OneVision: Centralized to Distributed Controller Synthesis with Delay Compensation

Jiayi Wei[1], Tongrui Li[1], Swarat Chaudhuri[1], Isil Dillig[1], and Joydeep Biswas[1]

***Abstract*— We propose a new algorithm to simplify the controller development for distributed robotic systems subject to external observations, disturbances, and communication delays. Unlike prior approaches that propose specialized solutions to handling communication latency for specific robotic applications, our algorithm uses an arbitrary centralized controller as the specification and automatically generates distributed controllers with communication management and delay compensation. We formulate our goal as nonlinear optimal control—using a regret minimizing objective that measures how much the distributed agents behave differently from the delay-free centralized response—and solve for optimal actions w.r.t. local estimations of this objective using gradient-based optimization. We analyze our proposed algorithm's behavior under a linear time-invariant special case and prove that the closed-loop dynamics satisfy a form of input-to-state stability w.r.t. unexpected disturbances and observations. Our experimental results on both simulated and real-world robotic tasks demonstrate the practical usefulness of our approach and show significant improvement over several baseline approaches.**

I. Introduction

We are interested in distributed multi-agent control of robots in environments with unknown conditions or obstacles. Examples of such settings include autonomous convoy driving following a human driver and autonomous formation control of a fleet that needs to change formations in response to obstacles. Unlike in applications with fixed formation [21], [10] or trajectory control [12], the agents' behavior can vary significantly based on environmental observations, such as the observed trajectory of the lead car in the convoy setting or a narrow tunnel for the formation switching setting. Thus, it is preferable to specify the *behavior* of the robotic fleet, rather than their *execution*, as a desired ideal central controller. Unfortunately, such ideal central controllers cannot be executed directly in a distributed setting since each agent is only capable of observing their own local state, and communication latency leads to delayed observations of other agents. While there have been a few specialized solutions for handling communication latency for specific controllers such as formation control [19] and coordinated path following [11], synthesizing distributed controllers from arbitrary central controllers while accounting for communication delays has remained an open problem until now.

In this paper, we present OneVision, an algorithm for distributed control of multi-agent systems with local observations, in the presence of external disturbances and communication delays. OneVision accepts an ideal central control function for a multi-agent system as well as a system dynamics and observation model. Given the ideal central control function, OneVision generates local plans at every time step by minimizing a *regret loss* using gradient-based optimization. This regret loss is defined as the difference between the predicted future states and actions and an *ideal fleet trajectory* computed by forward-predicting the central controller on delay-compensated local observations from all agents. Since the ideal fleet trajectory cannot be locally computed in real time due to communication delays, each synthesized distributed controller also computes a local approximation of the ideal fleet trajectory and plans its future actions against this approximated objective.

Although OneVision works with arbitrary discrete-time multi-agent controllers, we limit our theoretical analysis to cases where the system dynamics and centralized controller are linear time-invariant. We prove that the distributed agents' execution generated by OneVision converges to the ideal fleet trajectory and is stable in the sense that smaller external disturbances lead to staying closer to the ideal trajectory. In addition, we provide empirical evidence of convergence and stability for a number of non-linear examples.

We summarize our contributions as follows:

- We present OneVision, a general algorithm for synthesizing distributed controllers from a centralized controller specification, under the presence of unknown local observations and disturbances.
- We analyze the close-loop behavior of our algorithm in a linear time-invariant special case and provide theoretical guarantees on the resulting performance. Our analysis provides an error bound that is independent of the amount of the delays.
- We implement the proposed algorithm, experimentally evaluate our algorithm on 4 multi-agent tasks, and demonstrate the practical usefulness of our approach.

II. Related Work

a) Synthesis Techniques for Multi-Robot Systems: There has been a long line of work on synthesizing reactive controllers from temporal logic specifications for multi-robot systems [14], [15]. These approaches typically create a discrete abstraction of the system and synthesize hybrid controllers that fulfill the logical specifications. In contrast, we focus on synthesizing continuous distributed controllers from a centralized controller specification.

b) Model Predictive Control (MPC): MPC has found use in several domains that are related to this work, including controlling distributed multi-agent systems [23] and

[1]Computer Science Department, University of Texas at Austin, USA. {jiayi, tongrui, swarat, isil, joydeepb}@cs.utexas.edu

distributed systems with time delays [18]. Recently, MPC has also been applied to robotics applications such as trajectory tracking [13], vehicle control [5], flight control [2], and cooperative landing [22]. Unlike many prior approaches where MPC is used to directly optimize a global performance objective defined across multiple agents, we use MPC mainly as a local control planning strategy to reduce the discrepancy between each agent's own trajectory and the corresponding (centrally predicted) ideal fleet trajectory.

c) Distributed Control Designs and Applications: In recent control and robotics literature, many specialized distributed controllers have been proposed for various application domains such as coordinated trajectory tracking and path following [1], [11], vehicle formation control [10], traffic control [24], and information consensus [25], [26]. In this work, we take a different approach and instead aim at reducing the future effort required to develop these distributed systems using a general controller synthesis framework applicable to different robotic applications.

d) Centralized Formation Control: Lastly, there are many prior works on multi-vehicle formation control using a centralized control law [17], [7]; some also deal with the challenging case of nonholonomic robots [9], [6]. In our 2D formation experiments, we employ a simple centralized control scheme based on reference point tracking [8] and rotational repulsive forces [7].

III. Problem Definition

Formally, our goal is to synthesize distributed robotic controllers from a given centralized controller, dynamics model, and observation model—subject to sensor noise, external disturbances and communication, actuation, and observation delays—such that the joint behaviors of the synthesized controllers approximate that of the centralized controller. For a setting with N robots, the inputs to our problem are:

(a) $\hat{f}_i$, the discrete-time dynamics models of robot i, for $i \in \{1 \dots N\}$, given in the form

$$x_i(t+1) = \hat{f}_i(x_i(t), u_i(t), t)\,, \tag{1}$$

where $t \in \mathbb{N}$ is the time index, $x_i(t) \in \mathbb{R}^{n_i^x}$ and $u_i(t) \in \mathbb{R}^{n_i^u}$ is robot i's state and actuation vector at time step t, respectively. Note that $\hat{f}_i$ can be different from the true dynamics f_i, with the difference being modeled as external disturbance. $\hat{f}_i$ may be provided either as an analytic kinematic function, or a learned kino-dynamic function [27].

