High-Speed Accurate Robot Control using Learned Forward Kinodynamics and Non-linear Least Squares Optimization

Pranav Atreya[1], Haresh Karnan[2], Kavan Singh Sikand[1], Xuesu Xiao[1], Garrett Warnell[1,4], Sadegh Rabiee[1], Peter Stone[1,3], and Joydeep Biswas[1]

***Abstract*— Accurate control of robots in the real world requires a control system that is capable of taking into account the kinodynamic interactions of the robot with its environment. At high speeds, the dependence of the movement of the robot on these kinodynamic interactions becomes more pronounced, making high-speed, accurate robot control a challenging problem. Previous work has shown that learning the inverse kinodynamics (IKD) of the robot can be helpful for high-speed robot control. However a learned inverse kinodynamic model can only be applied to a limited class of control problems, and different control problems require the learning of a new IKD model. In this work we present a new formulation for accurate, high-speed robot control that makes use of a learned forward kinodynamic (FKD) model and non-linear least squares optimization. By nature of the formulation, this approach is extensible to a wide array of control problems without requiring the retraining of a new model. We demonstrate the ability of this approach to accurately control a scale one-tenth robot car at high speeds, and show improved results over baselines.**

I. Introduction and Related Work

At moderate speeds, most existing mobile robot control systems can autonomously and reliably move robots from one point to another. These systems approximate the true underlying robot dynamics using a simple kinodynamics, or sometimes even kinematics, model. Such simplified models assume that robots only operate in a limited subspace of their entire state space, such as low acceleration and speed, minimum wheel slip, negligible tire deformation, and perfect nonholonomic constraints.

However, real-world robotic missions may entail violations of such over-simplified models. For example, in search and rescue missions where time is of the essence, robots need to move as fast as possible in order to reach the victims and therefore cause extensive side-ways slippage; in unstructured outdoor environments, terrain characteristics is not known a priori, and can cause inconsistent vehicle-terrain interaction with the model simplified for homogeneous surfaces. All these real-world challenges motivate a better kinodynamic model so that the control systems can confidently extend the robot operational space into more dynamic regimes in order to adapt to these real-world challenges.

[1]The University of Texas at Austin, Department of Computer Science, {pranavatreya, kvsikand}@utexas.edu, {xiao, joydeepb, pstone}@cs.utexas.edu

[2]The University of Texas at Austin, Department of Mechanical Engineering haresh.miriyala@utexas.edu

[3] Sony, AI

[4] Computational and Information Sciences Directorate, Army Research Laboratory garrett.a.warnell.civ@army.mil

Considering the difficulty in hand-crafting such a model that considers a variety of factors during real-world operation, roboticists have sought help from the machine learning community [1]. End-to-end learning is the most straightforward way to encapsulate both the model and controller in one function approximator and to train it with data, e.g., learning a neural network using imitation learning [2]–[8] or reinforcement learning [9]–[12]. Despite serving as a successful proof-of-concept, it has been reported that the learned systems usually do not perform as well as their classical counterparts [1]. Leaving both model learning and controller learning to data, such end-to-end approaches suffer from the common drawbacks of learning approaches in general, such as high requirement on massive training data, poor generalizability to unseen scenarios, and most importantly, a lack of safety assurance for navigation, which is commonly provided by classical planning and control algorithms.

To address such disadvantages of end-to-end learning, hybrid approaches have been introduced that leverage existing robot planning and control methods. For example, Wigness et al. [13] and Sikand et al. [14] used imitation learning to learn a cost function for existing navigation controllers to enable adaptive behaviors for semantics. Similar techniques have been applied to learn socially compliant navigation [15], [16]. Xiao et al. [17] introduced Adaptive Planner Parameter Learning, which learns to dynamically adjust existing motion planners' parameters to efficiently navigate through different obstacle configurations using teleoperated demonstration [18], corrective interventions [19], evaluative feedback [20], and reinforcement learning [21]. These approaches focuses on using machine learning to enable custom behaviors, such as semantic awareness, social compliance, or smooth obstacle-avoidance.

Machine learning has also been used for high-speed robot control [22]. Model Predictive Path Integral [23] utilizes a classical sampling based controller in an online learning fashion: it learns a sample distribution during online deployment that is likely to generate good samples [24]. However, it uses a simple unicycle forward model to predict future path based on the samples and compensates such an over-simplified model by a massive number of samples evaluated in parallel on GPUs [25]. Brunnbauer et al. [26] reported that model-based deep reinforcement learning substantially outperforms model-free agents with respect to performance, sample efficiency, successful task completion, and generalization in autonomous racing. Roboticists have also investigated accurate, high-speed, off-road navigation on

unstructured terrain [27] by learning an inverse kinodynamic model conditioned on onboard inertia observations.

