Meta Learning With Paired Forward and Inverse Models for Efficient Receding Horizon Control

Christopher D. McKinnon and Angela P. Schoellig

***Abstract*—This paper presents a model-learning method for Stochastic Model Predictive Control (SMPC) that is both accurate and computationally efficient. We assume that the control input affects the robot dynamics through an unknown (but invertable) nonlinear function. By learning this unknown function and its inverse, we can use the value of the function as a new control input (which we call the input feature) that is optimised by SMPC in place of the original control input. This removes the need to evaluate a function approximator for the unknown function during optimisation in SMPC (where it would be evaluated many times), reducing the computational cost. The learned inverse is evaluated only once at each sampling time to convert the optimal input feature from SMPC to a control input to apply to the system. We assume that the remaining unknown dynamics can be accurately represented as a model that is linear in a set of coefficients, which enables fast adaptation to new conditions. We demonstrate our approach in experiments on a large ground robot using a stereo camera for localisation.**

***Index Terms*—Field robots, model learning for control, robust/adaptive control.**

I. Introduction

ROBUST and Stochastic Model Predictive Control (SMPC) are effective tools for controlling mobile robots such as quadrotors [1], race cars [2], [3], and offroad vehicles [4], [5] in the presence of model uncertainty. Both methods rely on a forward dynamics model that predicts how a control input changes the state, including an estimate of uncertainty in this change. Although this model can be learned from data to improve its accuracy, a key challenge is the trade-off between the accuracy of the model and its computational efficiency since the model is evaluated many times for each control input applied to the system. In this paper, we present a method for leveraging expressive models for robot dynamics while incurring only a marginal increase in computational cost during trajectory optimisation in SMPC. Our approach is based on using a combination of forward and inverse models (trained offline) for part of the dynamics and Bayesian linear regression to adapt to changes in dynamics online. The proposed architecture is shown in Fig. 1.

The most popular approach to improve SMPC is to learn the forward model for the robot dynamics directly. To increase the accuracy of this model, researchers leverage various function approximators, such as Gaussian process regression [6]–[8], local linear regression [9]–[11], and neural networks [12], [13] to learn an unknown function in the forward dynamics model. However, since SMPC requires these models to be evaluated many times to compute each control input (since each control input is based on multi-step predictions of the state), there is a trade-off between the computational cost of these function approximators and their expressiveness. In our approach, we partition the model such that the computationally expensive component of the learned model is only evaluated once for each control input, mitigating this problem. Additionaly, we adapt a simple model for robot dynamics online to adapt to changes in operating conditions online and use this model to optimise the control input in SMPC. Partitioning the model into an expressive component learned offline and a simple component learned online to adapt to changes is inspired by a recent trend known as meta learning.

In meta learning, algorithms leverage large amounts of data available ahead of time to learn functions that can be adapted to new scenarios with a small amount of data and computation. An expressive model, such as a neural network, generates features [14], [15] or an initial guess [16] that can be adapted to changes using linear regression [14], [15] or a small number of gradient descent steps [16]. Functions learned using these methods adapt to changes efficiently because linear regression or a small number of gradient steps require a relatively small amount of data and computing power. These algorithms have been applied to reinforcement learning on real robotic platforms, used to learn forward models, and these forward models have been combined with robust control [15] and SMPC [17]. The main difference in our approach is that in addition to partitioning the model, we learn an inverse for part of the model so we do not have to use the full forward model during optimisation in SMPC.

While a forward model predicts the motion of a robot given the current state and control input, an inverse model predicts the control input that will produce the desired motion given the current state. Inverse models have long been used for controlling platforms such as robotic manipulators [18]. These models can be derived from physics [18], learned [19]–[22], or some combination [23]–[25]. The advantage is that they simplify complex,

Manuscript received October 15, 2020; accepted February 23, 2021. Date of publication March 4, 2021; date of current version March 22, 2021. This letter was recommended for publication by Associate Editor S. Weiss and Editor C. Gosselin upon evaluation of the reviewers' comments. This work was supported by the Natural Sciences and Engineering Research Council of Canada and the Canada CIFAR AI Chairs Program. *(Corresponding author: Christopher D. McKinnon.)*

The authors are with the Dynamic Systems Lab, The University of Toronto Institute for Aerospace Studies and the Vector Institute for Artificial Intelligence, Toronto M3H5T6, Canada (e-mail: chris.mckinnon@mail.utoronto.ca; schoellig@utias.utoronto.ca).

Digital Object Identifier 10.1109/LRA.2021.3063957

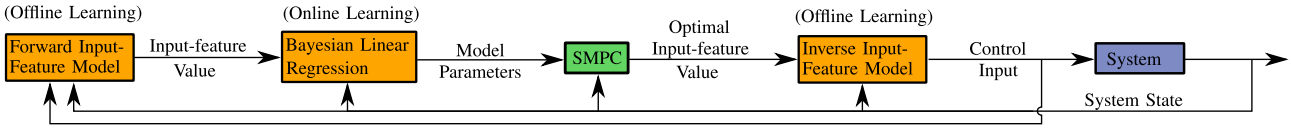


Fig. 1. A block diagram of the proposed approach. The forward and inverse input-feature models are learned offline and Bayesian Linear Regression is used to adapt to changes online and estimate model uncertainty. The input feature is the value of an unknown function in the dynamics for which we have learned an approximate forward input-feature model. The value of this unknown function is optimised in SMPC and then converted into a control input that can be applied to the system using the learned inverse input-feature model. The main benefits of this approach are that *(i)* the expressive but computationally expensive forward input-feature map does not have to be evaluated during optimisation in SMPC, and *(ii)*, multiple forward and inverse input-feature maps from different sources can be considered simultaneously. A detailed version of this block diagram that includes online model selection is shown in Fig. 2.

nonlinear dynamics into a simple, linear system. However, they typically do not combine this model with an MPC framework, which requires a forward model, and they adapt the inverse term (a complex, nonlinear function) directly to adapt to changing conditions. In contrast, by learning both a forward and inverse model offline and combining this with online linear regression, we simplify the online learning process to linear regression–which keeps the potentially challenging step of learning the inverse as an offline process where it can be validated before deployment–and leverage an SMPC framework.

