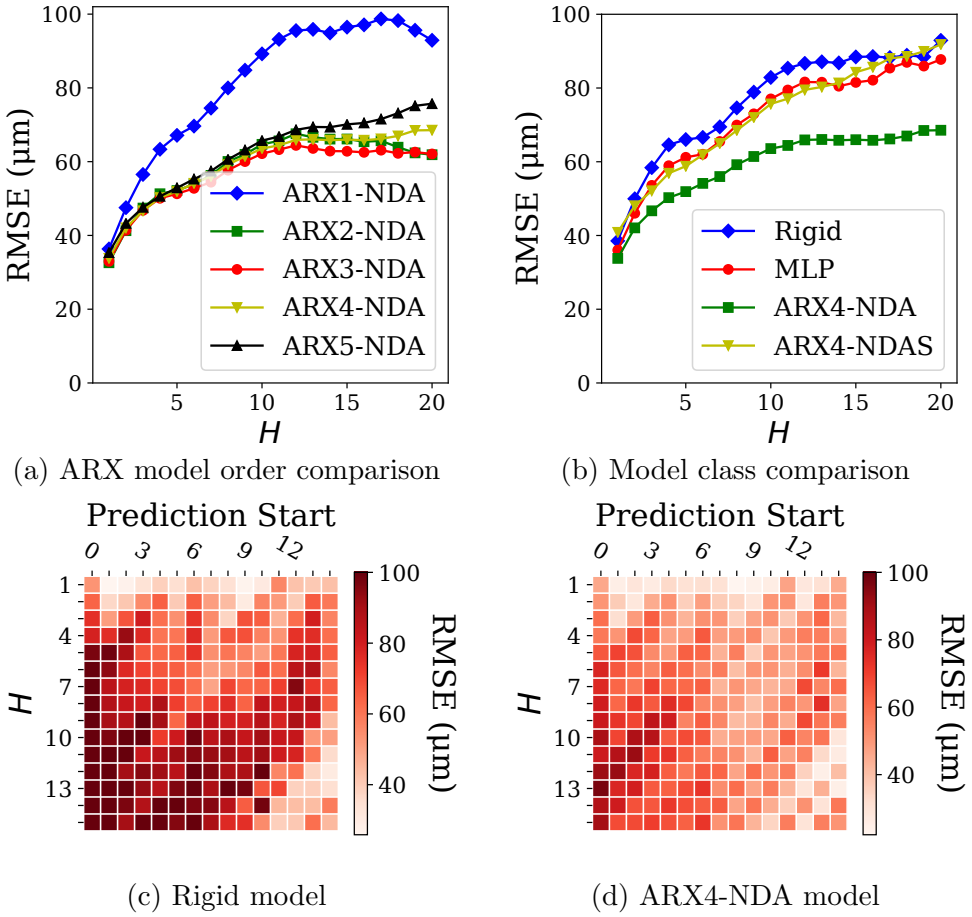
$\mathbf{A}_k$ is the learned parameter matrix, and k is a hyperparameter controlling the size of the history window, which is also known as the order of the model. Let $\mathbf{\Theta}_t^{(i)}$ denote the feature vector at the t-th time step of the i-th insertion. $\mathbf{A}_k$ is learned by using linear least-squares regression to minimize the objective

$$\sum_{i=1}^{N}\sum_{t=1}^{T_i-1}\left\|\mathbf{A}_k\mathbf{\Theta}_t^{(i)} - \mathbf{x}_{t+1}^{(i)}\right\|_2^2, \tag{3.2}$$

where $\|\cdot\|_2$ denotes the L2-norm. Since this ARX model considers features of the needle, depth, and apex, we refer it as the *ARXk-NDA* model. In order to make predictions within the first k time steps of an insertion, we also train separate ARX models with orders $1 \leq h \leq k$. We normalize the OCT coordinate frame by applying a translation so that the initial cornea apex position is at the origin. This makes the model invariant to translations of the cornea with respect to the OCT scanner.

3.4.2 Training and Model Selection

Using a dataset of 38 insertions from Draelos *et al.* [39], we combined 19 randomly selected insertions with 5 additional insertions taken from preliminary *ex vivo* experiments as a training set, and used the remaining 19 from Draelos *et al.* [39] for validation. Altogether,



(a) ARX model order comparison

(b) Model class comparison

(c) Rigid model

(d) ARX4-NDA model

Figure 3.5: Comparison of several models in needle tip (n_x, n_y) prediction accuracy. Heatmaps in (c,d) evaluate model accuracy at varying prediction horizons H and prediction starting timesteps.

this yields 771 time steps in the training set and 545 time steps in the validation set. We compare the prediction accuracy of several models on the validation set at varying prediction horizons. In Fig. 3.5a, we evaluate the impact of model order on the prediction accuracy of the ARXk-NDA model. We observe that at $k = 1$, the model prediction accuracy is significantly worse, while among $2 \leq k \leq 5$ the impact of model order is minor, only becoming apparent at the longest horizons. In Fig. 3.5b we compare ARXk-NDA to several other model classes, using $k = 4$ as a representative example. We compare against the rigid model and two other data-driven models. The multi-layer perceptron (MLP) model operates over the same feature space as the ARXk-NDA models, and uses hyperparameters automatically selected by AutoMPC [10]. The ARX4-NDAS model expands the feature space to include points sampled along the top and bottom cornea sufaces. For each surface, 10 points are sampled at even intervals in the x-axis. We find that the ARX4-NDA model significantly outperforms each of these alternatives at most prediction horizons. A key

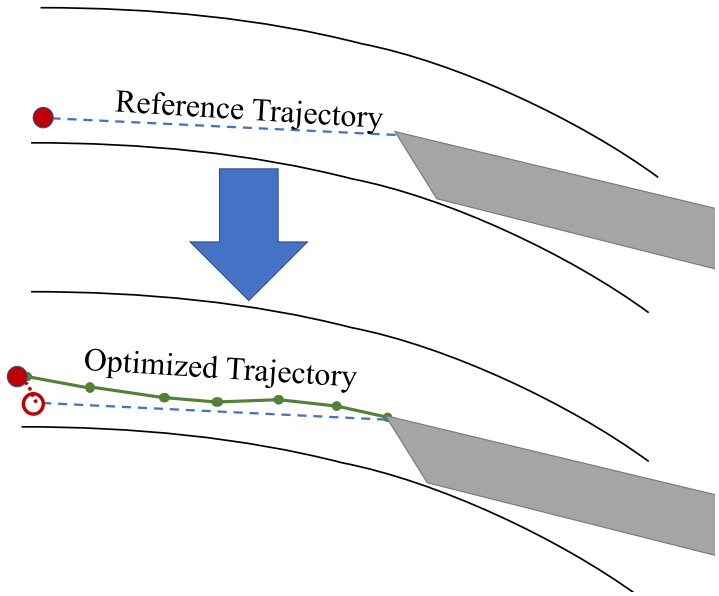


Figure 3.6: The needle advancement MPC begins by constructing a straight-line reference trajectory between the needle tip and goal, then optimizes a trajectory to track it, while predicting needle and apex motion.

property of the ARX model is that it considers a window of history, whereas the rigid and MLP models only use the most recent observation. The superior prediction accuracy of ARX suggests that historical observations and controls are very useful for predicting the needle-tissue interaction. On the other hand, the ARX4-NDAS model suffers from overfitting due to its larger feature space.

Figs. 3.5c and 3.5d evaluate the rigid and ARX4-NDA models respectively at varying prediction horizons and varying starting points in the insertion. We observe that the performance of the rigid model varies with the insertion starting point, with particularly high error occurring when prediction begins from the first few time steps. Though similar trends do hold for the ARX model, it does consistently outperform the rigid model at more than 90% of all starting points and prediction horizons.

3.5 NEEDLE ADVANCEMENT MPC

Using the data-driven model of cornea-needle interaction, we design a model predictive controller (MPC) to control the needle advancement phase. At a high level, the controller begins by constructing as a reference trajectory a straight line path between the needle tip and the goal position (see Fig. 3.6). The controller then optimizes for a trajectory consistent with the data-driven dynamics model which closely tracks the reference trajectory.

The optimized trajectory is penalized for deviations from the reference trajectory at both intermediary states and the terminal state.

3.5.1 MPC Optimization

During the advancement phase, the controller is triggered whenever a new OCT volume is acquired and processed, which normally occurs every $\sim$0.65 s. At this time, we observe the state $(n_x, n_y, n_p, a_x, a_y, d)$. The needle starting point is adjusted to account for the expected time for planning, which is denoted t_{lag}. This is computed as $s_x = n_x + t_{\text{lag}} v_x$, $s_y = n_y + t_{\text{lag}} v_y$, and $s_p = n_p + t_{\text{lag}} v_p$, where v_x, v_y, and v_p denote the current commanded needle tip velocities in x, y, and pitch respectively. The initial state for planning is $\mathbf{x}_{\text{init}} = [s_x\, s_y\, s_p\, a_x\, a_y\, d]^T$. The goal needle tip position is computed by adding a constant offset to the current apex position, $g_x = a_x + x_{\text{off}}$ and $g_y = a_y + y_{\text{off}}$. The goal needle pitch is horizontal $g_p = 0$ and the goal depth is set to $g_d = 90\%$. Finally, in order to limit cornea deformation, the goal apex is the same as the current apex position. Thus, the goal state is $\mathbf{x}_{\text{goal}} = [g_x\, g_y\, g_p\, a_x\, a_y\, g_d]^T$.

A reference trajectory is constructed for all state dimensions. The length of the reference trajectory is determined by the horizontal distance between the start position s_x and the goal position g_x as well as the desired x-axis speed v_x^{ref}. We have $m = \lfloor |s_x - g_x| / v_x^{\text{ref}} \rfloor$. The state reference trajectory is computed by linearly interpolating between $\mathbf{x}_{\text{init}}$ and $\mathbf{x}_{\text{goal}}$. That is, the i-th step of the reference trajectory is given by $\mathbf{x}_i^{\text{ref}} = \frac{i}{m}\mathbf{x}_{\text{goal}} + \frac{m-i}{m}\mathbf{x}_{\text{init}}$ for $0 \leq i \leq m$. The reference control trajectory is constructed as the sequence of controls needed to achieve the reference state trajectory assuming the rigid dynamics model. That is, for the x-axis, we have $[u_x^{\text{ref}}]_i = [n_x^{\text{ref}}]_{i+1} - [n_x^{\text{ref}}]_i$ for $0 \leq i < m$ and u_y^{ref} and u_p^{ref} are defined similarly.

Next, we optimize for a state and control trajectory which tracks the reference trajectory and is consistent with the data-driven dynamics model. This is done by constructing and solving a quadratic program (QP). Although the reference trajectory is m steps long, in order to account for movement of the needle after the robot has stopped moving, we solve for a longer $m + l + 1$ state trajectory with the final l steps having zero control input. The

quadratic program is given by

$$\begin{aligned}
\min_{\mathbf{x}_{0:m+l},\mathbf{u}_{0:m-1}} \quad & \left(\sum_{i=1}^{m} \bar{\mathbf{x}}_i^T Q \bar{\mathbf{x}}_i + \bar{\mathbf{u}}_i^T R \bar{\mathbf{u}}_i\right) + \tilde{\mathbf{x}}_m^T F \tilde{\mathbf{x}}_m + \tilde{\mathbf{x}}_{m+l}^T G \tilde{\mathbf{x}}_{m+l} \\
\text{s.t.} \quad & \mathbf{x}_0 = \mathbf{x}_{\text{init}} \\
& \mathbf{x}_i = \mathbf{A}_h \left[\mathbf{x}_{i-1} \dots \mathbf{x}_{i-k}\ \mathbf{u}_{i-1} \dots \mathbf{u}_{i-k}\ 1\right]^T \\
\text{where} \quad & \bar{\mathbf{x}}_i = \mathbf{x}_i - \mathbf{x}_i^{\text{ref}} \qquad \bar{\mathbf{u}}_i = \mathbf{u}_i - \mathbf{u}_i^{\text{ref}} \\
& \tilde{\mathbf{x}}_i = \left[[n_x]_i - [g_x]_i \quad [n_y]_i - [g_y]_i \quad [d]_i - g_d\right]^T \\
& h = \min(i, k).
\end{aligned} \tag{3.3}$$

Q, R, F, and G denote tune-able cost matrices. The QP is solved using the OSQP [61] solver.

To achieve good real-world performance, there are several additional practical considerations. To avoid large out-of-plane angular changes near the apex, replanning is stopped once the needle-apex horizontal distance is less than 0.5 mm and the last active plan is followed until completion. The controller must also be able to gracefully handle failures of perception, for which we use a simple outlier rejection scheme. When the observed needle position differs from the needle position predicted by the rigid model by more than a certain threshold, we assume a perception failure. In this case, the rigid model prediction is used to replace the observed needle position, and the observed needle depth is replaced by the observation at the previous time step. Outlier rejection does not apply to the apex observations which are less susceptible to perception failures.