(b) $\hat{h}_i$, the local observation model of robot i, for $i \in \{1 \dots N\}$, given in the form

$$z_i(t+1) = \hat{h}_i(z_i(t), t)\,, \tag{2}$$

where $z_i \in \mathbb{R}^{n_i^z}$ is the observation of robot i.[1] Similarly, $\hat{h}$ can be different from h, resulting in unexpected observations.

[1]Note that here we require observations to be defined as some state-independent quantities. For example, instead of using the distance to an obstacle as z (which depends on the position of the robot) we can define z as the obstacle's position (whose true value does not depend on where we perform the measurement).

(c) π^c, the centralized controller of the form

$$u(t) = \pi^c(x(t), z(t), t)\,, \tag{3}$$

where $x(t) = [x_1(t), \dots, x_N(t)]^\intercal \in \mathbb{R}^{Nn^x}$ is the global state vector formed by vertically concatenate all robot states, and similarly, $z(t) = [z_1(t), \dots, z_N(t)]^\intercal \in \mathbb{R}^{Nn^z}$ and $u(t) = [u_1(t), \dots, u_N(t)]^\intercal \in \mathbb{R}^{Nn^u}$.

(d) $T^x, T^u, T^c \in \mathbb{N}^+$, the discrete-time observation, actuation, and communication delay of this robotic fleet.

From the inputs given above, we want to synthesize N distributed controllers $\pi_i^d,\ \forall i \in \{1 \dots N\}$ of the form

$$u_i(t + T^u) = \pi_i^d(\mathcal{X}_i(t), \mathcal{Z}_i(t), \mathcal{U}_i(t), t) \tag{4}$$

where $\mathcal{X}_i(t), \mathcal{Z}_i(t), \mathcal{U}_i(t)$ denote the parts of the entire fleet's state history, observation history, and actuation history that are available to agent i at time t, subject to constraints imposed by the delays. For example, we have $\mathcal{X}_i(t) = \{x_i(\tau) | \tau \le t - T^x\} \cup \{x_j(\tau) | j \ne i, \tau \le t - T^x - T^c\}$.

Since our goal is to make the distributed agents behave like they were controlled by the centralized controller π^c, we need to formally define a loss that measures the distance between π^d and π^c. In this work, we have considered two different ways to define such a loss.

Option 1: Action Loss Since our goal is to make every agent take actions similar to the ones given by the centralized controller, an intuitive way to define such a loss is as

$$\ell_{\text{act}}(t) = \|\pi^c(x(t), z(t), t) - u(t)\|\,,$$

which simply measures the difference between the actuation output by the centralized controller (given the current state and observation) and the actual actuation $u(t)$ output by the distributed controllers.

Minimizing this loss requires each agent to accurately predict the current state $x(t)$ and observation $z(t)$ of the entire fleet, such that we can define the output of π_i^d as $\pi_i^c(\hat{x}^{(i)}(t), \hat{z}^{(i)}(t), t)$, where $\hat{x}^{(i)}(t) \approx x(t), \hat{z}^{(i)}(t) \approx z(t)$ are the ith agent's prediction of the current fleet state and observation.

However, predicting $\hat{x}$ and $\hat{z}$ in the closed-loop dynamics can lead to an infinite recursion. To see this, note that each agent's actuation depends on its prediction of other agent's states since

$$\forall i,\ u_i(t) = \pi_i^c(\hat{x}^{(i)}(t), \hat{z}^{(i)}(t), t)\,.$$

But for agent i to predict agent j's state, it will further need to predict j's action at the previous time step $t' = t - 1$:

$$\forall i, j,\ \hat{x}_j^{(i)}(t) = \hat{f}_j(\hat{x}_j^{(i)}(t'), u_j^{(i)}(t'), t')\,.$$

But this in turn requires predicting how agent j would have predicted other agent's states:

$$\forall i, j,\ u_j^{(i)}(t') = \pi_j^c(\hat{x}^{(i,j)}(t'), \hat{z}^{(i,j)}(t), t')\,,$$

where the notation $\hat{x}^{(i,j)}(t')$ denotes agent i's prediction (made at t) of agent j's prediction (made at t') of the fleet state at time t'. Therefore, we can keep unrolling the right hand side, resulting in an infinite recursion.

Option 2: Regret Loss In this formulation, instead of requiring each agent to predict the current states of other agent, we introduce the notion of an *ideal fleet trajectory* that—although not locally computable by each agent in real time—will always become computable later once more information become available through communication. We then define the loss as the difference between the actual fleet trajectory (x, u) and this ideal fleet trajectory (x^*, u^*):

$$\ell_{\text{reg}}(t) = \|x(t) - x^*(t)\|_{Q_x} + \|u(t) - u^*(t)\|_{Q_u} \quad (5)$$

for some positive definite Q_u and positive semi-definite Q_x, with the notation $\|a\|_B = a^\intercal B a$ denoting the quadratic norm of a defined by matrix B.

Assuming the true observation dynamics $h(z,t) = \hat{h}(z,t) + \delta z(t)$, and the true system dynamics $f(x,u,t) = \hat{f}(x,u,t) + \delta x(t)$, where δz and δx are the observation and state disturbance (both unknown a priori to us), we define the ideal fleet trajectory x^* and u^* as the solution to the closed-loop dynamics obtained by combining equation (1)–(3), with $\hat{f}$ and $\hat{h}$ replaced by f and h:

$$\begin{aligned} x_i^*(t+1) &= f_i(x_i^*(t), u_i^*(t), t)\,, \\ u^*(t) &= \pi^c(x^*(t), z^*(t), t)\,, \\ z_i^*(t+1) &= h_i(z_i^*(t), t)\,. \end{aligned} \quad (6)$$

Note that since δx_i and δz_i can be measured at time $t+1$ from the observed state and actuation (assuming no observation noise) using eq. (1) and (2) as

$$\begin{aligned} \delta x_i(t) &= x_i(t+1) - \hat{f}_i(x_i(t), u_i(t), t))\,, \\ \delta z_i(t) &= z_i(t+1) - \hat{h}_i(z_i(t), t)\,, \end{aligned}$$

each agent will eventually be able to locally compute the same ideal fleet trajectory once other agents' δx and δz become available through communication. However, because communication takes time, the most recent part of the ideal fleet trajectory will have to be predicted initially and revised later, hence loss (5) in general cannot be zero and is caused by the "regret" of each agent's past predictions.