Leveraging a hybrid paradigm to address high-speed robot control problems, our approach, which we call Optim-FKD, falls into the model learning regime [28] and utilizes numerical optimization to find optimal control sequences based on the learned model in order to enable high-speed, accurate robot motions. To be specific, we employ the direct modeling paradigm [28] and learn a forward kinodynamics model to be used in different downstream optimization tasks. The contribution of this paper can be summarized as follows:

1) A novel formulation for robotic control using a learned forward kinodynamic function and numerical optimization. We demonstrate that this formulation is easily extensible to a range of control tasks without requiring the retraining of a new forward kinodynamic model.
2) A novel learning formulation that enables a highly accurate forward kinodynamic model to be learned.
3) A detailed description of the system architecture required to enable the presented approach to run on real robot hardware in real time.
4) Empirical results demonstrating that the presented approach outperforms baselines for various robot control tasks.

II. Mathematical Formulation

High speed, accurate robot control as a problem can be formulated in many different ways. Here we present two different formulations of the problem, and show that each formulation can be solved by the same class of solution, namely a nonlinear least squares optimization that uses a forward kinodynamic model.

A. Preliminaries

Let X represent the state space of the robot. X not only consists of configuration space variables (such as position and orientation) but also dynamics variables (such as linear and angular velocity). Let U represent the control space of the robot. Consider a period of time of operation of the robot Δt. It is assumed that controls are executed on the robot in a piecewise constant manner. Let τ be the duration for which a particular constant control is executed. In the time period of operation Δt, the robot will execute $n = \frac{\Delta t}{\tau}$ constant controls.

To model the response of the robot from the executed controls, we introduce a state transition likelihood function $\rho : X \times U^n \times X^n \rightarrow [0, 1]$. ρ takes as input the initial state of the robot $x_0 \in X$, a length-n piecewise constant control sequence $u_{1:n}$, and a length-n state sequence $x_{1:n}$. Each $x_i \in x_{1:n}$ represents the state of the robot at time $i * \tau$. The output of ρ is the probability that the state sequence $x_{1:n}$ is observed after executing $u_{1:n}$ beginning from x_0.

We assume that the motion of the robot obeys the Markov property, that is, the probability of reaching a state x_i depends only on the previous state x_{i-1} and the constant control executed beginning at that previous state u_i. This induces a local state transition likelihood function $\rho_i(x_{i-1}, u_i, x_i)$ for every $i \in 1...n$. We can thus model ρ as

$$\rho(x_0, u_{1:n}, x_{1:n}) = \prod_{i=1}^{n} \rho_i(x_{i-1}, u_i, x_i) \tag{1}$$

Equation 1 represents the probability that $x_{1:n}$ is observed after executing $u_{1:n}$ from x_0. It is also useful to consider what the maximum likelihood state sequence $\widehat{x_{1:n}}$ is after executing $u_{1:n}$ from x_0.

$$\widehat{x_{1:n}} = \arg\max_{x_{1:n}} \rho(x_0, u_{1:n}, x_{1:n}) \tag{2}$$

$$= \arg\max_{x_{1:n}} \prod_{i=1}^{n} \rho_i(x_{i-1}, u_i, x_i) \tag{3}$$

For each ρ_i, consider the conditional probability $p(x_i|u_i, x_{i-1})$. We assume that $p(x_i|u_i, x_{i-1})$ follows a normal distribution: $p(x_i|u_i, x_{i-1}) \sim \mathcal{N}(\bar{x}_i, \sigma_{x_i})$. We represent the maximum likelihood estimate of $p(x_i|u_i, x_{i-1})$ as the forward kinodynamic function $\pi(u_i, x_{i-1}) = \bar{x}_i$. With this definition of π, equation 3 can be rewritten as

$$\widehat{x_{1:n}} = (\pi(u_1, x_0), ..., \pi(u_n, \widehat{x_{n-1}})) \tag{4}$$

With these preliminaries we will now show that various robot control problems can be expressed as nonlinear least squares optimizations that use the forward kinodynamic function π.

B. Formulation 1: Path Following

The problem we consider here is that of following a predefined path as closely as possible. This problem becomes noteworthy at high speeds where accurate control of the robot becomes increasingly more difficult.

