Delft University of Technology
Master's Thesis in Embedded Systems

Autopilot Design for Software-in-the-Loop Validation of Fixed-wing UAV Guidance Laws

Arun Geo Thomas

TU Delft Delft University of Technology

Autopilot Design for Software-in-the-Loop Validation of Fixed-wing UAV Guidance Laws

Master's Thesis in Embedded Systems

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Arun Geo Thomas
agthomas@student.tudelft.nl

1st August 2019

Abstract

UnmannedAerial Vehicles(UAVs) have multi-domain applications and fixed-wing UAVs are a widely used class. There is ongoing research on topics in view to optimize the control and guidance of UAVs. This work explores the design, implementation and Software-in-the-Loop validation of an autopilot using adaptive guidance laws with emphasis on formation control of multiple fixed-wing UAVs. The work is done on Raspberry Pis in C++ which can be interfaced to standard autopilots as companion computers.

The work splits a mission given by the user into primitive missions and uses an adaptive vector field approach for following it. For formation control, the work implements a discretized version of the Model Reference Adaptive Control synchronisation laws for multi-agent systems. Simulations are done in a distributed setting with a server program designed for the purpose. The server program handles the user inputs and configurations of the UAVs.

Contents

List of Figures

Chapter 1

Introduction

Unmanned Aerial Vehicles (UAVs) or drones are flying electro-mechanical systems which can operate autonomously, or be operated by remote control, or by a combination of both [1][2]. UAVs are cutting-edge technology systems and have multi-domain applications ranging from disaster management missions to home delivery of goods [3][4][5]. With the advancement of technology, sophisticated UAVs are being developed; of which, fixed-wing UAVs are a widely used type [6]. Fixed-wing UAVs have extended flying hours in comparison to Rotary-wing UAVs, as fixed-wing UAVs support gliding in air with less energy consumption. However, their dynamics, actuation, take-off and landing involve complicated aerodynamics and mathematics [7][8]. Figure 1.1 depicts Fixed-wing and Rotary-wing type of UAVs. Control and guidance of UAVs is an active area of research and this work also primarily focusses on the same. In this chapter, we introduce the problem statement and the structure of the thesis.

1.1 Research Problem Introduction

According to [9], formation control tries to drive multiple agents to follow a set of constraints on their states. In the application level, formation control enables a set of agents or UAVs to fly in a pre-specified spacial formation (Y,T,V) in the inertial frame to execute a mission. For each type of the formation, we can obtain formation gaps that the UAVs should maintain with each other. For flying, fixed-wing UAVs rely on aerodynamic lift generated due to the relative motion of UAV with air. Thus, unlike quad-rotor drones, fixed-wing UAVs cannot stay stationary in air and constantly needs to be in motion in-order to be airborne. Thus highly mathematical control algorithms are needed to achieve the formation. Also, the complicated lateral dynamics of fixed-wing UAVs add to the difficulty in achieving formation control through ordinary methods.

With emphasis on implementation, the formation control problem can be

(a) Fixed-wing UAV.

(b) Rotary-wing(Quad-copter) UAV.

Figure 1.1: Fixed-wing and Rotary-wing UAVs.
[Image source: Internet.]

seen as two sub-problems: 1) *path planning and following* - the ability to execute the inertial missions provided by the user 2) *synchronization* - the need to fly in the formation respecting the formation gaps. Thus as part of this work, we try to achieve an implementation which addresses both the sub-problems.

For the first sub-problem, by using guidance laws in literature for path planning and following, the component developed should be able to plan the path for the formation of a set of UAVs.

"Agent" is a multi-disciplinary term which can be defined as any entity which listens and acts on its own environment based on an autonomous logic [10]. UAVs fit into the above definition and qualify to be called an agent. Thus the works in literature for multi-agents system can also be applied to UAVs.

In multi-agent systems, synchronization is the term used to represent the idea of achieving the same objective by having information about neighbouring agents. Flying in formation can be considered as an extension of the same with the only difference in having the formation gaps. Thus using the autopilot component developed, the UAVs should be able to synchronize its states.

Hardware-in-the-Loop simulation (HITL) is a real-time approach to validate a designed algorithm or software by deploying them in the targeted hardware with simulated plant and feedback sensors [11]. On the other hand, Software-in-the-Loop (SITL) is an approach for validating the developed software or algorithm with the help of mathematical models.

Autopilot is a component used to control the trajectory or orientation of the aircraft by automated control mechanisms reducing the human effort [12]. The software component developed in this work fits in the definition of an autopilot component for UAVs. SITL and HITL simulations are widely used for the validation of autopilot software components in UAVs. The SITL

simulation of the implemented autopilot component and algorithms is in the purview of this work.

In a nutshell, as part of this work, a design, implementation and SITL simulation of an autopilot component giving emphasis to formation control is required. For the simulations, the multiple UAVs in the system should have a synchronized output while executing the missions provided by the user. The simulations need to be done in a distributed environment extendable to HITL or real-word scenarios.

1.2 State of the Art

Autopilot components, comprising of both software and hardware, enable the basic control and navigation of UAVs [13][14]. Developers can use the framework and support materials to develop advanced tailor-made applications. The autopilot components provide Application Programming Interfaces (APIs) to extend the ability of autopilot components. The established autopilot components can handle multiple UAVs at a time, but each UAV is treated individually with a separate user input. An autopilot component which handles UAVs operations as a group is absent in the existing implementations. This poses the question: *How the existing autopilot components can be extended to execute formation flight enabling operations as a group with a single user input?*

For path planning of UAVs, the complex missions for a UAV can be split into primitive missions which can be individually executed[15]. The path following algorithms generate a commanded course angle which will gradually put the UAV into the required mission trajectory. The authors in [16] do a survey of path following strategies for UAVs. The geometric methods for UAV path following uses a virtual target point (VTP) along the path and follow it with pure pursuit or Line of Sight (LOS) guidance laws. Carrot chasing algorithm is the simplest of all and makes the UAV constantly chase a VTP named as carrot using the guidance laws. Nonlinear Guidance Law (NLGL) uses a different approach for calculation of VTP and the stability is established by Lyapunov analysis. The advantage of NLGL is that it works independent of the type of the path. Pure pursuit and LOS-based path following (PLOS) uses the pure pursuit for driving the UAV close to the waypoint, and the LOS guidance law for directing the UAV towards the LOS. Vector Field (VF) based approaches define a field of commanded course angle for each point around the paths. By following the commanded angle at each point, the UAVs can track the path gradually with the lowest cross track error of all algorithms [17]. The adaptive vector field approach proposed in [18] improves the VF approach by estimating the ground velocity used in the control law. This also accounts for the unmodelled dynamics of first order approximations assumed for the VF method [19][20]. But, *Soft-*

ware implementation and SITL validation of adaptive vector field approach is an open question.

Work in [21][22][23][24][25][26] are towards addressing the problem of multi-agent synchronization. For formation control of fixed-wing UAVs, [27] proposed a graph based method using linearised model and a game theory based Riccati solution. An artificial potential field based approach is proposed for formation flight in [28]. The approach doesn't consider the uncertainties in UAV parameters and uses a linearised model for simulation. In [23], a Model Reference Adaptive Control (MRAC) graph based approach is introduced. Parameters such as mass, inertial tensor etcetera of UAVs can be uncertain because of three reasons: 1) errors in measurements, 2) variation of parameters due to physical changes in each UAV, and 3) parameter variation from UAV to UAV due to manufacturing imperfections. Adaptive control laws have the advantage of online estimation of control law parameters, and are thus independent of parameter uncertainties. The work in [23] proposes continuous-time estimation dynamics and simulates it using MATLAB ordinary differential equation (ode) solver, which is far from the reality of distributed discrete-time digital systems communicating through a network. Thus we frame the question, *Can MRAC adaptive synchronization work in a distributed setting with periodic inter-UAV communications through a digital network?*

[29][30] gives analysis and approach for simulating the dynamics of UAVs. But for formation control there involves a team of UAVs. Xplane and Gazebo are established simulators for UAVs. Both Xplane and Gazebo execute in a centralized setting and does not support HITL simulation of multiple fixed-wing UAVs. For quad-copters, *Simulation in Hardware* is a comparatively new approach and is in the very early stages of development by the PX4 Development Team. The idea is to simulate drone dynamics in the autopilot hardware itself, instead of depending on a separate machine for the simulation of the plant dynamics. Motivated from this idea, we frame the question, *Can the fixed-wing UAV plant dynamics be simulated in the accompanying hardware itself?*

1.3 Research Objective

Though the research in control and guidance of UAVs are intense, there are no autopilot components which has a design and implementation emphasising formation control of fixed-wing UAVs. The objectives of this work are:

- Design, implement and simulate an autopilot component with emphasis on formation control of fixed-wing UAVs.
- Software-in-the-Loop validation of adaptive vector field approach for

path following of fixed-wing UAVs.

- Validate MRAC adaptive synchronisation for multi-agent systems in a distributed setting.
- Simulate the dynamics of multiple fixed-wing UAVs.

1.4 Report Outline

The organization of this report is as follows:

- **Chapter 2** introduces the UAV as an agent, analyses the dynamics of fixed-wing UAVs, and describes the approach for simulating the dynamics.
- **Chapter 3** describes the approach for path planning and following for fixed-wing UAVs.
- **Chapter 4** discusses the approach for synchronization of multiple UAVs as multi-agent system.
- **Chapter 5** introduces the system architecture planned for the HITL and the reduced architecture for the simulations in this work. The chapter also discusses the software architecture for the implementation.
- **Chapter 6** introduces the protocol, approach and server design for exchange of data between nodes.
- **Chapter 7** presents the simulations and the results for the implementations done.
- **Chapter 8** concludes the work and points out the directions for future research.

Chapter 2

UAV as an Agent

This chapter describes the modelling of UAV as an agent and approach for the simulation of its dynamics. Section 2.1 introduces the important frames of references and Section 2.2 explains the wind triangle. Section 2.3 derives the Euler Lagrange dynamics of a UAV using basic mechanics. Section 2.4 expounds the forces and moments in an airborne fixed-wing UAV due to aerodynamics and propulsion. Section 2.5 introduces the low-level controllers introduced by autopilot software and modelling their dynamics. Section 2.6 models the fixed-wing UAV dynamics into components for the purpose of simulations. Section 2.7 deals with the approach for simulating fixed-wing UAV dynamics in a C++ environment.

2.1 Frame of References for UAVs

Frames of references help to uniquely locate and orient in space. In this section, we define the frames of references which are important for UAV dynamics. They are:

1. *The earth frame* $\mathscr{F}^e$: is a frame whose origin is fixed to a point on earth. The frame is assumed to be an inertial frame and having a flat $\mathbf{i}^e$-$\mathbf{j}^e$ plane. According to the NED convention, the unit vectors of this frame $\mathbf{i}^e$, $\mathbf{j}^e$, and $\mathbf{k}^e$ are directed towards north, east, and down respectively. In this work, we use the term *inertial frame* $\mathscr{F}^i$ also as a synonym to address this frame.

2. *The vehicle frame* $\mathscr{F}^v$: represents the inertial frame translated onto the center of mass of the vehicle. The frame is depicted in Figure 2.1

3. *The body frame* $\mathscr{F}^b$: is defined as shown in the Figure 2.1 and attached to the centre of mass of the vehicle. The roll axis $\mathbf{i}^b$ is aligned to the longitudinal direction from tail to head of the vehicle, pitch axis $\mathbf{j}^b$ to the transverse direction along wings, and yaw axis $\mathbf{k}^b$ perpendicular to the other two axes forming a right angled coordinate system.

4. *The stability frame* $\mathscr{F}^s$: The airspeed $\mathbf{V}_a$ is the relative velocity between aircraft and surrounding air. Angle of attack α is the angle airspeed makes to the $\mathbf{i}^b$-$\mathbf{j}^b$ plane. Stability frame is a frame obtained by rotating the body frame around $\mathbf{j}^b$ by the angle of attack α towards the direction of airspeed.

5. *The wind frame* $\mathscr{F}^w$: The airspeed vector may also not lie in the $\mathbf{i}^b$-$\mathbf{k}^b$ plane, and makes an angle to the plane called as side-slip angle β. By rotating the stability frame by angle β around $\mathbf{k}^s$ axes to have i axes aligned with the direction of airspeed vector, we obtain the wind frame.

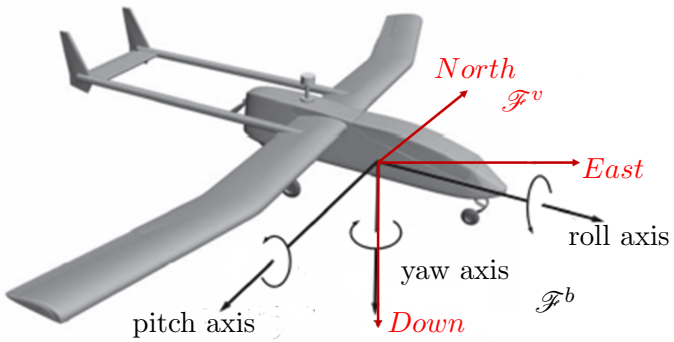


Figure 2.1: Fixed-Wing UAV body Frame.

The orientation of any given body frame relative to the vehicle frame can be represented using three rotations around body frame axes. The set of the angle measure for the three rotations are called as *euler angles* [31]. Figure 2.2 shows the three successive rotations for finding euler angles according to ZYX convention. The yaw rotation happens from the vehicle frame $\mathscr{F}^v$ to an intermediate body frame $\mathscr{F}^{b1}$ by an angle of ψ about k^v axis. The pitch rotation happens about j^{b1} axis of $\mathscr{F}^{b1}$ by an angle of θ. By an angle of ϕ, the roll rotation happens about the i^{b2} axis to obtain the body frame $\mathscr{F}^b$ that need to be represented. The euler angles are ϕ, θ, ψ.

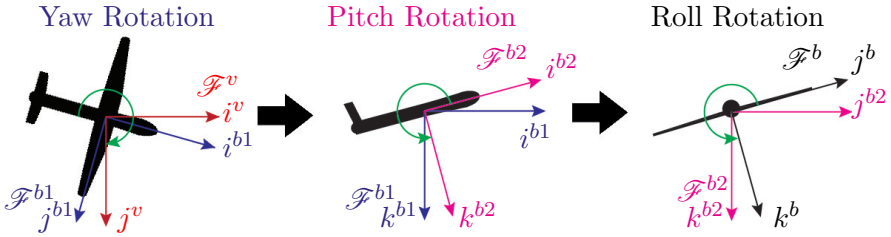


Figure 2.2: Elementary rotations defining euler angles (ZYX convention).