Since minimizing this loss only requires each agent to predict the ideal fleet trajectory, which is considerably easier than predicting the actual fleet trajectory, we use this second loss definition in this work.

IV. OneVision Overview

At a high level, our algorithm makes control decisions based on three main steps: *forward prediction*, *self state estimation*, and *local planning*. In forward prediction, each agent tries to locally compute an *estimated ideal fleet trajectory* using all the information currently available to itself. As we will show in Section V, an important property of this step is that despite agents only having access to limited state information about other agents, the differences between these predicted fleet trajectories computed by different agents do not accumulate over time. Then, in self state estimation, each agent uses its locally recorded actuation history and observed past state to predict its future state at $\tau + T^u$. Lastly, in local planning, each agent tries to plan its next action

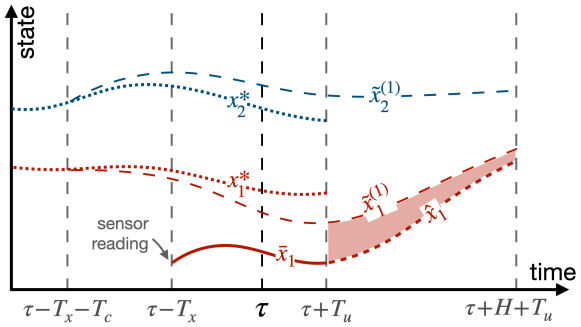

Fig. 1: **How OneVision works from the perspective of agent 1.** On the time axis, T^x, T^u, and T^c denote state, actuation, and communication delay, respectively. τ is the current time step, and H is the prediction horizon. The two dotted lines, x_1^* and x_2^*, represent the ideal fleet trajectories, while $\tilde{x}_1^{(1)}$ and $\tilde{x}_2^{(1)}$ represent the most probable ideal fleet trajectory predicted by agent 1. The solid line $\bar{x}_1$ represents the self estimation computed using agent 1's recorded action history. Line $\hat{x}_1$ represents a planned future trajectory obtained by minimizing the discrepancy between $\hat{x}_1$ and $\tilde{x}_2^{(1)}$ (shown as the colored area.)

using model predictive control, with the goal of minimizing the discrepancy between its predicted future trajectory and the corresponding part in the estimated ideal fleet trajectory. These three steps are illustrated in Figure 1.

Let τ be the current time step, we now describe each of these three steps in more details:

1. Forward Prediction. Every robot i uses the newest information available to itself (as defined in eq. (4)) to forward-predict the most probable future ideal fleet trajectory $\tilde{x}^{(i)}$ by solving the following initial value problem for the time span $\tau - T^x - T^c - 1 \le t < \tau + T^u + H$, where H is the prediction horizon:

$$\begin{aligned} \tilde{x}_j^{(i)}(t+1) &= \hat{f}_j(\tilde{x}_j^{(i)}(t), \tilde{u}_j^{(i)}(t), t) + \delta\tilde{x}_j^{(i)}\,, \\ \tilde{u}^{(i)}(t) &= \pi^c(\tilde{x}^{(i)}(t), \tilde{z}^{(i)}(t), t)\,, \\ \tilde{z}_j^{(i)}(t+1) &= \hat{h}_j(\tilde{x}_j^{(i)}(t), \tilde{z}_j^{(i)}(t), t) + \delta\tilde{z}_j^{(i)}\,, \end{aligned} \quad (7)$$

for $j \in \{1 \dots N\}$. Let $\tau_{\text{init}} = \tau - T^x - T^c - 1$, the above problem is subject to the initial conditions

$$\tilde{x}_j^{(i)}(\tau_{\text{init}}|\tau) = \tilde{x}_j^{(i)}(\tau_{\text{init}}|\tau - 1)\,, \quad (8)$$

where the notation $\tilde{f}_j^{(i)}(t|\tau)$ means "the prediction of $\tilde{f}_j(t)$ made by robot i at time τ". The disturbance terms $\delta\tilde{x}^{(i)}$ and $\delta\tilde{z}^{(i)}$ are defined to be zero unless the corresponding trajectory information is available to robot i, i.e.,

$$\delta\tilde{x}_j^{(i)}, \delta\tilde{z}_j^{(i)} = \begin{cases} \delta x_j, \delta z_j & j = i \text{ and } \tau_{\text{init}} \le t < \tau - T^x \\ & \text{or } j \ne i \text{ and } t = \tau_{\text{init}}\,, \\ 0, 0 & \text{otherwise.} \end{cases} \quad (9)$$

2. Self State Estimation. Every robot i then estimates its actual state at time $\tau + T^u$ using its actuation history u_i by solving the following initial value problem for $\tau - T^x \le t < \tau + T^u$:

$$\bar{x}_i(t+1) = \hat{f}(\bar{x}_i(t), u_i(t), t) \quad (10)$$

subject to the initial condition

$$\bar{x}_i(\tau - T^x) = x_i(\tau - T^x)\,. \tag{11}$$

This gives the self state estimation $\bar{x}_i(\tau + T^u)$, which will be used in the next step.

3. Local Planning. Every robot i then uses the predicted $\tilde{x}^{(i)}$ and $\tilde{u}^{(i)}$ from step 1 as the *most probable approximation* to the ideal fleet trajectory x^* and u^* and tries to locally minimize its future regret by solving the following optimization problem for the control time span $\tau+T^u \le t < \tau+T^u+H$:

$$\underset{\{\hat{u}_i(t)|t\}}{\text{Minimize}} \sum_{t=\tau+T^u}^{\tau+T^u+H-1} \ell_i(t)\,, \tag{12}$$

where

$$\ell_i(t) = \|\hat{x}_i(t) - \tilde{x}_i^{(i)}(t)\|_{Q_x} + \|\hat{u}_i(t) - \tilde{u}_i^{(i)}(t)\|_{Q_u}\,,$$

subject to the initial condition

$$\hat{x}_i(t) = \bar{x}_i(t)\,, \text{ for } t = \tau + T^u \tag{13}$$

and the dynamics constraints

$$\hat{x}_i(t+1) = \hat{f}(\hat{x}_i(t), \hat{u}_i(t), t)\,, \tag{14}$$

for $\tau + T^u \le t < \tau + T^u + H$. Robot i then takes the first actuation from the optimal solution as its next actuation, i.e., we have $u_i(\tau + T^u) = \hat{u}_i(\tau + T^u|\tau)$.

