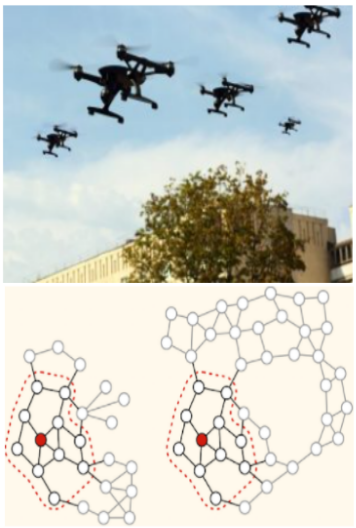
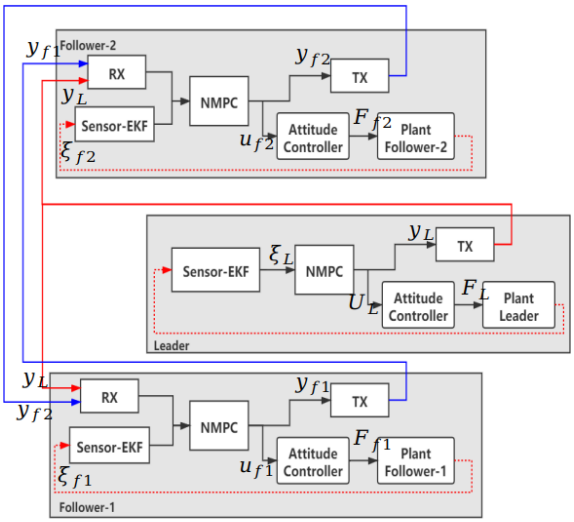
Spring semester 2021, SMT-ROB, DISAL-SP150	Start:	22.02.2021
	Finish:	11.06.2021

Distributed Model Predictive Control Architectures for Multi-Rotor Micro Aerial Vehicles (MAVâs)

Jianhao Zheng





Professor:	Alcherio Martinoli
Assistant:	Kagan Erunsal

This page intentionally left blank.

Contents

Chapter 1 Introduction

Multi-rotor Micro Aerial Vehicles (MAVs) are drawing growing attention due to their agility and ability to perform tasks that humans are unable to do such as infrastructure inspection, search, rescue operations and resources exploration [1]. Nowadays, one of the most popular topics is the cooperation strategies of multiple MAVs to perform complex missions. Among them, the formation control is a basis for these tasks and one efficient method to carry out them is to adopt a Model Predictive Control (MPC) paradigm.

Centralized, Decentralized and Distributed MPC are the three major types of MPC architectures commonly utilized for formation control [2]. Compared to Centralized MPC, Decentralized or Distributed strategy can have inherent advantages such as additional flexibility, robustness and expandability [3]. There are many successful implementations of Decentralized MPC for the formations of multi-agent systems. Among those, the leader-follower formation control by Erunsal et al. [4], dynamic encirclement by Marasco et al. [5] and formation reconfiguration by Chevet et al. [6] can be considered.

Since the Distributed MPC leverages communication, there are many possible network topologies and architectures to be considered. Dunbar and Murray proposed a distributed implementation of receding horizon control based-on inputs sharing and proved the stability and continuous feasibility [7]. Liu et al. introduced a Lyapunov-based iterative Distributed MPC algorithm for coupled nonlinear systems [8]. Van Parys and Goele Pipeleers presented a novel Distributed MPC strategy for formation control of multi-vehicle systems and decomposed the system by Alternating Direction Method of Multipliers (ADMM) [9]. Moreover, Hou et al. provided a Distributed MPC framework for building control applications [10]. Their framework utilizes Proximal Jacobian ADMM to decompose the coupled constraints and objective function. In Lee et al. [11], a novel dynamic cooperatively coevolving particle swarm optimization (CCPSO) based Distributed MPC was proposed to control a multirobot formation.

The main goal of this project is to compare the effectiveness of different Distributed MPC architectures and a Decentralized MPC schemes proposed by Erunsal et al. [4]. Three most prominent Distributed MPC structures for multi-robot formation control are formulated and implemented in MAT-

LAB. A leader-follower formation control problem is proposed as the benchmark problem. Several metrics regarding to the formation performance and computational complexity are introduced to do the comparison. Since Distributed MPC includes communicating information among the agents and some problems such as delay, packet dropout and jitter commonly exists in the real experiment, the robustness to communication uncertainty is also compared among the three types. From MATLAB simulation, the best Distributed MPC is selected and applied in a high-fidelity, open-source framework consisting of the Webots simulator and the Robotic Operating Systems (ROS). Finally, the best Distributed MPC architecture in the presence of communication delay and packet dropout is compared with the Decentralized MPC controller.

Chapter 2 Problem Formulation

In this chapter, the dynamic equation of a single drone is defined, several assumptions are stated and our benchmark leader-follower formation problem is described. In the rest part of the report, the following notation is used. $\mathcal{N}_i$ is the set of neighbors of agent i. $\boldsymbol{v}[m|n]$ denotes the value of variable $\boldsymbol{v}$ at discrete instant m, predicted at time step n. $\|\cdot\|$ denotes the Euclidean norm of a vector or the spectral norm of a real matrix and $\|\cdot\|_P := \sqrt{x^T P x}$ (with $P \in \mathbb{R}^{n\times n}$ and $P \succ 0$) stands for the weighted Euclidean norm of $x \in \mathbb{R}^n$.

Let $\{n\}$ be the Earth-fixed inertial frame and $\{b\}$ be the body-fixed frame of an aerial robotic agent. The state of a single MAV is defined as follows:

$$\boldsymbol{X}_i = [\ \boldsymbol{x}^T \quad \boldsymbol{v}^T \quad \boldsymbol{t}^T \quad \boldsymbol{w}^T\ ]^T \tag{2.1}$$

where $\boldsymbol{x} \in \{n\}$ and $\boldsymbol{v} \in \{n\}$ represent the position and linear velocity of the drone with respect to $\{n\}$ expressed in $\{n\}$, $\boldsymbol{t}$ is the Euler angles and $\boldsymbol{w} \in \{b\}$ denotes the angular velocities expressed in $\{b\}$. Based on this definition, the 6-DoF rigid body dynamics of the aerial robot can be written by the following equations [12].

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{v} \\ m\dot{\boldsymbol{v}} &= m\boldsymbol{g} + \boldsymbol{R}_b^n \boldsymbol{F} \\ \dot{\boldsymbol{t}} &= \boldsymbol{T}\boldsymbol{w} \\ \boldsymbol{I}_b\dot{\boldsymbol{w}} &= \boldsymbol{\tau} - \boldsymbol{w} \times \boldsymbol{I}_b\boldsymbol{w} \end{aligned} \tag{2.2}$$

where m denotes the mass of the airframe, $\boldsymbol{g} \in \{n\}$ represents the gravitational acceleration vector, $\boldsymbol{T}$ is the angular transformation matrix, $\boldsymbol{R}_b^n$ represents the rotation matrix $R \in \mathbb{SO}^3$ that transforms a vector expression from $\{b\}$ to $\{n\}$, $\boldsymbol{F} := [0 \quad 0 \quad F_z] \in \{b\}$ is the thrust aligned with the body's z-axis, $\boldsymbol{\tau} \in \{b\}$ denotes the torque applied to the body and $\boldsymbol{I}_b$ is the inertia of the vehicle with respect to its center of mass.

Despite the traditional model using thrust and torque or propeller speeds as inputs, our model and inputs can be simplified due to the cascaded control structure we use. As shown in **Figure 2.1**, each quadrotor first estimates its own state. Based on the state estimation, the nonlinear MPC computes the thrust the drone needs to generate F_z and the reference of Euler angles $\boldsymbol{t}_{ref}$.

According to that information, attitude controller calculates the propeller speeds $\boldsymbol{w}$ and apply it to the quadrotor.

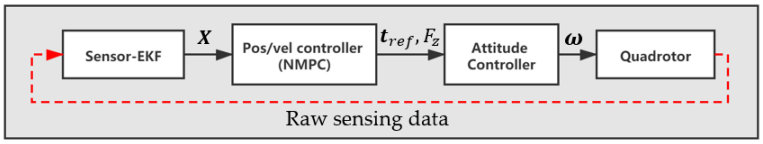


Figure 2.1: Cascaded control/estimation architecture for a single agent

Therefore, Euler angles dynamics are not considered in MPC controller. The input we compute in the nonlinear MPC is the reference Euler angles $\boldsymbol{t}_{ref}$ and the thrust along the z-axis of the drone's local coordination F_z. The dynamic model used in MPC is:

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{v} \\ m\dot{\boldsymbol{v}} &= m\boldsymbol{g} + \boldsymbol{R}_b^n \boldsymbol{F} \\ \dot{\boldsymbol{t}} &= \frac{1}{\tau}(k\boldsymbol{t}_{ref} - \boldsymbol{t}) \end{aligned} \tag{2.3}$$

where τ is the time constant and k_L is the gain of the first order dynamical model for roll, pitch and yaw angles.

To reduce the computational burden, our benchmark leader-follower control problem consists of one leader and two followers. Each of the control architectures, however, can be generalized to control more followers. Only the leader has access to its absolute position by a global localization system. Hence, the leader is responsible for the trajectory tracking activity. Since only three vehicles are involved, all the rest agents are considered to be the neighbors of a follower, i.e. $\mathcal{N}_L = \{f_1, f_2\}, \mathcal{N}_{f_1} = \{L, f_2\}, \mathcal{N}_{f_2} = \{L, f_1\}$. Meanwhile, all agents are assumed to be equipped with an optic flow-sonar sensor couple to obtain linear velocities and an IMU to acquire linear accelerations, rotational velocities and Euler angles. It is assumed that all followers can measure the inter-vehicle positions and orientations of neighbor vehicles by an on-board relative localization system. All sensors are characterized by zero-mean Gaussian noise. Therefore, an Extended Kalman Filter (EKF) is implemented for each agent to fuse the sensor information and carry out the state estimation. Finally, each pair of agents can communicate information with each other. Possible communication delays and packet losses are considered in the simulation.