The relative orientation between any two frames can be included in calculations using a *rotation matrix*. Rotation about any frame axes by an angle

θ_{rot} to obtain a second frame relates the coordinates as,

$$p^1 = R_0^1(\theta_{rot})p^0 \tag{2.1}$$

where $p^0 \in \mathbb{R}^3$ and $p^1 \in \mathbb{R}^3$ are the coordinates of points in frame 0 and frame 1 respectively, $R_0^1 \in \mathbb{R}^{3\times3}$ is the rotation matrix and a function of the angle of rotation θ_{rot}.

Using the euler angle definition, the rotation matrix from earth frame (or vehicle frame) to body frame, can be written as,

$$\mathcal{R}_e^b(\phi,\theta,\psi) = \mathcal{R}_{b_2}^b(\phi)\mathcal{R}_{b_1}^{b_2}(\theta)\mathcal{R}_e^{b_1}(\psi) \tag{2.2}$$

Thus, the rotation matrix from earth to body frame $\mathcal{R}_e^b$ is obtained as,

$$\begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & \sin\phi\cos\theta \\ \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi & \cos\phi\cos\theta \end{bmatrix} \tag{2.3}$$

An interesting property of rotation matrices is given below,

$$\mathcal{R}_b^e = (\mathcal{R}_e^b)^{-1} = (\mathcal{R}_e^b)^T \tag{2.4}$$

2.2 Wind Triangle and Course Angle

The effect of wind on UAV dynamics can be calculated using the so called wind triangle, as shown in Figure 2.3 [19]. In vector sense, the wind triangle can be written as,

$$\mathbf{V}_g = \mathbf{V}_a + \mathbf{V}_w \tag{2.5}$$

where $\mathbf{V}_a$ is the velocity of aircraft relative air known as airspeed, $\mathbf{V}_w$ is wind velocity, and $\mathbf{V}_g$ is the velocity of aircraft with respect to ground called *ground velocity*. The angle between projection of $\mathbf{V}_g$ in xy plane of $\mathscr{F}^i$ and north direction is called as *course angle* χ.

2.3 Euler Lagrange Dynamics of UAV Agents

A UAV as an agent, perceives the relative location and orientation respective to the earth frame using onboard sensors and implement control actions using actuators to achieve a mission in its environment. Here we model the Euler Lagrange (EL)dynamics for a UAV. An airborne UAV has six Degrees Of Freedom (DOF) of a rigid body - three for translation and three for rotation.

Let $m \in \mathbb{R}$ be the mass of the UAV, $X_e = [x, y, x]T \in \mathbb{R}^3$ is the inertial position of the body, $V_b = [u, v, w]^T \in \mathbb{R}^3$ is the linear velocity of body

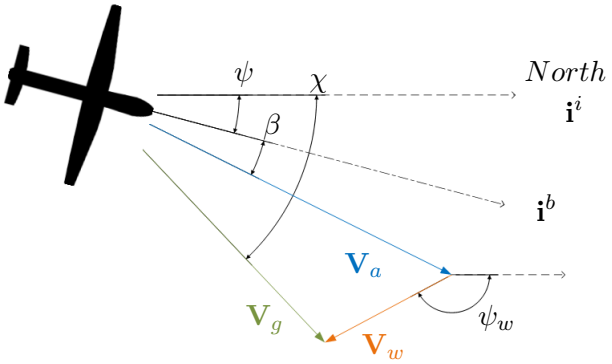


Figure 2.3: Wind triangle for a UAV.

expressed in body frame, $E = [\phi, \theta, \psi]^T \in \mathbb{R}^3$ are the euler angles of the body frame from the inertial frame, $\omega_b = [p, q, r]^T \in \mathbb{R}^3$ be the angular velocity of body in body frame, $F_b \in \mathbb{R}^3$ be the net forces acting on the body expressed in body frame, and $M_b \in \mathbb{R}^3$ be the net torque acting on the body expressed in body frame.

The equation of motion in inertial frame is given by Newton's third law of motion as,

$$m\dot{V}_e = F_e \tag{2.6}$$

where $V_e \in \mathbb{R}^3$ is the linear velocity of the body in inertial frame, and $F_e \in \mathbb{R}^3$ be the net forces acting in the body expressed in inertial frame.

Using results in [32], as explained in [33], the relation in body frame can be written as,

$$m\dot{V}_b + m(\omega_b \times V_b) = F_b \tag{2.7}$$

Let $\omega_e \in \mathbb{R}^3$ be the angular velocity of the body in inertial frame, $M_e \in \mathbb{R}^3$ be the torques acting on the body expressed in inertial frame, and $I \in \mathbb{R}^{3\times3}$ is the inertia tensor which is assumed to be constant. For rotational motion, by Euler's law we can write,

$$(I\dot{\omega}_e) = M_e \tag{2.8}$$

The dynamics of the rotational velocity in body frame [33]is,

$$I\dot{\omega}_b + \omega_b \times I\omega_b = M_b \tag{2.9}$$

Equations (2.7) and (2.9) constitutes the 6 DOF motion equation for an airborne UAV. Both equations can be written using the components of

vectors as,

$$\begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & I_x & 0 & -I_{xz} \\ 0 & 0 & 0 & 0 & I_y & 0 \\ 0 & 0 & 0 & -I_{xz} & 0 & I_z \end{bmatrix} \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} 0 & -mr & mq & 0 & 0 & 0 \\ mr & 0 & -mp & 0 & 0 & 0 \\ -mq & mp & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I_z r - I_{xz} p & -I_y q \\ 0 & 0 & 0 & -I_z r + I_{xz} p & 0 & I_x p - I_{xz} r \\ 0 & 0 & 0 & I_y q & -I_x p + I_{xz} r & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ p \\ q \\ r \end{bmatrix} = \begin{bmatrix} F_b \\ M_b \end{bmatrix} \tag{2.10}$$

F_b and M_b are the net forces and torques acting on the body, which includes the action of gravity. Gravitational force acts at the centre of gravity of any body. For small bodies like UAVs, the centre of gravity and the centre of mass are the same. Thus, the moment by gravitational action on the UAV is negligible.

Gravitational force always acts in the positive z-axis direction of the earth frame. Gravitational force on the body represented in earth frame is,

$$F_{ge} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{2.11}$$

Thus, gravitational force in body frame can be expressed as,

$$\begin{aligned} F_{gb} &= R_e^b F_{ge} \\ &= \begin{bmatrix} -mg\sin\theta \\ mg\sin\phi\cos\theta \\ mg\cos\phi\cos\theta \end{bmatrix} \end{aligned} \tag{2.12}$$

F_b and M_b can be re-written as,

$$\begin{bmatrix} F_b \\ M_b \end{bmatrix} = \begin{bmatrix} -mg\sin\theta \\ mg\sin\phi\cos\theta \\ mg\cos\phi\cos\theta \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \\ \tau_4 \\ \tau_5 \\ \tau_6 \end{bmatrix} \tag{2.13}$$

where $[\tau_1, \tau_2, \tau_3]^T$ and $[\tau_4, \tau_5, \tau_6]^T$ are the sum of forces and moments acting on the body other than gravity. Using (2.13) in (2.10), we obtain the Euler Lagrange dynamics for a UAV,

$$\underbrace{\begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & I_x & 0 & -I_{xz} \\ 0 & 0 & 0 & 0 & I_y & 0 \\ 0 & 0 & 0 & -I_{xz} & 0 & I_z \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix}}_{\ddot{q}} + \underbrace{\begin{bmatrix} 0 & -mr & mq & 0 & 0 & 0 \\ mr & 0 & -mp & 0 & 0 & 0 \\ -mq & mp & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I_z r - I_{xz} p & -I_y q \\ 0 & 0 & 0 & -I_z r + I_{xz} p & 0 & I_x p - I_{xz} r \\ 0 & 0 & 0 & I_y q & -I_x p + I_{xz} r & 0 \end{bmatrix}}_{C(\dot{q})} \underbrace{\begin{bmatrix} u \\ v \\ w \\ p \\ q \\ r \end{bmatrix}}_{\dot{q}}$$

$$+ \underbrace{\begin{bmatrix} sin\theta mg \\ -sin\phi cos\theta mg \\ -cos\phi cos\theta mg \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{g} = \underbrace{\begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \\ \tau_4 \\ \tau_5 \\ \tau_6 \end{bmatrix}}_{\tau} \tag{2.14}$$

The state-space representation of EL dynamics in (2.14) is,

$$\underbrace{\begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix}}_{\dot{x}} = \underbrace{\begin{bmatrix} 0 & \mathbb{1} \\ 0 & -D^{-1}C \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} q \\ \dot{q} \end{bmatrix}}_{x} + \begin{bmatrix} 0 \\ -D^{-1}g \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ D^{-1} \end{bmatrix}}_{B} \tau \tag{2.15}$$

2.4 Propulsion and Aerodynamic Effects

The actuation in a fixed-wing UAV is done by the propeller (throttle) and the control surfaces such as rudder, aileron and elevator. Propeller imparts forward thrust, rudder steers the aircraft, aileron controls the roll and elevator controls the pitch of the aircraft. Using the throttle input, the forces and moments of propulsion can be calculated. The forces and moments due to aerodynamics can be calculated with the deflections in rudder, elevator and aileron.

The force of propulsion can be calculated [19] using the formula,

$$F_p = \frac{1}{2}\rho S_p C_{prop}[(R\Omega)^2 - V_a^2] \tag{2.16}$$

where ρ is the density of air, S_p is the area swept by the propeller, C_{prop} is the rotor thrust coefficient, R is the radius of the propeller, $Omega = k_{motor}\delta_t + q_{motor}$ is the motor speed, k_{motor},q_{motor} are motor constants, V_a is the magnitude of velocity od UAV relative to air, and δ_t is the throttle input.

In UAVs, often the propeller is mounted in slightly slanting position to the $\mathbf{i}^b$-$\mathbf{j}^b$ plane of aircraft, but in $\mathbf{i}^b$-$\mathbf{k}^b$ plane. The angle by which the propeller is tilted is known as the mount angle. Let the mount angle be δ_{mount}, the force of propulsion in body frame is given by,

$$F_{pb} = \begin{bmatrix} F_p \cos\delta_{mount} \\ 0 \\ F_p \sin\delta_{mount} \end{bmatrix} \tag{2.17}$$

Force due to propulsion F_p is in $\mathbf{i}^b$-$\mathbf{k}^b$ plane. The turning moment due to propulsion M_{pb},

$$M_{pb} = r_{com-p} \times F_{pb} \tag{2.18}$$

where r_{com-p} is a vector from centre of mass to any point along the line of action of force F_{pb}. Also, r_{com-p} is expressed in body frame.

For longitudinal dynamics, the aerodynamic lift, drag and pitch moments acting in the $\mathbf{i}^b$-$\mathbf{k}^b$ plane expressed in stability frame can be calculated [19] using,

$$\begin{aligned} F_{lift} &= \frac{1}{2}\rho V_a^2 S C_L(\alpha, q, \delta_e) \\ F_{drag} &= \frac{1}{2}\rho V_a^2 S C_D(\alpha, q, \delta_e) \\ M &= \frac{1}{2}\rho V_a^2 S c C_m(\alpha, q, \delta_e) \end{aligned} \tag{2.19}$$

where S is the planform area, as defined in [34], of single wing, c is the main chord of the wing, and C_L, C_D, C_m dimensionless coefficients. The value of coefficients is determined by the angle of attack (α), the rate of pitch (q), and the deflection of elevator (δ_e). These forces in body frame can be written as,

$$\begin{bmatrix} f_x \\ 0 \\ f_y \end{bmatrix} = R_s^b(\alpha) \begin{bmatrix} -F_{lift} \\ 0 \\ -F_{drag} \end{bmatrix} \tag{2.20}$$

where R_s^b is the rotational matrix from stability to body frame.

For lateral aerodynamics, a force f_y directly acts in direction of $\mathbf{j}^b$, rolling moment L and yawing moment N written as [19],

$$\begin{aligned} f_y &= \frac{1}{2}\rho V_a^2 S C_Y(\beta, p, r, \delta_a, \delta_r) \\ L &= \frac{1}{2}\rho V_a^2 S b C_l(\beta, p, r, \delta_a, \delta_r) \\ N &= \frac{1}{2}\rho V_a^2 S b C_n(\beta, p, r, \delta_a, \delta_r) \end{aligned} \tag{2.21}$$

where b is the wingspan, C_Y, C_l, C_m are dimensionless aerodynamic coefficients, β is slide-slip angle, δ_a is deflection in ailerons, and δ_e is deflection in elevator.

Thus the aerodynamic force F_{aero-b} in body frame is,

$$F_{aero-b} = \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} \tag{2.22}$$

The net force and moment acting on the body are,

$$\begin{aligned} F &= F_{gb} + F_{pb} + F_{aero-b} \\ M &= M_{pb} + \begin{bmatrix} L \\ M \\ N \end{bmatrix} \end{aligned} \tag{2.23}$$

2.5 Autopilot Low-level Controllers in Fixed-Wing UAVs

Standard autopilots for fixed-wing UAVs realize low-level controllers such as [35], 1) yaw rate control, 2) pitch angle control, 3) roll angle control, 4) Total Energy Control System (TECS) [36] for height and velocity control.

The controllers are realized as discrete time control loops. The continuous time dynamics equations can be reverse-engineered from the discretized control loops of tuned autopilot controllers, which can be represented as,

$$\dot{x_c} = f_c(x_c, u_c) \tag{2.24}$$

where x_c is the states introduced by the controllers, and u_c is the control input comprising of height, velocity, course angle set points. The control surface (elevator, rudder, ailerons) deflections and throttle input can be written as a function of the state x_c, i.e.,

$$\begin{bmatrix} \delta_e \\ \delta_r \\ \delta_a \\ \delta_t \end{bmatrix} = g_c(x_c) \tag{2.25}$$

2.6 UAV Model for Simulations

For simulation of Unmanned Aerial Vehicle (UAV) dynamics, any UAV can be modelled into components as shown in Figure 2.4. Block 1 represents the motion of an object, in 3D space, having six degrees of freedoms under the effect of the net force and net moment acting on it. The ordinary differential equation can be written as in (2.7) and (2.9) for the motion using basic mechanics.