3.5.2 Cross-Simulation Controller Selection

The designer of a data-driven MPC must make a number of key choices, such as feature selection, model class selection, cost matrix tuning, and regularization. Each of these can have a significant impact on system performance, so the designer must weigh each option carefully. The gold standard would be to evaluate each design option with a sufficiently large sample size of physical experiments in order to achieve a reliable comparison. However in many applications, this approach would be prohibitively expensive and time-consuming for comparing more than handful of candidate designs. Instead, designers often use simulators as a proxy for physical experiments. This approach often suffers from the so-called *sim-to-real* gap, where inaccuracies in the simulation create bias in the estimation of MPC performance, a problem which can be especially significant in hard to model systems such as those involving deformable objects. Moreover, in a novel application, developing a high-

quality simulator may be itself a more challenging problem than designing a data-driven MPC. We faced both of these challenges in the DALK task. Physical experiments are not only time and labor-intensive, but consume a limited supply of cornea tissue samples, and the interaction of the needle with deformable tissue is very hard to model in simulation.

Instead we propose *cross-simulation controller selection*. We randomly re-sample the training set for the planning model in order to obtain a simulation model training set, which is used to train an ARX simulation model. We then simulate the needle advancement MPC using the planning model in the controller and the simulation model in place of a simulator. This process is repeated, each time with a new simulation model created through random re-sampling. This technique allows us to assess how robust the MPC is to model uncertainty caused by the limited size of the training set, and we find it to be empirically a good predictor of real-world performance.

We use the cross-simulation technique to compare several candidate controllers, which mainly differ in the choice of planning model. For each controller, we run 100 trials, each with a different simulation model and initial configuration. Each simulation model is of the same class as the planning model, but uses a randomly bootstrapped training set. Each initial configuration is sampled randomly from real-world data. For each trial, we evaluate the final depth error $|d - g_d|$ and the final error in vertical position $|n_y - g_y|$. We present the summary statistics in Table 3.1.

First, we evaluate the impact of ARX model order on controller performance. We consider orders $3 \leq k \leq 5$ and denote the corresponding controllers ARX3-NDA-MPC, ARX4-NDA-MPC, and ARX5-NDA-MPC. For brevity, we will drop the -MPC suffix when this does not create ambiguity. We find that of these three controllers, ARX4-NDA achieves the lowest error both in final depth and final vertical position, which leads us to choose $k = 4$ as the model order. We note that this occurs in spite of the fact that the ARX3-NDA model had slightly better prediction accuracy, suggesting that prediction accuracy alone is not sufficient to evaluate the suitability of a model for control. Qualitatively, we observe that every controller produces some outliers with much higher depth and vertical error, creating relatively high standard deviations. Typically, this is caused by sudden deformations of the needle or apex in the last few steps of the simulation. When the model fails to predict these deformations, the needle cannot correct, so errors remain high.

In the interest of avoiding overfitting and unnecessary complexity in trajectory optimization, we next consider performing feature selection amongst ARX models. We propose several controllers which use *hybrid models*, where ARX is used to predict only some state dimensions, while the rigid model is used to predict others. For example, the *ARX4-DA-RN-MPC* controller uses the rigid model to predict the needle tip, but uses ARX for the

Table 3.1: Results of cross-simulation controller selection. 100 trials are run per controller. We report final error in relative depth and vertical position (smaller is better), with ±95% CI.

	Depth Error (%)		**Vertical Error (µm)**	
	μ	σ	μ	σ
ARX3-NDA-MPC	6.9±1.0	5.1	41.3±8.4	42.9
ARX4-NDA-MPC	6.1±0.9	4.8	40.4±8.1	41.3
ARX5-NDA-MPC	7.1±1.2	6.2	65.9±12.8	65.1
ARX4-DA-RN-MPC	41.5±5.0	25.7	1,137.2±90.1	459.9
ARX4-ND-RA-MPC	7.7±1.0	4.9	50.6±8.3	42.2
ARX4-NA-MPC	7.2±0.9	4.5	36.3±5.3	26.8

apex and depth dimensions. The *ARX4-ND-RA-MPC* controller is similar, but uses the rigid model to predict the apex. Since the rigid model cannot be used to predict the needle depth, we cannot use the hybrid model to assess the impact of modelling the depth dimension. Instead, the *ARX4-NA-MPC* controller entirely removes the depth feature from the model, as well as the associated cost terms in the MPC formulation. For these comparisons, we always take the simulation model to be the ARX4-NDA, rather than choosing the same class as planning model. We find that while the ARX4-NA slightly outperforms ARX4-NDA in terms of vertical accuracy at $36.43 \pm 5.2\,\mu m$ vs $40.4 \pm 8.1\,\mu m$, none of the three outperform ARX4-NDA in terms of depth error, which is the most clinically relevant factor. Thus, we choose to proceed with ARX4-NDA-MPC in the *ex vivo* experiments.

3.6 EX VIVO EXPERIMENTS

We evaluate our controller on human cadaver corneas in an *ex vivo* setting, simulating intraocular conditions using an artificial anterior chamber (Katena Products; Denville, NJ). This model has been well-validated in literature [34, 62], and we used only cornea samples with short preservation times suitable for transplantation, so the effects of post-mortem deterioration should be minimal. We used 6 corneas and performed 8 needle insertions per cornea, for a total of 48 trials. The baseline controller is the automatic line planner used in [39]. For each cornea, we randomly assigned half of the trials to the MPC and half to the baseline. Two of the MPC trials were excluded due to needle tracking failures in the final steps of the insertion which prevented accurate measurement of the final depth. We choose the target depth to be 90% and measure the actual final depth based on the needle and cornea segmentations given by the perception system (referred to as auto-graded depth).

Table 3.2 summarizes the results.

We find that the MPC significantly ($p < 0.05$) outperforms the baseline in terms of final depth error, and also outperforms the baseline in terms of final vertical error, though this result is not statistically significant due to the large variance in the baseline. This variance is caused by a number of factors, including perception noise, unpredictable needle and tissue deformation, and differences in cornea geometry and mechanical properties. We also note that under the baseline planner, the needle perforated the posterior cornea surface in 2 of 24 insertions, which would result in a clinical failure. No perforations were observed in the MPC insertions. Fig. 3.7 shows an example plan generated by the MPC in one of the *ex vivo* experiments. Qualitatively, we observe that the needle tracks the planned path well, and that the MPC correctly anticipated that the needle would take a lower path than the rigid model would have predicted.

To better understand the factors influencing performance in the *ex vivo* experiments, we perform *post hoc* analyses. First, we calculate the plan tracking error for both MPC and baseline. For a starting time t and a planning horizon H_{plan}, the plan tracking error is the minimum distance between planned path computed at time t and the actual position $[(n_x, n_y)]_{t+H_{\text{plan}}}$. The plan tracking error is averaged over all insertions and starting points. In essence, this metric measures how accurately the needle reaches the positions intended by the planner. Fig. 3.8a compares the plan tracking errors of the MPC and baseline over varying planning horizons. We observe that while the MPC tracking error is higher than the baseline at the one-step planning horizon, the MPC exhibits lower tracking error for all longer horizons. This indicates that the MPC is superior to the baseline in producing realistic plans that can be followed by the needle. In the baseline trials, we frequently observe that the needle deflects significantly early in the insertions, but this deflection decreases as the needle is inserted deeper. As a result, the plan tracking accuracy of the baseline actually improves as the horizon increases from 10 to 15 time steps. Although the cause of this phenomenon is not known, it may be that as the needle is inserted, the tension created by needle deflection increases until some other part of the system begins to give, allowing the needle to return closer to its rigid position.

We also analyze model prediction accuracy on a testing set taken from the *ex vivo* experiments (Fig. 3.8b), comparing the ARX and rigid models. Both models perform worse on the testing set compared to the validation, but ARX still outperforms the rigid model at all prediction horizons. The difference in model accuracy between the validation and testing set may be partially explained by domain shift, since the MPC drives the system to a different distribution of states than the planners used in the training and validation sets.

Finally, we note that while the results presented here based on the auto-graded depth are

Table 3.2: Results of $N = 46$ insertions (22 MPC, 24 Baseline) on 6 *ex vivo* corneas. We report # of perforations, final error in relative depth (auto-graded), and vertical position (smaller is better), with ±95% CI. P-values are from a two-sample t-test.

	Perforations	**Depth Error (Auto) (%)**		**Vertical Error (µm)**	
		μ	σ	μ	σ
MPC	0	3.96±1.48	3.35	43.29±15.78	35.60
Baseline	2	7.80±3.37	7.98	75.71±33.90	80.27
P-Value		0.039		0.082	

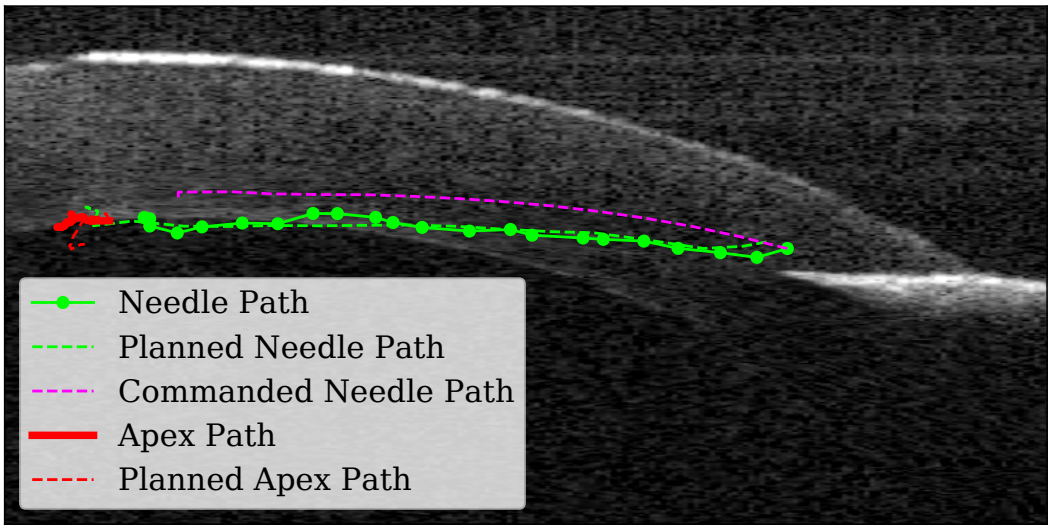
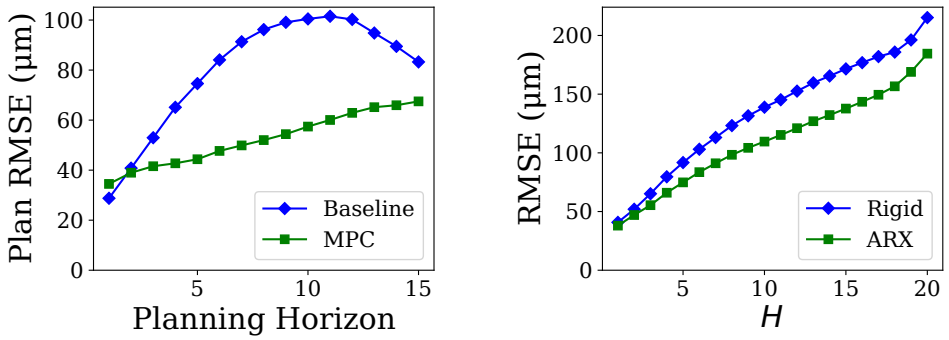


Figure 3.7: Example plan produced by MPC in testing.

significant, we also manually graded the depths to correct for perception errors and did not find a significant difference between the MPC and the baseline in terms of depth error (MPC $5.40 \pm 1.96\%$ vs Baseline $5.45 \pm 3.11\%$). This indicates that the performance of the planner is constrained by limitations of the perception system. Improving the perception accuracy is outside the scope of this work, but an important area of future research.

3.7 DISCUSSION & CONCLUSION

In this chapter, we demonstrate that data-driven techniques can accurately model the interaction between cannulation needle and cornea in the DALK procedure. We also demonstrate that the technique of cross-simulation controller selection can evaluate controller per-



(a) Plan tracking error

(b) Model accuracy on testing

Figure 3.8: Analysis of the *ex vivo* experiments. 3.8a compares the plan tracking accuracy in the needle tip (n_x, n_y) dimension of the MPC vs the baseline planner; 3.8b evaluates model prediction accuracy in the needle tip (n_x, n_y) dimension on the *ex vivo* data.

formance with offline data, and the results are predictive of real-world performance. The data-driven model predictive controller (MPC) developed using this approach enables the surgical robot to achieve superior accuracy in needle placement compared to the baseline. These improvements have the potential to significantly improve the reliability and consistency of the DALK procedure. In future work, we hope to directly evaluate the impact data-driven MPC on the pneumodissection success rate, particularly when combined with an improved perception system.