We're give as input $x^*_{1:n}$ which describes a path to follow. Following this path as closely as possible amounts to solving the following problem:

$$\arg\max_{u_{1:n}} \rho(x_0, u_{1:n}, x^*_{1:n}) \tag{5}$$

From equation 4, this is equivalent to the problem

$$\arg\min_{u_{1:n}} ||x^*_{1:n} - \widehat{x_{1:n}}||_2^2 \tag{6}$$

This is a nonlinear least squares formulation where each $\widehat{x}_i \in \widehat{x_{1:n}}$ is determined from the forward kinodynamic function π.

C. Formulation 2: Optimal Connectivity

Another variant of the robot control problem that we consider is traversing from a start state x_i to a goal state x_f in as little time as possible. Problems of this type appear frequently in optimal sampling-based motion planning where algorithms like RRT* [29] and BIT* [30] require a steering function that can time-optimally connect arbitrary states.

Consider the maximum likelihood state sequence $\widehat{x_{1:n}}$ from earlier. If we wanted the final state of the robot to be as close as possible to the goal state x_f, we would optimize

the following

$$\arg\min_{u_{1:n}} ||x_f - \widehat{x_n}||_2^2 \tag{7}$$

This formulation however keeps the time that the goal state x_f is reached fixed. Specifically, the state $\widehat{x_n}$ is reached after time $n*\tau$. To also minimize the time taken to reach the goal, n is introduced as an optimization parameter, and a slightly different objective function is used:

$$\arg\min_{u_{1:n},n} ||x_f - \widehat{x_n}||_2^2 + (\alpha(n*\tau))^2 \tag{8}$$

Here α is a scaling parameter that trades off time to reach the goal and the distance to the goal. Like the path following formulation, this formulation is a nonlinear least squares optimization where $\widehat{x_n}$ is determined from the forward kinodynamic function π.

III. Forward Kinodynamic Model Learning

In this section we present how the forward kinodynamic model π is learned. Since π is an integral component to the nonlinear least squares optimizations introduced earlier, it is key that π models the true forward kinodynamics effectively. We learn π for a scale one-tenth autonomous robot car.

A. Dataset Generation

The FKD model π is trained in a supervised manner, and thus needs a dataset to learn from. We obtain this dataset by teleoperating the robot at various speeds and recording at every timestep the state estimates of the robot and the joysticked control commands. This results in a dataset D of trajectories $T_1, ..., T_m$ where each trajectory $T_i \in D$ is a tuple of the form $(v_x(t), v_y(t), \omega(t), x(t), y(t), \theta(t), \delta(t), \psi(t))$. Here, $v_x(t)$ is the velocity in the x-direction, $v_y(t)$ is the velocity in the y-direction, $\omega(t)$ is the angular velocity, $x(t)$ is the x-position, $y(t)$ is the y-position, $\theta(t)$ is the orientation, $\delta(t)$ is the commanded forward velocity, and $\psi(t)$ is the commanded angular velocity. All of these functions are time-dependent and are defined in the domain $[0, t_f^{(i)}]$ where $t_f^{(i)}$ is the termination time for trajectory T_i.

B. Learning Formulation

The formulation presented earlier for π maps an initial state of the robot and a constant control to the most likely next state of the robot. Learning this exact formulation, while in theory would work fine, in practice does not yield good performing FKD models. This is because the model is tasked with predicting the state after τ units of time, which in most cases is very close to the current state since τ is selected to be small. This means the model can get away with simply predicting the current state without incurring much loss. In order to ensure the model's predictions are of high quality, the model needs to learn to model the state of the robot after a time period much greater than τ.

We achieve this longer prediction horizon by training π in a recurrent fashion. Simply increasing τ would not suffice since that would forego the fine-grained prediction capabilities of the model. The basic structure of the recurrence formulation is as follows. The model predicts the next state $x_i = \pi(u_i, x_{i-1})$. For timestep $i+1$, instead of being given access to the ground truth value of x_i, the model uses its previous prediction as the starting state: $x_{i+1} = \pi(u_{i+1}, x_i)$. This process continues for the number of timesteps in the prediction horizon.

This simple recurrent approach has a few limitations however. The base timestep duration τ is selected to be small so as to capture minute changes in the state of the robot. In our experiments we set τ to 0.05 seconds. For the model to predict the state of the robot after time t_{pred}, $\frac{t_{\text{pred}}}{\tau}$ forward passes through the model are needed. In our experiments, we set t_{pred} to 3.0 seconds, requiring 60 forward passes through the model. Since π is to be used in a real-time optimization framework, the number of forward passes through π need to be limited to maintain computational efficiency. We achieve this by introducing a model prediction time t_{model}. The FKD model π, in one forward pass, outputs the next $\frac{t_{\text{model}}}{\tau}$ states given the next $\frac{t_{\text{model}}}{\tau}$ controls. It also takes in as input the previous $\frac{t_{\text{model}}}{\tau}$ states, enabling recurrence. In our experiments t_{model} was set to 0.5 seconds. This approach enables both computational efficiency and fine-grained prediction.