In the benchmark test problem, the leader is considered as a separate and independent trajectory tracking agent. Though the two followers have no information about the trajectory, they will follow the leader to navigate along the reference by trying to maintain the formation structure. At each time step, the controllers of the followers attempt to control the agents to keep the desired formation structure (i.e. $\boldsymbol{x}_j - \boldsymbol{x}_i = \Delta\boldsymbol{x}_{ref,ij}, \forall j \in \mathcal{N}_i$).

Chapter 3 Methodology

In this chapter, three types of Distributed MPC for multi-agents formation control is formulated. The first type decomposes the coupled system by sharing the estimated velocity with their neighbors. Additional slack variables are introduced and ADMM is implemented to distribute the centralized problem in the second type of Distributed MPC. In the third type, a stochastic optimization algorithm, PSO, is applied to solve the MPC problem.

3.1 Velocity Sharing Based Distributed MPC

In most literature about information sharing based Distributed MPC [7, 13], the estimated input is shared with their neighbors. Given the dynamic model of their neighbors and the estimated input, the coupled state of the neighbors is computed in the agents' subsystem. In the formation control problem, the only coupled state is the neighbor's linear velocity. Therefore, we can directly share the estimated velocity instead of saving the computational burden in each subsystem, which leads to our first type of Distributed MPC effort.

3.1.1 State Definition

Since the follower can not access to its own absolute position by our assumption, we need to redefine the state of the followers, while the state of the follower can be the same as that in **Equation 2.1**. To clarify, new notation $\boldsymbol{\xi}_i$ will be used to represent the state of agent i in the rest part of the report. The state and input of leader and followers are defined as:

$$\begin{aligned}
\boldsymbol{\xi}_L &= [\ \boldsymbol{x}_L{}^T \quad \boldsymbol{v}_L{}^T \quad \boldsymbol{t}_L{}^T\ ]^T \\
\boldsymbol{\xi}_{f_1} &= [\ \boldsymbol{v}_{f_1}{}^T \quad \boldsymbol{t}_{f_1}{}^T \quad \Delta\boldsymbol{x}_{f_1,L}{}^T \quad \Delta\boldsymbol{x}_{f_1,f_2}{}^T\ ]^T \\
\boldsymbol{\xi}_{f_2} &= [\ \boldsymbol{v}_{f_2}{}^T \quad \boldsymbol{t}_{f_2}{}^T \quad \Delta\boldsymbol{x}_{f_2,L}{}^T \quad \Delta\boldsymbol{x}_{f_2,f_1}{}^T\ ]^T
\end{aligned} \tag{3.1}$$

where $\boldsymbol{\xi}_L$ is the state of the leader, $\boldsymbol{\xi}_{f_1}$ and $\boldsymbol{\xi}_{f_2}$ are the state of follower-1 and follower-2 and $\Delta\boldsymbol{x}_{i,j}$ is the relative position of vehicle j respect to vehicle i.

As the yaw angle is part of the requirement of formation control, the desired yaw angle is directly sent to the attitude controller. The input of each agent computed by MPC controller is:

$$\boldsymbol{u}_i = [\ \theta_{ref,i} \qquad \phi_{ref,i} \qquad F_i\ ]^T \tag{3.2}$$

where $\theta_{ref,i}$ and $\phi_{ref,i}$ is the desired roll and pitch angle.

The dynamic equation of the state of the leader can directly refer to **Equation 2.3**. Regarding to the followers, the dynamic of relative position can simply be derived by the difference of positions of the two agents. Hence, the dynamic model of the followers can be defined as:

$$\begin{aligned} m\dot{\boldsymbol{v}}_i &= m\boldsymbol{g} + \boldsymbol{R}_{b_i}^n \boldsymbol{F}_i \\ \dot{\boldsymbol{t}}_i &= \frac{1}{\tau_i}(k_i \boldsymbol{t}_{ref,i} - \boldsymbol{t}_i) \\ \Delta\dot{\boldsymbol{x}}_{i,j} &= \dot{\boldsymbol{x}}_j - \dot{\boldsymbol{x}}_i = \boldsymbol{v}_j - \boldsymbol{v}_i \end{aligned} \tag{3.3}$$

3.1.2 Centralized MPC problem

Before illustrating the structure of the Distributed MPC, let's first write down the centralized MPC problem. The cost function of the leader and the followers are defined as:

$$\min_{\boldsymbol{u}_L, \boldsymbol{u}_{f_1}, \boldsymbol{u}_{f_2}} J_L(\boldsymbol{\xi}_L, \boldsymbol{u}_L) + J_{f_1}(\boldsymbol{\xi}_{f_1}, \boldsymbol{u}_{f_1}) + J_{f_2}(\boldsymbol{\xi}_{f_2}, \boldsymbol{u}_{f_2}) \tag{3.4}$$

subject to the following constraints,

$$\begin{gathered} \boldsymbol{\xi}_L[k+n+1|k] = f_L(\boldsymbol{\xi}_L[k+n|k], \boldsymbol{u}_L[k+n|k]) \\ \boldsymbol{\xi}_{f_1}[k+n+1|k] = f_{f_1}(\boldsymbol{\xi}_{f_1}[k+n|k], \boldsymbol{u}_{f_1}[k+n|k], \boldsymbol{v}_L[k+n|k], \boldsymbol{v}_{f_2}[k+n|k]) \\ \boldsymbol{\xi}_{f_2}[k+n+1|k] = f_{f_2}(\boldsymbol{\xi}_{f_1}[k+n|k], \boldsymbol{u}_{f_1}[k+n|k], \boldsymbol{v}_L[k+n|k], \boldsymbol{v}_{f_2}[k+n|k]) \\ \boldsymbol{\xi}_i[k+n|k] \in \boldsymbol{\Xi}, \qquad \boldsymbol{u}_i[k+n|k] \in \boldsymbol{U}, \qquad \boldsymbol{\xi}_i[k|k] = \boldsymbol{\xi}_i'[k] \\ n \in \{0, 1, ..., N\}, \quad i \in \{L, f_1, f_2\} \end{gathered} \tag{3.5}$$

where $\boldsymbol{\xi}_i = [\boldsymbol{\xi}_i^T[k|k],\ \boldsymbol{\xi}_i^T[k+1|k],\ ...\ ,\ \boldsymbol{\xi}_i^T[k+N|k]]^T$ and $\boldsymbol{u}_i = [\boldsymbol{u}_i^T[k|k],\ \boldsymbol{u}_i^T[k+1|k],\ ...\ ,\ \boldsymbol{u}_i^T[k+N-1|k]]^T$ are the compact expression of the state and input in the whole horizon, $\boldsymbol{\xi}_i'[k]$ denotes the initial state estimated by EKF at time k, f_L is the discrete version of dynamic equation of the leader stated in **Equation 2.3**, f_{f_1} and f_{f_2} are the discrete version of dynamic equation of the followers illustrated in **Equation 3.3**, $\boldsymbol{\Xi}$ and $\boldsymbol{U}$ are the bounding set of state and input due to the safety constraints and the physical limits of the propellers and electric motors on the aerial agent, N is the prediction (and control) horizon and J_L, J_{f_1} and J_{f_2} denotes respectively the cost function

of the leader, follower-1 and follower-2, which is defined as the following:

$$\begin{aligned}
&J_{k,L}(\boldsymbol{\xi}_L[k+n|k], \boldsymbol{u}_L[k+n|k]) = \\
&\quad \|\boldsymbol{x}_L[k+n|k] - \boldsymbol{x}_{ref,L}\|_{Q_{x,L}} + \|\boldsymbol{v}_L[k+n|k]\|_{Q_{v,L}} + \\
&\quad \|\boldsymbol{u}_L[k+n|k]\|_{Q_{u,L}} + \|\Delta\boldsymbol{u}_L[k+n|k]\|_{Q_{\Delta u,L}} \\
&J_{N,L}(\boldsymbol{\xi}_L[k+N|k]) = \\
&\quad \|\boldsymbol{x}_L[k+N|k] - \boldsymbol{x}_{ref,L}\|_{Q_{x_N},L} + \|\boldsymbol{v}_L[k+N|k]\|_{Q_{v_N},L} \\
&J_L(\boldsymbol{\xi}_L, \boldsymbol{u}_L) = \sum_{n=0}^{N-1} J_{k,L}(\boldsymbol{\xi}_L[k+n|k], \boldsymbol{u}_L[k+n|k]) + J_{N,L}(\boldsymbol{\xi}_L[k+N|k])
\end{aligned} \tag{3.6}$$

where $\boldsymbol{x}_{ref,L}$ is the trajectory reference and $\Delta\boldsymbol{u}_L[k+n|k] = \boldsymbol{u}_L[k+n|k] - \boldsymbol{u}_L[k+n|k-1]$ is the deviation of the computed input from that computed in the previous time step, this term is one of the key requirements for stability [7].