As discussed in previous sections, for an airborne body, the major forces and moments acting on the body are due to propulsion, aerodynamics and gravity. Block 2 calculates the net force and moments acting on an airborne body. Block 3 has the dynamics introduced by the low level controllers which generate throttle input and control surface deflections based.

In this work, for the path planner node simulation, we use the model which comprises of all the three blocks above. For the simulation of leader and

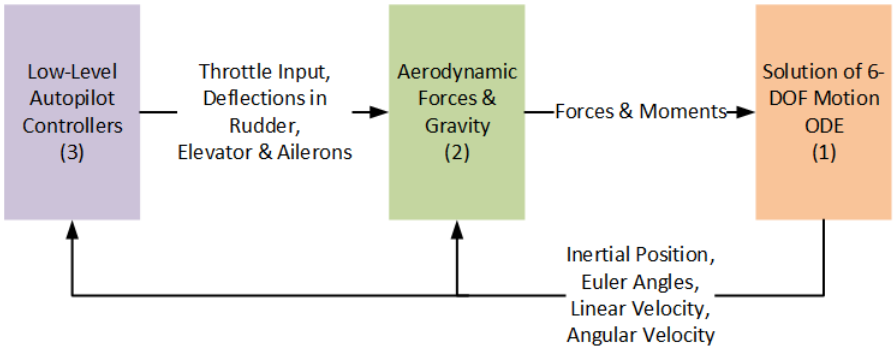


Figure 2.4: Modelling of UAV for simulations.

follower nodes, we use the model with only Block 1 which can be extended in future to accommodate for dynamics by Block 2 & Block 3.

2.7 Simulation of UAV Dynamics

Consider the state vector of the body,

$$X = \begin{bmatrix} X_e \\ V_b \\ E \\ \omega_b \end{bmatrix} \tag{2.26}$$

where all the symbols have meaning as defined in the previous section. In this section, we derive the expression for $\dot{X}$ for the body and establish a method in C++ to numerically solve $X(t)$ at any time instant t, for a known initial condition $X(0)$.

The time derivative of X can be written as,

$$\dot{X} = \begin{bmatrix} V_e \\ \dot{V}_b \\ \dot{E} \\ \dot{\omega}_b \end{bmatrix} \tag{2.27}$$

where $V_e \in \mathbb{R}^3$ is the linear velocity of the body in inertial frame.

Equation (2.7) can be rewritten as,

$$\dot{V}_b = \frac{F_b}{m} - \omega_b \times V_b \tag{2.28}$$

Also, The velocity of body expressed in inertial and body frames are related by,

$$V_e = R_b^e V_b \tag{2.29}$$

Equation (2.9) can be rearranged as,

$$\dot{\omega}_b = I^{-1}(M_b - \omega_b \times I\omega_b) \tag{2.30}$$

Given ϕ,θ,$\psi \in \mathbb{R}$ are the euler angles as per Z-Y-X convention, a relationship between euler rates and angular velocity w_b can be established [37] as,

$$\begin{aligned} \dot{w}_b &= \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \mathcal{R}^b_{b2} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \mathcal{R}^b_{b2}\mathcal{R}^{b2}_{b1} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \\ &= J^{-1} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \end{aligned} \tag{2.31}$$

where $\mathcal{R}$ is the rotational matrix, $b1$, $b2$ are the two intermediate frames in euler angles definition, and $J = \begin{bmatrix} 1 & (\sin\phi\tan\theta) & (\cos\phi\tan\theta) \\ 0 & \cos\phi & -\sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \end{bmatrix}$.

Thus,

$$\dot{E} = J\dot{w}_b \tag{2.32}$$

Using (2.29),(2.28), (2.32), and (2.30) in (2.27), we obtain the dynamics of X as,

$$\dot{X} = \begin{bmatrix} R^e_b V_b \\ \frac{F_b}{m} - \omega_b \times V_b \\ J\dot{w}_b \\ I^{-1}(M_b - \omega_b \times I\omega_b) \end{bmatrix} \tag{2.33}$$

Since the dynamics of X in (2.33) is non-linear, we can only solve for $X(t)$ numerically from the known initial condition $X(0)$.

'Odeint' is a C++ library for numerically solving ordinary differential equations [38], and provides various solvers like Runge-Kutta4, Dormand-Prince etc. The odeint solvers need a functor or a function having the dynamics as in (2.33). Thus the dynamics of the body is simulated in C++ using a functor and the odeint Dormand-Prince solver.

The C++ code for simulation implements a templated class *Drone* with dynamics in Equations (2.33) which takes force and moment inputs. Also, another template specialization of class *Drone* inherits from previous class *Drone* implementation and adds dynamics in (2.24) utilizing Equations (2.23) and (2.25). The simulations use the aerodynamic coefficients of a *Bixler* fixed-wing UAV, and the continuous time dynamics of the controllers in *ardupilot* autopilot tuned for it. This enables us to simulate the complete dynamics for a fixed-wing UAV.

Chapter 3

Path Planning and Following in Fixed-wing UAVs

This chapter deals with the approach for planning a path based on the user mission inputs and following the planned path.

Section 3.1 explains the strategy for path planning of a UAV to execute an inertial mission given by the user. Section 3.2 introduces the concept of vector fields and application of them in following a path. Section 3.3 describes the adaptive version of the vector field approach for following a path in fixed-wing UAVs.

3.1 Planning of Path Based on User Mission Inputs

In this work, *path planning* is defined as the process of deciding a trajectory the UAVs will traverse to execute a mission provided by the user. It is not in the sense of obstacle detection and detours for obstacle avoidance.

A mission is defined as a path or points the UAV should traverse while it is airborne. From existing UAV software, we list the three standard aerial missions the UAVs need to execute. They are:

1. *Straight Line Mission:* In this mission, the UAV needs to traverse a straight line in the inertial frame $\mathscr{F}^i$. The slope and a point through which the straight line passes defines the mission. The straight line mission can be written as an ordered pair,

$$\mathcal{SL} = (\mathcal{P}, \mathcal{S}) \tag{3.1}$$

where, $\mathcal{P} = (x_p, y_p)$ is a point in the xy plane of $\mathscr{F}^i$ called *line origin*, and $\mathcal{S} = (x_s, y_s)$ is a vector denoting the direction in the xy plane of $\mathscr{F}^i$ called *line slope*. Figure 3.1 depicts a candidate for a straight line mission.

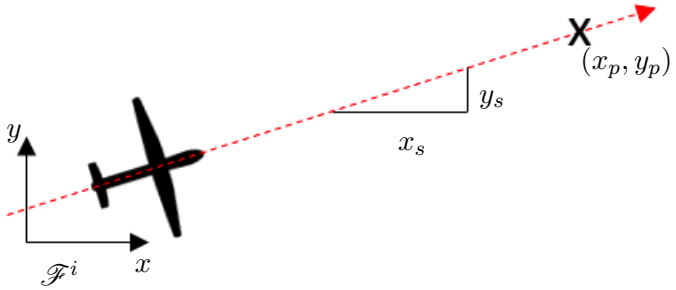


Figure 3.1: Straight line mission.

2. *Loitering Mission:* In a loitering mission, the UAV needs to continuously loiter in a circular orbit. The orbit is defined by the centre point and the orbit radius in the $\mathscr{F}^i$. The loitering mission can be written

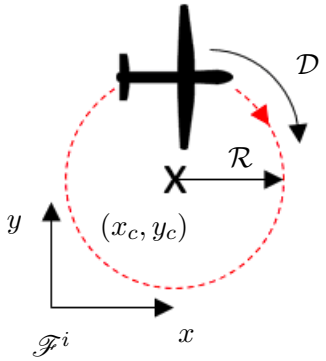


Figure 3.2: Loitering mission.

as a 3-tuple,

$$\mathcal{O} = (\mathcal{D}, \mathcal{R}, \mathcal{C}) \tag{3.2}$$

where, $\mathcal{D}$ is the direction of loitering (1 for clockwise and -1 for anti-clockwise), $\mathcal{R}$ is a distance in the xy plane of $\mathscr{F}^i$ called *orbit radius*, and $\mathcal{C} = (x_p, y_p)$ is a point in the xy plane of $\mathscr{F}^i$ called *orbit centre*, . An example for a loitering mission is shown in figure 3.2.

3. *Waypoints Mission:* The UAV needs to traverse through or fly close-by to a given set of points in inertial frame $\mathscr{F}^i$. The set of points defining these mission are called waypoints. The waypoints mission can be written as,

$$\mathcal{W} = (\mathcal{R}, \mathcal{P}) \tag{3.3}$$

where, $\mathcal{R}$ is a distance in the xy plane of $\mathscr{F}^i$ called *turn radius*, and $\mathcal{P}$ is a sequence of ordered pairs (x_n, y_n) such that (x_n, y_n) is a point

in the xy plane of $\mathscr{F}^i$ for $n = 1...N$. Figure 3.3 depicts an example of a mission with three waypoints.

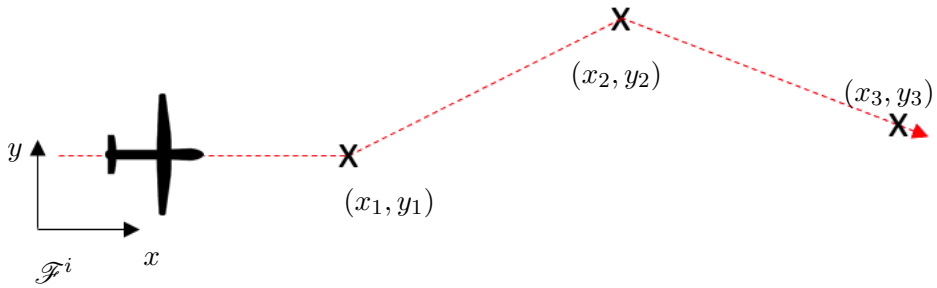


Figure 3.3: Waypoints mission.

Straight Line and Loitering Missions are basic (primitives) types. Every waypoints mission can be split into sub-missions of straight lines and loitering. In [15], the authors present an approach to split a given waypoints mission into primitive missions. Figure 3.4 shows the combinations of straight line and loitering missions that can be achieved using the approach. Dashed red lines connect the way points, and blue curve is the path planned for the UAV. Green circles show the circles to which the arcs in the path belong. In this work, we use the *Type II* combination for path planning.

For any three consecutive waypoints W_1, W_2 and W_3, we can split into primitive missions as shown in Figure 3.5. Let the current position of UAV be L, and turn radius be R. The primitive missions obtained are,

- *Mission 1 - Straight Line:* The mission is a straight line through current location and point M_1. In vector sense,

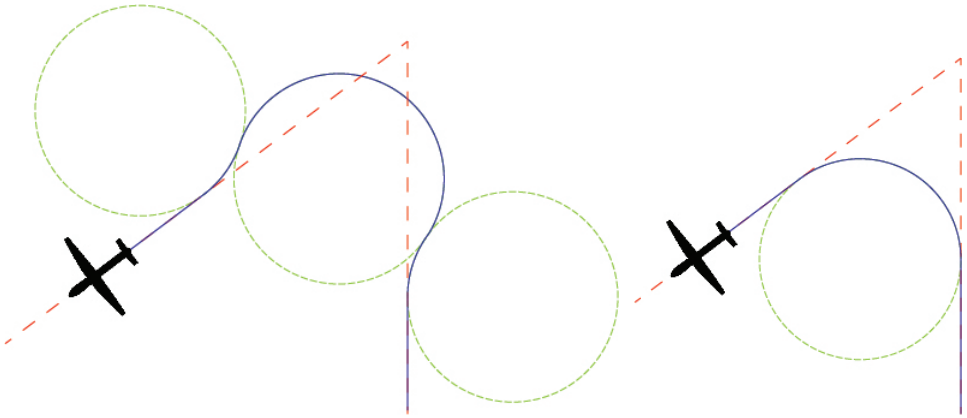
$$M_1 = W_1 + \frac{R}{tan(cos^{-1}(\frac{(L-W_1)}{|L-W_1|} \cdot \frac{L-W_1+W_2-W_1}{|L-W_1+W_2-W_1|}))} \frac{(L-W_1)}{|L-W_1|} \tag{3.4}$$

- *Mission 2 - Loitering:* A circular orbit centered at M_2 with a radius of R. In vector sense, M_2 can be written as,

$$M_2 = W_1 + \frac{R}{sin(cos^{-1}(\frac{L-W_2)}{|L-W_2|} \cdot \frac{L-W_1+W_2-W_1}{|L-W_1+W_2-W_1|}))} \frac{L-W_1+W_2-W_1}{|L-W_1+W_2-W_1|} \tag{3.5}$$

- *Mission 3 - Straight Line:* The mission is a straight line through points W_1 and M_3. In vector sense, M_3 can be written as,

$$M_3 = W_2 + \frac{R}{tan(cos^{-1}(\frac{(W_1-W_2)}{|W_1-W_2|} \cdot \frac{W_1-W_2+W_3-W_2}{|W_1-W_2+W_3-W_2|}))} \frac{(W_1-W_2)}{|W_1-W_2|} \tag{3.6}$$



(a) Type I. (b) Type II.

Figure 3.4: Combinations of straight line and loitering missions.

- *Mission 4 - Loitering:* A circular orbit centered at M_4 with a radius of R. In vector sense, M_4 can be written as,

$$M_4 = W_2 + \frac{R}{sin(cos^{-1}(\frac{W_1-W_2)}{|W_1-W_2|} \cdot \frac{W_1-W_2+W_3-W_2}{|W_1-W_2+W_3-W_2|}))} \frac{W_1 - W_2 + W_3 - W_2}{|W_1 - W_2 + W_3 - W_2|} \tag{3.7}$$

- *Mission 5 - Straight Line:* The mission is a straight line through points W_2 and W_3.

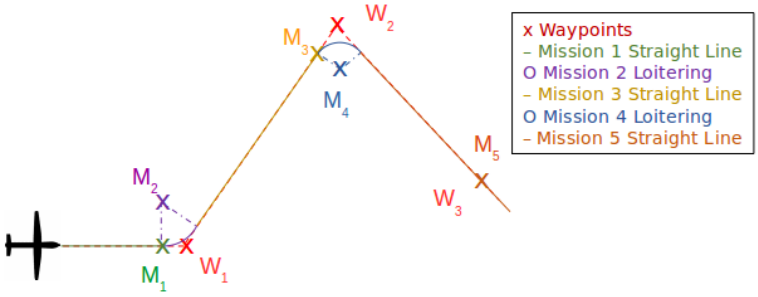


Figure 3.5: Splitting of waypoints mission into primitive missions.