Having motivated the model formulation, we now present the learning objective. For each trajectory T_i in our dataset D, we evenly sample k starting times $(t_1, ..., t_k)$ from the range $[t_{\text{model}}, t_f^{(i)} - t_{\text{pred}}]$. For each starting time $t_a \in (t_1, ..., t_k)$, we will use the model to predict the states at times $t_a + b\tau$ for $b \in [0, 1, ..., \frac{t_{\text{pred}}}{\tau}]$. For brevity, let S_{t_a} be the state variables $v_x(t), v_y(t), \omega(t), x(t), y(t)$, and $\theta(t)$ sampled evenly with spacing τ from the time period $[t_a, t_a + t_{\text{model}}]$. Additionally let M_{t_a} be the control variables $\delta(t)$ and $\psi(t)$ sampled in the same manner. The model π takes in as input M_{t_a} and $S_{t_a - t_{\text{model}}}$ and produces as output $\tilde{S}_{t_a}$. $\tilde{S}_{t_a}$ differs from S_{t_a} in that the former is the model's prediction whereas the latter is the ground truth. Taking everything into account, we obtain the following learning objective

$$\arg\min_{\Theta} \sum_{T_i \in D} \sum_{t_a \in (t_1,...,t_k)} \sum_{i=0}^{\frac{t_{\text{pred}}}{t_{\text{model}}}-1} ||\pi(\tilde{S}_{t_a + t_{\text{model}}*(i-1)}, M_{t_a + i*t_{\text{model}}}) - S_{t_a + i*\text{model}}||_2^2 \tag{9}$$

where Θ is the parameter set of π and $\tilde{S}_{t_a - t_{\text{model}}} = S_{t_a - t_{\text{model}}}$ for the case $i = 0$.

IV. Optimization System Architecture

Here we describe the system architecture of the proposed approach. We discuss how to integrate the optimization procedure and calls to the FKD model in a manner that enables real-time control on real robot hardware. Figure 1 shows a block diagram of the system components. There are four key components that all operate asynchronously: the state estimator, optimizer, updater, and executor.

A. State Estimator

In order for a predefined path to be followed accurately or for a goal state to be reached as fast as possible, accurate state estimates of the robot are essential. These state estimates

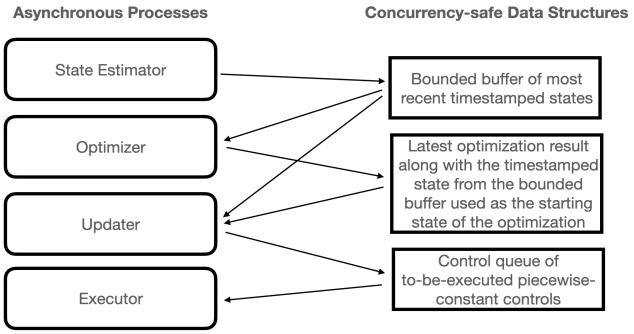


Fig. 1: System architecture block diagram.

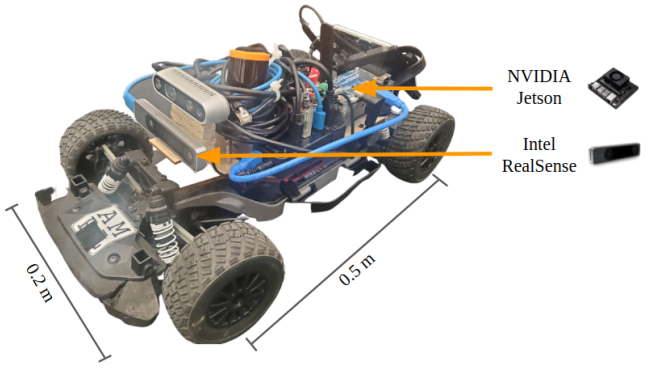


Fig. 2: UT-Automata F1Tenth Robot Car

define the robot's position, orientation, velocity, and angular velocity with respect to some coordinate frame. The state estimates themselves can come from a variety of sources such as a LIDAR based localization algorithm or visual odometry. It is assumed that there exists some delay ϵ between the actual, real-world state of the robot and what the state estimator outputs. The state estimator operates asynchronously, continuously updating a fixed sized buffer of the most recent state estimates. It is critical that each state estimate in the buffer is timestamped.