CHAPTER 4: AUTOMPC PACKAGE DESIGN

In this section, we describe the design of `AutoMPC`, an open-source Python package for automatic synthesis of data-driven model predictive controllers (MPC). First, in Sec. 4.1, we define the MPC synthesis problem and describe our approach based on simulation with learned surrogate dynamics. Second, in Sec. 4.2, we discuss the details of our implementation of `AutoMPC`, including including the specific System ID models, optimization algorithms, and objective functions which our package provides.

4.1 DATA-DRIVEN MPC TUNING

We automate MPC using the framework illustrated in Fig. 4.1. As input, the user provides a dataset $\mathcal{D}$ of state/control trajectories, and a task $\tau = (J, \mathbf{u}_{\min}, \mathbf{u}_{\max}, \mathbf{x}_{\min}, \mathbf{x}_{\max}, \mathbf{I})$, where J is a performance metric which assigns numerical scores to trajectories, and $(\mathbf{u}_{\min}, \ldots, \mathbf{x}_{\max})$ give bounds for the controls and states, and $\mathbf{I}$ is a set of initial states. A complete MPC controller consists of a SysID model, learned from a training subset, and an optimizer. Each component has number of tunable *hyperparameters* (Sec. 4.1.2), with a hyperparameter setting denoted a *configuration*. The controller defined by a configuration is optimized end-to-end using Bayesian optimization (Sec. 4.1.3).

At the start of tuning, $\mathcal{D}$ is randomly partitioned a SysID training set $\mathcal{D}_I \subset \mathcal{D}$ and a holdout dataset $\mathcal{D}_H = \mathcal{D} \setminus \mathcal{D}_I$, which is used to train a *surrogate dynamics model*. We then evaluate configurations in an order selected by Bayesian optimization. To evaluate a configuration, we synthesize the corresponding MPC by learning a dynamics model from $\mathcal{D}_I$, initializing the optimizer, and selecting the objective function based on the configuration hyperparameters. Control actions are then selected by the optimizer using the dynamics

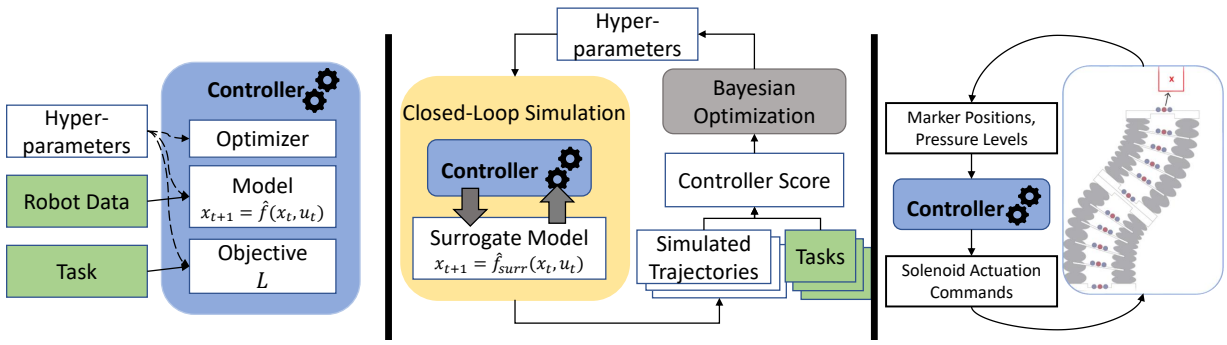


Figure 4.1: AutoMPC controller synthesis, tuning, and deployment.

model and objective function. We then evaluate the closed-loop performance of the MPC from the initial states $\mathbf{I}$ using the surrogate dynamics model. After a fixed number of iterations, the best performing controller is provided as output. Details are given below.

4.1.1 Problem Definition

We assume a fully-observable discrete-time dynamical system with state $x_t \in \mathbb{R}^n$ and control $u_t \in \mathbb{R}^m$. We use the notation $\mathbf{x}_{t:r}$ and $\mathbf{u}_{p:q}$ to denote the sequences $(x_t, x_{t+1}, \dots x_r)$ and $(u_p, u_{p+1}, \dots, u_q)$ respectively. The dynamics of the system are written as

$$x_{t+1} = f(x_t, u_t). \tag{4.1}$$

At each time step, MPC optimizes a trajectory over a fixed horizon H with respect to some objective function L and constraints ϕ. This optimization has the form

$$\begin{aligned} &\min_{\mathbf{x}_{t:t+H}, \mathbf{u}_{t:t+H-1}} L(\mathbf{x}_{t:t+H}, \mathbf{u}_{t:t+H-1}) \\ &\text{s.t. } x_{i+1} = f(x_i, u_i) \text{ and } \phi(\mathbf{x}_{t:t+H}, \mathbf{u}_{t:t+H-1}) \leq 0. \end{aligned} \tag{4.2}$$

We do not have access to the true system dynamics, so we must estimate an approximate model $\hat{f}$ identified from $\mathcal{D}$.

The user designates a performance metric $J(\mathbf{x}_{1:T}, \mathbf{u}_{1:T})$ which scores rolled-out trajectories. We allow J to have arbitrary form. The terminal time T may be constant or dynamically determined by some goal condition, also specified.

Note that we distinguish the objective L used in the optimizer from the performance metric J, for several reasons. First, certain optimization methods require L to have a particular structure. For example, LQR requires L to be quadratic while iLQR requires L to be twice differentiable. Second, J is evaluated over the entire trajectory while L is only optimized over a fixed horizon H, and optimizing repeatedly over a fixed horizon may not lead to good overall performance. This is particularly true for metrics that include terminal cost, since terminal cost does not provide useful guidance to the MPC until near the end of the trajectory. Third, certain optimizers cannot accept state or control constraints, so the objective function may encode constraints as barriers in the objective function. Finally, L may include regularization terms that penalize deviation from the training data.

Note that learning $\hat{f}$ from $\mathcal{D}$ can be viewed as supervised learning, but the MPC context adds additional considerations. For example, a simpler, less accurate model may be preferred to a more complex, more accurate one if the former is cheaper to evaluate, suffers from fewer

local minima, or is less prone to overfitting. Moreover, the generalization performance of the model outside of the data distribution is critical, as the MPC may guide the system away from the distribution in order to "exploit" the model inaccuracies and produce unrealistic trajectories. This is known as *distribution shift* and can be partially alleviated by tuning L to penalize trajectories which deviate from the data distribution.

4.1.2 Hyperparameters

Hyperparameters can take on a mixture of continuous, integer, and categorical values. We also allow for conditional relationships between hyperparameters. For example, a continuous hyperparameter specifying the weight of a particular term in the objective function can be conditioned on a boolean hyperparameter which turns the term on or off.

First, we let $\mathcal{H}_S$ denote the set of hyperparameters for the SysID method. For example, the linear autoregression model uses an integer k to control the size of the state history. Next, we let $\mathcal{H}_L$ denote the set of objective function hyperparameters. For example, a quadratic cost function might rescale the diagonal values of the cost matrices. Finally, we let $\mathcal{H}_O$ denote the set of optimizer hyperparameters. This includes the planning horizon H as well as any other optimizer-specific settings, e.g., MPPI sets the number of trajectories that are sampled in each iteration. Considering these components together, we have a joint hyperparameter space $\mathcal{H} = \mathcal{H}_S \times \mathcal{H}_L \times \mathcal{H}_O$ for the end-to-end system.

4.1.3 Tuning with Surrogate Functions

`AutoMPC` implements tuning in three modes: 1) End-to-end tuning, which tunes the entire pipeline for closed-loop performance, 2) SysID tuning, which only tunes the dynamics model for accuracy, 3) Decoupled tuning, which first performs SysID tuning and then tunes the optimizer for closed-loop performance.

To search the hyperparameter space $\mathcal{H}$, we use the Bayesian optimization algorithm Sequential Model-based Algorithm Configuration (SMAC) [63], as implemented in the Python package `smac3`. SMAC builds a model using a random forest to predict the performance of a configuration h before it is evaluated. This model is used to select the next configuration to evaluate. The use of a random forest, in contrast to the Gaussian Process models used by many other Bayesian optimization algorithms, allows SMAC to handle structured hyperparameter spaces that contain a mixture of discrete and continuous hyperparameters, and conditional hyperparameter relationships. The details of the three tuning modes are as follows:

Algorithm 4.1: End-to-end AutoMPC tuning.

Input: Task τ, dataset $\mathcal{D}$, surrogate config h_{surr}, number of iterations n
Randomly partition $\mathcal{D}_I \cup \mathcal{D}_H = \mathcal{D}$, $\mathcal{D}_I \cap \mathcal{D}_H = \varnothing$
$\hat{f}_{surr} \leftarrow$ `TRAIN`$(\mathcal{D}_H, h_{surr})$
history = { }
for $i \leftarrow 1$ to n **do**
  $h \leftarrow$ `BAYES-OPT`(history)
  $\hat{f} \leftarrow$ `TRAIN`$(\mathcal{D}_I, h)$
  $L \leftarrow$ `BUILD-OBJECTIVE`(h)
  optimizer $\leftarrow$ `BUILD-OPTIMIZER`(h)
  controller $\leftarrow$ `BUILD-CONTROLLER`($\hat{f}$, L, optimizer)
  trajectory $\leftarrow$ `SIMULATE`(controller, $\hat{f}_{surr}$, τ)
  $\hat{\mathbb{J}}[h] \leftarrow$ `SCORE`(trajectory, τ)
  history $\leftarrow$ history $\cup$ $(h, \hat{\mathbb{J}}[h])$
end for
Return h with minimum $\hat{\mathbb{J}}[h]$

End-to-End tuning Given a configuration $h \in \mathcal{H}$, we learn $\hat{f}$ from $\mathcal{D}_I$ and derive an MPC controller $\pi_{h,\hat{f}}$. We define the controller's *true performance* as

$$\mathbb{J}[h] = \sum_{s^i \in \mathbf{I}} J(\mathbf{x}^i_{1:T}, \mathbf{u}^i_{1:T-1}) \tag{4.3}$$

where $\mathbf{I}$ is the set of initial states, $x^i_1 = s_i$, and each trajectory is generated by a closed-loop rollout of $\pi_{h,\hat{f}}$ to the true dynamics f.

Without access to f, we define a *surrogate performance* $\hat{\mathbb{J}}$ that is identical to (4.3) except the rollout is performed with respect to a learned surrogate dynamics model $\hat{f}_{surr}$, which is learned on the holdout set $\mathcal{D}_H$. We avoiding sharing data between system ID and surrogate training sets to ensure that the surrogate is not identical to model $\hat{f}$ used for control. We use Bayesian optimization to minimize $\hat{\mathbb{J}}$ over $\mathcal{H}$. Although $\hat{\mathbb{J}}$ is an imperfect approximation of $\mathbb{J}$, our experiments show that it is still useful for tuning. Specifically, for two controllers π_1, π_2 with a non-negligible difference in $\mathbb{J}(\pi_1)$ and $\mathbb{J}(\pi_2)$, it almost always holds that $\hat{\mathbb{J}}(\pi_1) < \hat{\mathbb{J}}(\pi_2)$.

The end-to-end tuning procedure is described in Alg. 4.1.

SysID Tuning This optimizes the SysID hyperparameters $\mathcal{H}_S$ for accuracy, akin to classical model selection. `AutoMPC` allows the choice to tune for 1-step prediction accuracy or k-step prediction accuracy on the testing set $\mathcal{D}_H$.

Decoupled tuning This approach first performs SysID tuning to fix $\hat{f}$ and then tunes the configuration over $\mathcal{H}_L \times \mathcal{H}_O$ to optimize performance $\hat{\mathbb{J}}$.

4.2 IMPLEMENTATION

We provide an implementation of `AutoMPC` as an open-source Python library. We designed the package so that: *1)* `AutoMPC` should be accessible enough that a non-expert should be able to achieve a performant MPC with minimal manual tuning; *2)* `AutoMPC` should be useful to experts, providing tools to analyze system performance, and allowing a combination of automatic and manual fine-tuning; *3)* `AutoMPC` should provide a uniform API for components, allowing users to implement their own methods to be tuned by `AutoMPC`. The components implemented in the current version of `AutoMPC` and their hyperparameters are summarized in Tab. 4.1 and described in more detail next.