Finally, as an alternative to learning one model that adapts efficiently to new scenarios, several models can be trained ahead of time. The one that performs the best in the robot's current operating conditions is selected at runtime [26]. This simplifies the training process since the training for each model is de-coupled and can be combined with online learning to adapt further to specific operating conditions [27]–[29]. While multiple paired forward and inverse models have been used for control before [30], our approach is, to the best of our knowledge, the first that combines this capability with probabilistic, online model learning and SMPC.

Notation: We denote matrices with boldface uppercase letters, column vectors with boldface lowercase letters, and scalars with non-boldface letters. The mean of a Gaussian random variable v is $\bar{v}$. The time derivative of variable v is $\dot{v}$. Functions are followed by round brackets; e.g. $\mathbf{v}(\cdot)$, if the output is a vector, and $v(\cdot)$ if the output is a scalar. A probability density function (pdf) of v given parameters $*$ will be denoted by $p(v \mid *)$. $v \sim \mathcal{N}(\mu, \sigma^2)$ means that samples of v are distributed according to a Gaussian distribution with mean μ and variance σ^2.

II. Problem Statement

The goal of this work is to learn a model for the robot dynamics where the robot is performing a path following task in changing conditions and the control approach is SMPC. The key requirements for the model are: *(i)* high accuracy predictions of the robot motion given the current state and control input, *(ii)* realistic bounds on modelling error to maintain safety, and *(iii)* computational efficiency to allow for long prediction horizons in SMPC. We assume that a geometric path and input and state constraints (e.g. maximum speed and lateral error) are given.

We consider the robot dynamics to be of the form where part of the state $\mathbf{s}$ evolves according to known dynamics $\mathbf{h}(\mathbf{s}, \xi)$ and part of the state ξ evolves according unknown dynamics. We assume the unknown dynamics are a linear combination of a known set of features $\boldsymbol{\phi}(\mathbf{s}, \xi)$ and an unknown but smooth and invertible function $f(\mathbf{s}, \xi, u)$, the output of which we call the input feature, that depends on the states and the control input u:

$$\dot{\mathbf{s}} = \mathbf{h}(\mathbf{s}, \xi) \tag{1}$$

$$\dot{\xi} = [f(u, \mathbf{s}, \xi),\ \boldsymbol{\phi}^T(\mathbf{s}, \xi)]\,\mathbf{w} + \eta, \tag{2}$$

where $\mathbf{w}$ is a vector of unknown weights, and $\eta \sim \mathcal{N}(0, \sigma^2)$ where σ^2 is unknown and slowly changing. The reason for including $\boldsymbol{\phi}(\cdot)$ in addition to $f(\cdot)$ is to allow parts of the dynamics that depend on each element in $\boldsymbol{\phi}(\cdot)$ and $f(\cdot)$ to vary independently through the associated weight in $\mathbf{w}$ (facilitating online adaptation). That is, $\boldsymbol{\phi}(\cdot)$ represents prior knowledge about ways that the dynamics can change in new operating conditions and $f(\cdot)$ allows us to leverage prior data to learn unknown dynamics in a particular set of operating conditions. In the case where we have no prior knowledge, $\boldsymbol{\phi}(\cdot)$ can be omitted. There may be multiple states with unknown dynamics (i.e. ξ may be a vector), however we assume that each one depends on a separate input u. In this case, there would be one independent instance of (2) for each input. To further motivate this problem set-up, we present a simple example.

Simple Example: Consider a cart on a rail with forward velocity v, mass m, viscous drag with drag coefficient b, input u, which is related to a force through an unknown function $f'(u)$, and a disturbance with units of force which takes the form of zero-mean Gaussian noise $\eta_0 \sim \mathcal{N}(0, \sigma_0^2)$:

$$m\dot{v} = f'(u) - bv + \eta_0. \tag{3}$$

Let $f(u, v) = f'(u) - bv$. Then the dynamics of a cart with any mass and drag coefficient can be expressed as:

$$\dot{v} = [f(u, v),\ v]\,\mathbf{w} + \eta. \tag{4}$$

This is of the form (2) with $\xi = v$. Here, including v in addition to $f(\cdot)$ enables changes in both m and b to be captured by $\mathbf{w}$ rather than just changes in m. Learning the entire right-hand side of (3) as $f(\cdot)$ offline means that we do not have to know the initial value of b.

This can be extended to a unicycle using (1) as follows. Let ω be the turn rate and assume that it is constant in this example. Let x and y be the position of the robot and θ be its heading. Then:

$$(1)\begin{cases} \dot{x} = v\cos(\theta), \\ \dot{y} = v\sin(\theta), \\ \dot{\theta} = \omega, \end{cases} \tag{5}$$

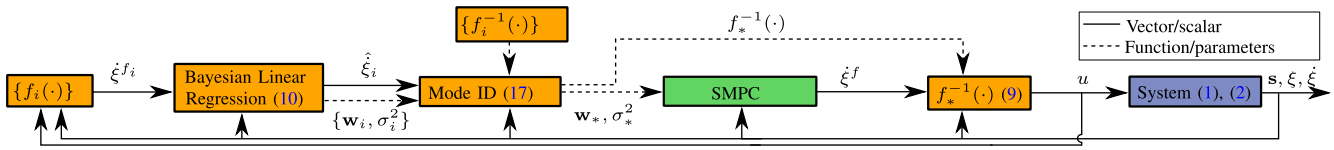


Fig. 2. Block diagram of the proposed approach with multiple models. Curly braces indicate a discrete set of functions or values. Dashed lines indicate functions or parameters, and solid lines indicate scalar or vector quantities. The orange blocks are relevant for our model learning approach. The subscript i on $f_i(\cdot)$ and $f_i^{-1}(\cdot)$ indicates the mode. Equations relevant to each block are referenced in round braces.

$$(2)\ \left\{ \dot{v} = [f(u, v),\ v]\,\mathbf{w} + \eta. \right. \tag{6}$$

which is of the form (1) and (2) with $\mathbf{s} = [x, y, \theta]^T$ and $\xi = v$. For this example, changes in m and b only affect $\mathbf{w}$ and σ^2. As a result, it is sufficient to learn $\mathbf{w}$ and σ^2 online.

Additional examples of systems with dynamics that can be factored such that model parameters appear linearly include serial manipulators, pendulums, cartpoles [31], and quadrotors with unknown mass and inertia properties [32] or which have nested control loops that have dynamics close to first order systems [33]. The nonlinearity $f'(\cdot)$ could come from, for example, a DC motor with a nonlinear saturation curve [34] (see Section 8, Fig 4).

