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

Pranav Atreya\textsuperscript{1}, Haresh Karnan\textsuperscript{2}, Kavan Singh Sikand\textsuperscript{1}, Xuesu Xiao\textsuperscript{3}, Garrett Warnell\textsuperscript{1,4}, Sadegh Rabiee\textsuperscript{1}, Peter Stone\textsuperscript{1,3}, and Joydeep Biswas\textsuperscript{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.

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 $\Delta t$. It is assumed that controls are executed on the robot in a piecewise constant manner. Let $\tau$ be the duration for which a particular constant control is executed. In the time period of operation $\Delta t$, 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 $\rho : X \times U^n \times X^n \rightarrow [0,1]$. $\rho$ 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 $i = 1, \ldots, n$. The output of $\rho$ 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 $i \in 1 \ldots n$. We can thus model $\rho$ as

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

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 $\tilde{x}_{1:n}$ is after executing $u_{1:n}$ from $x_0$.

$$\tilde{x}_{1:n} = \arg\max_{x_{1:n}} \rho(x_0, u_{1:n}, x_{1:n})$$

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

For each $\rho_i$, 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: $p(x_i|u_i, x_{i-1}) \sim \mathcal{N}(\bar{x}_i, \sigma_i)$. 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 $\pi$, equation 3 can be rewritten as

$$\tilde{x}_{1:n} = (\pi(u_1, x_0), \ldots, \pi(u_n, \tilde{x}_{n-1}))$$

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 $\pi$.

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:

$$\arg\max_{u_{1:n}} \rho(x_0, u_{1:n}, x'_{1:n})$$

From equation 4, this is equivalent to the problem

$$\arg\min_{u_{1:n}} \|x'_{1:n} - \tilde{x}_{1:n}\|^2_2$$

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

C. Formulation 2: Optimal Connectivity

Another variant of the robot control problem that we consider is traversing from a start state $x_f$ 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 $\tilde{x}_{1:n}$ 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

$$\arg \min \| x_f - \tilde{x}_n \|_2$$

(7)

This formulation however keeps the time that the goal state

$$x_f$$ is reached fixed. Specifically, the state \( \tilde{x}_n \) 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:

$$\arg \min \| x_f - \tilde{x}_n \|_2^2 + (\alpha(n + \tau))^2$$

(8)

Here \( \alpha \) 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 \( \tilde{x}_n \) is determined from the forward
kinodynamic function \( \pi \).

III. FORWARD KINODYNAMIC MODEL LEARNING

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

A. Dataset Generation

The FKD model \( \pi \) 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 joy-
sticked control commands. This results in a dataset \( D \) of tra-
jectories \( T_1, ..., T_m \) 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 \( \pi \) 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 \( \tau \) units of time, which in most
cases is very close to the current state since \( \tau \) 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 \( \tau \).

We achieve this longer prediction horizon by training \( \pi \)
in a recurrent fashion. Simply increasing \( \tau \) would not suffice
since that would forego the fine-grained prediction capabili-
ties 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 \( \tau \) is selected to be small
so as to capture minute changes in the state of the robot. In
our experiments we set \( \tau \) to 0.05 seconds. For the model to
predict the state of the robot after time \( \tau_{\text{pred}} \), \( \tau_{\text{pred}} \) forward
passes through the model are needed. In our experiments, we
set \( \tau_{\text{pred}} \) to 3.0 seconds, requiring 60 forward passes through
the model. Since \( \pi \) is to be used in a real-time optimization
framework, the number of forward passes through \( \pi \) need to
be limited to maintain computational efficiency. We achieve
this by introducing a model prediction time \( \tau_{\text{model}} \). The FKD
model \( \pi \), in one forward pass, outputs the next \( \tau_{\text{model}} \) states
given the next \( \tau_{\text{model}} \) controls. It also takes in as input the
previous \( \tau_{\text{model}} \) states, enabling recurrence. In our exper-
iments \( \tau_{\text{model}} \) 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 \([\tau_{\text{model}}, \tau_{\text{pred}} - \tau_{\text{model}}]\). 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 \( b \in [0, 1, ..., \tau_{\text{model}}] \). For brevity, let \( S_{t_a} \) be
the state variables \( v_x(t), v_y(t), \omega(t), x(t), y(t), \theta(t) \) and \( \delta(t) \)
sampled evenly with spacing \( \tau \) from the time period \([t_a, t_a + \tau_{\text{model}}]\).
Additionally let \( M_{t_a} \) be the control variables \( \delta(t) \) and \( \psi(t) \)
sampled in the same manner. The model \( \pi \) takes \( M_{t_a} \) as in input
\( M_{t_a} \) and \( S_{t_a} - \tau_{\text{model}} \) and produces as output \( \hat{S}_{t_a} \). \( \hat{S}_{t_a} \) 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

$$\arg \min \sum_{T_i \in D} \sum_{t_a \in (t_i, t_{i+1})} \tau \left| \left| S_{t_a + i \tau_{\text{model}}} - S_{t_a + i \tau_{\text{model}}} \right| \right|_2$$

(9)

where \( \Theta \) is the parameter set of \( \pi \) and \( \hat{S}_{t_a - \tau_{\text{model}}} =
S_{t_a - \tau_{\text{model}}} \) 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
define the robot’s position, orientation, velocity, and angular
delay in the state estimators measurements, and \( \gamma_i \),
velocity with respect to some coordinate frame. The state
evaluated. Since \( \gamma_i \) is different for every run of the optimizer,
estimates themselves can come from a variety of sources
we have the robot traverse from an initial starting state to
such as a LIDAR based localization algorithm or visual
what the state estimator outputs. The state estimator operates
localization and its Nvidia Jetson TX2 for compute.
 denotes the previously introduced
constrained which path it needs to take to get to the goal
involving a different form of the robot control problem.

B. Optimizer

The optimizer begins by obtaining from the state buffer
the state. This task corresponds to the optimization objective
the most recent estimated state of the robot \( \xi \). This state
the model, meaning the controls output by the optimization
the current system time and the timestamp of \( \xi \) that was used
is stored in a concurrency-safe data structure.

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. If the forward kinodynamic model \( \pi \) had been learned well,
the optimizer over the next \( \Delta t \) of controls by making
\( \frac{\Delta t}{t_{model}} \) calls to the FKD model. The result \( u^* \)
position \( s \in P \) that minimizes \( ||\xi - s||^2_2 \). \( g \) can then
ordered in increasing order by which each position
localization and its Nvidia Jetson TX2 for compute.

C. Updater

The main role of the updater is dealing with latencies
that are characterstic of real robot systems. We consider the
we make use of the car’s Intel RealSense T265 tracking camera for
involves a different variant of the robot control problem.

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. If the forward kinodynamic model \( \pi \) had been learned well,
the optimizer over the next \( \Delta t \) of controls by making
\( \frac{\Delta t}{t_{model}} \) calls to the FKD model. The result \( u^* \)
along with \( \xi \) is stored in a concurrency-safe data structure.

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 \( \pi \) had been learned well,
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.
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).

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
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 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 $3\frac{\pi}{2}$ whereas the maximum speed of the robot traversing the green path was $2.6\frac{\pi}{7}$.

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