3.2 Vector Field Approach for Path Following

In the previous section, we introduced path planning and a method to obtain primitive missions from the mission provided by the user. At this juncture,

we need a mechanism for the fixed-wing UAV to perform the obtained primitive missions.

Path following is the process of a UAV navigating in space to execute a primitive mission. The vector field approach for fixed-wing UAVs employed in [19][17][39] suits this purpose. *Cross-track error* denotes the deviation from the required path. In comparison to other approaches, with the least control effort, the vector field approach gives the lowest cross-track error [19].

The vector field gives a direction in every point in the inertial plane, along which the UAV should move to track the path. Since all the coordinates around the path get a direction eventually leading to the path, it's like a field but with only interest in the direction.

Recall the course angle χ defined in Section 2.2. With the vector field approach, the objective is to asymptotically drive the cross-track error to zero by controlling the course angle. Thus we construct a vector field around the desired path to provide the course angle commands to the UAV.

Over the existing low-level controllers from autopilot software, UAVs can be equipped with basic controllers as designed in [19], to have an approximated first order course angle χ dynamics given as,

$$\dot{\chi} = \alpha_\chi(\chi_c - \chi) \tag{3.8}$$

where, χ_c is the *commanded course angle*, α_χ is first order time constant.

The vector field around a straight line path is described by,

$$\chi_d(e_{py}) = \chi_q - \chi_\infty \frac{2}{\pi} tan^{-1}(k_{sl} e_{py}) \tag{3.9}$$

where, e_{py} is cross track error i.e. lateral deviation from path, χ_q is angle between north and the straight line, k_{sl} and χ_∞ are tuning parameters. An example vector field generated for straight line path following is given in 3.6 [19].

The control law which drives $\chi \to \chi_d$ and $e_{py} \to 0$ while $t \to \infty$ is,

$$\chi_c = \chi - \chi_\infty \frac{2}{\pi} \frac{\beta_s V_g}{\alpha_\chi} sin(\chi - \chi_q) - \frac{\kappa_{sl}}{\alpha_x} sat(\frac{\tilde{\chi}}{\epsilon_{sl}}) \tag{3.10}$$

where, $\beta_s = \frac{k_{sl}}{1+(k_{sl}e_{py})^2}$, $\tilde{\chi} = \chi - \chi_d$, $V_g = \|\mathbf{V}_g\|$, and $\kappa_{sl}, \epsilon_{sl}$ are tuning parameters.

For orbit, the vector field which describes reference course is given by,

$$\chi_d(\tilde{d}) = \gamma + \lambda(\frac{\pi}{2} + tan^{-1}(k_o \tilde{d})) \tag{3.11}$$

where $\tilde{d} = d - R$, d is distance from orbit center, γ is angle between North and vector from UAV to orbit center, λ is direction of rotation in orbit, k_o

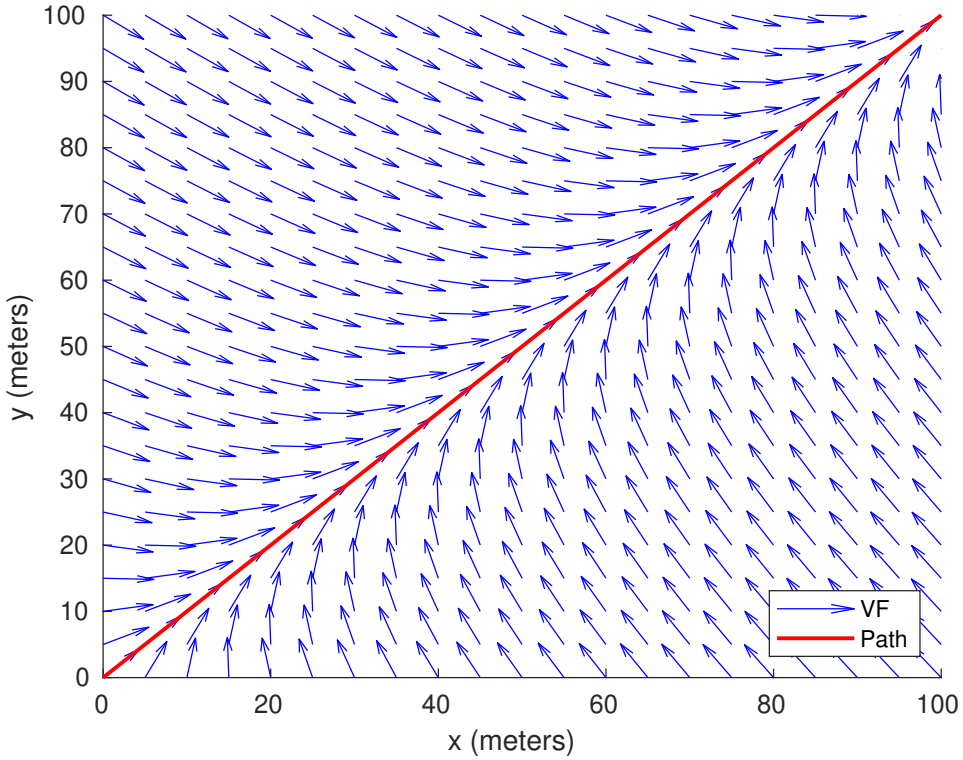


Figure 3.6: Vector field for a straight line path following.

is a tuning parameter. An example for a vector field described by equation 3.11 is as shown in Figure 3.7 [19].

The control law which drives $\chi \to \chi_d$ and $\tilde{d} \to 0$ while $t \to \infty$ is,

$$\chi_c = \chi + \frac{V_g}{\alpha_\chi d} sin(\chi - \gamma) + \beta_o \frac{\lambda V_g}{\alpha_\chi} cos(\chi - \gamma) - \frac{\kappa_o}{\alpha_\chi} sat(\frac{\tilde{\chi}}{\epsilon_o}) \tag{3.12}$$

where $\beta_o = \frac{k_o}{1+(k_o d)^2}$, and κ_o, ϵ_o are tuning parameters.

3.3 Adaptive Vector Field Approach for Path Following

There are two issues in the vector field approach discussed in section 3.2 which reduces the accuracy and have scope for improvement. They are:

- VF approach works with assumption of course angle dynamics $\dot{\chi} = \alpha_\chi(\chi_c - \chi)$. But in real UAV, this is a first order approximation of a higher order system dynamics. The approximation to first order dynamics, leaves out some dynamics as un-modelled. The bode plot given in Figure 3.8 depicts the difference between the first order approximation and actual UAV dynamics [19].

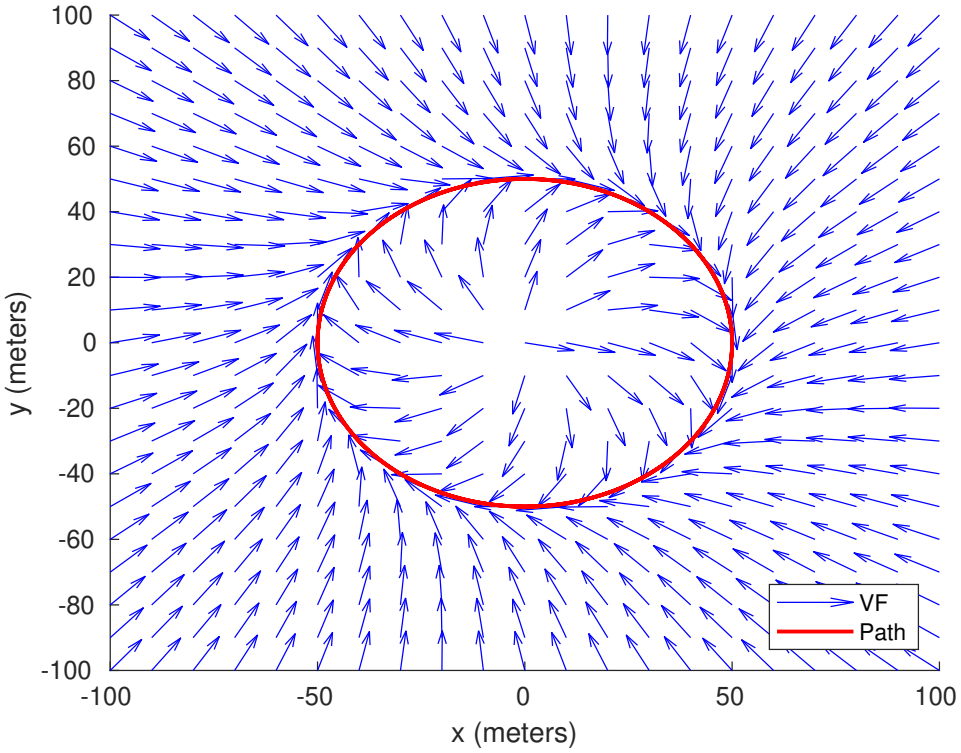


Figure 3.7: Vector field for a orbit path following.

- For the vector field approach $V_g = \|\mathbf{V}_g\|$) should be known. But $\mathbf{V}_g$ is the vector sum of velocity of UAV w.r.t air and wind velocity, where wind velocity is an unknown.

The accuracy of the above approach can be increased by adaptively tuning some parameters of VF in such a way as to compensate for unmodelled dynamics and unknown wind. Motivated from literature, parameter V_g can be adaptively estimated to achieve this. The resultant new approach is called as the *adaptive vector field* approach. In [18][19] the estimation dynamics for the adaptive estimation of V_g is derived.

The estimation dynamics of V_g for a straight line following is given as,

$$\dot{\hat{V}}_g = \Gamma_{sl}\mu_{sl}\tilde{\chi}\chi_\infty\beta_s\frac{2}{\pi}\sin(\chi - \chi_q) + F_{sl} - \sigma_{sl}\Gamma_{sl}\hat{V}_g \tag{3.13}$$

where Γ_{sl} is the estimator gain, μ_{sl} is a weighting term, σ_{sl} adds a damping action, and F_sl is a feed-forward term defined as,

$$F_{sl} = \frac{\partial\hat{V}_g}{\partial\chi}\left[-\chi_\infty\frac{2}{\pi}\beta_s\hat{V}_g\sin(\chi - \chi_q) - \kappa_{sl}sat\left(\frac{\tilde{\chi}}{\epsilon_{sl}}\right)\right] \tag{3.14}$$

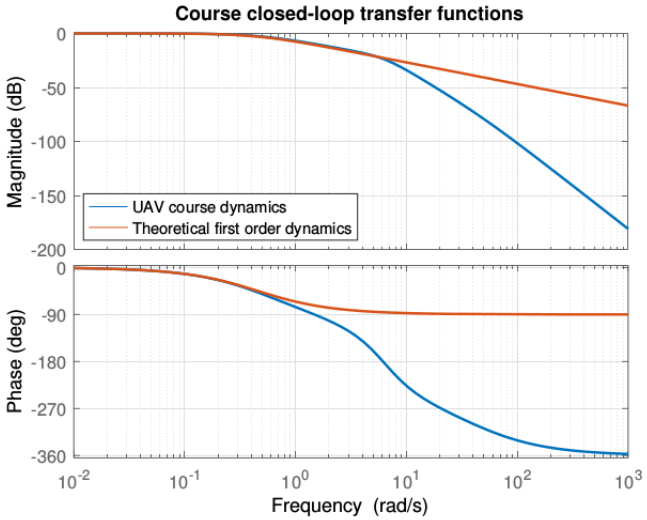


Figure 3.8: Difference in bode plot of course angle caused by approximation. [*Taken from [20].*]

where

$$\frac{\partial \hat{V}_g}{\partial \chi} \simeq W_s \sin(\psi_{w,s} - \chi) + (V_a^2 - W_s^2 \sin^2(\psi_{w,s} - \chi))^{-\frac{1}{2}} \\ W_s^2 \sin(\psi_{w,s} - \chi) \cos(\psi_{w,s} - \chi) \tag{3.15}$$

where $W_s = \|V_{w,s}\|$, $V_{w,s}$ is the velocity of steady-state wind component, and $\psi_{w,s}$ is the angle steady-state wind component makes with the north direction [20].

The estimation dynamics of V_g for an orbit path following is given as,

$$\dot{\hat{V}}_g = -\Gamma_o \mu_o \tilde{\chi} \left(\frac{1}{d} \sin(\chi - \gamma) + \lambda \beta_o \cos(\chi - \gamma) \right) + F_o - \sigma_o \Gamma_o \hat{V}_g \tag{3.16}$$

where

$$F_o = \frac{\partial \hat{V}_g}{\partial \chi} \left[\frac{\hat{V}_g}{d} \sin(\chi - \gamma) + \lambda \beta_o \hat{V}_g \cos(\chi - \gamma) - \kappa_o sat \left(\frac{\tilde{\chi}}{\epsilon_o} \right) \right] \tag{3.17}$$

Chapter 4

Team of UAVs as a Multi-Agent System

This chapter discusses the approach for synchronization of multiple UAVs as multi-agent system.

Section 4.1 formulates the multi-agent system for UAVs and introduces the overall approach for synchronization. Section 4.2 introduces the communication graph and types of nodes in UAV multi-agent synchronization. Section 4.3 discuss the control objectives for each type of node. Section 4.4 explains the approach for generating the reference dynamics to which the leaders and followers should synchronize. Section 4.5 and Section 4.6 discuss the approach for synchronizing leaders and followers to reference dynamics using Model Reference Adaptive Control (MRAC).

4.1 UAV Muti-Agent Systems

In Section 2.3, we introduced UAV as an agent and derived the Euler Lagrange dynamics for UAVs. When we have multiple UAVs,we can consider that team as a multi-agent system having dynamics as,

$$D_i(q_i)\ddot{q}_i + C_i(q_i, \dot{q}_i)\dot{q}_i + g_i(q_i) = \tau_i \tag{4.1}$$

where subscript $i \in \{1...N\}$ uniquely identifies each agent in the team of N UAVs.

In this work, a particular UAV in the team follows a free designed path based on user inputs and all other agents synchronize to a dynamics generated by the UAV.

For synchronisation of multi-agent systems having EL dynamics the graph based Model Reference Adaptive Control (MRAC) approach presented in [23] can be used. The approach estimates the parameters of MRC control laws online to drive the systems of agents into synchronization. Control allocation is the process by which a required force and moment input is

distributed across the propeller and control surfaces of a UAV. This work does not account for the control allocation and assumes the generated forces and moments from the control law can be directly implemented.