Initialization OneVision needs an initial history of x, z, and u to start with. In our implementation, we assume the initial condition of the entire fleet is available to all robots during initialization, and we simply initialize the history trajectories as constant functions whose value equals the corresponding initial condition[2].

Remark (algorithm scalability)**.** The running time of the OneVision algorithm is $\mathcal{O}(N)$ for each agent due to the forward prediction step (whereas both self state estimation and local planning take constant time w.r.t. N). Hence, this work mainly targets middle-ground applications where the number of coordinating agents are not too large. For larger applications, applying distributed control algorithms or exploiting some form of sparsity structure will be necessary, which we leave as future work.

V. Theoretical Analysis

In this section, we analyze our algorithm's closed-loop behavior in the special case of linear time-invariant (LTI) dynamics and controller. Our main goal is to answer the following two questions: 1) When our model error δx and δz is zero, how fast does the true fleet trajectory x converge to the ideal fleet trajectory x^*? 2) When δx and δz is small, does x stays close to x^*? We will answer these questions in Proposition V.5 at the end of this section, stated in the form of input-to-state stability.

[2]Our algorithm is not sensitive to the errors introduced during initialization as long as the initialization guarantees that all robots make the same forward prediction $\tilde{x}^{(i)}$ initially.

Lemma V.1 (Inital condition of forward prediction)**.** *Initial condition* (8) *in the forward prediction step initializes $\tilde{x}^{(i)}$ to the ideal fleet state x^* (as shown in Figure 1). i.e.,*

$$\tilde{x}_j^{(i)}(\tau_{init}|\tau) = x_j^*(\tau_{init}),\ \forall j,\ \forall \tau \ge 0\,.$$

Proof. We prove this by induction. The base case at $\tau = 0$, $\tilde{x}_j^{(i)}(\tau_{\text{init}}|\tau) = x_j^*(\tau_{\text{init}})$ holds because of the initialization step. For the inductive case, assume $\tilde{x}_j^{(i)}(\tau_{\text{init}}|\tau) = x_j^*(\tau_{\text{init}})$ at time τ, we aim to show that $\tilde{x}_j^{(i)}(\tau_{\text{init}}+1|\tau+1) = x_j^*(\tau_{\text{init}}+1)$. Compare the dynamics (6) of x^* with the dynamics (7) of $\tilde{x}^{(i)}$, we see that they become identical if $\delta\tilde{x}^{(i)}(t) = \delta x(t)$ and $\delta\tilde{z}^{(i)}(t) = \delta z(t)$. And from the history constraints (9), we see that this is indeed the case at $t = \tau_{\text{init}}$; hence, we have $\tilde{x}_j^{(i)}(\tau_{\text{init}}+1|\tau) = x_j^*(\tau_{\text{init}}+1)$. Finally, apply the initial condition (8), we have $\tilde{x}_j^{(i)}(\tau_{\text{init}}+1|\tau+1) = x_j^*(\tau_{\text{init}}+1)$. ■

Next, we introduce the following LTI assumptions about the system dynamics and controllers.[3]

Assumption 1. For $t \ge 0$, all robots have the LTI dynamics of the form

$$\hat{f}_i(x_i, u_i, t) = A_i x_i + B_i u_i + \hat{w}_i(t)\,, \tag{15}$$
$$f_i(x_i, u_i, t) = A_i x_i + B_i u_i + w_i^*(t)\,, \tag{16}$$

in which A_i and B_i are constant matrices, and the time-dependent terms satisfy $w_i^*(t) - \hat{w}_i(t) = \delta x(t)$. We also assume that norm $|\lambda_k| \le 1$ for all eigenvalues λ_k of A_i, i.e., the system dynamics is not exponentially unstable.

Assumption 2. For $t \ge 0$, the observation dynamics have LTI dynamics of the form

$$\hat{h}(z_i, t) = C_i z_i + \hat{\mu}_i(t)\,, \tag{17}$$
$$h(z_i, t) = C_i z_i + \mu_i^*(t)\,, \tag{18}$$

in which C_i has all its eigenvalues λ_k's norm $|\lambda_k| \le 1$, and $\mu^*(t) - \hat{\mu}_i(t) = \delta z(t)$.

Assumption 3. For $t \ge 0$, the centralized controller π^c is also LTI w.r.t. x and z and has the form

$$\pi^c(x, z, t) = -K_x x + K_z z + v(t)\,,$$

and K_x stabilizes the closed-loop dynamics, i.e., $|\lambda_k| < 1$ for all eigenvalues λ_k of the matrix $A - BK_x$. Here, $A = \text{diag}\{A_i|\,\forall i\}$ and $B = \text{diag}\{B_i|\,\forall i\}$ are the block-diagonal matrices of the overall system.

We next prove the following error bounds of the forward prediction and self state estimation step.

Lemma V.2 (Self estimation error)**.** *Let $\Delta\bar{x}_i = \bar{x}_i - x_i$, $\Delta\hat{w}_i = \hat{w}_i - w_i^*$. At any given time $\tau \ge 0$, we have*

$$\|\Delta\bar{x}_i(\tau + T^u)\| \le \beta_i(T^x + T^u)\,\lceil\Delta\hat{w}_i\rceil\,,$$

[3]For mildly nonlinear systems, the following assumptions can hold temporarily by linearizing the system behavior around the current operating point.

where β_i is a positive definite polynomial of order m_i (when A_i is stable, β_i reduces to a constant), m_i depends on the number of unit-norm eigenvalues of A_i, and

$$\lceil \Delta\hat{w}_i \rceil = \max_{-T^x \le t \le T^u} \|\Delta\hat{w}_i(\tau + t)\|$$

is the maximal norm of $\Delta\hat{w}_i$ between $\tau - T^x$ and $\tau + T^u$.