The cost function of the follower-1 and the follower-2 are nearly the same. The definition is the following:

$$\begin{aligned}
&J_{k,i}(\boldsymbol{\xi}_i[k+n|k], \boldsymbol{u}_i[k+n|k]) = \\
&\quad \sum_{j\in\mathcal{N}_i} \|\Delta\boldsymbol{x}_{i,j}[k+n|k] - \Delta\boldsymbol{x}_{ref,ij}\|_{Q_{\Delta x,i}} + \\
&\quad \sum_{j\in\mathcal{N}_i} \|\boldsymbol{v}_i[k+n|k] - \boldsymbol{v}_j[k+n|k]\|_{Q_{v,i}} + \|\boldsymbol{u}_i[k+n|k]\|_{Q_{u,i}} \\
&J_{N,i}(\boldsymbol{\xi}_i[k+N|k]) = \\
&\quad \sum_{j\in\mathcal{N}_i} \|\Delta\boldsymbol{x}_{i,j}[k+N|k] - \Delta\boldsymbol{x}_{ref,ij}\|_{Q_{\Delta x_N,i}} + \\
&\quad \sum_{j\in\mathcal{N}_i} \|\boldsymbol{v}_i[k+N|k] - \boldsymbol{v}_j[k+N|k]\|_{Q_{v_N,i}} \\
&J_i(\boldsymbol{\xi}_i, \boldsymbol{u}_i) = \sum_{n=0}^{N-1} J_{k,i}(\boldsymbol{\xi}_i[k+n|k], \boldsymbol{u}_i[k+n|k]) + J_{N,i}(\boldsymbol{\xi}_i[k+N|k])
\end{aligned} \tag{3.7}$$

where the index i here can be either f_1 or f_2 and $\Delta\boldsymbol{x}_{ref,ij}$ is the desired relative displacement of the neighbor j to agent i.

3.1.3 Distributed MPC Architecture

From the centralized optimization problem in **Equation 3.4** and **Equation 3.5**, the leader can be decoupled directly from the whole system. The followers, however, have the coupled term in their dynamic equation. To decompose the whole system, the neighbor's future velocity is required in order to estimate the relative displacement of the neighbors to the follower.

The strategy of the velocity sharing based Distributed MPC is to communicate the predicted future velocity of each agent with their neighbors. Then, the followers utilized the velocity they received at the previous time step to solve the MPC problem.

To prevent confusion, new notations are introduced as the following: $\bar{\boldsymbol{v}}_i[k] = [\bar{\boldsymbol{v}}_i^T[k|k],\ \bar{\boldsymbol{v}}_i^T[k+1|k],\ \dots\ ,\ \bar{\boldsymbol{v}}_i^T[k+N|k]]^T$ represents the predicted velocity of agent i computed by nonlinear MPC at time step k. $\hat{\boldsymbol{v}}_{i,j}[k] = [\hat{\boldsymbol{v}}_{i,j}^T[k|k],\ \hat{\boldsymbol{v}}_{i,j}^T[k+1|k],\ \dots\ ,\ \hat{\boldsymbol{v}}_{i,j}^T[k+N|k]]^T$ denotes the predicted velocity of agent j processed in the MPC controller of agent i at time step k.

At each time step, the followers will receive the predicted future velocity from their neighbors. Nevertheless, these information is computed at the previous step. The estimated velocity of the last horizon in the current step isn't included. Hence, the velocity in the last two horizons are assumed to be constant and the transmission between $\bar{\boldsymbol{v}}_i[k-1]$ and $\hat{\boldsymbol{v}}_i[k]$ is defined as follows:

$$\hat{\boldsymbol{v}}_{i,j}[k+n|k] = \begin{cases} \bar{\boldsymbol{v}}_j[k+n|k-1], & n \in [0, N-1] \\ \bar{\boldsymbol{v}}_j[k+N-1|k-1], & n = N \end{cases} \tag{3.8}$$

With the predicted velocity of neighbors, the whole system can be decomposed. The leader can be directly decomposed without any modifications. The MPC problem it solves at each times step is defined as:

$$\begin{aligned} \min_{\boldsymbol{u}_L} \quad & J_L(\boldsymbol{\xi}_L, \boldsymbol{u}_L) \\ \text{s.t.} \quad & \boldsymbol{\xi}_L[k+n+1|k] = f_L(\boldsymbol{\xi}_L[k+n|k], \boldsymbol{u}_L[k+n|k]) \\ & \boldsymbol{\xi}_L[k+n|k] \in \boldsymbol{\Xi}, \quad \boldsymbol{u}_L[k+n|k] \in \boldsymbol{U} \\ & \boldsymbol{\xi}_L[k|k] = \boldsymbol{\xi}_L'[k], \quad n \in \{0, 1, ..., N\} \end{aligned} \tag{3.9}$$

The optimization problem solved in the follower controller in **Equation 3.10**. The index i can be either f_1 or f_2 and $j_1, j_2 \in \mathcal{N}_i$ are the neighbors of agent i.

$$\begin{aligned} \min_{\boldsymbol{u}_i} \quad & J_i(\boldsymbol{\xi}_i, \boldsymbol{u}_i) \\ \text{s.t.} \quad & \boldsymbol{\xi}_i[k+n+1|k] = f_i(\boldsymbol{\xi}_i[k+n|k], \boldsymbol{u}_i[k+n|k], \hat{\boldsymbol{v}}_{i,j_1}[k+n|k], \hat{\boldsymbol{v}}_{i,j_2}[k+n|k]) \\ & \boldsymbol{\xi}_i[k+n|k] \in \boldsymbol{\Xi}, \quad \boldsymbol{u}_i[k+n|k] \in \boldsymbol{U} \\ & \boldsymbol{\xi}_i[k|k] = \boldsymbol{\xi}_i'[k], \quad n \in \{0, 1, ..., N\} \end{aligned} \tag{3.10}$$

Algorithm 1 summarizes the velocity sharing based Distributed MPC strategy. The followers start with the assumption that the initial future velocity of their neighbors is $\mathbf{0}$. At each time step, all agents first estimate their own state by the sensors and EKF. Then, each follower receives the predicted future velocity of their neighbors and obtain $\hat{\boldsymbol{v}}_{i,j_1}[k], \hat{\boldsymbol{v}}_{i,j_2}[k]$ according to **Equation 3.8**. Based on those information, all agents solve

their local optimization problem as stated in **Equation 3.9** and **Equation 3.10**. Such MPC optimization problem can be solved either by a MPC solver or a nonlinear programming solver. After solving the optimization problem, all agents apply the first control input in their plant and transmit their predicted future velocity to the neighbors. Then, the whole system moves to next time step. Such loop will continue until the leader reaches the final destination.

Algorithm 1: Velocity sharing based Distributed MPC

Initialization: at time k_0
  Each follower i assumes $\hat{\boldsymbol{v}}_{i,j_1}[0] = \mathbf{0}, \hat{\boldsymbol{v}}_{i,j_2}[0] = \mathbf{0}$.
Main loop: at any time k, $k > k_0$
  Each robot estimates $\boldsymbol{\xi}'[k]$ by EKF.
  Each follower receives $\bar{\boldsymbol{v}}_{j_1}[k-1], \bar{\boldsymbol{v}}_{j_2}[k-1]$.
  Each follower i predicts $\hat{\boldsymbol{v}}_{i,j_1}[k], \hat{\boldsymbol{v}}_{i,j_2}[k]$ by **Equation 3.8**.
  All robots solve **Equation 3.9** or **Equation 3.10**.
  Each agent i apply the first computed control input $\boldsymbol{u}_i^*[k|k]$.
  Each drone i transmit $\bar{\boldsymbol{v}}_i[k]$ to their neighbors.
Terminate when leader reaches the final destination.

3.2 ADMM-based Distributed MPC

Inspired by Van Parys [9], this second type of Distributed MPC introduces some slack variables so that we can first separate the cost function into two parts. The terms that can be decoupled will be minimized in primal variables and the coupled terms will be handled in slack variables. Based on that, the whole system is decomposed and each agent can locally solve its sub-system.

The decomposition procedure in this section takes the advantage of the alternating direction method of multipliers (ADMM), a simple but powerful algorithm that is well suited to distributed optimization problem [14]. The algorithm solves problems in the form:

$$\begin{aligned} &\text{minimize} && f(x) + g(z) \\ &\text{subject to} && Ax + Bz = c \end{aligned} \tag{3.11}$$

the corresponding augmented Lagrangian is:

$$L_\rho(x, z, y) = f(x) + g(z) + y^T(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2 \tag{3.12}$$

where x and z are called primal variable and y is the dual variable.

The algorithm solves the problem by iteratively updating the primal and

dual variables:

$$\begin{aligned} x^{k+1} &:= \arg\min_x L_\rho(x, z^k, y^k) \\ z^{k+1} &:= \arg\min_x L_\rho(x^{k+1}, z, y^k) \\ y^{k+1} &:= y^k + \rho(Ax^{k+1} + Bz^{k+1} - c) \end{aligned} \tag{3.13}$$

3.2.1 Notation

In this section, some additional variables are introduced. We first stack the state and input of a single agent into a new variable $\boldsymbol{y}_i$:

$$\boldsymbol{y}_i = [\ \boldsymbol{\xi}_i{}^T \quad \boldsymbol{u}_i{}^T\ ]^T, \quad \forall i \in \{L, f_1, f_2\} \tag{3.14}$$

As we can see in the previous section, the coupled state is the relative position of the followers and each follower needs the velocity of their neighbors to decompose the whole problem. Therefore, several slack variables, $\boldsymbol{z}_{f_1}, \boldsymbol{z}_{f_1,f_2}, \boldsymbol{z}_{f_1,L}, \boldsymbol{z}_{f_2}, \boldsymbol{z}_{f_2,f_1}, \boldsymbol{z}_{f_2,L}$ are introduced. For those additional slack variables with two indexes, i.e. $\boldsymbol{z}_{i,j}$, the first index i means in which vehicle these variables are updated, while the second j means $\boldsymbol{z}_{i,j}$ should represent the velocity of the j vehicle, i.e. it should satisfy $\boldsymbol{z}_{i,j} = \boldsymbol{v}_j$. The formal definition of the constraints of these slack variables are:

$$\begin{aligned} \boldsymbol{z}_{f_1} &:= \boldsymbol{v}_{f_1}, \quad \boldsymbol{z}_{f_2,f_1} := \boldsymbol{v}_{f_1} \\ \boldsymbol{z}_{f_2} &:= \boldsymbol{v}_{f_2}, \quad \boldsymbol{z}_{f_1,f_2} := \boldsymbol{v}_{f_2} \\ \boldsymbol{z}_{f_1,L} &:= \boldsymbol{v}_L, \quad \boldsymbol{z}_{f_2,L} := \boldsymbol{v}_L \end{aligned} \tag{3.15}$$