4.2.1 System Identification

Each SysID technique estimates the dynamics function f from a dataset $\mathcal{D}_I$.

ARX: A linear autoregression predicting the state as a linear function of the state and control history for the previous k time steps. That is

$$x_{t+1} = [x_t, \dots x_{t-k+1}, u_t, \dots, u_{t-k+1}]\theta \tag{4.4}$$

with θ the model coefficients. Training is performed using least-squares regression on the prediction error. The hyperparameter for ARX is the size of the history window k.

Koopman Operators learn a linear operator over an augmented state $\bar{x} = [x, \phi_1(x), \dots, \phi_s(x)]^T$, where $\phi_1, \dots, \phi_s$ are referred to as basis functions [64]. We use hyperparameters to select the basis functions, which can include polynomial terms x^s with $2 \leq s \leq 8$ and trigonometric terms $\sin(\omega x)$ and $\cos(\omega x)$ where $1 \leq \omega \leq 8$.

Sparse Identification of Nonlinear Systems (SINDy) represents dynamics in the form

$$f(x_t, u_t) = \sum_{i=1}^{N} a_i f_i(x_t, u_t), \tag{4.5}$$

where $f_1, \dots, f_N$ are nonlinear basis functions [65]. SINDy uses a fixed set of candidate functions $g_1, \dots, g_M$ and performs sparse linear regression to identify a subset of $N < M$ basis functions for the dynamics. We use the pySINDy library [66] in our implementation

Table 4.1: For each SysID method, objective function, and optimization method, we list the total number of hyperparameters, and the names of hyperparameters. The possible number of active hyperparameters, which varies depending on the choice of values, is listed in parentheses.

SysID	**#hyper. (act.)**	**Hyperparameters**
ARX	1 (1)	`history`
Koopman	4 (3-4)	`usepolybasis, polydegree, usetrigbasis, trigfreq`
SINDy	4 (2-4)	`usepolybasis, polydegree, usetrigbasis, trigfreq`
GP	1 (1)	`inducingcount`
MLP	7 (4-7)	`numhiddenlayers, hiddensize{1,2,3,4}, learnrate, activation`
Objective	**#hyper (act.)**	**Hyperparameters**
Simple Quadratic	$2n+m$ $(2n+m)$	`qdiagvals{1,..,n}, fdiagvals{1,..,n}, rdiagvals{1,..,m}`
Gaussian Reg. Term	2 (2)	`stateregweight, controlregweight`
Optimization	**#hyper. (act.)**	**Hyperparameters**
LQR	2 (1-2)	`ishorizonfinite, horizon`
Direct Transcription	1 (1)	`horizon`
iLQR	1 (1)	`horizon`
MPPI	4 (4)	`horizon, numtrajs, noisemagn, costscale`

and use hyperparameters to select the set of candidate functions in the same way that we select basis functions for the Koopman operator.

Gaussian Processes (GPs) are a non-parametric model that has been commonly used for MPC [4]. Standard GPs use the full training set for inference, so they do not scale well to larger data sets. Instead, we use an approximate variational GP [67] implemented by the `gPyTorch` library [68], which selects a learnable subset of training points to use for inference. This subset is referred to as the inducing set. The hyperparameter we use for GP is the size of the inducing set.

Multi-layer Perceptrons (MLP) A feed-forward neural network architecture. We use hyperparameters to control the number of hidden layers in the network, the number of neurons in each layer, the choice of activation function (from ReLU, SeLU [69], tanh and sigmoid), and the learning rate during training. The number of training iterations are currently fixed.

4.2.2 Objective Functions

The next tunable component is the optimizer's objective function L. `AutoMPC` allows users to specify custom objective functions with tunable hyperparameters. We provide several predefined objectives which can be used as building blocks to define an objective. First, we consider the simple **quadratic objective**

$$\begin{aligned} L_Q(\mathbf{x}_{t:t+H}, \mathbf{u}_{t:t+H-1}; h_L) &= (x_{t+H} - \mathbf{x}_{\text{goal}})^T F (x_{t+H} - \mathbf{x}_{\text{goal}}) \\ &+ \sum_{i=t}^{t+H} [(x_i - \mathbf{x}_{\text{goal}})^T Q (x_i - \mathbf{x}_{\text{goal}}) + u_i^T R u_i], \end{aligned} \tag{4.6}$$

with hyperparameters $h_1 \dots h_{2n+m}$ dictating the matrices

$$\begin{aligned} &Q = \text{diag}(h_1, \dots, h_n), F = \text{diag}(h_{n+1}, \dots, h_{2n}), \\ &\text{and } R = \text{diag}(h_{2n+1}, \dots, h_{2n+m}). \end{aligned} \tag{4.7}$$

This objective can be effective for simple tasks which drive the system to a target state, such as the pendulum and cart-pole swing-up tasks.

The objective function can also include a **regularization term** to guide the optimization toward regions of the state space where model accuracy is better. We implement an option to penalize deviation from the data distribution of $\mathcal{D}$ as modeled by a multivariate Gaussian with mean μ_x and covariance matrix Σ_x. Similarly we model the controls in $\mathcal{D}$ with mean μ_u and covariance Σ_u. We add a regularization term to the objective:

$$\begin{aligned} L_R(\mathbf{x}_{t:t+H}, \mathbf{u}_{t:t+H-1}; h_L) = \sum_{i=t}^{t+H} & h_1 (x_i - \mu_x)^T \Sigma_x^{-1} (x_i - \mu_x) \\ & + h_2 (u_i - \mu_u)^T \Sigma_u^{-1} (u_i - \mu_u) \end{aligned} \tag{4.8}$$

with hyperparameters h_1, h_2.

4.2.3 Optimizers

We currently implement four optimizers. Each method has a hyperparameter for the planning horizon.

Linear Quadratic Regulators (LQR) solve the optimal control problem for tasks with linear models and quadratic cost functions [70]. We introduce a categorical hyperparameter to choose between finite and infinite horizon LQR.

Direct Transcription (DT) is a common method for formulating trajectory optimization as a nonlinear programming (NLP) problem [71]. DT requires the system model and the cost function to be differentiable. Our implementation uses IPOPT [72] to solve the NLP.

Iterative Linear Quadratic Regulator (iLQR) is a popular method for trajectory optimization similar to DT [1]. iLQR requires the system model to be differentiable and the cost function to be twice-differentiable.

Model Predictive Path Integral (MPPI) is a sampling-based optimizer that can be used with non-differentiable system models, and has been demonstrated to work effectively with neural networks [18]. We introduce hyperparameters for the number of trajectories sampled on each iteration, the magnitude of the noise, and the cost scaling factor.

Note that the choice of optimizer is constrained by the choices of system model and objective function, and vice versa. For example, if the model is linear and the objective function is quadratic, then an LQR controller may be used, but a nonlinear system model requires the use of a more advanced optimizer. iLQR and Direct Transcription require the system model to be differentiable, while MPPI can work with non-differentiable models. At the moment, the model and optimizer classes must be chosen manually, but in future work we are exploring auto-tuning both.

CHAPTER 5: AUTOMPC RESULTS

We consider three robotics tasks: pendulum swing-up, cart-pole swing-up, and half-cheetah running as implemented in the HalfCheetah-v2 environment in OpenAI Gym [73]. Each training set consists of 1,000 trajectories generated by applying, at each time step, controls chosen randomly from a uniform distribution. Each trajectory lasts for 10 s and the time step is 0.05 s. The train-test split is 50-50, and a MLP is trained as the surrogate model. The pendulum and cart-pole tasks are to move from the pole-down state to the pole-up state x_{goal}, and use the following rollout performance metric

$$J(\mathbf{x}_{1:T}, \mathbf{u}_{1:T-1}) = \sum_{i=1}^{T} \begin{cases} 1 & \mid x_i - x_{\text{goal}} \mid_\infty > \delta \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

where $\delta = 0.1$ for the pendulum and $\delta = 0.2$ for the cart-pole. For the half-cheetah, we use a static initial state and set

$$J(\mathbf{x}_{1:T}, \mathbf{u}_{1:T-1}) = 200 - R(\mathbf{x}_{1:T}, \mathbf{u}_{1:T-1}), \tag{5.2}$$

where R is the reward function defined in Gym.

5.1 SURROGATE FUNCTION TUNING

The first experiment, shown in Fig. 5.1 evaluates the correlation between the surrogate performance $\hat{\mathbb{J}}$ and the true performance $\mathbb{J}$. Each point in the scatter plot corresponds to a configuration h for a MLP-iLQR-Quad pipeline on the cart-pole task. The surrogate value varies depending on the random draw of $\mathcal{D}_H$, so the surrogate estimation procedure was repeated 10 times for each configuration. The error bars in Fig. 5.1 indicate the inter-quartile range. For some configurations the inter-quartile range is 0. The correlation between $\mathbb{J}$ and $\hat{\mathbb{J}}$ is not perfect and some configurations have considerably higher variance in $\hat{\mathbb{J}}$ than others. However, the relationship between $\mathbb{J}$ and $\hat{\mathbb{J}}$ is still mostly monotonic. This suggests that $\hat{\mathbb{J}}$ is a useful metric for tuning.

5.2 SYSTEM ID TUNING

Our next experiment compares the performance of each system ID method on each sample system. The system ID data set $\mathcal{D}_I$ is split into a training set, validation set, and testing

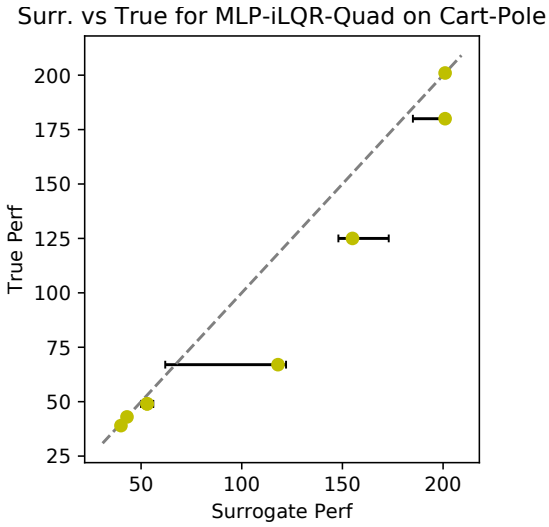


Figure 5.1: Comparing true performance $\mathbb{J}$ and the surrogate performance $\hat{\mathbb{J}}$. Error bars indicate interquartile range over 10 random draws of the dataset. Equality is indicated by the dotted line.

set containing 70%, 15%, and 15% of the trajectories respectively. Each method is tuned for 100 iterations (or until the search space is exhausted) by SMAC based on the 1 s rollout RMSE prediction error on the validation set.

Tab. 5.1 shows the 1 s error on the testing set for the tuned models. Note that the performance of the methods considered here varies significantly, both across methods for a given task and across tasks for a given method. For example, SINDy is the best performing scheme for the pendulum and cart-pole tasks, but is not for the half-cheetah task. This is because the dynamics of the pendulum and cart-pole systems can be represented as a sum of the nonlinear basis functions implemented for SINDy, while the dynamics of the half-cheetah are more complicated. This variability in performance highlights that methods are typically optimal only with respect to certain systems, motivating the need for auto-tuning to avoid the tedious process of searching among models.

5.3 OPTIMIZER TUNING

Next, we compare the performance of each optimizer on all three tasks. For each system, we test with a hand-tuned system ID model (SINDy for pendulum and cart-pole and MLP for half-cheetah). For the LQR optimizer, the dynamics are linearized around the target state. The objective and optimizer hyperparameters are tuned for 100 iterations by SMAC based on the surrogate performance. Since the outcome of the tuning process varies, we tune each setting three times, randomizing the dataset, model weights, and SMAC seed.

Table 5.1: Performance of SysID methods for three systems. Each combination is tuned and evaluated based on the RMSE at a 1 s time horizon.

Sys. ID	**Pendulum**	**Cart-pole**	**Half-cheetah**
ARX	2.08	1.87	5.68
Koopman	2.48	2.04	5.62
SINDy	0.02	0.00	5.70
GP	1.52	3.76	6.10
MLP	0.21	0.17	5.05

Tab. 5.2 reports the best tuning outcome for each task and optimizer. We also report the results of the iLQR optimizer with the Gaussian regularization objective. For the pendulum task, we observe that all methods perform comparably, though iLQR is slightly worse. For the cart-pole, we observe that all methods perform identically except for LQR which fails to complete the task. For the half-cheetah, iLQR achieves the best performance, while MPPI also performs decently. We note that as with system ID, the best optimizer varies from system to system.