B. Optimizer

The optimizer begins by obtaining from the state buffer the most recent estimated state of the robot ξ. This state will be used as the start state in the optimization procedure. Now, depending on the optimization objective, different steps must be taken. For the path following objective, it is citical to first localize the robot on the map that the robot is desired to follow. Let P be the path that the robot is assigned to follow at speed v_{desired}. P consists of a series of robot positions x, y, θ ordered in increasing order by which each position is to be reached by the robot. Each optimization will plan the next Δt controls for the robot. To do this, the goal state $g \in P$ the robot needs to reach after time Δt needs to be determined. Localizing the robot in P amounts to finding the position $s \in P$ that minimizes $||\xi - s||_2^2$. g can then be obtained by computing $g = p \rightsquigarrow v_{\text{desired}} * \Delta t$ where the $a \rightsquigarrow b$ operator looks ahead in P from position a by b distance.

Next, the optimizer must prepare the input required by the forward kinodynamic model, namely the past t_{model} units time of robot state information. This is done by running time synchronization on the states in the state buffer by making use of the states' timestamps to sample $\frac{t_{\text{model}}}{\tau}$ evenly spaced states. Finally the optimization procedure is called, which internally will optimizer over the next Δt of controls by making $\frac{\Delta t}{t_{\text{model}}}$ calls to the FKD model. The result u^* along with ξ is stored in a concurrency-safe data structure.

C. Updater

The main role of the updater is dealing with latencies that are characterstic of real robot systems. We consider the two most impactful latencies: ϵ, the previously introduced latency in the state estimators measurements, and γ_i, the time required for optimization i to complete. Since γ_i is different for every run of the optimizer, it needs to first be computed. This is done by computing the difference between the current system time and the timestamp of ξ that was used as the starting state of the optimization procedure. The first $\gamma_i + \epsilon$ units time of controls are discarded from u^*, and the remainder replaces the contents of the control buffer.

D. Executor

Finally, the executor asynchronously executes the commands stored in the control queue one at a time on the robot. If the forward kinodynamic model π had been learned well, the evolution of the state of the robot after executing the controls will closely match the predicted state sequence by the model, meaning the controls output by the optimization procedure are the desired ones.

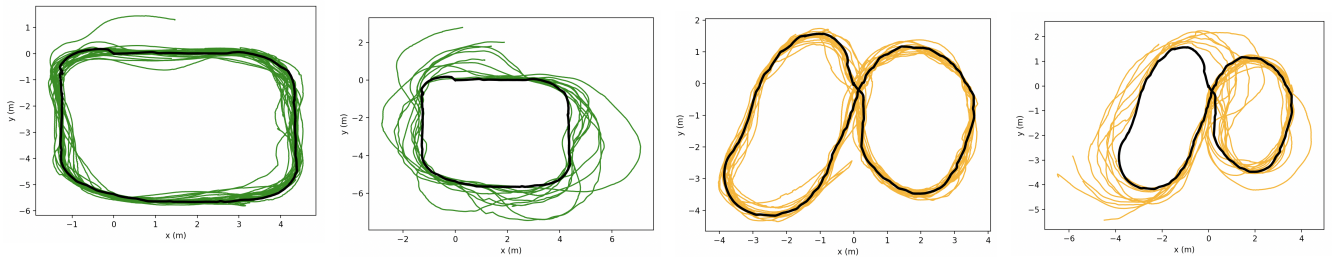
V. Experiments

To evaluate the performance of our proposed Optim-FKD approach, we perform two sets of experiments, each involving a different variant of the robot control problem. We demonstrate the ability of Optim-FKD to successfully complete both, and show improved performance over an optimization-free, IKD model baseline.

A. Experimental Setup

The robot platform used for our experimentation is the UT-Automata F1tenth Car depicted in figure 2. We make use of the car's Intel Realsense T265 tracking camera for localization and its Nvidia Jetson TX2 for compute.

We consider two different experimental setups, each involving a different form of the robot control problem and thus a different optimization objective. For the first task we consider having the robot follow two different predefined paths as accurately as possible at speeds varying from $1.0\ m/s$ to $3.0\ m/s$. This task corresponds to the optimization objective presented in equation 6. For the second task we have the robot traverse from an initial starting state to a goal state in as little time as possible, without explicitly constraining which path it needs to take to get to the goal state. This task corresponds to the optimization objective presented in equation 8.