3.2.2 Centralized Optimization Problem

With the introduction of the slack variables, the objective function stated in **Equation 3.4** can be separated by decoupled terms expressed in primal variables and coupled terms expressed in slack variables. Therefore, the cost function of the leader, which has no coupled term, can remain the same as that in **Equation 3.6**.

$$\begin{aligned} J'_{k,L}(\boldsymbol{y}_L[k+n|k]) =& J_{k,L}(\boldsymbol{\xi}_L[k+n|k], \boldsymbol{u}_L[k+n|k]) \\ J'_{N,L}(\boldsymbol{y}_L[k+N|k]) =& J_{N,L}(\boldsymbol{\xi}_L[k+N|k]) \end{aligned} \tag{3.16}$$

The cost function of the primal variables of the follower has to remove all the coupled term:

$$J'_{k,i}(\boldsymbol{y}_i) = \|\boldsymbol{u}_i[k+n|k]\|_{Q_{u,i}}, \quad J'_{N,i}(\boldsymbol{y}_i) = 0, \quad \forall i \in \{f_1, f_2\} \tag{3.17}$$

The overall cost function for primal variables are defined as:

$$J'_i = \sum_{n=0}^{N-1} J'_{k,i}(\boldsymbol{y}_i) + J'_{N,i}(\boldsymbol{y}_i), \quad \forall i \in \{L, f_1, f_2\} \tag{3.18}$$

The coupled terms for formation requirement that removed from the objective function of the followers are defined in the cost function of the slack variables:

$$\begin{aligned} G_{ij}(\boldsymbol{z}_i, \boldsymbol{z}_{i,j}) = \sum_{n=1}^{N} \|[\Delta \boldsymbol{x}_{ij}[k|k] + \sum_{t=0}^{n} (\boldsymbol{z}_{i,j}[k+t|k] - \boldsymbol{z}_i[k+t|k])\Delta t] \\ - \Delta \boldsymbol{x}_{ij}^{desire}[k+n+1|k]\|_{\Delta x}^2 \end{aligned} \tag{3.19}$$

where Δt is the time duration of each time step, i.e. $dt = t[k+1|k] - t[k|k]$.

With all these definitions, the centralized optimization problem can be rewritten as the following:

$$\begin{aligned} \min_{\substack{\boldsymbol{y}_i, \boldsymbol{z}_i, \boldsymbol{z}_{i,j} \\ i \in \{L, f_1, f_2\}, j \in \mathcal{N}_i}} \quad & J'_L(\boldsymbol{y}_L) + J'_{f_1}(\boldsymbol{y}_{f_1}) + J'_{f_2}(\boldsymbol{y}_{f_2}) + G_{f_1,f_2}(\boldsymbol{z}_{f_1}, \boldsymbol{z}_{f_1,f_2}) + \\ & G_{f_1,L}(\boldsymbol{z}_{f_1}, \boldsymbol{z}_{f_1,L}) + G_{f_2,f_1}(\boldsymbol{z}_{f_2}, \boldsymbol{z}_{f_2,f_1}) + G_{f_2,L}(\boldsymbol{z}_{f_2}, \boldsymbol{z}_{f_2,L}) \end{aligned} \tag{3.20}$$

subject to,

$$\begin{aligned} & \boldsymbol{y}_i \in \boldsymbol{Y}_i \\ & \boldsymbol{z}_i = \boldsymbol{v}_i, \ \boldsymbol{z}_{i,j} = \boldsymbol{v}_j, \quad j \in \mathcal{N}_i \\ & \forall i \in \{L, f_1, f_2\} \end{aligned} \tag{3.21}$$

where $\boldsymbol{Y}_i$ is the compact set that contained all the possible values of $\boldsymbol{y}_i$ satisfying the constraints in **Equation 3.5**. The second constraint is how we define the slack variables in **Equation 3.15**. It can also be written as $\boldsymbol{z}_i = P_i \boldsymbol{y}_i, \ \boldsymbol{z}_{i,j} = P_j \boldsymbol{y}_j, \quad j \in \mathcal{N}_i$ where P_i is the indicator matrix to extract the velocity term from y_i.

The difference of velocity between an agent and its neighbors is not penalized in this new centralized problem as an improvement of formation control performance is observed in the simulation when removing these terms. Except that, the new centralized optimization problem is equivalent to that described in previous section.

Because the leader is not responsible for maintaining the formation, there won't be any slack variable and dual variable updated in leader and the constraints $\boldsymbol{z}_{f_1,L} := \boldsymbol{v}_L, \ \boldsymbol{z}_{f_2,L} := \boldsymbol{v}_L$ will be strictly satisfied. The rest four slack variable constraints, however, will be dualized and that's how we decompose the whole system by ADMM. The augmented Lagrangian

function of the centralized problem is:

$$\begin{aligned}\mathcal{L}_\rho =& J'_L(\boldsymbol{y}_L) + \sum_{i\in\{f_1,f_2\}} (J'_i(\boldsymbol{y}_i) + \boldsymbol{\lambda}_i^T(\boldsymbol{v}_i - \boldsymbol{z}_i) + \frac{\rho}{2}\|\boldsymbol{v}_i - \boldsymbol{z}_i\|_2^2 + \\ & \sum_{j\in\mathcal{N}_i} G_{i,j}(\boldsymbol{z}_i, \boldsymbol{z}_{i,j}) + \boldsymbol{\lambda}_{i,j}^T(\boldsymbol{v}_j - \boldsymbol{z}_{i,j}) + \frac{\rho}{2}\|\boldsymbol{v}_j - \boldsymbol{z}_{i,j}\|_2^2) \\ =& \sum_{i\in\{L,f_1,f_2\}} \mathcal{L}_{\rho,i}(\boldsymbol{y}_i, \boldsymbol{z}_i, \boldsymbol{\lambda}_i, \boldsymbol{v}_j, \boldsymbol{z}_{i,j}, \boldsymbol{\lambda}_{i,j}) \\ =& \sum_{i\in\{L,f_1,f_2\}} \mathcal{L}_{\rho,i}(\boldsymbol{y}_i, \boldsymbol{z}_i, \boldsymbol{\lambda}_i, \boldsymbol{v}_i, \boldsymbol{z}_{j,i}, \boldsymbol{\lambda}_{j,i})\end{aligned} \tag{3.22}$$

where $\boldsymbol{\lambda}_i$ and $\boldsymbol{\lambda}_{i,j}$ represents the dual variables associated with the dualized constraints and ρ is the penalty parameter of ADMM.

3.2.3 Problem Decomposition

For better illustration, $\tilde{\boldsymbol{x}}$ will represent that the variable $\boldsymbol{x}$ is obtained by communication from the neighbors.

The last line in **Equation. 3.22** can be decomposed for each agent to operate the primal variable y update and the second last line can be separated to local system for the slack variable z update. To be more specific, the leader and follower will update their primal variable by solving the following optimization problem:

$$\begin{aligned}\boldsymbol{y}_L[k] &= \underset{\boldsymbol{y}_L\in\boldsymbol{Y}_L}{\arg\min}\ J'_L(\boldsymbol{y}_L) \\ \boldsymbol{y}_{f_1}[k] &= \underset{\boldsymbol{y}_{f_1}\in\boldsymbol{Y}_{f_1}}{\arg\min}\ J'_{f_1}(\boldsymbol{y}_{f_1}) + \boldsymbol{\lambda}_{f_1}^T(\boldsymbol{v}_{f_1} - \boldsymbol{z}_{f_1}[k-1]) + \frac{\rho}{2}\|\boldsymbol{v}_{f_1} - \boldsymbol{z}_{f_1}[k-1]\|_2^2 \\ &\qquad + \tilde{\boldsymbol{\lambda}}_{f_2,f_1}^T[k-1](\boldsymbol{v}_{f_1} - \tilde{\boldsymbol{z}}_{f_2,f_1}[k-1]) + \frac{\rho}{2}\|\boldsymbol{v}_{f_1} - \tilde{\boldsymbol{z}}_{f_2,f_1}[k-1]\|_2^2 \\ \boldsymbol{y}_{f_2}[k] &= \underset{\boldsymbol{y}_{f_2}\in\boldsymbol{Y}_{f_2}}{\arg\min}\ J'_{f_2}(\boldsymbol{y}_{f_2}) + \boldsymbol{\lambda}_{f_2}^T(\boldsymbol{v}_{f_2} - \boldsymbol{z}_{f_2}[k-1]) + \frac{\rho}{2}\|\boldsymbol{v}_{f_1} - \boldsymbol{z}_{f_2}[k-1]\|_2^2 \\ &\qquad + \tilde{\boldsymbol{\lambda}}_{f_1,f_2}^T[k-1](\boldsymbol{v}_{f_2} - \tilde{\boldsymbol{z}}_{f_1,f_2}[k-1]) + \frac{\rho}{2}\|\boldsymbol{v}_{f_2} - \tilde{\boldsymbol{z}}_{f_1,f_2}[k-1]\|_2^2\end{aligned} \tag{3.23}$$

The leader has no slack variable to update. The followers will update their slack variables according to the following equations.