5.4 END-TO-END TUNING

Next, we evaluate the end-to-end system performance of several tuning approaches, and compare with a hand-tuned baseline. We use the example of MLP-iLQR-Quad on the cart-pole, and tune using the following approaches: 1) We tune the system ID for accuracy in the same manner as in Sec. 5.2, while the objective and optimizer are fixed to the hand-tuned baseline; 2) We tune the system ID model based on the surrogate performance, keeping objective and optimizer fixed; 3) Using a system ID pre-tuned on data (i.e. the result of tuning under the first mode), we tune the optimizer and objective hyperparameters; 4) We perform full pipeline tuning of all hyperparameters simultaneously.

Fig. 5.2 compares the results against the hand-tuned baseline over 100 tuning iterations. We run each tuning method five times and plot the median scores. Except for tuning system ID for accuracy, all methods perform comparably and are able to exceed the baseline.

Table 5.2: Performance of optimizers for three systems. For each system, we fix a hand-tuned SysID, and tune the optimizer and objective function hyperparameters by an MLP surrogate. For each combination, we show the best result out of three tunes. Lower values indicate better performance.

Optimizer	Pendulum	Cart-pole	Half-cheetah
LQR	31.0	201.0	261.5
iLQR	35.0	21.0	-29.5
iLQR w/ Gauss. Reg.	31.0	21.0	134.1
Direct Transcription	30.0	21.0	221.8
MPPI	31.0	21.0	52.2

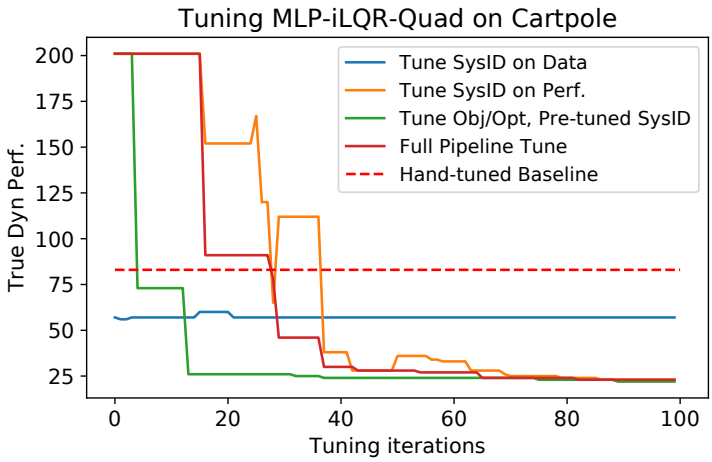
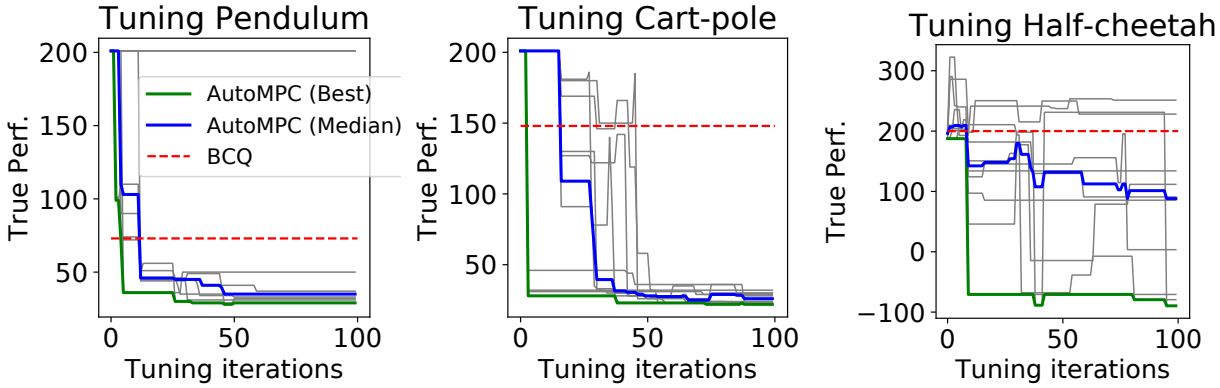


Figure 5.2: Comparing auto-tuning procedures. The y-axis plots true performance on the ground truth dynamics, median of 5 trials.



(a) (b) (c)

Figure 5.3: Tuning curves for three robotics tasks compared to the offline RL algorithm BCQ. For each system, we run ten tuning trials, plotted in grey. The median and best performances for each system are highlighted. Performance is evaluated with respect to ground truth dynamics.

5.5 COMPARISON TO OFFLINE RL

Next, we compare the true performance of full pipeline tuning to the offline RL algorithm Batch-constrained Deep Q Learning (BCQ) [24]. We train BCQ for one million iterations with the same dataset size and distribution as used in prior sections. To limit the hyperparameter space for the half-cheetah, we use a custom quadratic objective which only tunes the weights for the vertical position, horizontal velocity, and front thigh, shin, and foot angles. The custom objective also allows the target velocity to be tuned. For each system, we repeat the tuning procedure ten times, randomizing the dataset, model weights, and SMAC seed. For the half-cheetah system, we observe that the surrogate model evaluation occasionally produces highly implausible estimates, and so we modify the tuning procedure to reject performance estimates which fall outside of a predefined plausible range. Results in Fig. 5.3 demonstrate that in the median case, our method achieves superior performance to BCQ on each task. We remark that BCQ was originally presented as learning on data produced from a moderately good controller. In these experiments, we consider the much more challenging task of learning on data generated by a completely uninformed (random) controller. Even with uninformed initial data, `AutoMPC` achieves reasonable performance in almost all cases. In the more challenging half-cheetah system, the controllers produced by `AutoMPC` exhibit larger variations in performance, but at least some perform well. For such problems, we suggest to run the tuning process multiple times on the same dataset to obtain different controllers, then acquire a new dataset on the physical system using those controllers, and finally re-run tuning on the new dataset.

5.5.1 Analysis of Selected Hyperparameters

We observe a few interesting trends in the hyperparameters selected by `AutoMPC`. First, `AutoMPC` rarely selects ReLU activations for MLP system ID models. Instead, it more often selects tanh or SeLU activations. This may suggest that the discontinuities of ReLU cause problems for optimizations. Additionally, `AutoMPC` typically selects at least two MLP layers at least one of which has more than 150 neurons. Second, we observe that `AutoMPC` typically selects longer prediction horizons for the half-cheetah system, but prediction horizon appears to be less relevant for the simpler systems. Finally, for the pendulum and cart-pole systems, `AutoMPC` typically places a large objective weight on pendulum angle compared to other state dimensions.

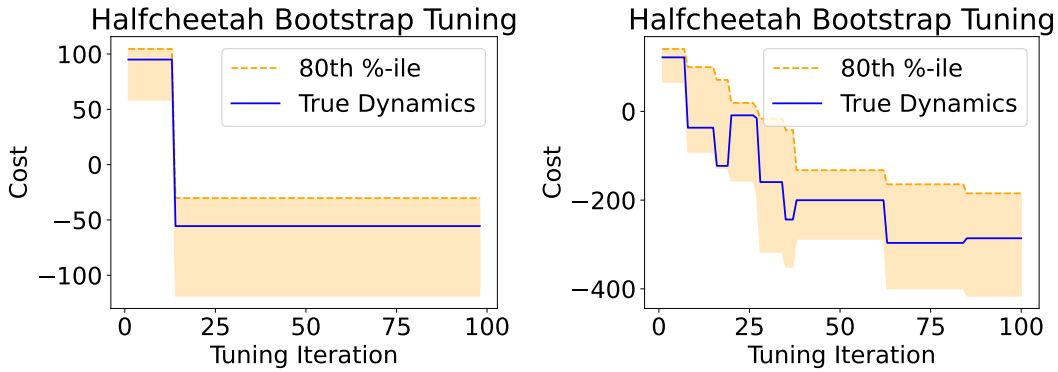
CHAPTER 6: AUTOMPC EXTENSIONS

In this section, we introduce several futher extensions to `AutoMPC`. In Sec. 6.1, we introduce an ensemble-based tuning method to increase the robustness of `AutoMPC` tuning to modelling uncertainty. In Sec. 6.2, we introduce a method for synthesizing controllers which can be applied to multiple control tasks and describe a method for tuning controllers to achieve good task generalization. In Sec. 6.3, we describe the architecture we use to scale `AutoMPC` in high-performance computing environments. Finally, in Sec. 6.4, we present results of physical experiments with `AutoMPC` on an underwater soft robot.

6.1 ENSEMBLE-BASED TUNING

In Sec 4.1, we introduced the surrogate-based controller tuning procedure. Since we do not assume access to the ground truth dynamics during the tuning procedure, we began by training a surrogate dynamics model $\hat{f}_{surr}$ on the holdout dataset $\mathcal{D}_H$. We then defined the estimated performance metric $\hat{\mathbb{J}}$, computed by performing closed-loop simulation of the candidate controller with respect to the surrogate model $\hat{f}_{surr}$. This metric $\hat{\mathbb{J}}$ was used for evaluating controllers during the tuning process. In Sec. 5.1, we empirically demonstrated that $\hat{\mathbb{J}}$ is a reasonable estimate for $\mathbb{J}$, and in Sec. 5.4, we demonstrated that tuning based on $\hat{\mathbb{J}}$ is able to produce controllers which perform well on the ground truth dynamics. However, these experiments also showed that surrogate-based tuning is not fully robust on the higher dimensional HalfCheetah benchmark. In several trials, `AutoMPC` made no progress beyond its starting configuration, while in others, high-performing configurations were later discarded in favor of low-performers. These issues can be caused by a) the difficulty of searching the combinatorially large controller design space, and b) discrepancies between the estimated controller performance $\hat{\mathbb{J}}$ and the true controller performance $\mathbb{J}$, which are in turn caused by inaccuracies in the surrogate dynamics model $\hat{f}_{surr}$. In this section, we propose a bootstrapping-based ensemble-evaluation method to address the latter.

Discrepancies between simulated and ground truth dynamics is a widely studied issue in robotics, particularly in the area of reinforcement learning, where it is commonly referred to as the *sim-to-real gap* [74]. RL policies trained under simulated dynamics frequently fail to generalize to the real world. A variety of approaches have been taken to address this issue, including domain randomization [75, 76], policy distillation [77], and generative adversarial networks to improve simulation realism [78]. In this work, we use a bootstrapping-based ensemble method to replace the point estimate $\hat{\mathbb{J}}$ with an uncertainty distribution estimate



(a) (b)

Figure 6.1: Ensemble-based tuning for HalfCheetah system. Shaded region shows 20th-80th percentile of ensemble estimates.

$\hat{\mathcal{J}}$. We then tune a controller to be robust against modeling uncertainty.

Inaccuracy of the surrogate dynamics model can be caused by either the inductive bias inherent to the model class or by model variance due insufficient coverage of the training data. Model variance can be particularly large in high-dimensional systems, since the amount of training data required for good coverage of the state space scales exponentially with the dimensionality. Analytically describing the impact of model variance on the variance of the estimate performance metric $\hat{\mathbb{J}}$ is very difficult due to the effect of the closed-loop nonlinear controller. Bootstrapping is a common method used in statistics to estimate the variance of an estimator in the absence of an analytic approach [79]. The available data is repeatedly resampled with replacement and the estimator is computed for each resample. The resulting empirical distribution has been shown to often be a good estimate for the uncertainty of the estimator.

Inspired by this approach, we propose an ensemble-based tuning method. The hold out dataset $\mathcal{D}_H$ is randomly resampled with replacement N times. Each sample is used to train a separate dynamics model, giving a surrogate ensemble $\{\hat{f}_{surr,1}, \ldots, \hat{f}_{surr,N}\}$. To evaluate a configuration, the candidate controller is simulated N times against each of the models in the surrogate ensemble, giving the empirical performance distribution $\hat{\mathcal{J}}$. Finally, $\hat{\mathcal{J}}$ is aggregated to obtain an evaluation metric for tuning. There are several options for this aggregator, including the median, worst-case, or fixed percentile. The choice of aggregator represents a tradeoff between confidence and performance in the tuned controller.

6.1.1 Ensemble-Based Tuning Results

To evaluate the capabilities of ensemble-based tuning, we perform two tuning trials

on the HalfCheetah benchmark. As in Sec. 5.4, we use datasets with 1000 trajectories of 200 time steps each, generated by application of uniform random controls. The ensembles contain 10 MLP models trained on bootstrap samples. We aggregate the empirical performance distribution of each candidate controller by taking the 80th percentile performance. For each trial we use independently random dataset generation and bootstrap sampling. In Fig. 6.1, we visualize the controller performance under the true dynamics, as well as the distribution of estimated performance (shaded region is 20th to 80th percentile of performance estimates). We observe that the true dynamics performance consistently falls within the range of estimates of the surrogate ensemble, suggesting that ensemble-based tuning is an effective means of quantifying uncertainty during the tuning process. We also observe that in both trials, `AutoMPC` is able to make significant improvements in true dynamics performance over the course of tuning, suggesting that ensemble-based tuning is effective at tuning controllers for high-dimensional systems.