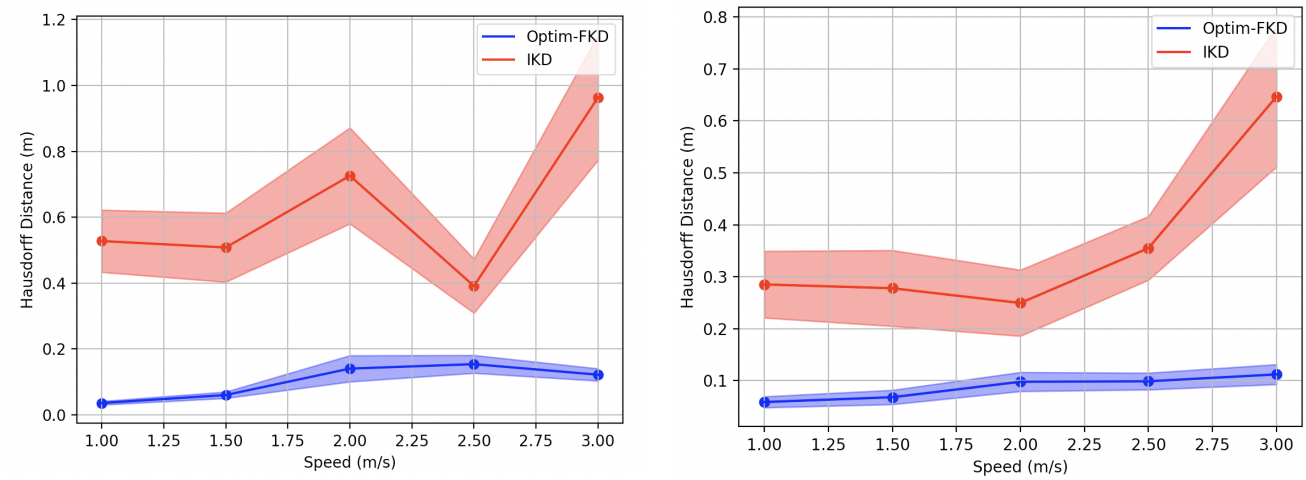
(a) Optim-FKD on rounded rectangle

(b) IKD on rounded rectangle

(c) Optim-FKD on figure 8

(d) IKD on figure 8

Fig. 3: Superposition of execution traces of both the Optim-FKD and IKD algorithms running at different speeds on the rounded rectangle and figure 8 paths. In (a) and (c), which correspond to Optim-FKD, the execution trace matches the desired path (black) very closely. In contrast, there are significant errors in the baseline IKD model shown in (b) and (d).



(a) Rounded rectangle path

(b) Figure 8 path

Fig. 4: Average Hausdorff distances between the executed paths and the desired paths across different speeds. Each speed's average is computed for five rollouts.

The baseline algorithm we consider for all of these experiments is an inverse kinodynamic model. In contrary to the forward kinodynamic model, which maps a sequence of controls to the most likely next sequence of states, the inverse kinodynamic model attempts to infer what controls led to a particular state sequence. To ensure fairness in the comparison, the underlying neural network used in both the FKD and IKD models was identical, with the same architecture and activations; the dataset that both were trained on was identical; and finally the number of iterations each was trained for was identical.

B. Experiment 1: Path Following

Here we assess the control capabilities of the Optim-FKD model on the task of accurately following a path. Figure 3 shows a visualization in black of the two paths used: a rounded rectangle path and a figure 8 path. Both paths were generated by joysticking the robot at a slow speed and recording only the position estimates. Each algorithm is assessed on how accurately it can follow the paths both at slower and higher speeds. For each speed, five full traversals through the desired path are completed. We measure an algorithm's ability to closely follow a path at a particular speed by computing the Hausdorff distance between the executed path and the desired path.

Figure 4 shows the evaluation of the Hausdorff distance metric for (a) the rounded rectangle path and (b) the figure 8 path. We can see that Optim-FKD produces execution traces that are significantly closer to the desired trajectories than the IKD model. For both paths, the Optim-FKD algorithm is strictly under 0.2 Hausdorff distance, while the IKD algorithm is strictly greater than 0.2. Figure 3 provides a visual argument for these results: in (a) and (c) the executed path hugs the desired path much more closely than in (b) and (d). For both algorithms and across both paths, we observe that increases in velocity tend to result in higher Hausdorff distances. This phenomenon is expected, since at high velocities the kinodynamic responses of the robot with respect to the terrain become more pronounced, making accurate control more difficult.