Proof. Using (1), (10), we can write down the error dynamics as

$$\Delta\bar{x}_i(t+1) = A_i\Delta\bar{x}_i(t) + \Delta\hat{w}_i(t) \tag{19}$$

with the initial condition (due to (11))

$$\Delta\bar{x}_i(\tau - T^x) = 0\,.$$

Solving this linear system gives us the value of $\Delta\bar{x}$ at $\tau + T^u$:

$$\Delta\bar{x}_i(\tau + T^u) = \sum_{t=1}^{T^x+T^u} (A_i)^t \Delta\hat{w}_i(\tau + T^u - t)\,.$$

Since A_i contains no eigenvalues whose norm is greater than 1, (Assumption 1), we can bound $\|(A_i)^t\|$ by a polynomial of order m_i' on t, where m_i' is the number of unit-norm eigenvalues of A_i whose algebraic multiplicity $\neq$ geometric multiplicity. Also bound $\|\Delta\hat{w}_i(t)\|$ by $\lceil \Delta\hat{w}_i \rceil$, we arrive at the conclusion by setting $m_i = m_i' + 1$. ∎

Similarly, we now provide a bound for the prediction error $\Delta\tilde{x}^{(i)}(t|\tau) = \tilde{x}^{(i)}(t|\tau) - x^*(t)$ and $\Delta\tilde{z}^{(i)}(t|\tau) = \tilde{z}^{(i)}(t|\tau) - z^*(t)$.

Lemma V.3 (Forward prediction error)**.** *At any time $\tau \ge 0$, we have*

$$\|\Delta\tilde{z}^{(i)}(\tau + T^u|\tau)\| \le \gamma_i(T^{all})\lceil \Delta\hat{\mu} \rceil\,, \tag{20}$$

$$\|\Delta\tilde{x}^{(i)}(\tau + T^u|\tau)\| \le a_i\lceil \Delta\hat{\mu} \rceil + b\lceil \Delta\hat{w} \rceil\,. \tag{21}$$

where $T^{all} = T^x + T^u + T^c$. γ_i is a polynomial of order n_i that depends on the eigenvalues of C_i's, and a_i, b are constants that depends on A, B, K_x, K_z, and C. $\Delta\hat{\mu} = \hat{\mu} - \mu^*$ is the prediction model error, and $\lceil \Delta\hat{\mu} \rceil$ and $\lceil \Delta\hat{w} \rceil$ are the corresponding maximal error norms between $\tau - T^x - T^c$ and $\tau + T^u$.

Proof. Take the difference of the dynamics of $\tilde{z}_i^{(i)}$ and z_i^* using (17), we can write down the error dynamics of $\Delta\tilde{z}_i^{(i)}(t)$ as

$$\Delta\tilde{z}_j^{(i)}(t+1) = C_j\Delta\tilde{z}_j^{(i)}(t) + \Delta\mu_j(t)$$

with the initial condition (which can be obtained from the history constraints (9))

$$\begin{cases} \Delta\tilde{z}_i^{(i)}(\tau_{\text{init}} + T^x) = 0\,, & j = i \\ \Delta\tilde{z}_i^{(i)}(\tau_{\text{init}}) = 0\,, & j \neq i\,. \end{cases}$$

And since C_i has no eigenvalues whose norm is greater than 1, similar to the argument in the proof of Lemma V.2, $\Delta\tilde{z}_i^{(i)}$ only grows at most at polynomial speed, hence we have

$$\|\Delta\tilde{z}^{(i)}(\tau + T^u)\| \le \gamma_i'(T^{\text{all}} - T^x)\lceil \Delta\hat{\mu}_i \rceil + \sum_{j \neq i} \gamma_j'(T^{\text{all}})\lceil \Delta\hat{\mu}_j \rceil\,.$$

We can then bound the right-hand-side with $\gamma_i(T^{\text{all}})\lceil \Delta\hat{\mu} \rceil$ and arrive at (20).

Now using the linear form of π^c given by Assumption 3, we can also write down the dynamics of $\Delta\tilde{x}^{(i)}$ as

$$\begin{aligned}\Delta\tilde{x}^{(i)}(t+1) = &(A - BK_x)\Delta\tilde{x}^{(i)}(t) \\ &+ BK_z\Delta\tilde{z}^{(i)}(t) + \Delta\tilde{w}_j^{(i)}(t)\end{aligned}$$

with the initial condition (which holds by Lemma V.1)

$$\Delta\tilde{x}^{(i)}(\tau_{\text{init}}) = 0\,.$$

Since $|\lambda_k| < 1$ for all eigenvalues λ_k of $A - BK_x$ (Assumption 3), and $\Delta\tilde{z}^{(i)}$ grows at most at polynomial speed, we can bound $\Delta\tilde{x}^{(i)}(\tau + T^u|\tau)$ using (21). ∎

To simplify the analysis of local planning, we also assume that the prediction horizon H is very long such that the resulting optimal action is linear feedback.

Assumption 4. The prediction horizon $H \to \infty$.

Lemma V.4 (Local planning provides linear feedback)**.** *The optimal actuation given by the local planning step has the form*

$$u_i(t) = \tilde{u}_i^{(i)}(t|t - T^u) - K_i^L(\bar{x}_i(t) - \tilde{x}_i^{(i)}(t|t - T^u))\,,$$

where K_i^L is some constant matrix that stabilizes the system.

Proof. Take the difference between (14) and (7), and notice that there are no history constraints during $t \ge \tau + T^u$, we obtain

$$\begin{aligned}\hat{x}_i(t+1) - \tilde{x}_i^{(i)}(t+1) = &\,A_i\left(\hat{x}_i(t) - \tilde{x}_i^{(i)}(t|\tau)\right) \\ &+ B_i\left(\hat{u}_i(t) - \tilde{u}_i^{(i)}(t|\tau)\right)\end{aligned}$$

for $t \ge \tau + T^u$, which—when combined with the objective (12) and the assumption $H \approx \infty$—matches the form of a linear quadratic regulator (LQR) problem. Hence, the optimal solution is a linear feedback law given by

$$\hat{u}_i(t) - \tilde{u}_i^{(i)}(t|\tau) = -K_i^L(\hat{x}_i(t|\tau) - \tilde{x}_i^{(i)}(t|\tau))\,,$$

where the gain K_i^L stabilizes the system and can be obtained by solving the discrete-time algebraic Riccati equation [16].

Take $t = \tau + T^u$ in the above, and notice that $\hat{x}_i(\tau + T^u|\tau) = \bar{x}_i(\tau + T^u)$ (equation (13)), we arrive at the conclusion. ∎

We are now ready to prove our main result.