4.2 Communication Graph and Types of Nodes

For UAV synchronization, we consider a system of UAVs where each UAV can share its data to others. Two UAVs are said to have a *directed connection* if there is a flow of information between one UAV to the other. The information shared can be the inertial measurements, control inputs, or any data which helps in formation control. The information flow between UAVs are better represented using directed graphs called *communication graphs.*

A *directed graph* or *digraph* is composed of nodes and *directed edges (arrows).* A directed graph can be written as an ordered pair,

$$\mathcal{D} = (\mathcal{V}, \mathcal{A}) \tag{4.2}$$

where, $\mathcal{V}$ is a set of *node* (or vertices), and $\mathcal{A}$ is a set of ordered pair of nodes called arrows [40].

Each UAV forms a node in the graph and the directed edges represent the allowed information flows between the UAVs.

For graph based approach of formation control, nodes in the graph can be classified into three based on information flow as,

- *Path Planner Node:* This node decides the path for the complete set of drones. The node doesn't receive information from any other nodes (UAVs) and also generates the dynamics to which all other nodes should synchronize. Thus the node is called as *pinner node* in literature.
- *Leader Node:* Leader nodes in the formation have access to data from the pinner node.
- *Follower Nodes:* The follower nodes have only access to data from nodes other than the pinner node.

The type and hierarchy of nodes identified in a typical formation is as shown in figure 4.1. The arrows represent the allowed information flows.

In this work we consider a system of UAVs having a communication graph depicted as in Figure 4.2. Node 0 is the pinner node. Note that node 1, the leader node, receives information from the pinner node but not vice versa. Node 2, the follower node, has access to the information from Node 1.

Mathematically the communication graph in Figure 4.2 can be written as,

$$\mathcal{D} = (\mathcal{V}, \mathcal{A}) \tag{4.3}$$

where $\mathcal{V} = \{0, 1, 2\}$, and $\mathcal{A} = \{(0, 1), (1, 2)\}$.

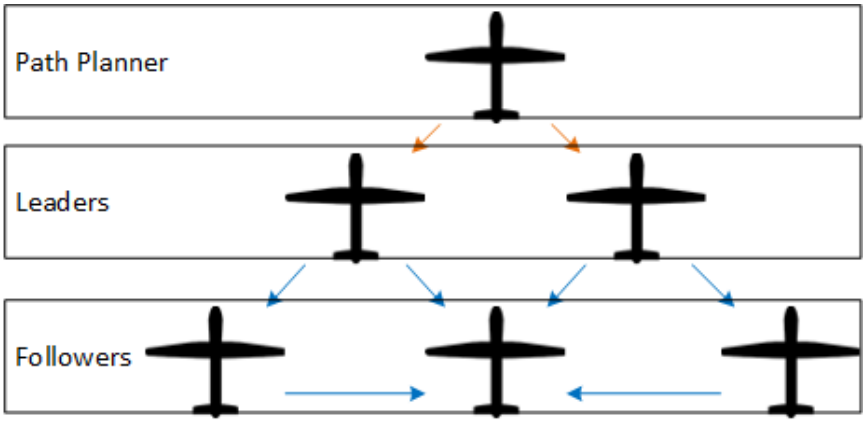


Figure 4.1: Type and hierarchy of nodes in a typical formation.

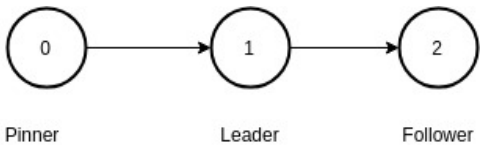


Figure 4.2: Communication graph for UAV formation control.

4.3 Objectives for Control

The complete set of UAVs together have to execute the missions provided by the user. In this work, the path planner UAV executes the received mission and all other UAVs in the formation synchronize with the path planner node. Thus we achieve a synchronized flying while performing the given mission.

In Section 4.2, we introduced the path planner (or pinner) node. It is important to recall that the path planner node does not receive any information inputs from other UAVs in the formation. However, the path planner node uses the information configured by the user for its operation.

Here we define the primary objectives of the path planner node. They are:

1. Plan the path for the complete formation based on the type of path input provided by the user.

2. Traverse the planned path using path following techniques.

3. Generate a dynamics to which all other UAVs should synchronize using their respective controllers.

The design of the path planner node is as such to address these objectives.

Objectives 1 & 2 are handled by using the approach in 3. The approach for achieving Objective 3 of path planner node is explained in Section 4.4.

The control objective of leader node is to synchronize to the reference dynamics by Model Reference Control (MRC). This requires the precise knowledge of plant parameters. But given the fact that there could be variation of plant parameters, a pure Model Reference Control (MRC) law with known parameters is impossible. Thus a modified control law to address the uncertainty in parameters is most needed.

The information of reference dynamics from the pinner node is only available to leader type of nodes and not to the follower nodes. The followers have access to control input of leaders or other neighbours. If a follower synchronizes to the dynamics of its neighbouring nodes, since the neighbouring nodes already have control inputs for synchronizing it to the reference dynamics, a synchronization is possible.

Thus the control objective for follower nodes is to synchronize to the reference dynamics by a Model Reference Controller utilizing information from neighbouring nodes. Since the plant parameters for both neighbours and current node cannot be precisely known. the control law should adaptively sychronize.

4.4 Reference Dynamics for Leader/Follower Synchronization

The graph based Leader/Follower formation control approach in [23] works by synchronizing to a given reference dynamics. The reference dynamics for this is formulated as,

$$\begin{bmatrix} \dot{q}_0 \\ \ddot{q}_0 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & \mathbb{1} \\ -K_p & -K_v \end{bmatrix}}_{A_m} \underbrace{\begin{bmatrix} q_0 \\ \dot{q}_0 \end{bmatrix}}_{x_m} + \underbrace{\begin{bmatrix} 0 \\ \mathbb{1} \end{bmatrix}}_{B_m} r \tag{4.4}$$

where $q_0, \dot{q}_0 \in \mathbb{R}^n$, x_m the reference model states, K_p, K_v are the proportional and derivative gains of the multivariable PD controller, and $r = \ddot{q}^d + K_v\dot{q}^d + K_p q^d$ is a control input.

Currently, the control in path planning UAV has a vector field based approach which does not ensure a dynamics as given in (4.4) for the UAV. Thus, the path planner node also needs to generate a reference dynamics to which all UAVs in the formation should synchronize. From (2.14), EL dynamics of a UAV can be expressed as

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau \tag{4.5}$$

On using an inverse dynamic based controller of the form in (4.6) we obtain the dynamics as in (4.4).

$$\tau = D(q)a + C(q,\dot{q})\dot{q} + g(q) \tag{4.6}$$

where the term a is defined as

$$a = \ddot{q}^d - K_v\dot{e} - K_p e \tag{4.7}$$

with $e = q - q^d$.

On applying (4.6) in (4.5), we obtain the error dynamics as,

$$\ddot{e} + K_v\dot{e} + K_p e = 0 \tag{4.8}$$

The equation in (4.8) can be re-written in state space form as,

$$\begin{bmatrix}\dot{e}\\ \ddot{e}\end{bmatrix} = \begin{bmatrix}0 & \mathbb{1}\\ -K_p & -K_v\end{bmatrix}\begin{bmatrix}e\\ \dot{e}\end{bmatrix} \tag{4.9}$$

Since K_p, K_v are positive gains, by construction the state matrix in (4.9) is Hurwitz. This implies as $t \to \infty$, $e \to 0$ [41] i.e. $q \to q_d$. Also, (4.9) can be easily re-written to obtain the form in (4.4).

In path planner node, we run the dynamics in (4.4) virtually with q^d, $\dot{q}^d$,and $\ddot{q}^d$ as the inertial measurements (trajectories, velocities, and accelerations) of the path planner UAV. By this method, the reference states x_m will be in close match to the states of the path planner UAV and have the dynamics in (4.4). The reference states x_m are further transmitted to the leader nodes for leader synchronization.

4.5 Synchronization of Leader Dynamics to Reference Dynamics

The state-space representation of EL dynamics for any UAV is given in (2.15) is,

$$\underbrace{\begin{bmatrix}\dot{q}_l\\ \ddot{q}_l\end{bmatrix}}_{\dot{x}_l} = \underbrace{\begin{bmatrix}0 & \mathbb{1}\\ 0 & -D_l^{-1}C_l\end{bmatrix}}_{A_l}\underbrace{\begin{bmatrix}q_l\\ \dot{q}_l\end{bmatrix}}_{x_l} + \begin{bmatrix}0\\ -D_l^{-1}g_l\end{bmatrix} + \underbrace{\begin{bmatrix}0\\ D_l^{-1}\end{bmatrix}}_{B_l}\tau_l \tag{4.10}$$

In this section, the subscript l in (4.10) represents the values for leader type UAVs. Model Reference Control is a typical control approach in which the plant is made to have the dynamics as of a reference model by using an appropriate control law. The control law is developed by first defining a control structure, and then finding matching conditions that makes the closed loop dynamics as that of the reference model. As per [23], the ideal Model Reference (MRC) Control law for this purpose would be,

$$\tau_l^* = D_l(-K_p q_l - K_v\dot{q}_l + r) + C_l\dot{q}_l + g_l \tag{4.11}$$

where D_l, C_l and g_l are matrices as in (2.14), and $r = \ddot{q}^d + K_v\dot{q}^d + K_p q^d$ is a control input used in the path planner node.

The control law proposed in (4.11) is ideal and requires knowledge of the matrices D_l, C_l and g_l. A Model Reference Adaptive Control (MRAC) law is proposed in [23] for using the estimates of the matrices. The control law is,

$$\tau_l = \hat{D}_l(-K_p q_l - K_v \dot{q}_l + r) + \hat{C}_l \dot{q}_l + \hat{g}_l \tag{4.12}$$

where $\hat{D}_l$,$\hat{C}_l$, and $\hat{g}_l$ are estimates of matrices D_l,C_l, and g_l respectively. The estimates can be split in linear-in-the-parameter form as,

$$\begin{aligned} \hat{D}_l &= \Theta'_{D_l} \xi_{D_l} \\ \hat{C}_l &= \Theta'_{C_l} \xi_{C_l} \\ \hat{g}_l &= \Theta'_{g_l} \xi_{g_l} \end{aligned} \tag{4.13}$$

where Θ matrices are the regressands, and ξ matrices are the regressors.

Using Lyapunov analysis the estimation laws can be established as,

$$\begin{aligned} \dot{\Theta}'_{D_l} &= -S_l B'_m P e_l (-K_p q_l - K_v \dot{q}_l + r)' \xi'_{D_l} \\ \dot{\Theta}'_{C_l} &= -S_l B'_m P e_l \dot{q}'_l \xi'_{C_l} \\ \dot{\Theta}'_{g_l} &= -S_l B'_m P e_l \xi'_{g_l} \end{aligned} \tag{4.14}$$

where $e_l = x_l - x_m$, $P = P' > 0$ is such that $PA_m + A'_m P = -Q, \quad Q > 0$, S_l is a matrix such that $D_l S_l = S'_l D'_l > 0$.

Here it is important to note that, the laws given in (4.14) depends on r and x_m data from the path planner node.

The composition of regressands and regressors in (4.13) are,

$$\Theta^*_{D_l} = \begin{bmatrix} m_l & 0 & 0 & 0 & 0 & 0 \\ 0 & m_l & 0 & 0 & 0 & 0 \\ 0 & 0 & m_l & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{x_l} & 0 & -I_{xz_l} \\ 0 & 0 & 0 & 0 & I_{y_l} & 0 \\ 0 & 0 & 0 & -I_{xz_l} & 0 & I_{z_l} \end{bmatrix} \qquad \xi_{D_l} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.15}$$

$$\Theta^{*\prime}_{C_l} = \begin{bmatrix} m_l & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & m_l & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & m_l & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{x_l} & 0 & 0 & I_{y_l} & 0 & 0 & I_{z_l} & 0 & 0 & I_{xz_l} & 0 & 0 \\ 0 & 0 & 0 & 0 & I_{x_l} & 0 & 0 & I_{y_l} & 0 & 0 & I_{z_l} & 0 & 0 & I_{xz_l} & 0 \\ 0 & 0 & 0 & 0 & 0 & I_{x_l} & 0 & 0 & I_{y_l} & 0 & 0 & I_{z_l} & 0 & 0 & I_{xz_l} \end{bmatrix}$$

$$\xi'_{C_l} = \begin{bmatrix} 0 & \bar{r}_l & -\bar{q}_l & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\bar{r}_l & 0 & \bar{p}_l & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \bar{q}_l & -\bar{p}_l & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \bar{q}_l & 0 & -\bar{r}_l & 0 & 0 & \bar{p}_l & 0 \\ 0 & 0 & 0 & 0 & 0 & -\bar{p}_l & 0 & 0 & 0 & \bar{r}_l & 0 & 0 & -\bar{p}_l & 0 & \bar{r}_l \\ 0 & 0 & 0 & 0 & \bar{p}_l & 0 & -\bar{q}_l & 0 & 0 & 0 & 0 & 0 & 0 & -\bar{r}_l & 0 \end{bmatrix} \tag{4.16}$$

$$\Theta^{*\prime}_{g_l} = \begin{bmatrix} m_l & 0 & 0 & 0 & 0 & 0 \\ 0 & m_l & 0 & 0 & 0 & 0 \\ 0 & 0 & m_l & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \xi_{g_l} = \begin{bmatrix} \sin\theta g \\ -\sin\phi\cos\theta g \\ -\cos\phi\cos\theta g \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{4.17}$$

4.6 Synchronization of Follower Dynamics to Reference Dynamics

The follower synchronizes to reference dynamics exploiting the signals of the neighbouring agents. Here we consider only followers which listen to the data from one neighbour.