For follower-1:

$$\begin{aligned}(\boldsymbol{z}_{f_1}[k], \boldsymbol{z}_{f_1,f_2}[k]) = \underset{\boldsymbol{z}_{f_1},\boldsymbol{z}_{f_1,f_2}}{\arg\min}\ & G_{f_1,L}(\boldsymbol{z}_{f_1}, \boldsymbol{z}_{f_1,L}) + G_{f_1,f_2}(\boldsymbol{z}_{f_1}, \boldsymbol{z}_{f_1,f_2}) \\ & + \boldsymbol{\lambda}_{f_1}^T(\boldsymbol{v}_{f_1}[k] - \boldsymbol{z}_{f_1}) + \boldsymbol{\lambda}_{f_1,f_2}^T(\tilde{\boldsymbol{v}}_{f_2}[k] - \boldsymbol{z}_{f_1,f_2}) \\ & + \frac{\rho}{2}\|\boldsymbol{v}_{f_1}[k] - \boldsymbol{z}_{f_1}\|_2^2 + \frac{\rho}{2}\|\tilde{\boldsymbol{v}}_{f_2}[k] - \boldsymbol{z}_{f_1,f_2}\|_2^2 \\ \textit{subject to}\quad & \boldsymbol{z}_{f_1,L} := \tilde{\boldsymbol{v}}_L[k]\end{aligned} \tag{3.24}$$

For follower-2:

$$\begin{aligned}(\boldsymbol{z}_{f_2}[k], \boldsymbol{z}_{f_2,f_1}[k]) = \underset{\boldsymbol{z}_{f_2}, \boldsymbol{z}_{f_2,f_1}}{\arg\min}\ & G_{f_2,L}(\boldsymbol{z}_{f_2}, \boldsymbol{z}_{f_2,L}) + G_{f_2,f_1}(\boldsymbol{z}_{f_2}, \boldsymbol{z}_{f_2,f_1}) \\ & + \boldsymbol{\lambda}_{f_2}^T(\boldsymbol{v}_{f_2}[k] - \boldsymbol{z}_{f_2}) + \boldsymbol{\lambda}_{f_2,f_1}^T(\tilde{\boldsymbol{v}}_{f_1}[k] - \boldsymbol{z}_{f_2,f_1}) \\ & + \frac{\rho}{2}\|\boldsymbol{v}_{f_2}[k] - \boldsymbol{z}_{f_2}\|_2^2 + \frac{\rho}{2}\|\tilde{\boldsymbol{v}}_{f_1}[k] - \boldsymbol{z}_{f_2,f_1}\|_2^2 \\ \textit{subject to}\quad & \boldsymbol{z}_{f_2,L} := \tilde{\boldsymbol{v}}_L[k]\end{aligned} \tag{3.25}$$

The final step is to update the dual variable in the two followers.
For follower-1:

$$\begin{aligned}\boldsymbol{\lambda}_{f_1}[k] &= \boldsymbol{\lambda}_{f_1}[k-1] + \rho(\boldsymbol{v}_{f_1}[k] - \boldsymbol{z}_{f_1}[k]) \\ \boldsymbol{\lambda}_{f_1,f_2}[k] &= \boldsymbol{\lambda}_{f_1,f_2}[k-1] + \rho(\tilde{\boldsymbol{v}}_{f_2}[k-1] - \boldsymbol{z}_{f_1,f_2}[k])\end{aligned} \tag{3.26}$$

For follower-2:

$$\begin{aligned}\boldsymbol{\lambda}_{f_2}[k] &= \boldsymbol{\lambda}_{f_2}[k-1] + \rho(\boldsymbol{v}_{f_2}[k] - \boldsymbol{z}_{f_2}[k]) \\ \boldsymbol{\lambda}_{f_2,f_1}[k] &= \boldsymbol{\lambda}_{f_2,f_1}[k-1] + \rho(\tilde{\boldsymbol{v}}_{f_1}[k-1] - \boldsymbol{z}_{f_2,f_1}[k])\end{aligned} \tag{3.27}$$

The architecture of the ADMM-based Distributed MPC is summarized in **Algorithm 2**. At each time step k, all agents first estimate their own state by the sensors and EKF. The leader simply does its own reference tracking without receiving anything from the followers. Follower-1 will receive $\tilde{\boldsymbol{z}}_{f_2,f_1}[k-1]$ and $\tilde{\boldsymbol{\lambda}}_{f_2,f_1}[k-1]$ from follower-2. Follower-2 will receive $\tilde{\boldsymbol{z}}_{f_1,f_2}[k-1]$ and $\tilde{\boldsymbol{\lambda}}_{f_1,f_2}[k-1]$ from follower-1. Then, both followers and leader update their primal variable $\boldsymbol{y}_i[k]$ according to **Equation 3.23**. They will extract the predicted future velocity $\boldsymbol{v}_i[k]$ from $\boldsymbol{y}_i[k]$ and send it to their neighbors. After that, follower-1 will receive $\tilde{\boldsymbol{v}}_L[k]$ from leader and $\tilde{\boldsymbol{v}}_{f_2}[k]$ from follower-2, update the slack variable $\boldsymbol{z}_{f_1}[k]$ and $\boldsymbol{z}_{f_1,f_2}[k]$ by **Equation 3.24** and compute the dual variable $\boldsymbol{\lambda}_{f_1}[k]$ and $\boldsymbol{\lambda}_{f_1,f_2}[k]$ according to **Equation 3.26**. Follower-2 will receive $\tilde{\boldsymbol{v}}_L[k]$ from leader and $\tilde{\boldsymbol{v}}_{f_1}[k]$ from follower-1, calculate the slack variable $\boldsymbol{z}_{f_2}[k]$ and $\boldsymbol{z}_{f_2,f_1}[k]$ based on **Equation 3.25** and update the dual variable $\boldsymbol{\lambda}_{f_2}[k]$ and $\boldsymbol{\lambda}_{f_2,f_1}[k]$ by **Equation 3.27**. Then, the newly updated slack variable and dual variable will be transmitted to their neighbors. Finally, each drone will extract and implement the first optimal inputs $\boldsymbol{u}_i^*[k|k]$ from $\boldsymbol{y}_i[k]$ and go to the next time step.

Algorithm 2: ADMM-based Distributed MPC

Initialization: at time k_0
  Agent i assumes $\tilde{\boldsymbol{v}}_L[0]$, $\tilde{\boldsymbol{v}}_j[0]$, $\tilde{\boldsymbol{z}}_{j,i}[0]$ and $\tilde{\boldsymbol{\lambda}}_{j,i}[0]$ all to be $\mathbf{0}$, $j \in \mathcal{N}_i$.
Main loop: at any time k, $k > k_0$
  Each robot estimates $\boldsymbol{\xi}'[k]$ by EKF.
  Follower i receives $\tilde{\boldsymbol{z}}_{j,i}[k-1]$ and $\tilde{\boldsymbol{\lambda}}_{j,i}[k-1]$ from neighbor j.
  Each agent i updates $\boldsymbol{y}_i[k]$ by **Equation 3.23**.
  Each agent i sends $\boldsymbol{v}_i[k]$ to their neighbors.
  Follower i receives $\tilde{\boldsymbol{v}}_j[k]$ from neighbor j.
  Each follower update the slack variable and dual variable according to **Equations 3.24 to 3.27**
  Follower i transmits $\boldsymbol{z}_{i,j}[k]$ and $\boldsymbol{\lambda}_{i,j}[k]$ to follower$j$.
  Each agent i apply the first computed control input $\boldsymbol{u}_i^*[k|k]$.
Terminate when leader reaches the final destination.

3.3 PSO-based Distributed MPC

The third type is similar to the velocity sharing based Distributed MPC architecture, the main difference is that Particle Swarm Optimization (PSO) is used to solve the optimization problem. PSO is a population-based meta-heuristic optimization algorithm proposed in [15]. Each particle in PSO is considered to be a potential solution, and navigates with an assigned randomized velocity through the search space. The position of each particle is updated iteratively depending on the experiences of it and its neighbors, which are called personal best (pbest) and global best (gbest).

To distinguish the notations in the previous section, the notation used in this section is a bit different from those are commonly used to describe PSO algorithm. Let $\boldsymbol{r}_l^i$ and $\boldsymbol{h}_l^i$ denote the position and velocity of the ith particle, $\boldsymbol{p}_l^i$ represents its personal best solution and $\boldsymbol{p}_l^g$ is the global best in the swarm. The subscript l here is the index indicating the iteration number. At each iteration, the velocity and position of each particle are updated as following:

$$\begin{aligned} \boldsymbol{h}_{l+1}^i &= \omega_l \boldsymbol{h}_l^i + c_1 R_1(\boldsymbol{p}_l^i - \boldsymbol{r}_l^i) + c_2 R_2(\boldsymbol{p}_l^g - \boldsymbol{r}_l^i) \\ \boldsymbol{r}_{l+1}^i &= \boldsymbol{r}_l^i + \boldsymbol{h}_{l+1}^i \end{aligned} \tag{3.28}$$

where ω_l is the inertia, c_1 and c_2 are the acceleration coefficients, R_1 and R_2 are the diagonal matrices where each diagonal element is a random number uniformly generated within $[0, 1]$. After generating the new position of each particle, their fitness will be reevaluated and the personal best and the global best will be updated.

Consider the centralized optimization problem described in **Equation 3.4 and 3.5**, each candidate particle will represent the predicted control input sequence of the leader and the two followers. In that case, the dimension of

one particle would be a large number and PSO will require more iterations to reach a good solution. In order to distribute the centralized problem, three swarms with each representing the control input sequence of a single agent are generated and used to search for the optimized solution in each agent. According to [11], The conventional PSO-based MPC approach decompose the coupled dynamic by communicating the global best particle, i.e. the optimal input sequence, among the agents. As discussed in **Section 3.1**, only sharing predicted velocity is sufficient and can reduce the computational complexity in each subsystem. Hence, only velocity will be shared in our PSO-based Distributed MPC.