III. Methodology

Our method partitions model learning into two phases: an offline phase, where a forward and inverse model for the input feature are learned; and an online phase, where a simple model uses the input feature to adapt to changes in the robot dynamics and estimate model uncertainty. From a model-learning perspective, partitioning of the model in this way enables us to match the complexity of each part of the model to the availability of data and computational power before and during deployment. We use expressive forward and inverse input-feature models that have large *data capacity* and are trained *offline*, and a model that is linear in coefficients that is *data efficient* and trained *online*. From a computational perspective for SMPC at runtime, trajectory optimisation (which requires many queries to the full model for robot dynamics) relies on the simple model, which is computationally inexpensive, and only optimises the *value* of the input feature. The optimal value of the input feature is then converted back into a control input using the inverse input-feature model, which is more computationally expensive, but this is done only once for each control input applied to the robot. Fig. 2 shows a detailed block diagram of the proposed approach.

A. Approach

To present our approach, we focus on the unknown dynamics and assume $\xi \in \mathcal{R}$ for clarity of presentation. Our method can be applied to systems with $\boldsymbol{\xi} \in \mathcal{R}^{n_\xi}$ so long as each element depends on a separate input which allows them to be treated independently using our approach.

1) Forward Input-Feature Model (Trained Offline): We assume that we are given a dataset $\mathcal{D} = \{u_i, \mathbf{s}_i, \xi_i, \dot{\xi}_i\}_{i=1}^{n_\mathcal{D}}$ of samples of the states, input, and time derivative of ξ at $n_\mathcal{D}$ different sampling times. Suppose that we have a system where the unknown dynamics are:

$$\dot{\xi} = [f'(u, \mathbf{s}, \xi) + \boldsymbol{\phi}^T(\mathbf{s}, \xi)]\mathbf{w}_0 + \eta_0. \tag{7}$$

where $\eta_0 \sim \mathcal{N}(0, \sigma_0^2)$, $f'(u, \mathbf{s}, \xi)$ is an unknown function, and $\mathbf{w}_0$ is unknown but fixed. We can use $\mathcal{D}$ to fit a function $f(u, \mathbf{s}, \xi) \approx [f'(u, \mathbf{s}, \xi) + \boldsymbol{\phi}^T(\xi, \mathbf{s})]\mathbf{w}_0$, which we assume to be invertible. Learning the mean of the right hand side of (7) means that we don't have to know the value of $\mathbf{w}_0$ and the dynamics can be expressed as:

$$\dot{\xi} = [f(u, \mathbf{s}, \xi),\ \boldsymbol{\phi}^T(\mathbf{s}, \xi)]\mathbf{w} + \eta \tag{8}$$

for any changes to the weight and the variance of the additive, Gaussian noise.

2) Inverse Input-Feature Model (Trained Offline): Let $\dot{\xi}^f \equiv f(u, \mathbf{s}, \xi)$. At a given time instant, ξ and $\mathbf{s}$ are fixed so the only parameter that we can change to influence $\dot{\xi}^f$ is u. Therefore, we can define:

$$u = f^{-1}(\mathbf{s}, \xi, \dot{\xi}^f). \tag{9}$$

which generates u that results in a desired $\dot{\xi}^f$ for a given $\mathbf{s}$ and ξ. To fit this function, we can use $f(\cdot)$ from the previous section to generate $\dot{\xi}^f$ for each sample in $\mathcal{D}$ yielding a new dataset $\mathcal{D}^f = \{u_i, \mathbf{s}_i, \xi_i, \dot{\xi}_i^f\}_{i=1}^{n_\mathcal{D}}$. This dataset can be used to fit $f^{-1}(\cdot)$ using standard regression techniques. Since $\mathcal{D}^f$ is generated using a deterministic function, this dataset is noise-free which facilitates learning $f^{-1}(\cdot)$. In cases where the inverse is not unique, approaches such as the distal-teacher approach can be employed to learn a particular inverse [35].

3) Simple Forward Model (Trained Online): Given $f(\cdot)$ from the previous section, a tuple $(u_i, \mathbf{s}_i, \xi_i, \dot{\xi}_i)$, and $\boldsymbol{\phi}(\cdot)$, we can generate a corresponding tuple $(\dot{\xi}_i^f, \boldsymbol{\phi}_i, \dot{\xi}_i)$, where $\dot{\xi}_i^f = f(u_i, \mathbf{s}_i, \xi_i)$ and $\boldsymbol{\phi}_i = \boldsymbol{\phi}(\mathbf{s}_i, \xi_i)$. Now, subbing into (2), we get:

$$\dot{\xi}_i = [\dot{\xi}_i^f,\ \boldsymbol{\phi}_i^T]\mathbf{w} + \eta_i \tag{10}$$

which is linear in coefficients $\mathbf{w}$ with additive, Gaussian noise $\eta_i \sim \mathcal{N}(0, \sigma^2)$. Functions of this form can be learned efficiently using Bayesian linear regression, which estimates the distribution of $\mathbf{w}$ and σ^2. See [11], section III b). These estimates can be updated recursively, which we leverage for adapting our estimate for $\mathbf{w}$ and σ^2 online using the live stream of data generated by the robot. Now, in combination with $\mathbf{h}(\cdot)$ from (1), we have all the components necessary to do trajectory optimisation in SMPC.

4) Stochastic MPC: We build on the SMPC formulation presented in [11]. Here, we give a brief overview. We consider a receding horizon control problem with H timesteps where

the robot states $\mathbf{s}$ and ξ evolve according to dynamics of the form (1) and (2) given input u. Using the method proposed in this paper, we use the mean value of $\dot{\xi}^f$ as a decision variable instead of u. Let $\ell(\mathbf{s}, \xi, \dot{\xi}, \ddot{\xi})$ be the non-negative scalar cost of the state and its derivatives with respect to time at a particular time instant and $\ell_f(\mathbf{s}, \xi)$ be the non-negative scalar cost of the final states in the horizon. Let $\mathbf{s}$ and ξ be constrained to be within the sets $\mathcal{S}$ and Ξ with a small acceptable probability ϵ of violating these constraints. The resulting receding horizon control problem solved at each sampling time k is then the following:

$$\min_{\dot{\xi}^f_{k:k+H-1}} \quad \ell_f(\bar{\mathbf{s}}_H, \bar{\xi}_H) + \sum_i \ell(\bar{\mathbf{s}}_{k+i}, \bar{\xi}_{k+i}, \dot{\bar{\xi}}_{k+i}, \ddot{\bar{\xi}}_{k+i}) \tag{11}$$