The state-space representation of EL dynamics for a follower node written from (2.15),

$$\underbrace{\begin{bmatrix} \dot{q}_f \\ \ddot{q}_f \end{bmatrix}}_{\dot{x}_f} = \underbrace{\begin{bmatrix} 0 & \mathbb{1} \\ 0 & -D_f^{-1}C_f \end{bmatrix}}_{A_f} \underbrace{\begin{bmatrix} q_f \\ \dot{q}_f \end{bmatrix}}_{x_f} + \begin{bmatrix} 0 \\ -D_f^{-1}g_f \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ D_f^{-1} \end{bmatrix}}_{B_f} \tau_f \tag{4.18}$$

Similarly, the dynamics of any neighbouring agent can be written as,

$$\underbrace{\begin{bmatrix} \dot{q}_n \\ \ddot{q}_n \end{bmatrix}}_{\dot{x}_n} = \underbrace{\begin{bmatrix} 0 & \mathbb{1} \\ 0 & -D_n^{-1}C_n \end{bmatrix}}_{A_n} \underbrace{\begin{bmatrix} q_n \\ \dot{q}_n \end{bmatrix}}_{x_n} + \begin{bmatrix} 0 \\ -D_n^{-1}g_n \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ D_n^{-1} \end{bmatrix}}_{B_n} \tau_n \tag{4.19}$$

The Model Reference Contol (MRC) law for any follower is given in [23] is,

$$\tau_f^* = C_f\dot{q}_f + D_fD_n^{-1}\tau_n - D_fD_n^{-1}C_n\dot{q}_n - D_f(K_p\bar{e}_{fn} + K_v\bar{\bar{e}}_{fn}) + g_f \tag{4.20}$$

where $\bar{e}_{fn} = q_f - q_n$, and $\bar{\bar{e}}_{fn} = \dot{q}_f - \dot{q}_n$.

The ideal control law proposed in (4.20) use matrices D_f, C_f and g_f of the plant, and D_n, C_n and g_n matrices of the neighbour. This implies, in addition to the knowledge of plant parameters, the parameters of the neighbour also should be known. Since parameters of both follower and neighbour may change, also considering that the parameters need to be sent over a network, the law is not the best to implement. In [23], authors propose a Model Reference Adaptive Control (MRAC) law using the estimates of the matrices. The control law is,

$$\tau_f = -\hat{D}_f(K_p\bar{e}_{fn} + K_v\bar{\bar{e}}_{fn}) + \hat{C}_f\dot{q}_f + \widehat{D_fD_n}\tau_n - \widehat{D_fD_nC_n}\dot{q}_n + \hat{g}_f \tag{4.21}$$

where $\hat{D}_f$,$\hat{C}_f$, $\hat{g}_f$,$\widehat{D_f D_n}$,$\widehat{D_f D_n C_n}$, are estimates of matrices D_f,C_f,g_f,$D_f D_n^{-1}$, and $D_f D_n^{-1} C_n$ respectively. The estimates can be split in linear-in-the-parameter form as,

$$\begin{aligned}
\hat{D}_f &= \Theta'_{D_f} \xi_{D_f} \\
\hat{C}_f &= \Theta'_{C_f} \xi_{C_f} \\
\hat{g}_f &= \Theta'_{g_f} \xi_{g_f} \\
\widehat{D_f D_n} &= \Theta'_{D_f D_n C_n} \xi_{D_f D_n C_n} \\
\widehat{D_f D_n C_n} &= \Theta'_{D_f D_n C_n} \xi_{D_f D_n C_n}
\end{aligned} \tag{4.22}$$

where Θ matrices are the regressands, and ξ matrices are the regressors.

The estimation dynamics for matrices in (4.22) can be established using Lyapunov analysis, and they are,

$$\begin{aligned}
\dot{\Theta}'_{D_f} &= S_f B'_m P e_{fn} (K_p \bar{e}_{fn} + K_v \dot{\bar{e}}_{fn})' \xi'_{D_f} \\
\dot{\Theta}'_{C_f} &= -S_f B'_m P e_{fn} \dot{q}'_f \xi'_{C_f} \\
\dot{\Theta}'_{g_f} &= -S_f B'_m P e_{fn} \xi'_{g_f} \\
\dot{\Theta}'_{D_f D_n} &= -S_f B'_m P e_{fn} \tau'_n \xi'_{D_f D_n} \\
\dot{\Theta}'_{D_f D_n C_n} &= S_f B'_m P e_{fn} \dot{q}'_n \xi'_{D_f D_n C_n}
\end{aligned} \tag{4.23}$$

where $e_{fn} = x_f - x_n$, $P = P' > 0$ is such that $PA_m + A'_m P = -Q, \quad Q > 0$, S_f is a matrix such that $D_f S_f = S'_f D'_f > 0$. The estimation dynamics (4.23) guarantees that the $e_{fn} \to 0$ as $t \to \infty$.

The composition of regressands and regressors in (4.22) are,

$$\Theta^*_{D_f} = \begin{bmatrix} m_f & 0 & 0 & 0 & 0 & 0 \\ 0 & m_f & 0 & 0 & 0 & 0 \\ 0 & 0 & m_f & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{x_f} & 0 & -I_{xz_f} \\ 0 & 0 & 0 & 0 & I_{y_f} & 0 \\ 0 & 0 & 0 & -I_{xz_f} & 0 & I_{z_f} \end{bmatrix} \qquad \xi_{D_f} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.24}$$

$$\Theta^{*\prime}_{C_f} = \begin{bmatrix} m_f & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & m_f & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & m_f & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{x_f} & 0 & 0 & I_{y_f} & 0 & 0 & I_{z_f} & 0 & 0 & I_{xz_f} & 0 & 0 \\ 0 & 0 & 0 & 0 & I_{x_f} & 0 & 0 & I_{y_f} & 0 & 0 & I_{z_f} & 0 & 0 & I_{xz_f} & 0 \\ 0 & 0 & 0 & 0 & 0 & I_{x_f} & 0 & 0 & I_{y_f} & 0 & 0 & I_{z_f} & 0 & 0 & I_{xz_f} \end{bmatrix}$$

$$\xi'_{C_f} = \begin{bmatrix} 0 & \bar{r}_f & -\bar{q}_f & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\bar{r}_f & 0 & \bar{p}_f & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \bar{q}_f & -\bar{p}_f & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \bar{q}_f & 0 & -\bar{r}_f & 0 & 0 & \bar{p}_f & 0 \\ 0 & 0 & 0 & 0 & 0 & -\bar{p}_f & 0 & 0 & 0 & \bar{r}_f & 0 & 0 & -\bar{p}_f & 0 & \bar{r}_f \\ 0 & 0 & 0 & 0 & \bar{p}_f & 0 & -\bar{q}_f & 0 & 0 & 0 & 0 & 0 & 0 & -\bar{r}_f & 0 \end{bmatrix} \tag{4.25}$$

$$\Theta^{*\prime}_{g_f} = \begin{bmatrix} m_f & 0 & 0 & 0 & 0 & 0 \\ 0 & m_f & 0 & 0 & 0 & 0 \\ 0 & 0 & m_f & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \xi_{g_f} = \begin{bmatrix} \sin\theta g \\ -\sin\phi\cos\theta g \\ -\cos\phi\cos\theta g \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.26}$$

$$\Theta^{*}_{D_f D_n} = \begin{bmatrix} \frac{m_f}{m_n} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{m_f}{m_n} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{m_f}{m_n} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{I_{z_n} I_{x_f} I_{xz_f}}{I_{x_n} I_{z_n} - I_{xz_n} I_{xz_n}} & 0 & \frac{I_{xz_n} I_{x_f} I_{xz_f}}{I_{x_n} I_{z_n} - I_{xz_n} I_{xz_n}} \\ 0 & 0 & 0 & 0 & \frac{I_{y_f}}{I_{y_n}} & 0 \\ I_{x_n} 0 & 0 & 0 & \frac{I_{xz_n} I_{xz_f} I_{z_f}}{I_{x_n} I_{z_n} - I_{xz_n} I_{xz_n}} & 0 & -\frac{I_{x_n} I_{xz_f} I_{z_f}}{I_{x_n} I_{z_n} - I I_{xz_n} I_{xz_n}} \end{bmatrix} \tag{4.27}$$

$$\xi_{D_f D_n} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\Theta^{*}_{D_f D_n C_n} = \begin{bmatrix} m_f & 0 & 0 & 0 & 0 & 0 \\ 0 & m_f & 0 & 0 & 0 & 0 \\ 0 & 0 & m_f & 0 & 0 & 0 \\ 0 & 0 & 0 & \Gamma_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \Gamma_1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \Gamma_1 \\ 0 & 0 & 0 & \Gamma_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \Gamma_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \Gamma_2 \\ 0 & 0 & 0 & \Gamma_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & \Gamma_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & \Gamma_3 \\ 0 & 0 & 0 & \Gamma_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & \Gamma_4 & 0 \\ 0 & 0 & 0 & 0 & 0 & \Gamma_4 \\ 0 & 0 & 0 & \Gamma_5 & 0 & 0 \\ 0 & 0 & 0 & 0 & \Gamma_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & \Gamma_5 \\ 0 & 0 & 0 & \Gamma_6 & 0 & 0 \\ 0 & 0 & 0 & 0 & \Gamma_6 & 0 \\ 0 & 0 & 0 & 0 & 0 & \Gamma_6 \\ 0 & 0 & 0 & \Gamma_7 & 0 & 0 \\ 0 & 0 & 0 & 0 & \Gamma_7 & 0 \\ 0 & 0 & 0 & 0 & 0 & \Gamma_7 \\ 0 & 0 & 0 & \Gamma_8 & 0 & 0 \\ 0 & 0 & 0 & 0 & \Gamma_8 & 0 \\ 0 & 0 & 0 & 0 & 0 & \Gamma_8 \\ 0 & 0 & 0 & \Gamma_9 & 0 & 0 \\ 0 & 0 & 0 & 0 & \Gamma_9 & 0 \\ 0 & 0 & 0 & 0 & 0 & \Gamma_9 \\ 0 & 0 & 0 & \Gamma_{10} & 0 & 0 \\ 0 & 0 & 0 & 0 & \Gamma_{10} & 0 \\ 0 & 0 & 0 & 0 & 0 & \Gamma_{10} \\ 0 & 0 & 0 & \Gamma_{11} & 0 & 0 \\ 0 & 0 & 0 & 0 & \Gamma_{11} & 0 \\ 0 & 0 & 0 & 0 & 0 & \Gamma_{11} \\ 0 & 0 & 0 & \Gamma_{12} & 0 & 0 \\ 0 & 0 & 0 & 0 & \Gamma_{12} & 0 \\ 0 & 0 & 0 & 0 & 0 & \Gamma_{12} \end{bmatrix} \qquad \xi_{D_f D_n C_n} = \begin{bmatrix} 0 & -\bar{r}_n & \bar{q}_n & 0 & 0 & 0 \\ \bar{r}_n & 0 & -\bar{p}_n & 0 & 0 & 0 \\ -\bar{q}_n & \bar{p}_n & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\bar{q}_n & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\bar{r}_n & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \bar{p}_n & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \bar{q}_n & 0 & 0 \\ 0 & 0 & 0 & 0 & \bar{r}_n & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\bar{p}_n & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \bar{r}_n & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\bar{p}_n & 0 \\ 0 & 0 & 0 & 0 & 0 & -\bar{q}_n \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\bar{r}_n \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \bar{p}_n \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \bar{q}_n \end{bmatrix} \tag{4.28}$$

Chapter 5

System & Software Architecture

This chapter introduces the system architecture for which the analysis, design and implementation in this work is targeted. The chapter also brief the software design of each type of nodes in synchronization.

Section 5.1, describes the architecture for planned Hardware-in-the-Loop (HITL) simulation. Section 5.2 discusses the motivation for the system architecture for which the work is done, and introduces the architecture. Section 5.3 briefly explains the software architecture for the path planner node. The software architecture for the leader and follower nodes are introduced in section 5.4 and 5.4 respectively.

5.1 System Architecture for Hardware-in-the-Loop Simulation

Unmanned Aerial Vehicles (UAV) need onboard flight controllers which enable stable flight operations, control and communication link to a ground station. The onboard flight controllers, called as autopilot, are realised using both hardware and software. For both the software and hardware parts, there are products from multiple vendors available in the market.

The system under consideration uses Pixhawk cube hardware shown in Figure 5.1a. Pixhwak cube has a 32bit STM32F427 Cortex-M4F core with Floating Point Unit (FPU) [42]. It has servo outputs which can be interfaced with actuators on the UAVs. The hardware board also has Inertial Measurement Units (IMUs) for measuring inertial parameters. The board comes with multiple features which aid in UAV flight control and navigation, which are not explained here.

The Pixhwak cube runs NuttX OS to have basic real-time operating system features for the upper software stack i.e. the autopilot software. The two established software stacks for UAV autopilot are - Ardupilot and PX4,

(a) Pixhawk Cube 2 (b) Raspberry Pi 3 Model B+

Figure 5.1: Hardware for HITL Simulations.

and Pixhwak cube supports both. The autopilot softwares have various modes of operations with or without a human in the loop. These stacks implement discrete time control loops for roll, pitch, yaw. For instance, the PX4 stack realizes a discrete time cascaded control loop for attitude control as shown in the Figure 5.2 [35]. The outer loop is a Proportional controller. This controller operates on the error between the setpoint and the estimated attitude to generate a rate setpoint. The inner Proportional-Integral (PI) controller uses the error in rates to compute the required angular acceleration. The autopilot software issues commands to the interfaced actuators for the required angular acceleration.

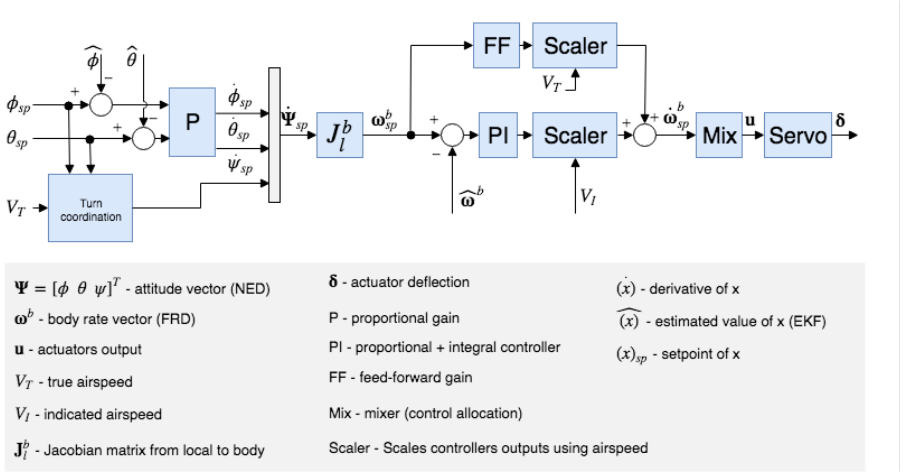


Figure 5.2: Atitude control loop for fixed-wing UAVs in PX4.
[Image source: [35].]