Illustrated in Algorithm 3, At each time step, all agents first estimate their current state by EKF. Then, the followers receive the predicted velocity of their neighbors $\bar{\boldsymbol{v}}_j[k-1]$ via communication and obtained $\hat{\boldsymbol{v}}_{i,j}[k]$ using same method as the first type. Based on these information, the centralized optimization is separated into three sub-problem as stated in **Equation 3.9** and **Equation 3.10**. Each agent will generate a swarm and search for the optimal solution of its corresponding sub-problem by PSO. After satisfying a terminal criterion, PSO will output the global best particle with minimum cost function value and that will be the optimal input sequence $\boldsymbol{u}_i^*$. Based on the input sequence, each robot will predict their future velocity $\bar{\boldsymbol{v}}_i[k]$ by the **dynamic equation 2.3** and transmit it to its neighbors. Finally, both leader and followers apply the first of the input sequence, i.e. $\boldsymbol{u}_i^*[k|k]$ in the real drone and turn to the next time step.

Algorithm 3: PSO-based Distributed MPC

Initialization: at time k_0
  Each follower i assumes $\hat{\boldsymbol{v}}_{i,j_1}[0] = \mathbf{0}, \hat{\boldsymbol{v}}_{i,j_2}[0] = \mathbf{0}$.
Main loop: at any time k, $k > k_0$
  Each robot estimates $\boldsymbol{\xi}[0]$ by EKF.
  Each follower receives $\bar{\boldsymbol{v}}_{j_1}[k-1], \bar{\boldsymbol{v}}_{j_2}[k-1]$.
  Each follower i predicts $\hat{\boldsymbol{v}}_{i,j_1}[k], \hat{\boldsymbol{v}}_{i,j_2}[k]$ by **Equation 3.8**.
  All robots generate a swarm of candidate particles and run PSO on the robot's own optimization problem as stated in **Equation 3.9** or **Equation 3.10**.
  Each agent i apply the first computed control input $\boldsymbol{u}_i^*[k|k]$.
  Each drone i transmit $\bar{\boldsymbol{v}}_i[k]$ to their neighbors.
Terminate when leader reaches the final destination.

3.4 Decentralized MPC

Formulating and implementing the decentralized MPC structure is not part of this semester project, but such controller will be used to compare with the best type of Distributed MPC in the Webots-ROS simulater. Hence, the idea of the Decentralized MPC will be briefly introduced in this section.

In decentralized MPC, all agents are totally separated. It is assumed that there is no explicit communication between agents. All the state of the agent and their neighbors are coming from sensor and EKF. Different from the distributed MPC, followers cannot obtain the neighbors future velocity by communication in Decentralized MPC. Therefore, the strategy to decouple the coupled state of relative position is to estimate the neighbors' current velocity by EKF at each time step. Then, assuming the neighbor will moves in constant velocity in the future, i.e. $\hat{\boldsymbol{v}}_{i,j}[k+n|k] = \boldsymbol{v}_j[k|k],\ \forall n \in [0, N]$.

The rest strategy is very similar to that of the velocity sharing based Distributed MPC, which can be referred to the work by Erunsal et al. [4].

Chapter 4 Experiments and Results

Several experiments have been simulated to compare the performance of different types of Distributed MPC and Decentralized MPC. The simulation experiments on the three types of Distributed MPC have been carried out in MATLAB 2020b with an Intel i7-8550U processor. In Webots-ROS simulation, the Webots version is R2020b and ROS melodic 1.14.11 has been used.

4.1 Metrics

Before showing the results, several metrics are introduced to compare the different types of controller.

- The overall average relative formation error e is defined as:

$$e = \frac{1}{N}\sum_{k=1}^{N}\frac{1}{M}\sum_{i=1}^{M}\sum_{j\in\mathcal{N}_i}\frac{||\boldsymbol{x}_i(k)-\boldsymbol{x}_j(k)-\Delta\boldsymbol{x}_{i,j}^d(k)||_2}{||\Delta\boldsymbol{x}_{i,j}^d(k)||_2} \tag{4.1}$$

where N is the total number of time steps, M is the number of followers (in our case M is 2), $\boldsymbol{x}_i(k)$ is the position of drone i at time step k and $\Delta\boldsymbol{x}_{i,j}^d(k)$ is the desired relative displacement between drone i and j.

- The average relative formation error at time step k is defined as:

$$e(k) = \frac{1}{M}\sum_{i=1}^{M}\sum_{j\in\mathcal{N}_i}\frac{||\boldsymbol{x}_i(k)-\boldsymbol{x}_j(k)-\Delta\boldsymbol{x}_{i,j}^d(k)||_2}{||\Delta\boldsymbol{x}_{i,j}^d(k)||_2} \tag{4.2}$$

- The definition of the formation maintenance error $e_{ij}(k)$ is:

$$e_{ij}(k) = ||\boldsymbol{x}_i(k)-\boldsymbol{x}_j(k)-\Delta\boldsymbol{x}_{i,j}^d(k)||_2 \tag{4.3}$$

- $P_{ij}(\epsilon)$ represents the probability of degradation. A time step is defined as degradation when the relative formation error exceeds threshold ϵ at that time step. The formal definition is:

$$P_{ij}(\epsilon) = \frac{\sum_{k=1}^{N} X_{k,\epsilon}}{N} \tag{4.4}$$

where $X_{k,\epsilon}$ is an indicator function:

$$X_{k,\epsilon} = \begin{cases} 1 & e_{ij}(k) > \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

4.2 Simulation in MATLAB

4.2.1 Experimental Setup

The ACADO toolkit with code generation by Houska et al. [16] is selected as the solver for the velocity sharing based Distributed MPC. Regarding to the ADMM-based Distributed MPC, the primal variable update as stated in **Equation 3.23** is a nonlinear programming and is solved by the **fmincon** function in **Optimization Toolbox** of MATLAB. The slack variable update process in **Equation 3.24 and 3.25** is a Quadratic Programming (QP) problem. This is solved the **quadprog** function in **Optimization Toolbox** of MATLAB. Besides, experiments on velocity sharing based Distributed MPC using **fmincon** as the solver have also been operated in order to fairly compare the two architectures. Finally, the PSO-based Distributed MPC select **particleswarm** function in **Global Optimization Toolbox** of MATLAB as the solver.

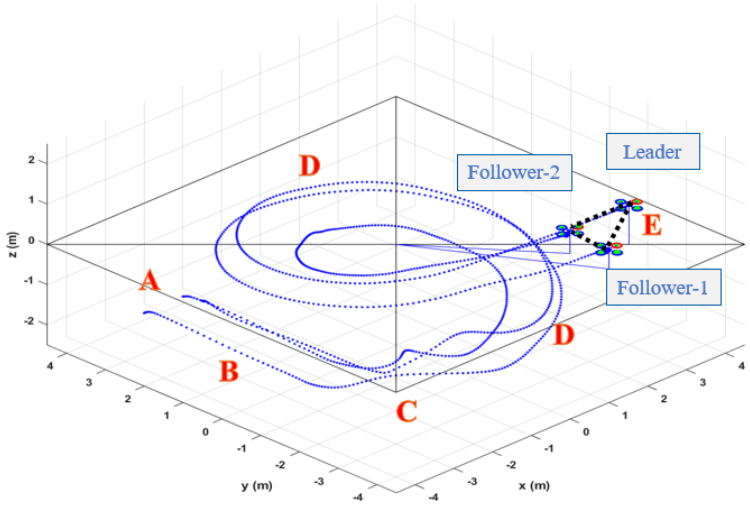


Figure 4.1: Reference trajectory in MATLAB. A: Starting point, B: Linear motion (2 m/s), C: Agile movement with a sudden turning and increase of height, D: Spiral motion (1.88 m/s), E: Destination.

The scenario we simluate in MATLAB consists of three vehicles, one

serving as leader and the other two as followers. There's no obstacle in the scenario and the reference trajectory we run is shown in **Figure 4.1**. As mentioned in the previous sections, all sensors have zero-mean noise. The standard deviation of the noise and other selected simulation parameters can refer to **Table 4.1**. To test the robustness to communication inaccuracy, communication delay and packet losses has been implemented and their definition is as belows:

$$\begin{aligned} &y_{rx}(k) = y_{tx}(k-\tau) \\ &P(y_{tx}(k) \text{ isn't received}) = \theta \end{aligned} \tag{4.6}$$

where $y_{rx}(k)$ denotes the information received in time step k, $y_{tx}(k)$ denotes the information transmitted in time step k, τ is the delay time and θ is packet loss probability. The experiments with different number of time steps delay have been simulated and there's a 5% of packet loss in every experiment. We run each simulation with different controller and delay 5 times in total. The mean and standard deviations are depicted in the following section.

Table 4.1: Selected simulation parameters.

General Parameters	**Value**	**Unit**
Vehicle lenth and mass	0.21, 1.37	m, kg
Duration of each tiem step	0.05	s
Fomration ref. F1 w.r.t L	[-1 -0.5 -0.5], 0	m, rad
Formation ref. F1 w.r.t. F2	[0 -1 0], 0	m, rad
Formation ref. F2 w.r.t. L	[-1 0.5 -0.5], 0	m, rad
Std. dev. of optic flow noise	0.25	m/s
Std. dev. of IMU noise (Attitude)	0.005	deg
Std. dev. of gyro noise	1	deg/s
Std. dev. of rel. loc. unit noise	0.025, 1	m, deg
MPC horizon length	15	-
Initial and running KKT tolerance for ACADO	10^{-3}, 10	-
Function tolerance (fmincon, particleswarm)	10^{-2}, 10^{-2}	-

4.2.2 Results and Discussion

The overall average relative formation error with different constant delay can be seen from **Table 4.2.2**. With perfect communication (no delay exists), the ADMM-based Distributed MPC has less overall average formation error than the velocity sharing based Distributed MPC if using same solver. However, the first type using ACADO code generator as the solver has the least formation among all the candidates, while PSO performs worst. The evolution of the average relative formation error without the presence of communication problem is illustrated in **Figure 4.2**. In most of time, the

PSO-based type has the greatest mean of relative error and the standard deviation is also higher than the rest. The velocity sharing type solved with fmincon performs the second worst. The mean of relative error in ADMM-based type is sometimes close to the first type with ACADO, but sometimes it has higher error.