$$\text{s.t.} \quad \dot{\mathbf{s}}_{k+i} = \mathbf{h}(\mathbf{s}_{k+i}, \xi_{k+i}) \tag{12}$$

$$\dot{\xi}_{k+i} = [\dot{\xi}^f_{k+i},\ \boldsymbol{\phi}(\mathbf{s}_{k+i}, \xi_{k+i})^T]\mathbf{w}_k + \eta_{k+i} \tag{13}$$

$$\eta_{k+i} \sim \mathcal{N}(0, \sigma^{\eta}_{k+i}) \tag{14}$$

$$p(\mathbf{s}_{k+i+1} \in \mathcal{S}) \geq 1 - \epsilon, \tag{15}$$

$$p(\xi_{k+i+1} \in \Xi) \geq 1 - \epsilon, \tag{16}$$

for $i = 0 \ldots H-1$, where $\bar{(\cdot)}$ indicates the mean. Noisy measurements of $\mathbf{s}$ and ξ at time k are assumed to be given and have additive, Gaussian uncertainty with known covariances. We integrate $\mathbf{s}$ and ξ using Eulers method and propagate uncertainty by linearising the dynamics model about the mean states and input. See [2] for a discussion of methods for uncertainty propagation for SMPC. For clarity of presentation, we have omitted path-reference values for $\mathbf{s}$ and ξ that appear in $\ell(\cdot)$ and $\ell_f(\cdot)$ and variables related to contouring control (see [36]), which is how we compute the reference for each timestep along the horizon.

In our approach, we include a penalty on $\dot{\xi}$ and $\ddot{\xi}$, which, for the example in Section II, is the forward acceleration and jerk of the unicycle. Penalising $\dot{\xi}$ and $\ddot{\xi}$ rather than the input $\dot{\xi}^f$ and its time derivative means that the cost function encourages the same motions regardless of the input feature. This is useful if we consider a set of input-feature model pairs, which we show next, in Section III-B.

B. Multi-Modal Learning

When a robot may be deployed in a wide range of environments, there may be changes in the parameters of the linear model as well as the nonlinear term $f(\cdot)$. Changes in $f(\cdot)$ cannot be captured using Bayesian linear regression. A key advantage of our approach is that multiple models, each using a different input-feature model pair, can be simultaneously adapted to the robots current environment given a stream of $\mathbf{s}, \xi, \dot{\xi}$, and u. This allows us to train a set of input-feature model pairs ahead of time and choose the best pair for the current operating conditions.

Let c be a discrete index where each value corresponds to a different forward and inverse input-feature model pair and its associated linear model. At sampling time k, the posterior probability of each model is calculated using a sliding window of n_w recent measurements $\mathcal{D}^- = \{\xi_i, \dot{\xi}_i, \mathbf{s}_i\}_{i=k-1-n_w}^{k-1}$ and a prior $p(c)$:

$$p(c = j|\mathcal{D}^-) \propto med(p(\dot{\xi}_i|u_i, \mathbf{s}_i, \xi_i, c = j))p(c = j) \tag{17}$$

for $i = k-1-n_w \ldots k-1$, where $p(\dot{\xi}_i|u_i, \mathbf{s}_i, \xi_i, c = j)$ is the probability of model j generating $\dot{\xi}_i$ given u_i, ξ_i, and $\mathbf{s}_i$. The model is switched if $p(c = j|\mathcal{D}^-)$ for the most likely model is more likely than the previously most likely model by at least a pre-defined threshold. This prevents rapid switching between two or more models that have similar probability. Using the median $med(\cdot)$ instead of the product is a departure from the more common assumption that each recent measurement is independent but we found it produced more reliable results in experiment. We estimate the probability of each model using the model for the unknown dynamics only (i.e. (2) for a given input-feature model pair, set of weights, and noise variance) assuming that all models share a common $\mathbf{h}(\cdot)$. If this is not the case, the full dynamics model would be used. Model selection is done asynchronously from the control loop. When the best model changes, a new linear model and inverse input-feature model are passed to the controller and used to compute the control at the next sampling time.

Fig. 3. The Clearpath Grizzly with mast-mounted stereo camera used for the experiments in this paper.

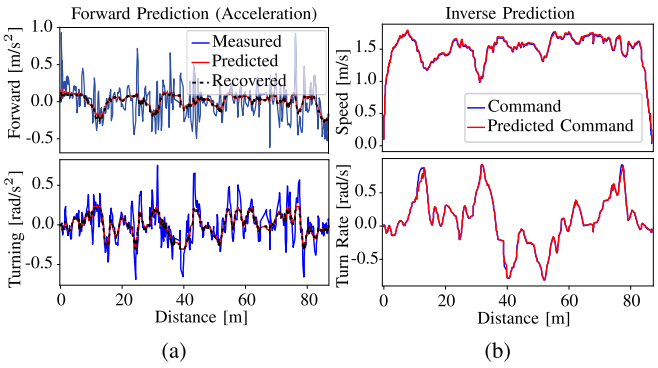
IV. Application to a Ground Robot

In this section, we describe how to apply our method to the Clearpath Grizzly shown in Fig. 3. The controller receives estimates of its pose and velocity from a localisation system that relies solely on a stereo camera (c.f. [37]).

A. Dynamics Model

The full dynamics model we use for the Clearpath Grizzly is the unicycle where we learn the speed and turn rate dynamics. Let x and y be the position of the vehicle, θ its heading, ω its turn rate, and v its forward speed. The model for robot dynamics is then:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \tag{18}$$



(a) (b)

Fig. 4. Measured values for the acceleration and the control commands applied to the vehicle are shown in blue. Single step predictions from forward 4(a) and inverse 4(b) input-feature models are shown in red. In this special case since, the training and test environments were the same, the forward input-feature is analogous to acceleration. The dashed black line shows the value of the input feature recovered using the control commands predicted by the inverse input-feature model (e.g., $f_v(v, \omega, f_v^{-1}(v, w, \dot{\xi}^{f,v}))$). Since the error induced by the approximate, learned inverse input-feature model was an order of magnitude smaller than the error in the forward model, we assumed that the inverse input-feature models were exact for our experiments.

$$\begin{bmatrix} \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} [\dot{\xi}^{f,v},\ v]\mathbf{w}^v + \eta^v \\ [\dot{\xi}^{f,\omega},\ \omega]\mathbf{w}^\omega + \eta^\omega \end{bmatrix} \tag{19}$$

where $\dot{\xi}^{f,v}$ and $\dot{\xi}^{f,\omega}$ are the input features for speed and turn-rate, respectively, $\eta^v \sim \mathcal{N}(0, \sigma_v^2)$, and $\eta^\omega \sim \mathcal{N}(0, \sigma_\omega^2)$. Equation (18) is analogous to (1) with $\mathbf{s} = [x, y, \theta]^T$ and $\boldsymbol{\xi} = [v, \omega]^T$, and each row of (19) is analogous to (2).