Theorem V.5 (closed-loop stability)**.** *Let $\Delta x = x - x^*$ be the distance between the actual fleet trajectory and the ideal fleet trajectory, we have the following bound*

$$\|\Delta x(t)\| \le c_1 e^{-\lambda t}\|\Delta x(0)\| + d_1\lceil \Delta\hat{w} \rceil + d_2\lceil \Delta\hat{\mu} \rceil\,, \tag{22}$$

where c_1, λ, d_1, and d_2 are all constants independent of the delays, and the maximal norm $\lceil \cdot \rceil$ is defined on the interval $0 \le t < \infty$.

Proof. We have the actual closed-loop dynamics

$$x(t+1) = Ax(t) + Bu(t) + w^*(t)\,,$$

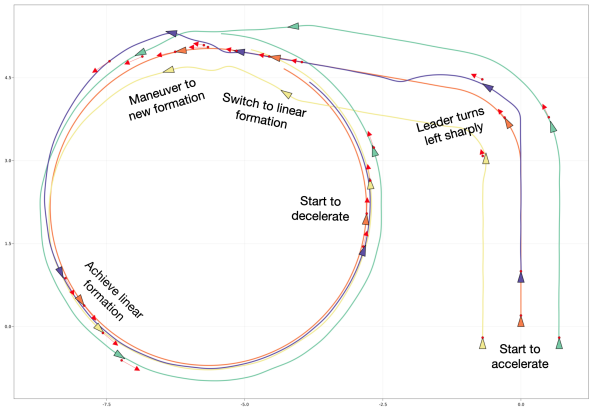

Fig. 2: **Simulation of 2D Formation Switching running OneVision**. The orange car is the formation leader and is controlled by external inputs, while the three other cars try to follow the leader. Red dots show formation reference points.

where $u(t)$ is given by Lemma V.4. We also have the ideal dynamics

$$x^*(t+1) = Ax^*(t) + Bu^*(t) + w^*(t)\,.$$

Take the difference, we obtain

$$\Delta x(t+1) = A\Delta x(t) + B\Delta u(t)\,.$$

Expand u and u^*'s definitions, and use the notation $\tilde{f}^{(\cdot)}$ to denote $[\tilde{f}_j^{(j)}|\ \forall j]^\intercal$, we have

$$\begin{aligned}\Delta u &= \tilde{u}^{(\cdot)} - u^* - K^L(\bar{x}(t) - \tilde{x}^{(\cdot)}) \\ &= [(K_z\Delta\tilde{z}^{(j)})_j|\ \forall j]^\intercal - K_x\Delta\tilde{x}^{(\cdot)} \\ &\quad - K^L(\Delta x + \Delta\bar{x} - \Delta\tilde{x}^{(\cdot)})\,,\end{aligned}$$

in which the time arguments are omitted for brevity. Substitute the above into the previous equation, we obtain

$$\Delta x(t+1) = (A - BK^L)\Delta x(t) + B\epsilon(t)\,, \qquad (23)$$

where the disturbance term

$$\epsilon(t) = [(K_z\Delta\tilde{z}^{(j)})_j|\ \forall j]^\intercal - (K_x - K^L)\Delta\tilde{x}^{(\cdot)} - K^L\Delta\bar{x}$$

is clearly bounded by the bound of $\Delta\tilde{z}^{(j)}$, $\Delta\tilde{x}^{(\cdot)}$, and $\Delta\bar{x}$ (given by Lemma V.3 and V.2). Thus, we can write the bound as

$$\|\epsilon(t)\| \le \alpha_1(t)\lceil\Delta\hat{\mu}\rceil + \alpha_2(t)\lceil\Delta\hat{w}\rceil$$

using some polynomial α_1 and α_2 (whose concrete forms are not needed for this proof).

Since $A - BK^L$ has only eigenvalues with norms < 1, and $\|\epsilon(t)\|$ grows at most at polynomial rate, we can bound the solution to (23) with (22). ■

VI. Experimental Evaluation

Implementation Our implementation of OneVision consists of around 2000 lines of Julia code and allows the user to write centralized controllers as ordinary Julia functions. To minimize the regret loss (5), we use the L-BFGS optimizer [4] implemented by the Optim package [20], which employs automatic differentiation to compute gradient information needed by the optimization process. To ensure convergence to the same local optimum between time steps when dealing with nonlinear dynamics, we always feed the solution found in the previous time step as the initial solution to the optimizer in the next time step.

Simulation Tasks We use the following 4 simulation tasks to compare OneVision with several other baseline controllers. We simulate 20 seconds for each task.

1) **1D Leader Linear:** 1-dimensional leader-follower driving task with a linear (PID) controller. The leader's goal is to reach a desired velocity, while the follower tries to stay close to the leader
2) **1D Leader With Obstacle:** 1-dimensional leader-follower task with an obstacle of unknown position. The leader can observe the obstacle when it is within its sensor's range and will brake once the obstacle becomes too close. To control velocity, both robots use bang-bang control instead of linear control.
3) **2D Formation Driving:** 4 car-like robots driving on a 2-dimensional plane. The leader car is assumed to be controlled by external commands (in the form of external observations), while the other four cars follow the leader and try to maintain a circular formation while avoiding collision with each other.
4) **2D Formation Switching:** Similar to the task above, but the leader also controls the formation of the fleet. At some point, the leader will switch the formation from a triangle to a straight line. A simulation result of this task is shown in Figure 2.

Baseline Controller Frameworks We compare OneVision with 3 other baseline controller frameworks:

- **Naive:** This is the simplest controller framework that runs the centralized controller without performing any delay compensation—each agent simply treats their own observations as well as the information broadcasted by other agents as the current state of the fleet.
- **Local:** Under this controller framework, delay compensation is limited to local information. Each agent uses its local actuation history and dynamics model to predict away its state and observation delays.
- **ConstU:** This controller framework employs simple heuristics to perform global compensation by assuming other agent's actuation remains constant during forward prediction. It also performs local compensation using the same strategy as Local.

Task-Sepcific Metrics To quantitively measure the performance on each task, in addition to the loss defined in (5), which is motivated by our proposed algorithm, we also define the following two method-independent, task-specific metrics.

- **Average Distance** (for task 1 and 2): defined as the time-averaged distance between the leader and follower in L2 norm, given by $\sqrt{\frac{1}{T}\int_0^T (p_1 - p_2)^2 dt}$, where p_1 and p_2 are the positions of each car, and T is the length of the simulation.