The user can configure or interact with the UAV using a Ground Station (GS) software installed in a Desktop or PC. QGroundControl (QGC) and

Mission Planner are two well-known ground station software. The ground station software can be used to upload the autopilot stacks to the UAV. The autopilot stack and ground station communicate using MAVLink protocol.Also, a single ground station software instance can handle multiple UAVs.

The system architecture for a single UAV can be summarised as shown in Figure 5.3.

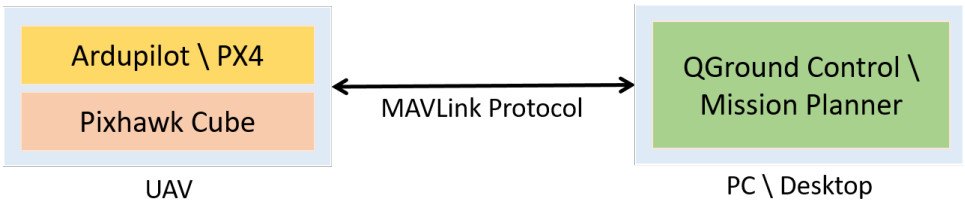


Figure 5.3: System architecture for a single UAV.

For formation control of a swarm of UAVs, each UAV needs to transmit or receive information to or from the neighbouring UAVs. Thus the UAVs need modules enabling inter-node communications.

With the data received, UAVs need to execute computation intensive formation control algorithms. The computational power of Pixhawk boards is only meant for enabling basic UAV operations and does not suffice for executing intensive algorithms. Pixhawk together with the autopilot software supports Offboard Mode. In Offboard Mode, the systems can receive commands from a companion computer of higher computational power.

Raspberry Pi is a series of small and cheap single-board computers widely used in computation intensive embedded applications. Raspberry Pi 3 Model B+ is the latest revision and has a 1.4GHz 64-bit quad-core processor along with a dual-band wireless LAN. Raspberry Pi 3 Model B+ can be interfaced as a companion computer for the UAV autopilot. Also, the in-built WLAN in Raspberry Pi board can be used to establish a communication mechanism for data exchange between the nodes. A picture of raspberry pi and its widely used logo is shown in Figure 5.1b.

Given the above information, a complete system architecture for UAV formation control can be proposed as shown in Figure 5.4. The multiple UAVs are configured using a single instance of Ground Station software. Each of the UAVs carries a Raspberry Pi 3 Model B+ interfaced as a companion computer. All the Raspberry Pis, of drones in formation, are connected to the server program through Wi-Fi. The server program has two primary functions: 1) Receive user input for the configuration for formation flying and configure the UAVs accordingly, 2) Receive and distribute data between the UAVs. The formation control program executes on the Raspberry Pi receiving configurations and data from the server. The program in

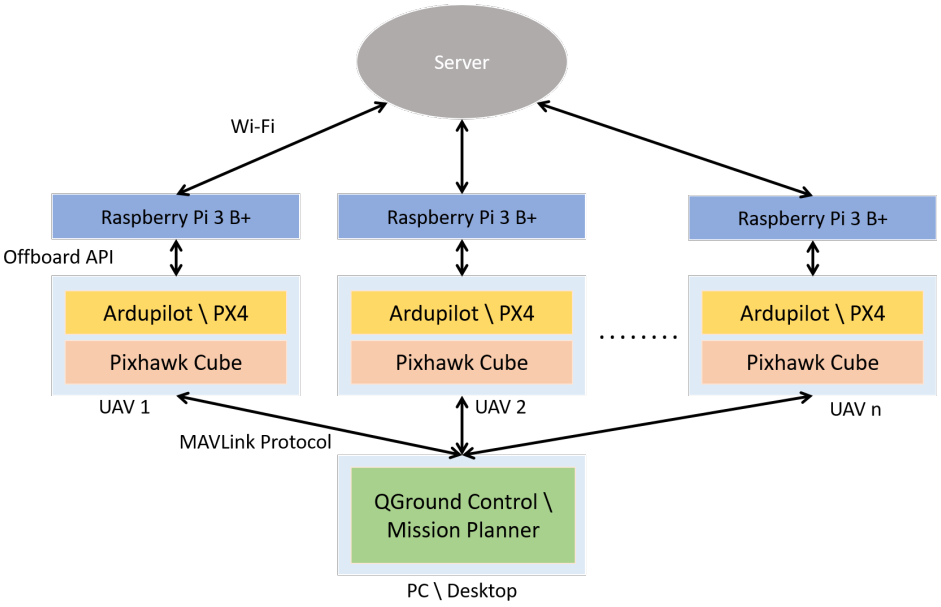


Figure 5.4: System architecture for UAV formation control.

Raspberry Pi issues commands to the autopilot using offboard APIs.

In Hardware-in-the-Loop (HITL) simulations, the computer program written for a feature is executed in the targeted hardware itself. That means the program written for Raspberry Pi in the above system should run in Raspberry Pi and the autopilot stack in Pixhawk. There are no real drones used in this simulation. But the dynamics of the drones are simulated using simulators like XPlane, Gazebo etc.

5.2 System Architecture for Software-in-the-Loop Simulation

From the background information and architecture introduced in the previous section, the UAVs carry Raspberry Pi Model 3+ as a companion computer. The formation control algorithm executes in the Raspberry Pi. Therefore, for the purpose of implementation and validation of the algorithm it is sensible to reduce the level of the simulation till the Raspberry Pis. A simulator for simulating the dynamics of UAVs is introduced in the Raspberry Pis itself.

It is also important to state that, the simulation of multiple fixed-wing UAV dynamics is not supported in the traditional simulators like Gazebo and XPlane. Also, the computation power needed for simulating multiple drones in one single machine is high. Adding a drone dynamics simulator in Raspberry Pi itself helps to address these shortcoming and serves the

purpose.

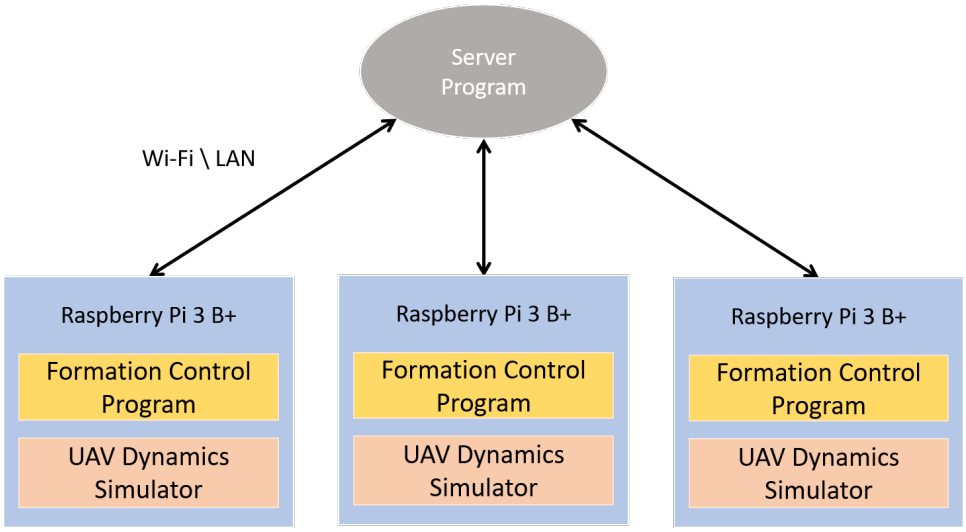


Figure 5.5: System architecture for UAV formation control SITL Simulation.

A reduced version of the architecture in Section 5.1 is used in the simulations in this work. The architecture is as given in Figure 5.5. The Raspberry Pis execute both the formation control and the UAV dynamics simulator program. The Raspberry Pi gets information of other drones through the server, and also reports back its own simulated information to the server to share across other UAVs. LAN or WLAN can be used for the simulation, as Raspberry Pis can connect to the server using either of them. The server receives configuration information from user and configures all UAVs. The data from UAVs is accumulated in server and is used to create plots of all the UAVs in formation.

5.3 Software Architecture for Path Planner Node

Logically, the components in path planner node implementation are as shown in Figure 5.6. The interfaces in the figure are the object or function by which each component access data from others. The * marked components are only used in software-in-the-loop simulations and can be replaced in real scenarios.

5.4 Software Architecture for Leader Node

The software implementation of leader node can be logically shown with a component diagram as as shown in Figure 5.7. The components exchange

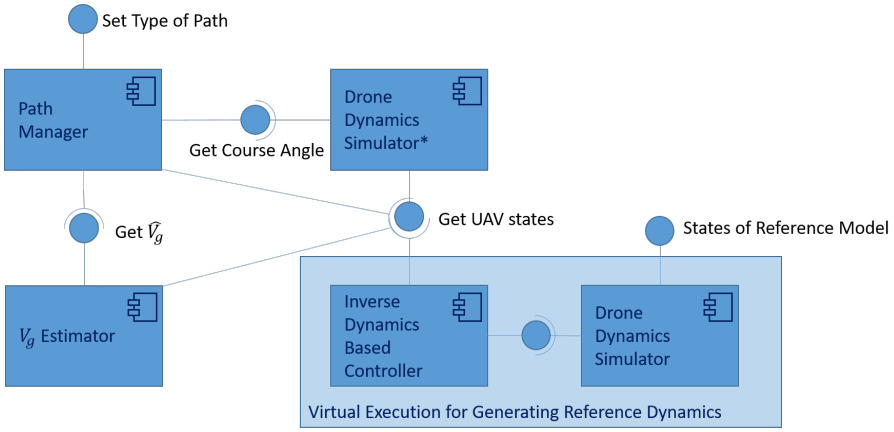


Figure 5.6: Component diagram for Path planner node.

data using the interfaces which are pure functions. The * marked components are only used in software-in-the-loop simulations and can be replaced in real scenarios.

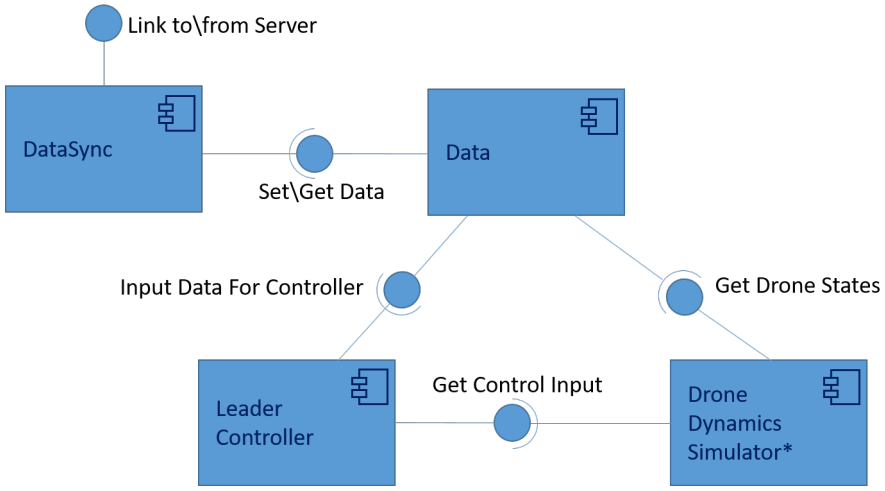


Figure 5.7: Component diagram for Leader node.

5.5 Software Architecture for Follower Node

The logical software components in follower node software implementation are as shown in Figure 5.8. The data interfaces, which are pure functions, helps in exchange of data between the components. The * marked components are only used in software-in-the-loop simulations and can be replaced in real scenarios.

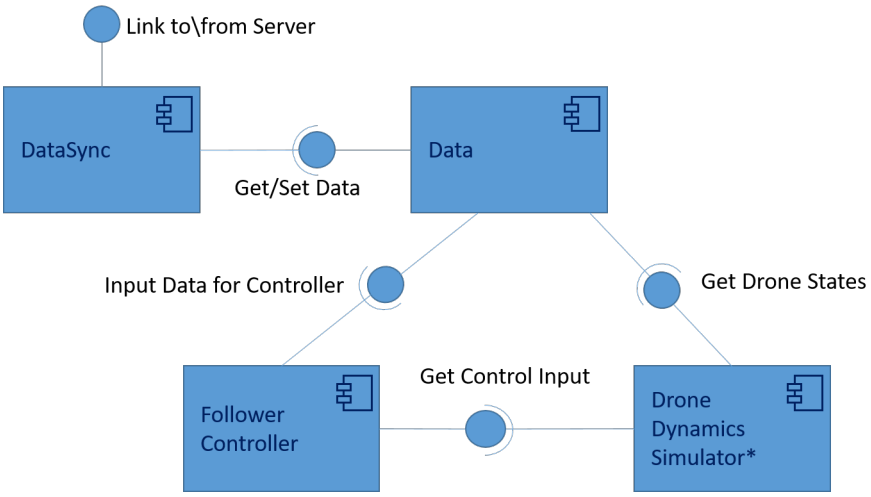


Figure 5.8: Component diagram for follower node.

Chapter 6

Synchronization of Data Between Nodes

This chapter describes the approach and implementation by which data is transferred between nodes to the server and the vice versa. Section 6.1 introduces the terminologies in computer networking and the settings in targeted architecture. Section 6.2 introduces the a packet-based protocol for data transfer between the nodes. Section 6.3 expounds how the packet-based protocol is used for the data synchronization. The server program is explained in Section 6.4 whereas Section 6.5 describes the software component in UAVs which enable the communication with the server. Section 6.6 describes how the system accepts and handles configurations and inputs from the user.

6.1 A Glimpse to Computer Networking

Computers can communicate with each other, and such connected computers are said to be in a *Network*. *Local Area Network (LAN)* are networks that connect computers or devices so that they can communicate with each other in a restricted area through wired connections. *Wireless Local Area Networks (WLAN)* are LANs in which the computers are linked through wireless communication. *Wi-Fi* is a collection of radio technologies used in WLANs [43]. A *Wireless Access Point (WAP)* is a type of network hub which connects a set of devices that forms a WLAN, relaying data between the connected devices [44].

Every device connected in a network is identified by a unique identifier called an address. For networks that use *Internet Protocol* (IP) for communication, these addresses are known as *IP addresses*. A network is logically subdivided into *subnetwork* or *subnet*. The portion of the IP address which represents the subnet can be found by using the *subnet mask*.

In this work, we follow a server-client architecture for data transfer, op-

erating in a network having only one subnet. The server computer is a computer which runs the server program for distributing data across the nodes. The server computer and all the companion Raspberry Pis of UAVs, connected to the network have unique IP addresses to identify them. The IP address of the server computer is fixed and stored in Raspberry Pis.