The learned forward input-feature models have the form:

$$\dot{\xi}^{f,v} = f_v(v, \omega, v_{cmd}), \tag{20}$$

$$\dot{\xi}^{f,w} = f_\omega(v, \omega, \omega_{cmd}), \tag{21}$$

where v_{cmd} and ω_{cmd} are the original speed and turn rate commands that are sent directly to the vehicle. The inverse models have the form:

$$v_{cmd} = f_v^{-1}(v, \omega, \dot{\xi}^{f,v}), \tag{22}$$

$$\omega_{cmd} = f_\omega^{-1}(v, \omega, \dot{\xi}^{f,w}). \tag{23}$$

Each model $f_*(\cdot)$ and $f_*^{-1}(\cdot)$ is parametrised as a neural network. The network architecture we used in all cases was a fully connected network with three hidden layers, 20 hidden units in each layer, and ReLU activation functions. We trained all networks using 25, 042 samples gathered in the UTIAS Mars Dome (sand and gravel terrain) with 20% of the samples used for validation and an L1 loss function.

B. Input-Feature Model Accuracy

In the special case when the dynamics model is tested in the same environment as it was trained, the input feature should directly predict the associated motion in the robot (e.g., $\dot{v} \approx \dot{\xi}^{f,v}$). In Fig. 4(a), we show that the forward input-feature model predicts a smoothed version of the acceleration of the vehicle. For eight runs of the vehicle driving in the UTIAS Mars Dome, the (50th, 75th, and 95th) percentiles of error between the input-feature (red in Fig. 4(a)) and the measured acceleration (blue in Fig. 4(a)) are $(0.074, 0.14, 0.29)$ m/s^2 for forward acceleration and $(0.064, 0.12, 0.23)$ rad/s^2 for turning acceleration. We can now use the input-feature value and the state at each sampling time to predict the control input using the inverse input-feature model. In Fig. 4(b), we show that it accurately recovers the control input with error percentiles $(0.014, 0.017, 0.045)$ m/s for v_{cmd} and $(0.003, 0.005, 0.011)$ rad/s for ω_{cmd}. Furthermore, we can feed these control inputs back into the forward input-feature model to see how much the input-feature changes when using the recovered control input (black in Fig. 4(a)) compared to the original control input. The percentiles of the change in the input-feature using the recovered vs. actual control inputs are (0.007, 0.017, 0.036) m/s^2 for forward acceleration and $(0.003, 0.006, 0.011)$ rad/s^2 for turning acceleration. Since these errors are approximately an order of magnitude smaller than the difference between the input-feature and the acceleration, we neglect error in the inverse input-feature models for our experiments.

The difference in accuracy between the forward and inverse input-feature models may result, in part, from the fact that the forward input-feature model is trained to model a physical process with an unknown dependence (the robot dynamics in the training environment). In contrast, the inverse input-feature model is trained to model a deterministic function with a known dependence (the inverse of the forward input-feature model). For example, in our case, the robot dynamics may be affected by local terrain properties, which may change from one run to the next as the vehicle displaces sand and gravel by driving over it. Since the forward input-feature model does not include local terrain properties as input, it cannot model the impact of these factors on the robot dynamics resulting in prediction errors. In addition, the output of the forward input-feature model is measured using sensors, which may introduce measurement noise. These problems do not exist for learning the inverse input-feature model since it is trained to invert the forward input-feature *model*, which is deterministic, has a known dependence, and is assumed to be invertible.

C. Cost Function

As mentioned in Section III-A4, the cost function should depend on the state and its derivatives and not the value of the input feature so that the optimal trajectory is consistent when the input-feature changes. Let $\mathbf{s}_k = [x_k, y_k, \theta_k]^T$, $\boldsymbol{\xi}_k = [v_k, \omega_k]^T$, $\ell_{lc}(\cdot)$ be a quadratic penalty on the position and heading error, $\ell_{v\omega}(\cdot)$ be a quadratic penalty for the speed and turn rate error, and $\mathbf{R}_{\dot{\xi}}$ and $\mathbf{R}_{\ddot{\xi}}$ be diagonal and positive semi-definite penalty matrices for acceleration and jerk. The cost function we used in SMPC is then:

$$\sum_{i=k+1}^{k+H} \ell_{lc}(\bar{\mathbf{s}}_i) + \sum_{i=k+1}^{k+H} \ell_{v\omega}(\bar{\boldsymbol{\xi}}_i) + \tag{24}$$

$$\sum_{i=k}^{k+H-1} \frac{1}{2}\dot{\bar{\boldsymbol{\xi}}}_i^T \mathbf{R}_{\dot{\xi}} \dot{\bar{\boldsymbol{\xi}}}_i + \sum_{i=k}^{k+H-1} \frac{1}{2}\ddot{\bar{\boldsymbol{\xi}}}_i^T \mathbf{R}_{\ddot{\xi}} \ddot{\bar{\boldsymbol{\xi}}}_i \tag{25}$$

TABLE I
SUMMARY OF DIFFERENCES BETWEEN THE METHODS COMPARED IN THIS SECTION. LEARNED INPUT FEATURE IS WHEN A LEARNED INPUT-FEATURE MODEL PAIR IS USED (SECTION III-A1 AND III-A2). ONLINE ADAPTATION IS WHEN THE LINEAR MODEL PARAMETERS ARE UPDATED ONLINE (SECTION III-A3). ONLINE MODEL SELECTION IS WHEN MULTIPLE INPUT-FEATURE MODEL PAIRS ARE CONSIDERED AT RUNTIME (SECTION III-B)

Name	Learned Input Feature	Online Adaptation	Online Model Selection
BLR	x	✓	x
DNN	✓	x	x
BLR-DNN	✓	✓	x
MM	✓	✓	✓

We have omitted the terms of the cost function for contouring control (see [36] for details on contouring control) and the reference values for $\mathbf{s}$ and $\boldsymbol{\xi}$ for brevity. Values for $\dot{\bar{\boldsymbol{\xi}}}$ and $\ddot{\bar{\boldsymbol{\xi}}}$ are calculated by differentiating $\bar{\boldsymbol{\xi}}$ using finite difference.