6.2 MULTI-TASK TUNING

Another limitation of the tuning procedure developed in Sec 4.1 is that the synthesized controllers can only perform a single task. For example, a controller synthesized for the Cart-pole system will always control the robot to a single target state. To control the robot to a second target state, an entirely new controller would need to be synthesized and tuned from scratch. This is highly inefficient, since one would intuitively expect considerable overlap in the solution manifolds of controllers performing similar tasks. Moreover, MPCs are commonly used to control systems on a distribution of possible tasks through modification of the objective function. The multi-task problem has been studied in the context of reinforcement learning with a variety of approaches, including an augmented observation space [80] and meta-reinforcement learning [81]. In this section, we present an extension to `AutoMPC` which allows a single controller to be executed on multiple tasks by modifying suitable terms of the objective function. We also present a method for explicitly tuning `AutoMPC` controllers to achieve good task generalization performance.

To enable `AutoMPC` controllers to generalize to multiple tasks, we introduce the notion of a task transformer Θ, which maps a task and configuration to an objective function $\Theta(\tau, h) \rightarrow L_{\tau,h}$. The task transformer functionally replaces the configurable objective function described in Sec. 4.1.1. To apply the same controller to two distinct tasks τ_1 and τ_2, the transformer is simply invoked to generate task-specific objective functions $L_{\tau_1,h}$ and $L_{\tau_2,h}$. The system ID model and all optimizer hyperparameters are shared. Although `AutoMPC` allows the transformer to have arbitrary structure, we introduce the notion of task properties to make

it easier to design general transformers. For example, any task that specifies a single goal state has the `goal` property. This allows a task transformer to use knowledge of the goal state without knowing anything else about the structure of the task or performance metric. For example, the quadratic task transformer produces an objective function of the form

$$\begin{aligned} L_Q(\mathbf{x}_{t:t+H}, \mathbf{u}_{t:t+H-1}; h_L) &= (x_{t+H} - \mathbf{x}_{\text{goal}})^T F (x_{t+H} - \mathbf{x}_{\text{goal}}) \\ &+ \sum_{i=t}^{t+H} [(x_i - \mathbf{x}_{\text{goal}})^T Q (x_i - \mathbf{x}_{\text{goal}}) + u_i^T R u_i], \end{aligned} \tag{6.1}$$

where $\mathbf{x}_{\text{goal}}$ is read from the input task properties. As with the tunable quadratic cost defined in Sec. 4.2.2, the diagonal values of the Q, R, and F matrices are determined by the configuration h.

Notice that though the task transformer allows a controller to be applied to multiple tasks, it makes no guarantees of good generalization performance. In particular, when a controller is tuned for performance on only a single task, it may not generalize well to others. To address this, we introduce multi-task tuning. Rather than providing a single task as input to the tuning process, the user provides a set of tasks $\{\tau_1, \ldots, \tau_N\}$. To evaluate a candidate configuration h, the tuner simulates the controller against each of the tasks separately. Each simulation uses a task-specific objective function generated by the transformer $L_{\tau_i,h}$ and is scored by the task-specific performance metric J_{τ_i}. Finally the task scores are aggregated to produce a single controller score. This aggregation can be performed in a number of ways, including mean, worst-case, or specified percentile performance.

The extended `AutoMPC` tuning procedure including both the ensemble- and multitask-extensions is described in Alg. 6.1.

6.2.1 Multi-task Tuning Results

To evaluate multi-task tuning, we propose a multi-task variation to the Cart-pole benchmark defined in Ch. 5. We define a set of five goal states for the cart-pole, each with the pole in the upright position, and with the base at positions -6, -3, 0, 3, and 6, respectively. Similar to Ch. 5, we define the performance metric as

$$J(\mathbf{x}_{1:T}, \mathbf{u}_{1:T-1}) = \sum_{i=1}^{T} \begin{cases} 1 & \mid x_i - x_{\text{goal}} \mid_\infty > \delta \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

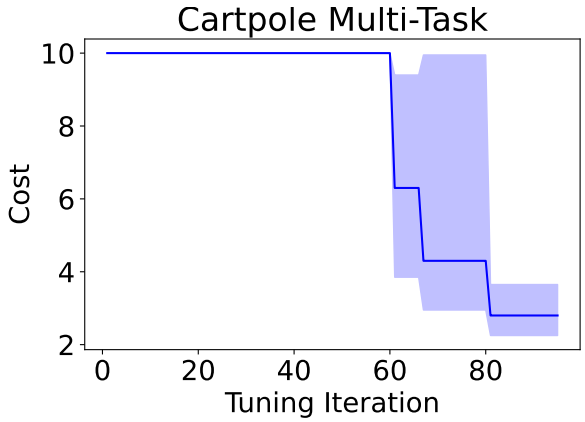


Figure 6.2: Multi-task tuning for the cartpole system. Line indicates median task performance. Shaded region indicates range of performance across tasks.

where $\delta = 0.2$ and x_{goal} varies with the task.

In Fig. 6.2, we show the results of 100 iterations of tuning an MLP-iLQR-QuadCost controller on the multi-task Cart-Pole benchmark. We use a surrogate ensemble of size 4 and tune based on the median performance across all tasks and ensemble members. For each iteration of tuning, we visualize the median and range of performance across the task-set of the best-known controller. We find that `AutoMPC` is able to make consistent progress in improving the median task performance and is eventually able to find a controller which achieves consistent performance across all tasks.

6.3 SCALING & PARALLELISM

An important practical consideration in the design of `AutoMPC` is efficient use of available computational resources. Each step of the main tuning loop (Alg 6.1, Line 10), requires both system ID model training and closed-loop controller simulation. Both of these operations can carry high cost in both time and memory. These requirements scale with the dimensionality of the system, since higher-dimensional systems require larger models and may also require a greater number of tuning iterations to adequately search the controller design space. The computation time required for model training is also impacted by the size of the input dataset. 100 iterations of tuning on the HalfCheetah can take 18-24 hours to run on a modern CPU. Moreover, the extensions introduced for ensemble-based tuning (Sec. 6.1) and multi-task tuning (Sec. 6.2)., significantly increase the computation requirements. In a setting with N tasks in the tuning set and M members in the ensemble model, a total of $N \cdot M$ independent controller simulations must be performed at each iteration of the main tuning loop. To effectively meet these requirements, we have explored the use of high-performance

Algorithm 6.1: `AutoMPC` tuning with ensemble- and multitask- extensions.

1: **Input:** Tuning tasks $\{\tau_1, \ldots, \tau_N\}$, dataset $\mathcal{D}$, surrogate config h_{surr}, ensemble size M, number of iterations n
2: Randomly partition $\mathcal{D}_I \cup \mathcal{D}_H = \mathcal{D}$, $\mathcal{D}_I \cap \mathcal{D}_H = \varnothing$
3:
4: **for** $i \leftarrow 1$ to n **do**
5: bootstrap_sample $\leftarrow$ resample($\mathcal{D}_H$)
6: $\hat{f}_{surr,i} \leftarrow$ `TRAIN`(bootstrap_sample, h_{surr})
7: **end for**
8: history = { }
9:
10: **for** $i \leftarrow 1$ to n **do**
11: $h \leftarrow$ `BAYES-OPT`(history)
12: $\hat{f}_{sysid} \leftarrow$ `TRAIN`($\mathcal{D}_I, h$)
13: optimizer $\leftarrow$ `BUILD-OPTIMIZER`(h)
14: $\hat{\mathcal{J}} \leftarrow \{\}$
15:
16: **for** $j \leftarrow 1$ to N **do**
17: $L \leftarrow$ `Transformer`(τ_j, h)
18: controller $\leftarrow$ `BUILD-CONTROLLER`($\hat{f}_{sysid}$, L, optimizer)
19:
20: **for** $k \leftarrow 1$ to M **do**
21: trajectory $\leftarrow$ `SIMULATE`(controller, $\hat{f}_{surr,k}$, τ_j)
22: score $\leftarrow$ `SCORE`(trajectory, τ_j)
23: $\hat{\mathcal{J}} \leftarrow \hat{\mathcal{J}} \cup \{\text{score}\}$
24: **end for**
25: **end for**
26: history $\leftarrow$ history $\cup \{(h, \texttt{AGGREGATE}(\hat{\mathcal{J}}))\}$
27: **end for**
28: **Return** h with minimum `AGGREGATE`($\hat{\mathcal{J}}$)

computing resources for `AutoMPC` tuning. We have extended `AutoMPC` with several additional features which allow it to efficiently exploit the highly parallel nature of these environments.

Since `AutoMPC` is a Python library, it is constrained by the global interpreter lock (GIL), a feature of the Python interpreter which only allows one thread to execute at a time. To avoid this constraint, we instead use process-based parallelism. We make use of three process classes:

- A single **master process** is responsible for overall program flow control and performs Bayesian optimization using the smac library.
- On each iteration of the main tuning loop (Alg. 6.1, Line 10), an **evaluation process**

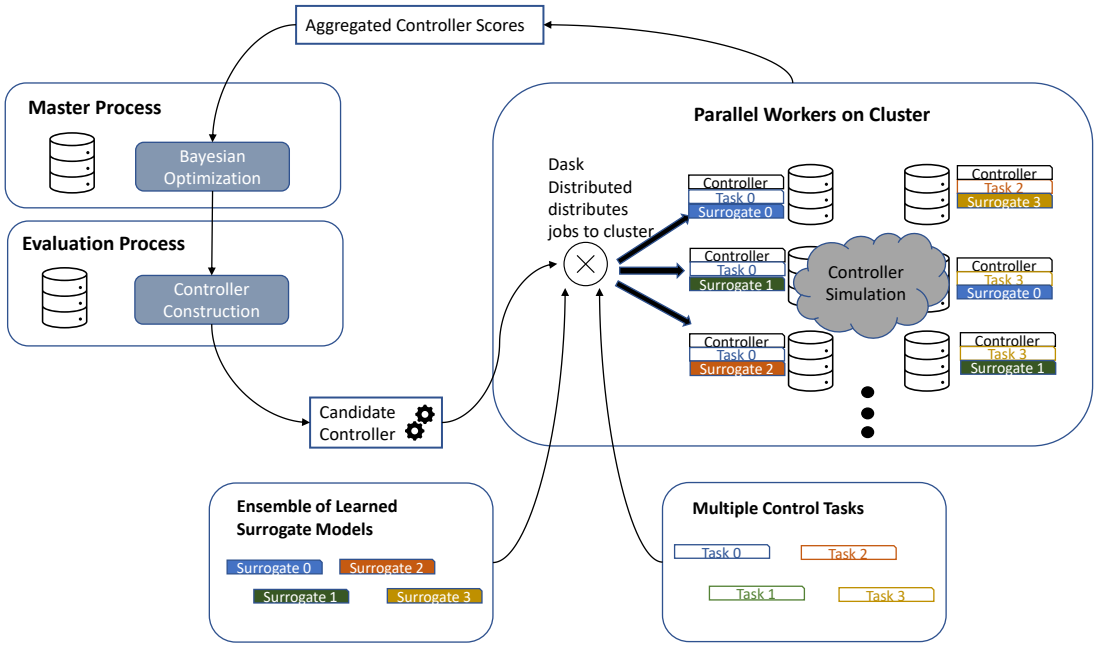


Figure 6.3: `AutoMPC` can use Dask to parallelize evaluations of multiple tasks and surrogate models across an HPC cluster.

is launched to run controller evaluation. The purpose of a separate evaluation process is two-fold. 1) `AutoMPC` makes use of a number of low-level libraries for model learning and optimization. These libraries may generate errors which cannot be handled by the Python exception system. To handle these cases, the master process monitors the evaluation process, and in the event of an unexpected crash, the configuration is recorded as having infinite cost and tuning proceeds gracefully. 2) The time required for model training and controller simulation can vary greatly between configurations and some configurations may require an intractable amount of time to evaluate. If evaluation exceeds a user-defined time limit, the master process terminates the evaluation process and again an infinite configuration cost is recorded.

- A number of **worker processes** are used to perform computations which can be run in parallel. Depending on the backend used, either a fixed number of worker processes are launched at the beginning of tuning and run for the duration, or worker processes are launched on demand.