TABLE I: Simulation Performance (log loss)

Task	Naive	Local	ConstU	OneVision
Leader Linear	1.444	1.169	0.912	**-0.353**
Leader With Obstacle	1.638	1.133	0.587	**-0.349**
Formation Driving	4.183	2.393	2.060	**0.262**
Formation Switching	4.150	2.446	2.150	**0.681**

TABLE II: Simulation Performance (task-specific metrics)

Task	Naive	Local	ConstU	OneVision
Leader Linear	0.101	0.100	0.027	**0.022**
Leader With Obstacle	0.083	0.082	0.027	**0.022**
Formation Driving	1.151	0.272	0.206	**0.204**
Formation Switching	1.234	0.343	0.281	**0.272**

- **Average Deviation** (for task 3 and 4): defined as the time-averaged distance between each follower and their supposed position in the fleet formation, given by $\sqrt{\frac{1}{T(N-1)}\int_0^T \sum_{i=2}^N \|p_i - \hat{p}_i\|^2 dt}$, where $\hat{p}_i$ is the supposed position of follower i.

Default Parameters Unless stated otherwise, we use the following parameters across different tasks: We run all controllers at 20Hz, with communication delay $T^c = 50$ms, observation delay $T^x = 30$ms, and actuation delay $T^u = 40$ms[4]. For sensor noise, we add Gaussian noise to the state vector at every time step, and for external disturbance, we also add Gaussian noise but it is limited only to velocity and steering angle. In both cases, the default noise strength is 0.005 (in SI units) at 100Hz. We set OneVision's prediction horizon to be $H = 20$, which corresponds to a time span of 1 second. By default, we also assume the dynamics models accurately match the true dynamics (excluding noise).

A. Performance under Default Parameters

We run OneVision along with the 3 other baselines under the default parameters and compare their performance in Table I (regret loss) and Table II (task-specific metrics). Each datum is obtained by averaging 100 random runs. OneVision achieves consistently the best performance in all tasks under both performance measurements.

B. Sensitivity Analysis

In this section, we modify the default parameters to study the impact of sensor noise, disturbance, delays, model inaccuracy, and prediction horizon. We vary each of these variables and compare the performance of different controller frameworks on all 4 simulation tasks (all results are measured using 10 random runs). Due to space constraints, we only present a few representative examples in this section and describe general trends we observed for the rest cases.[5]

Sensor Noise (Figure 3 upper left) We modify the sensor noise strength from 0 the 10 times the default strength and found that OneVision still achieves the best performance

[4]In our discrete-time formulation, to handle non-integral delays like 30ms (which is less than 1 time step under 20Hz control), we actually run our framework at 100Hz but only replan actuation at every 5th time step.

[5] Our full results available at `https://git.io/JqJ2x`

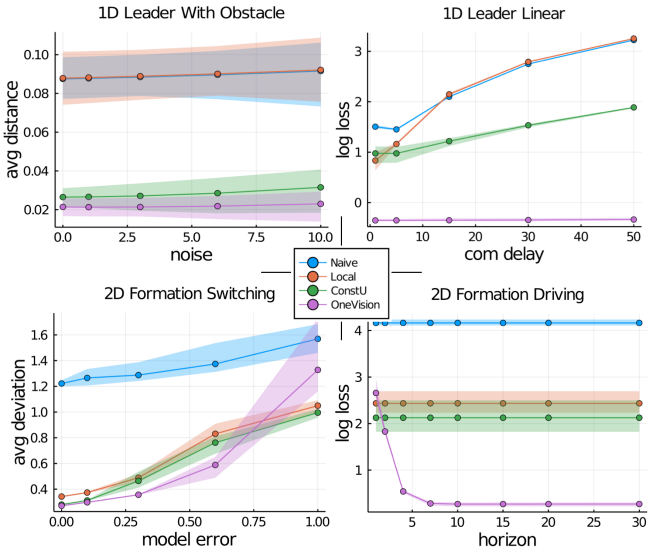

Fig. 3: Representative examples showing impact of sensor noise (upper left), communication delay (upper right), model inaccuracy (lower left), and prediction horizon (lower right).

across this range. We also observe similar trends when varying the amount of communication delay from 10ms to 500ms, both measured in log loss and in average distance/deviation. Particularly, for the 1D Leader Linear task, we notice that OneVision's performance is almost unaffected by the amount of the delay (Figure 3 upper right), confirming the results given by Theorem V.5.

Model Error (Figure 3 lower left)) To study the sensitivity w.r.t. model error, we define model error for task 1 and 2 as $r_1 - 1$, where r_1 is the ratio between the modeled car acceleration and the actual acceleration, and for task 3 and 4 as $r_2 - 1$, where r_2 is the ratio between the modeled car wheelbase and the true wheelbase. By ranging the model error from 0 to 100%, we found that although OneVision still has better performance than other baselines when the error is small ($\leq 60\%$), OneVision is generally more sensitive to model error due to its heavy reliance on forward prediction. We also observe similar trends when varying the amount of external disturbances from 0X to 10X.

Prediction Horizon (Figure 3 lower right)) As an ablation study, we change the prediction horizon H used by OneVision from 1 to 30 (default value is 20). We see that a very short horizon negatively impacts OneVision's performance, suggesting that the local planning step plays an important role, but a horizon longer than 10 (corresponding to 0.5s) does not further improve the performance.

C. Real-World Experiments

We implemented two real-world versions of the simulation tasks on the UT AUTOmata, a fleet of scale 1/10 autonomous cars. All sensing and computation is performed on-board—the cars are equipped with 2D LIDAR for sensing, and an Nvidia Jetson TX2 for computation. Each car runs Episodic non-Markov Localization [3] using observations from the LIDAR to estimate their pose in the world, and

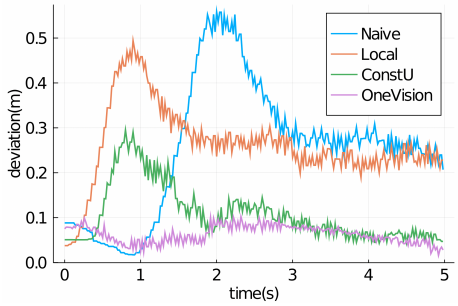

Fig. 4: Deviation over time in real-world experiment.

communicates to the other cars via WiFi. To ensure that the noise margin of localization is lower than the errors in distributed control stemming from delays, and to account for variability in transmission, the communication queue performs per-message throttling to ensures that all messages have a constant delay of 300ms for all controllers. We run the controllers at 50Hz and use the estimated delay parameters $T^x = 40$ms and $T^u = 80$ms. We summarize our results below and highlight the major differences from the simulation tasks.