D. Computational Advantage

A key advantage of our approach is that a simple model is used in SMPC and the forward and inverse input-feature models are evaluated only once per sampling time. This is important because the mean, uncertainty, and Jacobian of the model used in SMPC are queried 30 timesteps in the prediction horizon $\times$ 3 re-linearizations $\times$ 10 Hz $=$ 900 times per second whereas only the scalar output of the forward and inverse input-feature models are evaluated at 10 Hz $=$ 10 times per second. This means that the forward and inverse blocks can be much more computationally expensive (and therefore potentially more expressive) than any model used in SMPC. In our case, the linear model has 14 independent parameters and the forward and inverse models have a combined 3764 parameters which reflects the required computational requirements of each component. In addition, the linear model can provide a quantitative estimate of model uncertainty for SMPC. This allowed us to implement the forward and inverse model learning in Python on the CPU adding only 4.1 ms to 6.5 ms (50th to 95th percentile) to the run-time of the controller (small compared to the 100 ms sampling period of the controller).

V. EXPERIMENTS

We compare four variants of the proposed method which are summarised in Table I. These variants all use the unicycle model (18) but differ in whether or not a model is learned for the input features, whether the linear model parameters are updated online, and the number of input-feature model pairs considered at runtime. Initial values for the linear model parameters are calculated using the dataset used to train the input-feature model pairs.

First, to assess controller performance without using an input-feature model pair, we compare to using the original control inputs v_{cmd} and ω_{cmd} as the input-features directly and learning only the linear model parameters (e.g., $\mathbf{w}^v$ and σ_v^2) online (BLR). This is the special case when the forward and inverse input-feature models are identity and only depend on the original control input. Second, we compare to using a learned input-feature model pair but not adapting the linear model parameters online (DNN). Third, we compare to using *both* a learned input-feature model pair *and* adapting the linear model parameters online (BLR-DNN). Finally, we evaluate our online model selection approach by including both BLR and BLR-DNN as candidates and switching to the best model online using (17) with a uniform prior (MM).

A. Closed Loop Experiments

In this section, we demonstrate the effectiveness of the proposed approach in closed loop. We use the same network as in the previous section and conduct our tests over an 82 m course in a paved parking area. The high friction of the paved surface introduced a larger difference between BLR and BLR-DNN than driving offroad on dirt or over snowy terrain.

1) Tracking Performance Comparison: Fig. 5 shows the distribution of lateral error, step cost, and the cumulative cost when controlling the vehicle using BLR, BLR-DNN, MM, and DNN. All cases when the controller uses a model that incorporates a learned feature show a reduction in these metrics compared to BLR indicating that the learned feature captures more of the robot dynamics than using the control input directly. Furthermore, BLR-DNN out-performs DNN in all metrics. This indicates that learning the feature weights online translates into better closed-loop performance. One interesting point is that MM achieved lower lateral error and step cost than BLR-DNN but achieved similar cumulative cost. This is likely due to our model switching implementation, which assumes that the acceleration and jerk are zero when the model switches which causes the vehicle to slow down. At this lower speed, the vehicle is able to track the path more closely which reduces the lateral error and step cost, but it takes longer to complete the path which increases the cumulative cost.

2) Closed Loop Model Performance Comparison: In this section, we compare the prediction performance of BLR and BLR-DNN for the closed-loop experiment to gain insight into why their performance differed. The results are based on data from runs when the controller was using the corresponding method. We focus on the robots speed dynamics, but the results for turn rate show a similar trend.

Model prediction accuracy will be measured using Multi-step RMSE (M-RMSE) and the accuracy of the predicted uncertainty will be measured using the multi-step RMS Z-score (M-RMSZ) over the prediction horizon. For the speed v, the M-RMSE at sampling time k is calculated by comparing the predicted mean at each timestep i along the horizon $\bar{v}_{k,i}$ to the measured mean at the corresponding sample time $\bar{v}_{k+i,0}$:

$$\text{M-RMSE}_k = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(\bar{v}_{k,i} - \bar{v}_{k+i,0})^2}. \tag{26}$$

M-RMSZ is calculated the same way but each term in the sum is normalized by the predicted variance $\sigma^2_{v\,k,i}$.

Fig. 6 shows that the M-RMSE and M-RMSZ for speed was substantially lower for BLR-DNN than for BLR. This discrepancy highlights the importance of using a good feature for online learning in closed loop: large model errors induce tracking errors, which can result in larger control actions, which can compound the model errors further as more complex dynamics

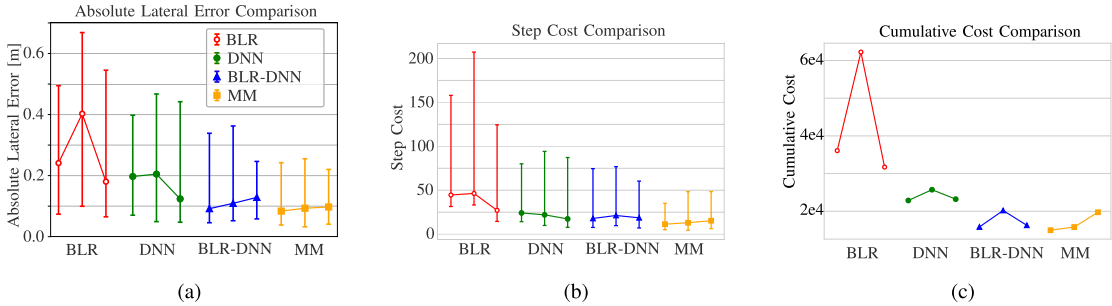


Fig. 5. This figure shows performance metrics for a closed-loop experiment demonstrating the effect of each component of the proposed algorithm. The robot traversed an 82 m path over a paved parking area. Each point along the horizontal axis represents a traversal of the path. All traversals where the controller leveraged a model using the learned input-feature out-performed **BLR**. Online adaptation (**BLR-DNN**) increased performance in all metrics compared to using a fixed model (**DNN**). The mixture model (**MM**) outperforms all models except for BLR-DNN, where it achieves a similar cumulative cost, for reasons explained in Section V-A.