Table 4.2: Mean of overall average relative formation error of the five simulation experiments with different constant delay.

Delay (time steps)	**Vel-share (ACADO)**	**Vel-share (fmincon)**	**ADMM-based**	**PSO-based**
0	0.1164	0.1834	0.1554	0.2125
3	0.1273	0.1764	0.1923	0.2155
5	0.1404	0.1784	0.2398	0.2056
7	0.1565	0.1823	0.2794	0.2042
10	0.1911	0.1928	0.3512	0.2100
13	0.2131	0.2093	0.4153	0.2309
15	0.2387	0.2175	0.4602	0.2376

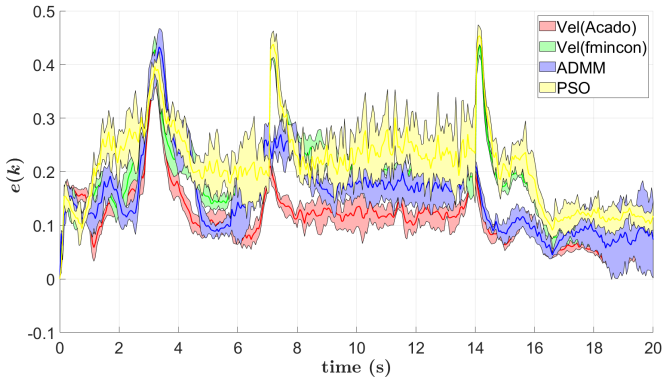


Figure 4.2: Average relative formation error with 0 delay.

The error norms of the relative position vectors of each pairs of the neighbors are given in **Figure 4.3**. With the detailed figure, the main difference is on the leader to follower formation maintenance where we can clearly see the outperformance of the ADMM-based type and velocity sharing type with ACADO. Regarding to the follower to follower formations, the four controllers have very close error.

Such phenomenon can also be observed in **Figure 4.4**. The percentage of degradation time for leader to leader formation is very close for the four controllers even with different thresholds. However, in the leader to follower, the advantage of velocity sharing with ACADO and ADMM-based type is very clear.

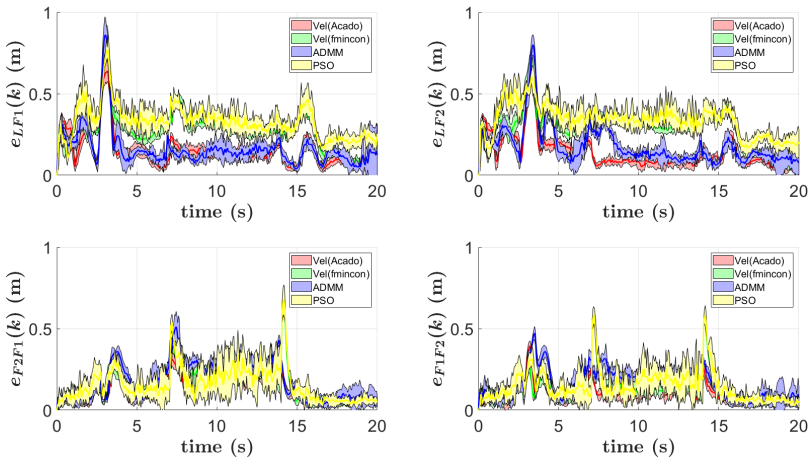


Figure 4.3: Formation maintenance errors for leader to follower-1, follower-2 to follower-1, leader to follower-2 and follower-1 to follower-2.

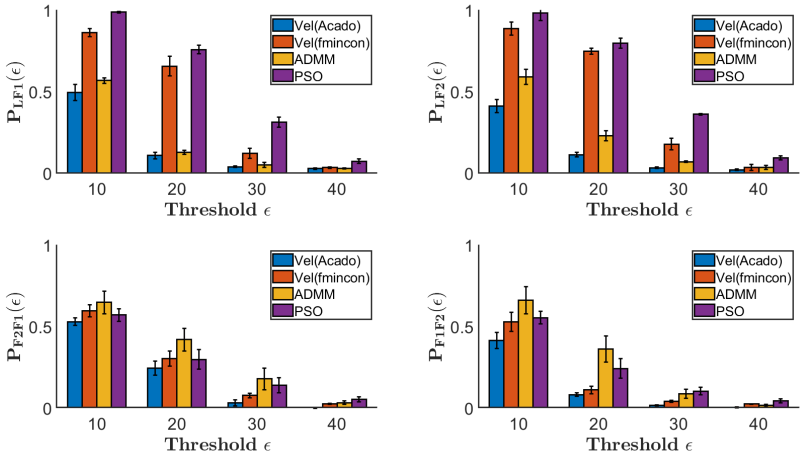


Figure 4.4: Percentage of degradation time with different thresholds for leader to follower-1, follower-2 to follower-1, leader to follower-2 and follower-1 to follower-2.

According to the performance of the first type with different solvers, fmincon solver is far less accurate than the ACADO toolkit with code generation. With the same solver, the ADMM-based has better performance than the velocity sharing one as the complex nonlinear optimization problem is separated into two parts to solve and more communication is involved in this ADMM-based type. Regarding to PSO-based type, both the mean and the standard deviation of the error is higher than the rest. This is probably because PSO is a stochastic optimization algorithm and is not suitable in this kind of problem where the deterministic solver can work.

As illustrated in **Figure 4.5**, the computational time of PSO-based is the highest. Even though the whole centralized problem is separated to three sub-problems, the dimension of each particles is still very large and PSO requires each candidate to randomly search in the possible region, which results in the high computational time it requires.

The first two types using fmincon as the solver uses less time to find the optimal solution. The average of computational time for ADMM-based is 0.6191s and that for velocity sharing is 0.1597s. However, that is still not sufficient to implement in the real case as the duration of each time step is 0.05s. Velocity sharing type with ACADO, whose average time is 0.0013s, can satisfy that requirement.

This is because in Acado, we export highly efficient and self-contained C code that is tailored to our MPC problem formulations. Computational speed is increased by hard-coding all problem dimensions, avoiding dynamic memory allocations, loop unrolling, symbolic simplifications and the use of a fixed-step integrator. This leads to significant speed-ups compared to generic implementations [17, 18].

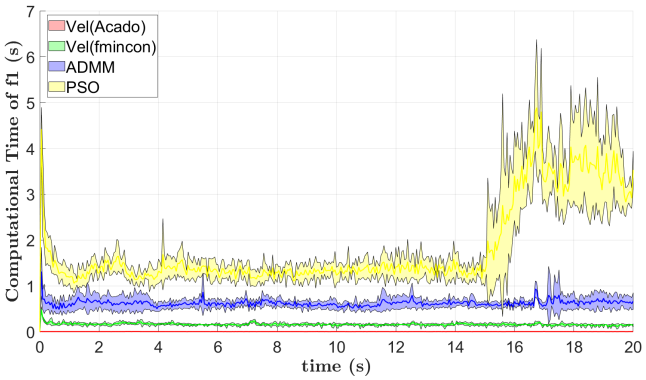


Figure 4.5: Computational time of follower-1 at each time step.

The mean and standard deviations of the overall average formation error with different communication delay are depicted in **Figure 4.6**. The error of the ADMM-based MPC increased significantly with the communication

delay while the error of the rest three doesn't suffer huge increase. It's quite reasonable as the information shared in this type is far more than the rest, which makes it very sensitive to the communication quality.

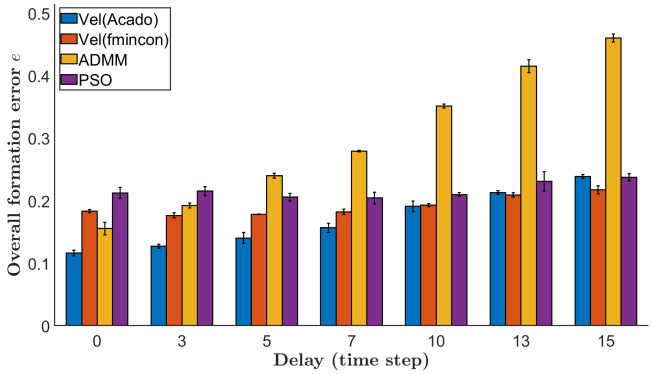


Figure 4.6: Overall average relative formation error with different communication delay (constant).

In summary, the PSO-based MPC has the worst performance on formation maintenance though it is very robust to the communication delay. The ADMM-based MPC has better performance on formation control than the velocity sharing based type. However, it is very sensitive to the communication quality and requires more computational time. Moreover, the velocity sharing type using ACADO code generate has far better performance than the ADMM-based type even with perfect communication. Considering both architecture performance and solver ability, we will choose the velocity sharing based one to simulate in the Webots-ROS framework. But, the ADMM-based Distributed MPC is also promising.

4.3 Simulation in Webots and ROS Framework

4.3.1 Experimental Setup

In this Webots-ROS simulator, the best Distributed MPC scheme, velocity sharing based Distributed MPC, and the Decentralized MPC controller proposed by Erunsal et al. [4] have been compared. Both controller use the ACADO toolkit as the solver.