The evaluation process is created and managed using the standard `multiprocessing` library and communicates with the master process via pipes. The worker processes may be managed by one of several backends, including Joblib[1], which provides easy parallelism

[1] `joblib.readthedocs.io`

Table 6.1: Time required for a tuning Cart-Pole with 40-member ensemble with varying numbers of workers

Number of Workers	Run Time
1	1h 51m 25s
5	0h 22m 36s
10	0h 12m 13s
40	0h 5m 58s

within a single computer, and Dask Distributed[2], which allows worker processes to be deployed across multiple nodes of a high-performance cluster. All data passed between processes are represented as Python objects encoded using the pickle library. One implementation challenge is the need to pass very large objects between processes, including datasets and trained models, which often cause performance issues for inter-process communication. To address this, we use a filesystem-based datastore for large objects. When needed, large objects are serialized to the datastore and replaced by a handle pointing to their location. The handle can be easily passed between processes and used to deserialize the large object when it is needed for computation. The datastore also reduces the memory requirements of `AutoMPC`, since large objects are kept on disk when they are not needed.

The worker processes are used in several places to parallelize computation. When ensemble-based tuning is used, the workers are used on Alg 6.1, Line 4 to train the elements of the surrogate ensemble in parallel, and Line 20, to perform controller simulations for each surrogate in parallel. When multi-task tuning is used, the workers are used on Line 16 to perform controller simulations for each task in parallel. This means that with a sufficient number of worker processes, ensemble- and multitask-tuning can be performed with no time penalties. Deployment of this architecture onto a cluster using Dask is illustrated in Fig. 6.3. In the future, we are also interested in modifying the Bayesian optimization loop to allow for multiple configurations to be evaluated in parallel, further increasing performance.

6.3.1 Scaling & Parallelism Results

To evaluate the impact of our scaling improvements, we measure the time required for an `AutoMPC` tuning trial under varying numbers of workers. In each trial, we perform 2 iterations of MLP-iLQR-QuadCost controller tuning on the Cart-pole benchmark with 200 trajectories

[2] `distributed.dask.org`

in the dataset and a 40-member surrogate ensemble. We repeated this experiment with 1, 5, 10, and 40 worker processes. All trials were run on the Delta Supercomputer at the National Center for Supercomputing Applications using the Dask Distributed backend. The total time required for each tuning trial is reported in Table 6.1. We observe that the required time consistently decreases with more workers, and that for 1, 5, and 10 workers, the speed-up is approximately linear. For 40 workers, the speed-up is slightly sublinear, due to fixed overhead.

6.4 PHYSICAL EXPERIMENTS ON UNDERWATER SOFT ROBOT

We also demonstrate the performance of `AutoMPC` using physical experiments on an underwater soft robot. The robot (shown in Fig. 6.4) consists of two modules, each with two parallel actuators. The water pressure in each actuator can be varied independently to alter the shape of the soft robot. Feedback is provided by pressure sensors in each actuator as well as ten computer vision markers placed along the spine of the robot. We define the state space of the robot as $\mathbf{x} = (\mathbf{x}_{\text{marker}}, \mathbf{x}_{\text{pressure}})$ consisting of eleven 2D Cartesian marker locations and four analog pressure readings, and its control $\mathbf{u}$ consisting of 8 binary solenoid inputs, controlling the intake and outtake valves for each actuator. For more details on the construction and design of the robot, see [82].

The task space for the underwater soft robot is divided up into 2x2 cm goal regions. Each

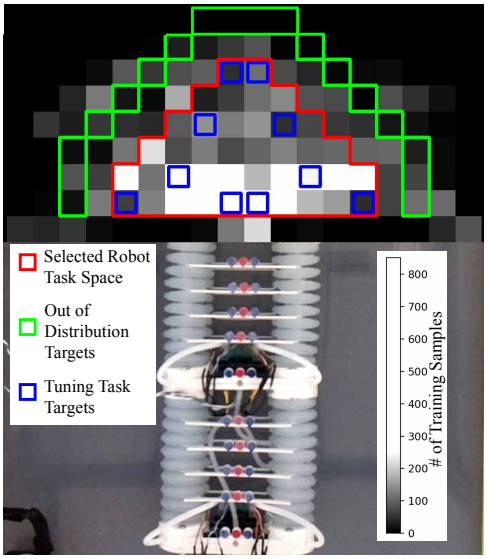


Figure 6.4: Training data coverage map visualized by a 2D histogram corresponding to number of data samples collected where the robot's end-effector is inside a given 2x2 cm square region. The ten blue targets are used for tuning, and the remainder are used for testing. Peripheral targets are used to test out-of-distribution generalization.

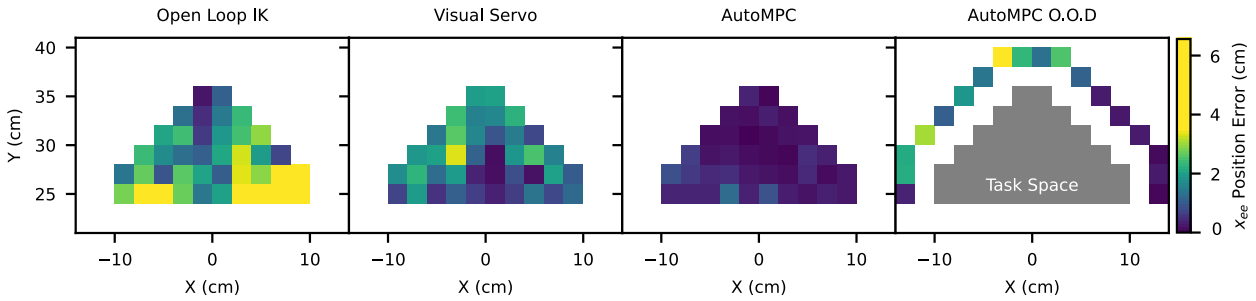


Figure 6.5: 2D Histograms of accuracy data for `AutoMPC` and baseline controllers on the chosen task space.

task τ is associated with a goal-region $\mathcal{G}_\tau$. The performance metric is defined to be the number of seconds for which the robot end-effector is not in the goal region, that is

$$J_\tau(\mathbf{x}_{1:T}, \mathbf{u}_{1:T-1}) = \Delta_t \sum_{j=1}^{T} \left(1 - 1_{\mathcal{G}_\tau}\left([\mathbf{x}_{ee}]_j\right)\right), \tag{6.3}$$

where Δ_t gives the controller period, and $1_{\mathcal{G}_\tau}$ gives the goal region indicator function. A subset of 10 tasks (shown in Fig. 6.4) were selected for tuning and a total of 11,358 time steps of data were collected from the robot.

Fig. 6.5 compares end-effector placement accuracy of the `AutoMPC` controller to two baseline controllers. The Open Loop IK baseline uses a learned IK model to predict the pressure levels needed to obtain the target position and a simple pressure controller drives the actuators to the intended pressure without any feedback from the comptuer vision markers. The Visual Servo baseline uses the pressure level feedback and a PI controller to adjust the target position and correct for steady-state error in the open-loop controller. The `AutoMPC` controller significantly outperforms both of these baseline controllers across most tasks. We also evaluate the `AutoMPC` controller on 18 tasks outside of the tuned task distribution. We find that `AutoMPC` generalizes generally well, though with slightly higher error than the in-distribution tasks.

CHAPTER 7: CONCLUSION

In this work, we studied data-driven techniques for the design of model predictive controllers (MPC) for novel robots with unknown dynamics. We began by reviewing the relevant literature (Chapter 2) and concluded that there was a need to study methods for automatic and offline tuning of MPC. Next, we presented a case study (Chapter 3) on the application of data-driven MPC to performing needle insertion in the deep anterior lamellar keratoplasty (DALK) ophthalmic surgical procedure. We proposed data-driven methods for offline MPC tuning, and tested our synthesized MPC in physical experiments, demonstrating superior performance to a baseline which had previously been shown to be comparable to human surgeons. Next, we presented the design of `AutoMPC` (Chapter 4), an open-source Python package for automatic, offline MPC synthesis. We conducted simulated experiments with `AutoMPC` (Chapter 5), evaluating its performance across several standard control benchmarks and demonstrating superior performance to a state-of-the-art offline reinforcement learning algorithm. Finally, we presented several extensions to `AutoMPC` (Chapter 6), including an ensemble-based method for improving tuning robustness, a method for synthesizing and tuning multiple controllers, scaling and performance improvements for deployment in high-performance computing environments, and evaluation of `AutoMPC` in physical experiments on an underwater soft robot.

We believe that there are many interesting future directions in the space of data-driven MPC design, including 1) the use of meta-learning techniques to improve the speed of controller tuning, 2) the use of active learning techniques to automatically identify informative strategies for robot data collection, 3) automatic tuning for adaptive controllers, 4) automatic tuning for stochastic environments, and finally 5) integration of `AutoMPC` with reinforcement learning and imitation learning methods.

REFERENCES

[1] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *International Conference on Intelligent Robots and Systems*, 2012, pp. 4906–4913.

[2] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *Neural Information Processing Systems (NeurIPS)*, 2018, pp. 4754–4765.

[3] I. Lenz, R. A. Knepper, and A. Saxena, "Deepmpc: Learning deep latent features for model predictive control." in *Robotics: Science and Systems*. Rome, Italy, 2015.

[4] G. Lee, S. S. Srinivasa, and M. T. Mason, "Gp-ilqg: Data-driven robust optimal control for uncertain nonlinear dynamical systems," *arXiv preprint arXiv:1705.05344*, 2017.

[5] I. Abraham, G. De La Torre, and T. D. Murphey, "Model-based control using Koopman operators," in *Robotics: Science and Systems (RSS)*, 2017.

[6] L. Ljung, "Perspectives on system identification," *Annual Reviews in Control*, vol. 34, no. 1, pp. 1–12, 2010.

[7] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Neural Information Processing Systems (NeurIPS)*, 2015, pp. 2962–2970.

[8] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 1946–1956.

[9] W. Edwards, Tang, K. A. Izatt, Joseph, and K. Hauser, "Model learning for automatic needle insertion in deep anterior lamellar keratoplasty," 2020, presented at IROS Cognitive Robotic Surgery Workshop.

[10] W. Edwards, G. Tang, G. Mamakoukas, T. Murphey, and K. Hauser, "Automatic tuning for data-driven model predictive control," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 7379–7385.

[11] W. Edwards, G. Tang, Y. Tian, M. Draelos, J. Izatt, A. Kuo, and K. Hauser, "Data-driven modelling and control for robot needle insertion in deep anterior lamellar keratoplasty," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1526–1533, 2022.

[12] W. Edwards, G. Tang, Y. Tian, M. Draelos, J. Izatt, A. Kuo, and K. Hauser, "Data-driven modelling and control for robot needle insertion in deep anterior lamellar keratoplasty," 2022, presented at IEEE Conference on Robotics and Automation.

[13] W. D. Null, W. Edwards, D. Jeong, T. Tchalakov, J. Menezes, K. Hauser, and Y. Z, "Automatically-tuned model predictive control for an underwater soft robot," 2023, under review for IEEE Conference on Robotics and Automation.

[14] D. H. Shim, H. J. Kim, and S. Sastry, "Decentralized nonlinear model predictive control of multiple flying robots," in *Conference on Decision and Control (CDC)*, vol. 4, 2003, pp. 3621–3626.

[15] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.

[16] A. Bemporad and M. Morari, "Robust model predictive control: A survey," in *Robustness in identification and control*. Springer, 1999, pp. 207–226.

[17] M. Fazel, R. Ge, S. M. Kakade, and M. Mesbahi, "Global convergence of policy gradient methods for the linear quadratic regulator," in *International Conference on Machine Learning (ICML)*, vol. 80, 2018.

[18] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic mpc for model-based reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1714–1721.

[19] S. Kamthe and M. Deisenroth, "Data-efficient reinforcement learning with probabilistic model predictive control," in *International Conference on Artificial Intelligence and Statistics*, 2018, pp. 1701–1710.

[20] G. Mamakoukas, M. L. Castano, X. Tan, and T. D. Murphey, "Derivative-based koopman operators for real-time control of robotic systems," *arXiv preprint arXiv:2010.05778*, 2020.

[21] Y. Wu, G. Tucker, and O. Nachum, "Behavior regularized offline reinforcement learning," *arXiv preprint arXiv:1911.11361*, 2019.

[22] S. Ross and J. A. Bagnell, "Agnostic system identification for model-based reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2012, p. 1905–1912.

[23] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *International Conference on Machine Learning (ICML)*, 2011, pp. 465–472.

[24] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *International Conference on Machine Learning (ICML)*, 2019, pp. 2052–2062.

[25] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine, "Stabilizing off-policy q-learning via bootstrapping error reduction," in *Neural Information Processing Systems (NeurIPS)*, 2019, pp. 11 784–11 794.

[26] R. Kidambi, A. Rajeswaran, P. Netrapalli, and T. Joachims, "Morel: Model-based offline reinforcement learning," *arXiv preprint arXiv:2005.05951*, 2020.