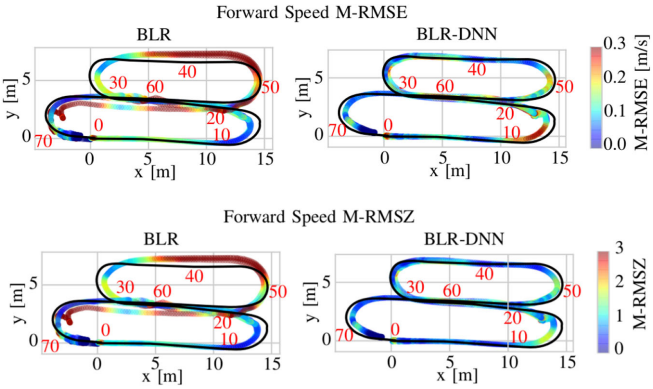


Fig. 6. This figure shows the model prediction accuracy using the parameters estimated online when **BLR** and **BLR-DNN** are used in closed loop in a paved parking lot. Results for each method over three runs (the same runs shown in Fig. 5) are super-imposed. This shows that, in the parking lot environment, **BLR-DNN** clearly out-performs **BLR** in model prediction accuracy, uncertainty estimation, in addition to closed loop performance as shown in Fig. 5.

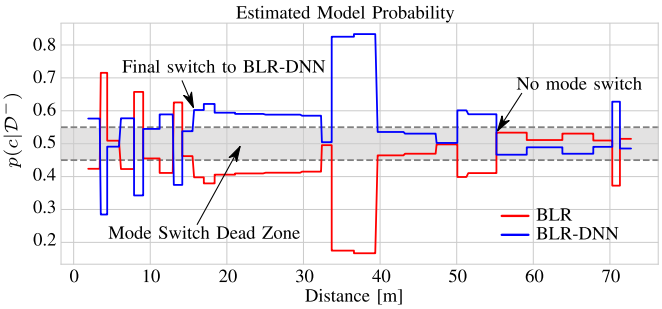


Fig. 7. This figure shows the estimated probability of each model in **MM** during one run of the closed loop experiments shown in Fig. 5. The probability of each model is estimated every 2 s which is why the estimated probability has the appearance of a square wave with varying width. We avoid chatter by requiring that the model only switches when the most likely model is at least 10% more likely than the currently active model. For two models, this leads to the 'dead zone,' shaded in gray, where no mode switches will occur even if the most likely model changes.

are excited and any time delays have a larger impact. While the controller using BLR does complete the course, the large M-RMSZ indicates that the uncertainty bounds are overconfident for several sections of the run. This means the vehicle may not always be able to complete the run without violating the path tracking error bounds if the it is close to the path tracking constraints at the same time that the model is overconfident.

3) Online Model Selection: Fig. 7 shows the estimated probability of each model calculated using (17) during one run from the results shown in Fig. 5. BLR-DNN is estimated as the most likely model for most of the run. This is consistent with our the results in Fig. 5 where BLR-DNN had better tracking performance, and Fig. 6, where it had better prediction accuracy.

To prevent rapid switching between models when two models have similar probability, we require that the most likely model be at least 10% more likely than the one used in the controller before switching. This results in the dead-band shown in Fig. 7. When the estimated probability of both models is similar, each model explains the recent dynamics similarly well so using either should result in similar performance.

VI. Discussion

An ongoing challenge in learning control is choosing the dependence of the unknown dynamics (i.e., the dependence of the input-feature model pair). Initially, it may be tempting to include many past states and rely on the function approximator to ignore irrelevant ones given enough data. We found that including past states as inputs to the input-feature model pair model induced large oscillations in closed loop even when output of SMPC $\dot{\xi}^{f,*}$ was smooth. In contrast, the simpler model used in the experiments above did not have this problem. A detailed study of feature selection for learning inverse models for closed loop control is beyond the scope of this paper.

While we assumed that the error induced by the approximate inverse input-feature model was negligible (justified empirically), further investigations into the impact of this error on the closed-loop system are planned for future work. In addition, investigating how to incorporate constraints on the original control input would enable this approach to be applied to systems that operate close to their input constraints.

VII. Conclusion

In summary, we presented a control approach specifically tailored for SMPC that leverages the combination of *(i)* an expressive input-feature model pair learned offline with *(ii)* Bayesian

linear regression to adapt to changes online and *(iii)* online model selection to leverage multiple possible input-feature model pairs. Our formulation allows learned input features to be used as a drop-in replacement for the control input in SMPC making it easy to apply to existing controllers. Through a series of experiments on a ground robot, we showed that the proposed approach significantly improved closed-loop performance over using the control input directly at minimal computational cost. We hope that the reader finds this an interesting method for control of systems with partially unknown dynamics that may be deployed in a wide range of operating conditions.

REFERENCES

[1] P. Bouffard, A. Aswani, and C. Tomlin, "Learning-based model predictive control on a quadrotor: Onboard implementation and experimental results," in *Proc. Int. Conf. Robot. Automat.*, 2012, pp. 279–284.

[2] L. Hewing, A. Liniger, and M. N. Zeilinger, "Cautious NMPC with gaussian process dynamics for miniature race cars," in *Proc. Eur. Control Conf.*, 2018, pp. 1341–1348.

[3] J. Kabzan, L. Hewing, A. Liniger, and M. Zeilinger, "Learning-based model predictive control for autonomous racing," *Robot. Autom. Lett.*, vol. 4, no. 4, pp. 3363–3370, 2019.

[4] C. Ostafew, A. P. Schoellig, and T. Barfoot, "Robust constrained learning-based NMPC enabling reliable mobile robot path tracking," *Intl. J. Robot. Res.*, vol. 35, no. 13, pp. 1547–1563, 2016.

[5] C. D. McKinnon and A. P. Schoellig, "Learn fast, forget slow: Safe predictive learning control for systems with unknown and changing dynamics performing repetitive tasks," *Robot. Autom. Lett.*, vol. 4, no. 2, pp. 2180–2187, 2019.

[6] L. Hewing, A. Liniger, and M. Zeilinger, "Cautious NMPC with gaussian process dynamics for autonomous miniature race cars," in *Proc. Eur. Control Conf.*, 2018, pp. 1341–1348.