Similar to that in MATLAB simulation, the scenario here is also composed of three vehicles, one is the leader and the other two is the followers. No obstacle is involved. The reference trajectory is illustrated in **Figure 4.7 and 4.8**. All sensors have zero-mean noise. To keep the fidelity of the simulation, Distributed MPC with different delay are tested. Instead of the constant delay in MATLAB, delay is generated as the Gaussian distribution

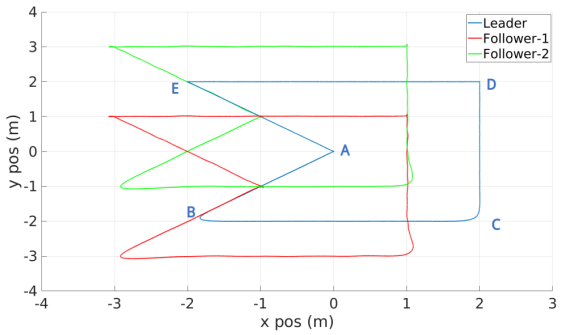


Figure 4.7: Top view of the reference trajectory in Webots-ROS. The leader starts with point A, does the way-point following in alphabetical order and comes back to point A. A to B takes 5 s and all rest segments take 12 s.

with standard deviation set to be 1, i.e. $\sigma = 1$ (sample time). Distributed MPC with perfect communication, 3-sample-time-mean delay (the cloest to real situation) and 10-sample-time-mean delay are tested.

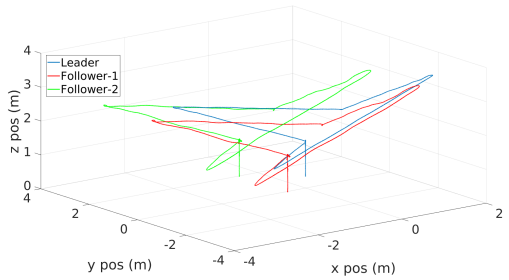


Figure 4.8: Orthographic view of the reference trajectory.

4.3.2 Results and Discussion

The overall average relative formation error is shown in Table 4.3.2. The error of velocity sharing based Distributed MPC increases with the mean of communication delay. Even when the the mean of the delay reaches 10 time step, which is nearly impossible in the reality, the overall formation error of Decentralized MPC is still higher than that of the velocity sharing based Distributed MPC.

Table 4.3: Overall average relative formation error.

-	**Decentralized**	**Distributed (0 delay)**	**Distributed (3 delay)**	**Distributed (10 delay)**
$\boldsymbol{e}$	0.0415	0.0261	0.0286	0.0334

Figure 4.9 shows the evolution of the average relative formation error. Except the time near 8 s, the average formation error of Decentralized MPC is higher than that of Distributed one even with presence of high delay.

The time when Distributed MPC has a higer error is corresponding to the time when the agents arrived the way-point B in **Figure 4.7** and start to do an agile turning. Since the followers can get the future velocity of their neighbors when controlled by Distributed MPC, they will know the future turning and the huge deviation of formation displacement on x,y axis and try to minimize that deviation as much as possible. In that case, they will care less about the deviation on z axis, resulting to huge formation error on z axis. The agents controlled by the decentralized MPC, however, don't have the future information of their neighbors. Thus, the formation error on z axis will be kept small.

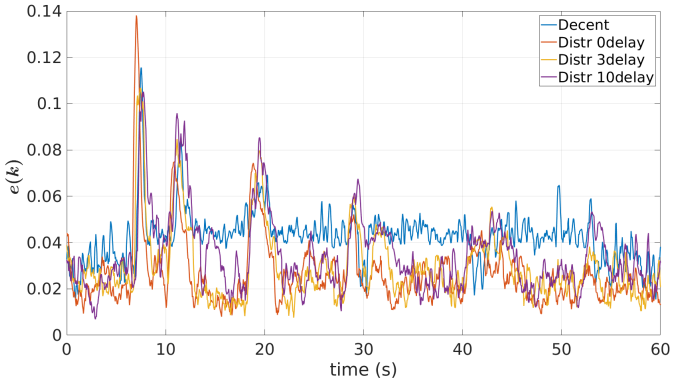


Figure 4.9: Average relative formation error in Webots-ROS.

Similar to our experience in the MATLAB simulation, the main difference of different controllers is in the leader to follower formation as we can see in **Figure 4.10**. Moreover, with the increase of communication delay, the performance of the Distributed MPC converges to that of the Decentralized MPC especially in the leader to follower formation maintenance.

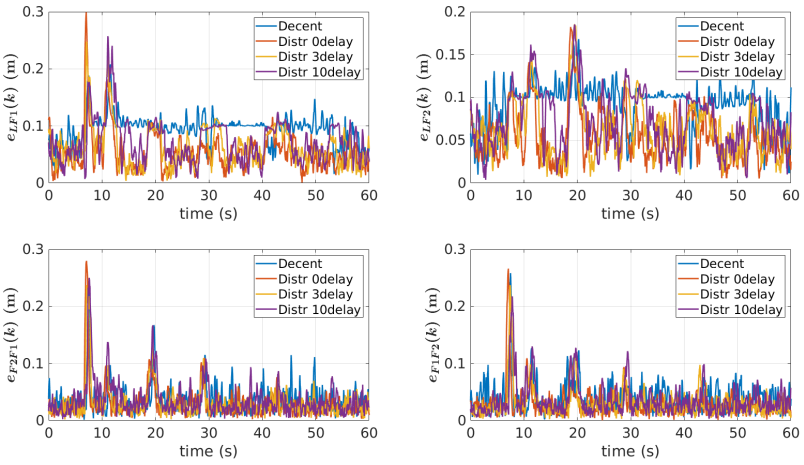


Figure 4.10: Detailed formation maintenance errors in Webots-ROS.

Chapter 5 Conclusion

In this project, three types of Distributed MPC architectures are theoretically formulated for the leader-follower formation control problem and implemented in MATLAB. According the simulation result in MATLAB, the ADMM-based type has the best formation maintenance if using the same solver and assuming perfect communication. Nevertheless, this type is very sensitive to the communication quality and its performance is worse than that of velocity sharing based type solved by ACADO. Considering both solver and architecture performance, the velocity sharing based Distributed MPC is selected to compare with the Decentralized MPC in the high-fidelity framework consisting of the Webots simulator and ROS. Despite some special moments, it turns out the Distributed MPC controller has less formation error than the Decentralized MPC controller most of time.

Though the velocity sharing based type is chosen, the competition between this type and ADMM-based type is unfair. When using the same solver the latter type has better performance in the condition of high communication quality. One possible future work is try to find some more efficient nonlinear programming solver to implement the ADMM-based type. Though the Webots simulator is already high-fidelity, the other future work is to implement these Distributed controllers in real drones and validating the results obtained in simulation experiments.

Bibliography

[1] M. Kamel, M. Burri, and R. Siegwart, "Linear vs nonlinear mpc for trajectory tracking applied to rotary wing micro aerial vehicles," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 3463–3469, 2017.

[2] U. Eren, A. Prach, B. B. Koçer, S. V. Raković, E. Kayacan, and B. Açıkmeşe, "Model predictive control in aerospace systems: Current state and opportunities," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 7, pp. 1541–1566, 2017.

[3] H. Ebel, E. S. Ardakani, and P. Eberhard, "Distributed model predictive formation control with discretization-free path planning for transporting a load," *Robotics and Autonomous Systems*, vol. 96, pp. 211–223, 2017.

[4] I. K. Erunsal, A. Martinoli, and R. Ventura, "Decentralized nonlinear model predictive control for 3d formation of multirotor micro aerial vehicles with relative sensing and estimation," in *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pp. 176–178, IEEE, 2019.

[5] A. J. Marasco, S. N. Givigi, C. A. Rabbath, and A. Beaulieu, "Dynamic encirclement of a moving target using decentralized nonlinear model predictive control," in *2013 American Control Conference*, pp. 3960–3966, IEEE, 2013.

[6] T. Chevet, C. Vlad, C. S. Maniu, and Y. Zhang, "Decentralized mpc for uavs formation deployment and reconfiguration with multiple outgoing agents," *Journal of Intelligent & Robotic Systems*, vol. 97, no. 1, pp. 155–170, 2020.

[7] W. B. Dunbar and R. M. Murray, "Distributed receding horizon control for multi-vehicle formation stabilization," *Automatica*, vol. 42, no. 4, pp. 549–558, 2006.

[8] J. Liu, X. Chen, D. Muñoz de la Peña, and P. D. Christofides, "Sequential and iterative architectures for distributed model predictive control

of nonlinear process systems," *AIChE Journal*, vol. 56, no. 8, pp. 2137–2149, 2010.

[9] R. Van Parys and G. Pipeleers, "Distributed mpc for multi-vehicle systems moving in formation," *Robotics and Autonomous Systems*, vol. 97, pp. 144–152, 2017.

[10] X. Hou, Y. Xiao, J. Cai, J. Hu, and J. E. Braun, "Distributed model predictive control via proximal jacobian admm for building control applications," in *2017 American Control Conference (ACC)*, pp. 37–43, IEEE, 2017.

[11] S.-M. Lee, H. Kim, H. Myung, and X. Yao, "Cooperative coevolutionary algorithm-based model predictive control guaranteeing stability of multirobot formation," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 1, pp. 37–51, 2014.

[12] R. Mahony, V. Kumar, and P. Corke, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor," *IEEE Robotics and Automation magazine*, vol. 19, no. 3, pp. 20–32, 2012.

[13] Z. Cai, H. Zhou, J. Zhao, K. Wu, and Y. Wang, "Formation control of multiple unmanned aerial vehicles by event-triggered distributed model predictive control," *IEEE Access*, vol. 6, pp. 55614–55627, 2018.

[14] S. Boyd, N. Parikh, and E. Chu, *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[15] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pp. 39–43, Ieee, 1995.

[16] B. Houska, H. J. Ferreau, and M. Diehl, "Acado toolkitâan open-source framework for automatic control and dynamic optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.

[17] B. Houska, H. J. Ferreau, and M. Diehl, "An auto-generated real-time iteration algorithm for nonlinear mpc in the microsecond range," *Automatica*, vol. 47, no. 10, pp. 2279–2285, 2011.

[18] H. J. Ferreau, T. Kraus, M. Vukov, W. Saeys, and M. Diehl, "High-speed moving horizon estimation based on automatic code generation," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pp. 687–692, IEEE, 2012.