1D Leader with Obstacle (quantitative study) We modify our 2D reference point tracking controller to fix the leader's reference point to be always on a straight line. We also use 3 cars instead of 2 and maintain a triangular formation. For each controller framework, we run the experiment 5 times and report the average deviation below.

Naive	Local	ConstU	OneVision
0.396	0.407	0.239	**0.192**

We also plot how deviation changes over time in Figure 4, which matches the expected trends and shows OneVision's superior performance compared to other baselines.

2D Formation Switching (qualitative study) We set up the formation so that the leader was tele-operated by one of the authors, while the other two cars followed the leader in formation. Upon command from the leader, the followers had to switch formations while performed obstacle avoidance. We observed robust performance and were able to successfully finish the task using OneVision. The followers were able to quickly converge back to their target formation in response to variation in the leader's action. We recorded an example execution as part of our supplementary video.

Acknowledgements

This work is supported in part by NSF awards CCF-2006404, CCF-1704883, and CAREER-2046955. We would also like to thank the IROS reviewers for their helpful comments and suggestions.

References

[1] A. P. Aguiar and A. M. Pascoal. Coordinated path-following control for nonlinear systems with logic-based communication. In *46th IEEE Conference on Decision and Control*, pages 1473–1479. IEEE, 2007.

[2] K. Alexis, C. Papachristos, R. Siegwart, and A. Tzes. Robust model predictive flight control of unmanned rotorcrafts. *Journal of Intelligent & Robotic Systems*, 81(3):443–469, 2016.

[3] J. Biswas and M. M. Veloso. Episodic non-markov localization. *Robotics and Autonomous Systems*, 87:162 – 176, 2017.

[4] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5):1190–1208, 1995.

[5] S. D. Cairano, D. Bernardini, A. Bemporad, and I. V. Kolmanovsky. Stochastic MPC with learning for driver-predictive vehicle control and its application to HEV energy management. *IEEE Transactions on Control Systems Technology*, 22(3):1018–1031, 2014.

[6] L. Consolini, F. Morbidi, D. Prattichizzo, and M. Tosques. Leader–follower formation control of nonholonomic mobile robots with input constraints. *Automatica*, 44(5):1343–1349, 2008.

[7] A. D. Dang, H. M. La, T. Nguyen, and J. Horn. Formation control for autonomous robots with collision and obstacle avoidance using a rotational and repulsive force–based approach. *International Journal of Advanced Robotic Systems*, 16(3), 2019.

[8] Danwei Wang and Guangyan Xu. Full-state tracking and internal dynamics of nonholonomic wheeled mobile robots. *IEEE/ASME Transactions on Mechatronics*, 8(2):203–214, 2003.

[9] J. P. Desai, J. P. Ostrowski, and V. Kumar. Modeling and control of formations of nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 17(6):905–908, 2001.

[10] W. B. Dunbar and R. M. Murray. Distributed receding horizon control for multi-vehicle formation stabilization. *Automatica*, 2006.

[11] R. Ghabcheloo, A. P. Aguiar, A. Pascoal, C. Silvestre, I. Kaminer, and J. Hespanha. Coordinated path-following in the presence of communication losses and time delays. *SIAM Journal on Control and Optimization*, 48(1):234–265, 2009.

[12] J. Ghommam and F. Mnif. Coordinated path-following control for a group of underactuated surface vessels. *IEEE Transactions on Industrial Electronics*, 56(10):3951–3963, 2009.

[13] M. Kamel, M. Burri, and R. Siegwart. Linear vs nonlinear MPC for trajectory tracking applied to rotary wing micro aerial vehicles. *arXiv:1611.09240 [cs]*, 2017.

[14] M. Kloetzer and C. Belta. Automatic deployment of distributed teams of robots from temporal logic motion specifications. *IEEE Transactions on Robotics*, 26(1):48–61, 2009.

[15] M. Kloetzer, X. C. Ding, and C. Belta. Multi-robot deployment from ltl specifications with reduced communication. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 4867–4872. IEEE, 2011.

[16] P. Lancaster and L. Rodman. *Algebraic riccati equations*. Clarendon press, 1995.

[17] N. Leonard and E. Fiorelli. Virtual leaders, artificial potentials and coordinated control of groups. In *Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No.01CH37228)*, volume 3, pages 2968–2973. IEEE, 2001.

[18] H. Li and Y. Shi. Distributed model predictive control of constrained nonlinear systems with communication delays. *Systems & Control Letters*, 62(10):819–826, 2013.

[19] F. Liao, R. Teo, J. L. Wang, X. Dong, F. Lin, and K. Peng. Distributed formation and reconfiguration control of VTOL UAVs. *IEEE Transactions on Control Systems Technology*, 25(1):270–277, 2017.

[20] P. K. Mogensen and A. N. Riseth. Optim: A mathematical optimization package for Julia. *Journal of Open Source Software*, 3(24):615, 2018.

[21] R. Olfati-Saber and R. M. Murray. Distributed cooperative control of multiple vehicle formations using structural potential functions. *IFAC Proceedings Volumes*, 35(1):495–500, 2002.

[22] L. Persson, B. Wahlberg, T. A. Johansen, KTH, and Skolan för elektroteknik och datavetenskap (EECS). *Autonomous and Cooperative Landings Using Model Predictive Control*. 2019.

[23] A. Richards and J. How. A decentralized algorithm for robust constrained model predictive control. In *Proceedings of the 2004 American Control Conference*, 2004.

[24] T. Tettamanti and I. Varga. Distributed traffic control system based on model predictive control. *Periodica Polytechnica Civil Engineering*, 54(1):3–9, 2010.

[25] E. M. A. Wei Ren, Randal W. Beard. Information consensus in multivehicle cooperative control. *IEEE Control Systems*, 27(2):71–82.

[26] G. Wen, Z. Duan, W. Yu, and G. Chen. Consensus in multi-agent systems with communication constraints. *International Journal of Robust and Nonlinear Control*, 22(2):170–182, 2012.

[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 (RA-L)*, pages 1–7, 2021.