[27] S. Shimmura and K. Tsubota, "Deep anterior lamellar keratoplasty," *Current opinion in ophthalmology*, vol. 17, no. 4, pp. 349–355, 2006.

[28] J. Sugita and J. Kondo, "Deep lamellar keratoplasty with complete removal of pathological stroma for vision improvement," *British Journal of Ophthalmology*, vol. 81, no. 3, pp. 184–188, 1997.

[29] M. Anwar and K. D. Teichmann, "Big-bubble technique to bare descemet's membrane in anterior lamellar keratoplasty," *Journal of Cataract & Refractive Surgery*, vol. 28, no. 3, pp. 398–403, 2002.

[30] V. M. Borderie, O. Sandali, J. Bullet, T. Gaujoux, O. Touzeau, and L. Laroche, "Long-term results of deep anterior lamellar versus penetrating keratoplasty," *Ophthalmology*, vol. 119, no. 2, pp. 249–255, 2012.

[31] M. Busin, V. Scorcia, P. Leon, and Y. Nahum, "Outcomes of air injection within 2 mm inside a deep trephination for deep anterior lamellar keratoplasty in eyes with keratoconus," *American journal of ophthalmology*, vol. 164, pp. 6–13, 2016.

[32] Y.-S. Yoo, W.-J. Whang, M.-J. Kang, J.-H. Hwang, Y.-S. Byun, G. Yoon, S. Shin, W. Jung, S. Moon, and C.-K. Joo, "Effect of air injection depth on big-bubble formation in lamellar keratoplasty: an ex vivo study," *Scientific reports*, vol. 9, no. 1, pp. 1–8, 2019.

[33] H. D. McKee, L. C. Irion, F. M. Carley, V. Jhanji, and A. K. Brahma, "Residual corneal stroma in big-bubble deep anterior lamellar keratoplasty: a histological study in eye-bank corneas," *British journal of ophthalmology*, vol. 95, no. 10, pp. 1463–1465, 2011.

[34] N. D. Pasricha, C. Shieh, O. M. Carrasco-Zevallos, B. Keller, D. Cunefare, J. S. Mehta, S. Farsiu, J. A. Izatt, C. A. Toth, and A. N. Kuo, "Needle depth and big bubble success in deep anterior lamellar keratoplasty: An ex vivo microscope-integrated oct study," *Cornea*, vol. 35, no. 11, p. 1471, 2016.

[35] S. Guo, N. R. Sarfaraz, W. G. Gensheimer, A. Krieger, and J. U. Kang, "Demonstration of optical coherence tomography guided big bubble technique for deep anterior lamellar keratoplasty (dalk)," *Sensors*, vol. 20, no. 2, p. 428, 2020.

[36] I. Park, H. G. Shin, K. Kim, H. K. Kim, and W. Kyun, "Robotic needle insertion using corneal applanation for deep anterior lamellar keratoplasty," *The Journal of Korea Robotics Society*, vol. 16, no. 1, pp. 64–71, 2021.

[37] A.-M. Jablonka, N. A. Maierhofer, H. Roodaki, A. Eslami, D. Zapp, A. Nasseri, and C. P. Lohmann, "The effect of needle entry angle on bubble formation in dalk using the big-bubble technique," *Investigative Ophthalmology & Visual Science*, vol. 61, no. 7, pp. 1862–1862, 2020.

[38] M. Draelos, B. Keller, G. Tang, A. Kuo, K. Hauser, and J. Izatt, "Real-time image-guided cooperative robotic assist device for deep anterior lamellar keratoplasty," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–9.

[39] M. Draelos, G. Tang, B. Keller, A. Kuo, K. Hauser, and J. A. Izatt, "Optical coherence tomography guided robotic needle insertion for deep anterior lamellar keratoplasty," *IEEE Transactions on Biomedical Engineering*, 2019.

[40] J. Troccaz, G. Dagnino, and G.-Z. Yang, "Frontiers of medical robotics: from concept to systems to clinical translation," *Annual review of biomedical engineering*, vol. 21, pp. 193–218, 2019.

[41] J.-Y. Lee, T. Mattar, T. J. Parisi, B. T. Carlsen, A. T. Bishop, and A. Y. Shin, "Learning curve of robotic-assisted microvascular anastomosis in the rat," *Journal of reconstructive microsurgery*, vol. 28, no. 07, pp. 451–456, 2012.

[42] S. J. Parekattil and A. Gudeloglu, "Robotic assisted andrological surgery," *Asian journal of andrology*, vol. 15, no. 1, p. 67, 2013.

[43] K. Grace, P. Jensen, J. Colgate, and M. Glucksberg, "Teleoperation for ophthalmic surgery: From the eye robot to feature extracting force feedback," *Automedica*, vol. 16, no. 4, pp. 293–310, 1998.

[44] X. He, J. Handa, P. Gehlbach, R. Taylor, and I. Iordachita, "A submillimetric 3-dof force sensing instrument with integrated fiber bragg grating for retinal microsurgery," *IEEE Transactions on Biomedical Engineering*, vol. 61, no. 2, pp. 522–534, 2013.

[45] M. D. de Smet, T. C. Meenink, T. Janssens, V. Vanheukelom, G. J. Naus, M. J. Beelen, C. Meers, B. Jonckx, and J.-M. Stassen, "Robotic assisted cannulation of occluded retinal veins," *PloS one*, vol. 11, no. 9, p. e0162037, 2016.

[46] H. Yu, J.-H. Shen, R. J. Shah, N. Simaan, and K. M. Joos, "Evaluation of microsurgical tasks with oct-guided and/or robot-assisted ophthalmic forceps," *Biomedical optics express*, vol. 6, no. 2, pp. 457–472, 2015.

[47] S. L. Charreyron, Q. Boehler, A. N. Danun, A. Mesot, M. Becker, and B. J. Nelson, "A magnetically navigated microcannula for subretinal injections," *IEEE transactions on biomedical engineering*, vol. 68, no. 1, pp. 119–129, 2020.

[48] F. Ullrich, J. Lussi, V. Chatzopoulos, S. Michels, A. J. Petruska, and B. J. Nelson, "A robotic diathermy system for automated capsulotomy," *Journal of Medical Robotics Research*, vol. 3, no. 01, p. 1850001, 2018.

[49] W. Sun, S. Patil, and R. Alterovitz, "High-frequency replanning under uncertainty using parallel sampling-based motion planning," *IEEE Transactions on Robotics*, vol. 31, no. 1, pp. 104–116, 2015.

[50] D. Glozman and M. Shoham, "Image-guided robotic flexible needle steering," *IEEE Transactions on Robotics*, vol. 23, no. 3, pp. 459–467, 2007.

[51] C. E. Reiley, E. Plaku, and G. D. Hager, "Motion generation of robotic surgical tasks: Learning from expert demonstrations," in *2010 Annual international conference of the IEEE engineering in medicine and biology*. IEEE, 2010, pp. 967–970.

[52] A. Murali, S. Sen, B. Kehoe, A. Garg, S. McFarland, S. Patil, W. D. Boyd, S. Lim, P. Abbeel, and K. Goldberg, "Learning by observation for surgical subtasks: Multilateral cutting of 3d viscoelastic and 2d orthotropic tissue phantoms," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 1202–1209.

[53] B. Keller, M. Draelos, K. Zhou, R. Qian, A. N. Kuo, G. Konidaris, K. Hauser, and J. A. Izatt, "Optical coherence tomography-guided robotic ophthalmic microsurgery via reinforcement learning from demonstration," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1207–1218, 2020.

[54] A. Takacs, S. Jordan, R.-E. Precup, L. Kovacs, J. Tar, I. Rudas, and T. Haidegger, "Review of tool-tissue interaction models for robotic surgery applications," in *2014 IEEE 12th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*. IEEE, 2014, pp. 339–344.

[55] N. Chentanez, R. Alterovitz, D. Ritchie, L. Cho, K. K. Hauser, K. Goldberg, J. R. Shewchuk, and J. F. O'Brien, "Interactive simulation of surgical needle insertion and steering," in *ACM SIGGRAPH 2009 papers*, 2009, pp. 1–10.

[56] G. Mamakoukas, M. L. Castano, X. Tan, and T. D. Murphey, "Derivative-based koopman operators for real-time control of robotic systems," *IEEE Transactions on Robotics*, 2021.

[57] B. Keller, M. Draelos, G. Tang, S. Farsiu, A. N. Kuo, K. Hauser, and J. A. Izatt, "Real-time corneal segmentation and 3d needle tracking in intrasurgical oct," *Biomedical optics express*, vol. 9, no. 6, pp. 2716–2732, 2018.

[58] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Sensor fusion IV: control paradigms and data structures*, vol. 1611. International Society for Optics and Photonics, 1992, pp. 586–606.

[59] V. Scorcia, M. Busin, A. Lucisano, J. Beltz, A. Carta, and G. Scorcia, "Anterior segment optical coherence tomography–guided big-bubble technique," *Ophthalmology*, vol. 120, no. 3, pp. 471–476, 2013.

[60] H. Akaike, "Autoregressive model fitting for control," in *Selected Papers of Hirotugu Akaike*. Springer, 1998, pp. 153–170.

[61] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020. [Online]. Available: https://doi.org/10.1007/s12532-020-00179-2

[62] P. K. Bhullar, O. M. Carrasco-Zevallos, A. Dandridge, N. D. Pasricha, B. Keller, L. Shen, J. A. Izatt, C. A. Toth, and A. N. Kuo, "Intraocular pressure and big bubble diameter in deep anterior lamellar keratoplasty: An ex-vivo microscope-integrated oct with heads-up-display study," *Asia-Pacific Journal of Ophthalmology (Philadelphia, Pa.)*, vol. 6, no. 5, p. 412, 2017.

[63] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International Conference on Learning and Intelligent Optimization*, 2011, pp. 507–523.

[64] M. Budišić, R. Mohr, and I. Mezić, "Applied Koopmanism," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 22, no. 4, p. 047510, 2012.

[65] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *National Academy of Sciences*, vol. 113, no. 15, pp. 3932–3937, 2016.

[66] B. de Silva, K. Champion, M. Quade, J.-C. Loiseau, J. Kutz, and S. Brunton, "Pysindy: A python package for the sparse identification of nonlinear dynamical systems from data," *Journal of Open Source Software*, vol. 5, no. 49, p. 2104, 2020. [Online]. Available: https://doi.org/10.21105/joss.02104

[67] J. Hensman, A. Matthews, and Z. Ghahramani, "Scalable variational Gaussian process classification," in *International Conference on Artificial Intelligence and Statistics*, vol. 38, 2015, pp. 351–360.

[68] J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson, "Gpytorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration," in *Neural Information Processing Systems (NeurIPS)*, 2018, pp. 7576–7586.

[69] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," in *NIPS*, 2017.

[70] H. Kwakernaak and R. Sivan, *Linear Optimal Control Systems*. Wiley-interscience New York, 1972, vol. 1.

[71] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[72] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.

[73] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," *Arxiv preprint arXiv:1606.01540*, 2016.

[74] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2020, pp. 737–744.

[75] B. Balaji, S. Mallya, S. Genc, S. Gupta, L. Dirac, V. Khare, G. Roy, T. Sun, Y. Tao, B. Townsend et al., "Deepracer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning," *arXiv preprint arXiv:1911.01562*, 2019.

[76] J. Vacaro, G. Marques, B. Oliveira, G. Paz, T. Paula, W. Staehler, and D. Murphy, "Sim-to-real in reinforcement learning for everyone," in *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)*, 2019, pp. 305–310.

[77] R. Traoré, H. Caselles-Dupré, T. Lesort, T. Sun, N. Díaz-Rodríguez, and D. Filliat, "Continual reinforcement learning deployed in real-life using policy distillation and sim2real transfer," *arXiv preprint arXiv:1906.04452*, 2019.

[78] O.-M. Pedersen, "Sim-to-real transfer of robotic gripper pose estimation-using deep reinforcement learning, generative adversarial networks, and visual servoing," M.S. thesis, NTNU, 2019.

[79] C. Z. Mooney, C. F. Mooney, C. L. Mooney, R. D. Duval, and R. Duvall, *Bootstrapping: A nonparametric approach to statistical inference*. sage, 1993, no. 95.

[80] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Conference on robot learning*. PMLR, 2020, pp. 1094–1100.

[81] J. Rothfuss, D. Lee, I. Clavera, T. Asfour, and P. Abbeel, "Promp: Proximal meta-policy search," in *International Conference on Learning Representations*, 2018.

[82] W. D. Null and Y. Z, "Development of a modular and submersible soft robotic arm and corresponding learned kinematics models," *Submitted*, 2022.