We posit that the better performance of the Optim-FKD approach is explained by an inherent advantage in FKD models over IKD models: better sample efficiency. Most

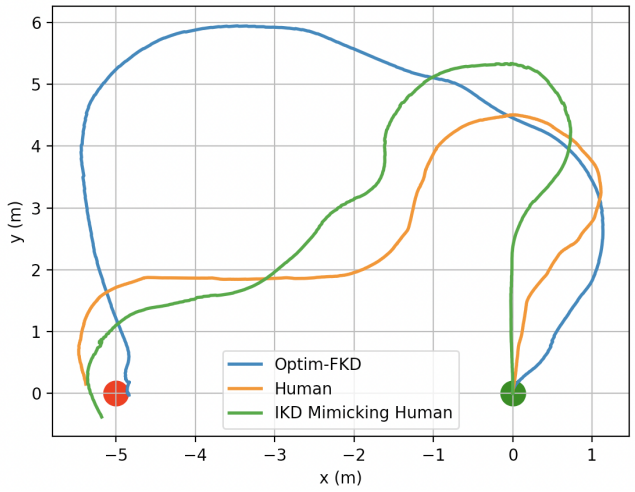


Fig. 5: Paths produced by Optim-FKD, a human, and the IKD model following the human's path. Objective is to reach from the start (green) state at $(0, 0)$ to the goal (red) state at $(-5, 0)$ in minimum time. The start and end velocities are constrained to be $0\ \frac{m}{s}$, the start heading is $\frac{\pi}{2}\ rad$, and the end heading must be $\frac{3\pi}{2}\ rad$.

robots where kinodynamic interactions with the environment are important are underactuated, i.e., the dimensionality of the control space is smaller than that of the state space. For a FKD model and an IKD model trained on the same dataset, the FKD model will learn a mapping from a lower dimensional space (the control space) to a higher dimensional space (the state space), whereas the IKD model learns the opposite mapping. Because of the control space's lower dimensionality, the training dataset will contain a larger fraction of the possible model inputs than the training dataset for the IKD model. This means that for the same amount of training data, the FKD model is exposed during training to a higher percentage of its input space than the FKD model. It follows that the FKD model is more reliable in estimating the effects of controls than the IKD model is in explaining what controls led to a particular state sequence. The optimization is able to leverage this improved prediction performance and can deal with cases where the IKD model would be in unknown territory. This argument is thus one of sample efficiency, with the FKD model better able to make use of its training dataset than the IKD model.

C. Experiment 2: Optimal Connectivity

With this experiment we demonstrate the improved generalizability of the Optim-FKD approach over the IKD approach. As demonstrated in section II-C, the optimal connectivity problem can be solved by Optim-FKD simply by altering the optimization objective. However, to solve the same problem with the IKD approach would necessarily involve the retraining of a new IKD model. This is because the IKD model has a fixed prediction horizon. Given a nearby state, it can output what controls to execute during this prediction horizon to reach that state. However in the general optimal connectivity problem, it may be possible that the goal state cannot be reached within the IKD model's prediction horizon, simply because it is too far away. Thus multiple evaluations of the IKD model would be necessary. But to do any one evaluation, the IKD model must know what the desired state is at time equal to its prediction horizon. This effectively means that the IKD model needs to be supplied with so called racing lines, i.e., the optimal path to take to reach the goal state.

It's more than likely that the provided racing lines are suboptimal, and in this experiment we demonstrate just that. Figure 5 depicts the problem of traversing from position $(0, 0)$ to $(-5, 0)$ in as little time as possible. This traversal cannot be done in the IKD model's prediction horizon, and so racing lines shown in yellow must be provided by a human. The IKD model then attempts to trace the racing line as fast as possible shown in green. In contrast Optim-FKD simply runs the optimization described in equation 8, and is thus not dependent on racing lines nor requires a new FKD model to be trained. The robot is able to execute the path in blue in $6.34\ s$ whereas it takes $7.20\ s$ for the robot to execute the path in green. Thus the Optim-FKD approach was able to find a more optimal path than the human provided one. Inspecting the blue path more reveals that because of the wide turn, it was able to traverse upwards of $3\ \frac{m}{s}$ whereas the maximum speed of the robot traversing the green path was $2.6 \frac{m}{s}$.

VI. Conclusion

In this work we presented Optim-FKD, a new algorithmic technique for accurate, high-speed robot control. We showed that solutions to various formulations of the robot control problem can be naturally expressed as a nonlinear least squares optimization with a FKD model. We demonstrate how such an FKD model can be learned effectively and integrated with the optimization on real robot hardware. Finally we evaluate our proposed approach on two robotic control tasks at high speeds and show that it outperforms the baseline.