[7] F. Meier and S. Schaal, "Drifting gaussian processes with varying neighborhood sizes for online model learning," in *Proc. Int. Conf. Robot. Automat.*, 2016, pp. 264–269.

[8] J. Kabzan, L. Hewing, A. Liniger, and M. Zeilinger, "Learning-based model predictive control for autonomous racing," *Robot. Autom. Lett.*, vol. 4, no. 4, pp. 3363–3370, 2019.

[9] L. Jamone, B. Damas, and J. Santos-Victor, "Incremental learning of context-dependent dynamic internal models for robot control," in *Proc. Int. Symp. Intell. Control*, 2014, pp. 1336–1341.

[10] J. Ting, A. D'Souza, S. Vijayakumar, and S. Schaal, "A Bayesian approach to empirical local linearization for robotics," in *Proc. Int. Conf. Robot. Automat.*, 2008, pp. 2860–2865.

[11] C. D. McKinnon and A. P. Schoellig, "Learning probabilistic models for safe predictive control in unknown environments," in *Proc. Eur. Conf. Control*, 2019, pp. 2472–2479, in press.

[12] G. Williams *et al.*, "Information theoretic MPC for model-based reinforcement learning," in *Proc. Int. Robot. Automat.*, 2017, pp. 1714–1721.

[13] N. Mohajerin and S. Waslander, "Multi-step prediction of dynamic systems with recurrent neural networks," *Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 11, pp. 3370–3383, 2019.

[14] J. Harrison, A. Sharma, and M. Pavone, "Meta-learning priors for efficient online bayesian regression," in *Proc. Intl. Workshop Algo. Found. Robot.*, 2018, pp. 318–337.

[15] M. O'Connell, G. Shi, X. Shi, and S.-J. Chung, "Meta-learning-based robust adaptive flight control under uncertain wind conditions," 2020, *arXiv:2103.01932*.

[16] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. Int. Conf. Mach. Learn.*, vol. 70. 2017, pp. 1126–1135.

[17] T. Lew, A. Sharma, J. Harrison, and M. Pavone, "Safe learning and control using meta-learning," *Openreview*, 2019.

[18] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Hoboken, NJ, USA: Wiley, 2020.

[19] S. Zhou, M. Helwa, and A. Schoellig, "Design of deep neural networks as add-on blocks for improving impromptu trajectory tracking," in *Proc. Conf. Decis. Control*, 2017, pp. 5201–5207.

[20] F. Meier, D. Kappler, N. Ratliff, and S. Schaal, "Towards robust online inverse dynamics learning," in *Proc. Int. Conf. Intell. Robots Syst.*, 2016, pp. 4034–4039.

[21] R. Calandra, S. Ivaldi, M. Deisenroth, E. Rueckert, and J. Peters, "Learning inverse dynamics models with contacts," in *Proc. Int. Conf. Robot. Automat.*, 2015, pp. 3186–3191.

[22] A. S. Polydoros, L. Nalpantidis, and V. Krüger, "Real-time deep learning of robotic manipulator inverse dynamics," in *Proc. Int. Conf. Intell. Robots Syst.*, 2015, pp. 3442–3448.

[23] D. Nguyen-Tuong and J. Peters, "Using model knowledge for learning inverse dynamics," in *Proc. Int. Conf. Robot. Automat.*, 2010, pp. 2677–2682.

[24] J. Sun de la Cruz, "Learning inverse dynamics for robot manipulator control," M.S. thesis, Univ. Waterloo, 2011.

[25] M. K. Helwa, A. Heins, and A. P. Schoellig, "Provably robust learning-based approach for high-accuracy tracking control of lagrangian systems," *Robot. Automat. Lett.*, vol. 4, no. 2, pp. 1587–1594, 2019.

[26] G. Aoude, B. Luders, J. Joseph, N. Roy, and J. How, "Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns," *Auton. Robots*, vol. 35, no. 1, pp. 51–76, 2013.

[27] R. Pautrat, K. Chatzilygeroudis, and J. Mouret, "Bayesian optimization with automatic prior selection for data-efficient direct policy search," in *Proc. Intl. Conf. Robot. Automat. (ICRA)*, 2018, pp. 7571–7578.

[28] M. Sorocky, S. Zhou, and A. P. Schoellig, "Experience selection using dynamics similarity for efficient multi-source transfer learning between robots," in *Proc. Int. Conf. Robot. Automat.*, 2020, pp. 2739–2745, accepted.

[29] C. D. McKinnon and A. P. Schoellig, "Learning multi-modal models for robot dynamics with a mixture of gaussian process experts," in *Proc. Int. Conf. Robot. Automat.*, 2017, pp. 322–328.

[30] D. Wolpert and M. Kawato, "Multiple paired forward and inverse models for motor control," *Neural Netw.*, vol. 11, no. 7–8, pp. 1317–1329, 1998.

[31] C. Xie, S. Patil, T. Moldovan, S. Levine, and P. Abbeel, "Model-based reinforcement learning with parametrized physical models and optimism-driven exploration," in *Proc. Int. Conf. Robot. Automat.*, 2016, pp. 504–511.

[32] G. Hoffmann, H. Huang, S. Waslander, and C. Tomlin, "Quadrotor helicopter flight dynamics and control: Theory and experiment," in *Proc. Guid., Navigation Control Conf.*, 2007, Art no. 6461.

[33] F. Augugliaro and R. D'Andrea, "Admittance control for physical human-quadrocopter interaction," in *Proc. Eur. Control Conf.*, 2013, pp. 1805–1810.

[34] V. Hagenmeyer and E. Delaleau, "Exact feedforward linearization based on differential flatness," *Intl. J. Control*, vol. 76, no. 6, pp. 537–556, 2003.

[35] M. I. Jordan and D. E. Rumelhart, "Forward models: Supervised learning with a distal teacher," *Cogn. Sci.*, vol. 16, no. 3, pp. 307–354, 1992.

[36] D. Lam, C. Manzie, and M. Good, "Model predictive contouring control," in *Proc. Conf. Decis. Control*, 2010, pp. 6137–6142.

[37] M. Paton, F. Pomerleau, K. MacTavish, C. Ostafew, and T. Barfoot, "Expanding the limits of vision-based localization for long-term route-following autonomy," *J. Field Robot.*, vol. 34, no. 1, pp. 98–122, 2017.