ACKNOWLEDGMENT

This work has taken place in the Autonomous Mobile Robotics Laboratory (AMRL) at UT Austin. AMRL research is supported in part by NSF (CAREER-2046955, IIS-1954778, SHF-2006404), ARO (W911NF-19-2-0333,W911NF-21-20217), DARPA (HR001120C0031), Amazon, JP Morgan, and Northrop Grumman Mission Systems. The views and conclusions contained in this document are those of the authors alone.

References

[1] X. Xiao, B. Liu, G. Warnell, and P. Stone, "Motion control for mobile robot navigation using machine learning: a survey," *Autonomous Robots*, 2022.

[2] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[3] L. Tai, S. Li, and M. Liu, "A deep-network solution towards model-less obstacle avoidance," in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2016, pp. 2759–2764.

[4] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots," in *2017 ieee international conference on robotics and automation (icra)*. IEEE, 2017, pp. 1527–1533.

[5] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. A. Theodorou, and B. Boots, "Imitation learning for agile autonomous driving," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 286–302, 2020.

[6] X. Xiao, B. Liu, G. Warnell, and P. Stone, "Toward agile maneuvers in highly constrained spaces: Learning from hallucination," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1503–1510, 2021.

[7] X. Xiao, B. Liu, and P. Stone, "Agile robot navigation through hallucinated learning and sober deployment," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 7316–7322.

[8] Z. Wang, X. Xiao, A. J. Nettekoven, K. Umasankar, A. Singh, S. Bommakanti, U. Topcu, and P. Stone, "From agile ground to aerial navigation: Learning from learned hallucination," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 148–153.

[9] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 31–36.

[10] M. Pfeiffer, S. Shukla, M. Turchetta, C. Cadena, A. Krause, R. Siegwart, and J. Nieto, "Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4423–4430, 2018.

[11] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with autorl," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2007–2014, 2019.

[12] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 285–292.

[13] M. Wigness, J. G. Rogers, and L. E. Navarro-Serment, "Robot navigation from human demonstration: Learning control behaviors," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1150–1157.

[14] K. S. Sikand, S. Rabiee, A. Uccello, X. Xiao, G. Warnell, and J. Biswas, "Visual representation learning for preference-aware path planning," in *2022 ieee international conference on robotics and automation (icra)*. IEEE, 2022.

[15] B. Kim and J. Pineau, "Socially adaptive path planning in human environments using inverse reinforcement learning," *International Journal of Social Robotics*, vol. 8, no. 1, pp. 51–66, 2016.

[16] H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard, "Socially compliant mobile robot navigation via inverse reinforcement learning," *The International Journal of Robotics Research*, vol. 35, no. 11, pp. 1289–1307, 2016.

[17] X. Xiao, Z. Wang, Z. Xu, B. Liu, G. Warnell, G. Dhamankar, A. Nair, and P. Stone, "Appl: Adaptive planner parameter learning," *arXiv preprint arXiv:2105.07620*, 2021.

[18] X. Xiao, B. Liu, G. Warnell, J. Fink, and P. Stone, "Appld: Adaptive planner parameter learning from demonstration," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4541–4547, 2020.

[19] Z. Wang, X. Xiao, B. Liu, G. Warnell, and P. Stone, "Appli: Adaptive planner parameter learning from interventions," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 6079–6085.

[20] Z. Wang, X. Xiao, G. Warnell, and P. Stone, "Apple: Adaptive planner parameter learning from evaluative feedback," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7744–7749, 2021.

[21] Z. Xu, G. Dhamankar, A. Nair, X. Xiao, G. Warnell, B. Liu, Z. Wang, and P. Stone, "Applr: Adaptive planner parameter learning from reinforcement," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 6086–6092.

[22] J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle, M. Behl, V. Krovi, and R. Mangharam, "Autonomous vehicles on the edge: A survey on autonomous vehicle racing," *arXiv preprint arXiv:2202.07008*, 2022.

[23] G. Williams, A. Aldrich, and E. A. Theodorou, "Model predictive path integral control: From theory to parallel computation," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 344–357, 2017.

[24] 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.

[25] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1433–1440.

[26] A. Brunnbauer, L. Berducci, A. Brandstätter, M. Lechner, R. Hasani, D. Rus, and R. Grosu, "Model-based versus model-free deep reinforcement learning for autonomous racing cars," *arXiv preprint arXiv:2103.04909*, 2021.

[27] X. Xiao, J. Biswas, and P. Stone, "Learning inverse kinodynamics for accurate high-speed off-road navigation on unstructured terrain," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 6054–6060, 2021.

[28] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive processing*, vol. 12, no. 4, pp. 319–340, 2011.

[29] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2. IEEE, 2000, pp. 995–1001.

[30] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 3067–3074.