The architecture of the network can be represented as in Figure 6.1. In figure, x is a fixed and known integer, such that $x \in (1, 254)$, forming a static address for the server computer. Also, $y_n \in (1, 254)$ for $n = 1...N$, $N \leq 253$ are pairwise distinct integers different from x.

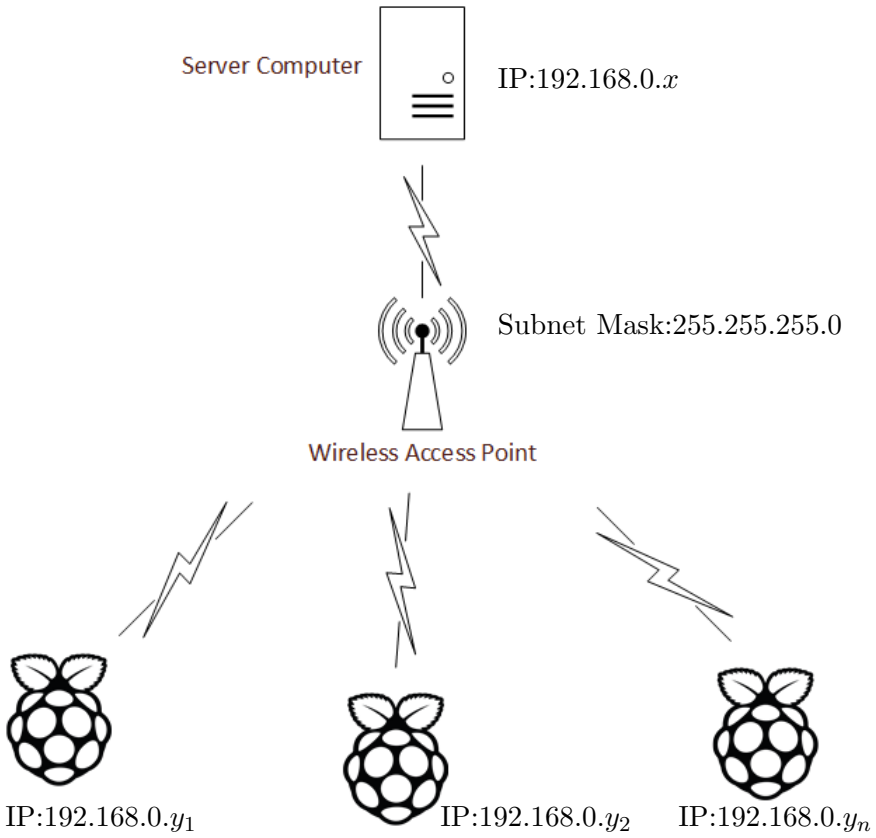


Figure 6.1: Network architecture.

Every program in a computer that uses the network, should bind to a port. A port, in computer networking, is a 16-bit unsigned number that specifically identifies a service offered by a program to a network of computers. Any second agent in the network can communicate to the program knowing the IP address of the computer and the port number. There are port numbers which are reserved by various standards to avoid possible clashes, and all other user programs are expected to use port numbers outside the reservations. In this work, the server program is bound to the port number 51717 with provisions

to be changed.

TCP/IP and UDP are two established protocols used for communication between computers through networks. In TCP/IP, data is transmitted as small packets with acknowledgements to ensure reliable data transfer. UDP operates in the same level of TCP/IP and does not offer any guarantee of data reception. The mechanism in TCP/IP to ensure reliable data transfer introduces an overhead in the turnaround time, which makes it unsuitable for the kind of purposes where the timing is of crucial importance. Thus, in this work, we use UDP for data transfer. The C/C++ programming in Linux environments to write computer applications using TCP/IP or UDP can be done using insights from [45].

6.2 Protocol for Synchronisation of Data Between Nodes

In this work, we define, *Data Synchronisation* as timed update of data of a node stored in a second node. TCP/IP or UDP are basic protocols which help in sending bytes of data through the network. We need a mechanism to create the byte streams from meaningful information that need to be transferred or vice versa. Thus, in this work we define a higher level protocol which operates above the Transport layer, customized for the application of formation control. The new protocol is positioned among TCP/IP protocol layers as shown in Figure 6.2.

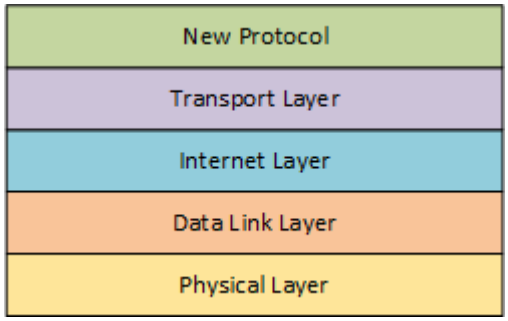


Figure 6.2: New protocol over TCP/IP Layers.

On analysis of formation control, the data that needs to be sent over the networks can be broadly classified into two. The classification helps in distinguishing between the data which is time critical and the data that needs acknowledgements of reception. The classification is as follows:

- *Plant Data*: The measurements and signals in UAVs, that change in real time. This consists of the inertial position, attitude, linear velocity, angular velocity, and control input. Plant Data is sent periodic-

ally without any acknowledgements. Table 6.1 summarizes the plant data that needs to be sent over the network based on the type of source UAV. Definitions of symbols x_m, r,q_l,$\dot{q}_l$, τ_l,q_f, $\dot{q}_f$, τ_f can be found in the respective sections.

UAV Type	**Plant Data to Send**	**Data Length**
Path Planner	x_m, r	18 bytes
Leader	q_l, $\dot{q}_l$, τ_l	18 bytes
Follower	q_f, $\dot{q}_f$, τ_f	18 bytes

Table 6.1: Plant data to be send over the network.

- *Configuration Data*: The data for configuring the control in UAVs like the communication graph, the estimation gains, the path inputs etc. The configuration packages need to be sent only on request and need acknowledgements to ensure reliable transmission.

Here we define, *Message* as a data or group of data that needs to be sent across the network using the protocol. The new protocol is based on a structured *packet*. A packet is a structured representation of the message to enable transmission and reception programmatically. Every message is structured into a packet and converted to a stream of bytes that are sent through the network using Transport Layer. The structure of a packet is as shown in Figure 6.3.

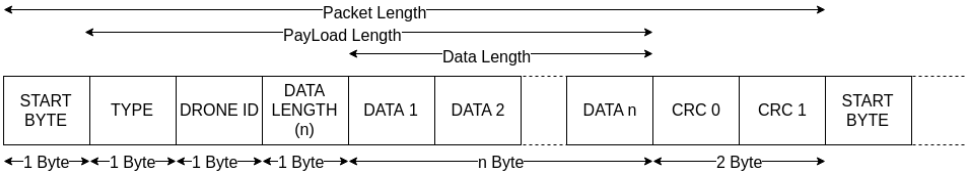


Figure 6.3: Structure of a packet.

The elements of the packet are described as:

1. START BYTE: An unsigned value of 1-byte length which signifies the start of a byte stream when each packet is converted to a stream of bytes.

2. TYPE: The element signifies the type of message. Theoretically, there can be 256 types since the value is one byte wide.

3. DATA LENGTH: The data in any message can be split into unsigned characters of 1-byte length. The DATA LENGTH signifies the length of the data being sent i.e. the bytes of the data in the message. Since the value is an unsigned byte, a maximum of 256 bytes can be sent over the network in a single packet.

4. DATA 1..n: Holds the unsigned characters split from the data in the message which needs to be transmitted.

5. CRC 0, 1: The two bytes helps in ensuring the integrity of the message in the packet. Cyclic Redundancy Check (CRC) is a widely used algorithm to detect errors in data transmitted over a network. CRC algorithm generates a reduced number of bytes, from a stream of bytes. The generated bytes remain the same for a given stream of bytes, irrespective of the time of execution. In this work, a CRC16 algorithm which generates two bytes of data is used. At any point, if there is a mismatch between the received CRC bytes in the packet and freshly calculated CRC, it signifies that there is a possible error in the contents of the packet.

The value of START BYTE and entries in TYPE byte are pre-fixed and universal to the implementation defined using C++ Macros.

A Payload is the actual information in the package intended to be sent over the network. This includes type of the message, unique id of the UAV, length of the data, and the data. The length of payload is depicted in Figure 6.3.

In packet structure description, we mentioned the types of messages. The types of messages are defined based on the contents and the purpose of the data being sent in the message. The types are summarized in Table 6.2. Column 'Source' lists the producer of the messages, and Column 'Target' denotes the consumer of the messages.

In C++ implmentation, the packet structure is realised as a class shown in Figure 6.4. All the elements of a packet introduced in Figure 6.3 are data members of type *unsigned char*. The class *packet* has member functions which help in creating packets of various message type from the data of the message. The function *GetByteStream* helps in getting the stream of bytes from the packet that needs to be sent over the network. Function *SetCRCValue* calculates and sets the CRC bytes of the packet.

6.3 Transfer of Packets Over the Network

In previous sections, we introduced the network architecture and the protocol. The messages are communicated across the nodes with a packet-based protocol.

Type of Message	**Source**	**Target**	**Purpose of Message**
T_MODE	Server	UAV	Set mode of UAV
T_TYPE	Server	UAV	Set type of UAV
T_DATA	UAV/ Server	Server/ UAV	Exchange plant data
T_EXIT	Server	UAV	Exit UAV programs
T_ACK	UAV	Server	Acknowledgment
T_CONFIG_KPKV	Server	UAV	Send K_p, K_v gains
T_CONFIG_SGAIN	Server	UAV	Send S_i gain matrix
T_CONFIG_PGAIN	Server	UAV	Send P matrix
T_CONFIG_COMM_GRAPH	Server	UAV	Send allowed comms.
T_PATHTYPE	Server	UAV	Configure type of path

Table 6.2: Types of messages.

On the sender node side, the message data is wrapped in an object of class *Packet* using the helper functions. The packet is serialized using the function *GetByteStream* and resultant byte stream is sent over the network to the receiver node using the IP address of the node.

The receiver node receives the serialized byte streams sent over the network and needs to retrieve the packet, and thereby the message data. Using the function *recvfrom* in C++ *socket* programming, the incoming data bytes can be read to a buffer. But the read data in buffer may contain serialized byte streams of multiple packets. We need a mechanism to read all the packets from the data buffer.

A start byte signifies the start of a packet. In order to read out a packet, we need to find the start byte of the packet. There is no guarantee that we spot a start byte on the first read from the buffer. This implies that we need to search for start bytes. The value we chose as a start byte could clash with a byte value from the data, and wrongly pose as a candidate for a start byte.

Also, if there are missing bytes of a packet, we need to drop the section and start searching for a start byte again. The packet length can vary depending on the size of the message. In order to ensure all these requirements, we introduce a Finite-State Machine (FSM) based algorithm to readout packets from the data available in the read buffer.

A Finite-State Machine (FSM) or state machine is a model used in computer science in which, at any time, the model can be in exactly one state among a set of pre-defined finite states. State diagrams depict the operation

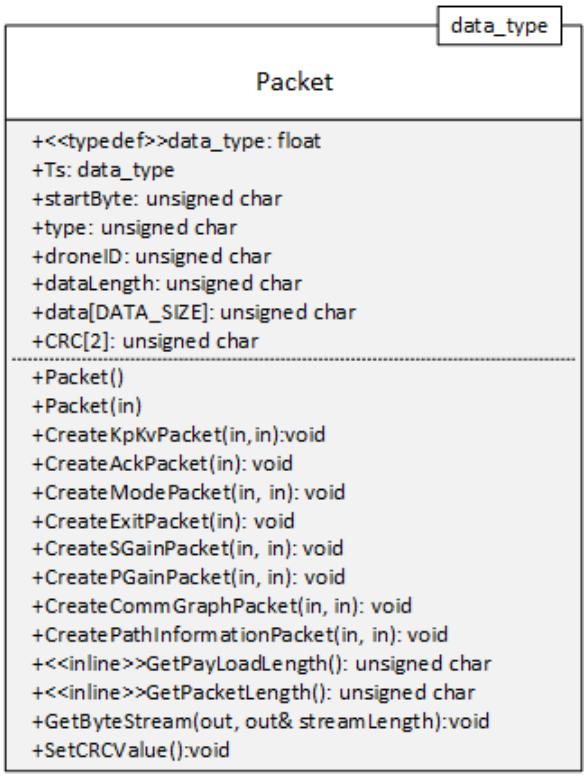


Figure 6.4: Class diagram for class *packet*.

of an FSM.

The state machine in packet reception algorithm has seven states. The finite states for the packet reception algorithm are: 1) Check Start Byte 2) Check Message Type 3) Get Data Length 4) Get Message 5) Check CRC0 6) Check CRC1 .

6.4 Server for Data Synchronisation

The Raspberry Pis send data to the server program using the protocol and mechanism introduced in the previous sections. The server needs to process the data packets from the bytes sent by each node and distribute it to other nodes which are in need.

The server program has to serve all the UAV nodes judiciously. The program should not listen and serve one node alone for a long period of time. In order to address this concern and to have enough decoupling, a multi-threaded server design is required.

There are four main tasks for the server, they are: 1) Handle user configurations and inputs. 2) Read bytes from all the nodes sent through the

network. 3) Process the read bytes to packets and check integrity. 4) Distribute the received packets to interested nodes.

The main thread of the server program executes task 1 and 2. Implementation for Task 1 is explained later in this chapter. For task 2, main thread continuously reads the data received from the network using the function *recevfrom*. In every call of the function *recevfrom*, together with the data, the address of the sender node is obtained. The main thread registers the received data in a map data structure with the address of the sender as the key.

6.5 Client for Data Synchronisation

Section 6.4 discusses how data is received and handled on the server side. The companion Raspberry Pi of UAVs also needs a software component to handle the data over the network. The software component has two purposes: 1) Send UAVs data to server 2) Receive data sent by server and process it.

Both the requirements are covered by implementing a class *Client*. Class *Client* runs a separate thread and operates on a singleton class *FormationData*. A singleton class has only one instantiated instance in a program. The same object is provided to all consumers through a static function. The implementation is made thread-safe using mutex locks.

6.6 Handling User Configurations & Inputs

XML files are widely used for configuration in software engineering. In this work also, the configuration is handled using a pre-formatted XML configuration file in the server. The XML nodes and their purpose in the configuration file are listed below.

1. *path*: User can set the type of path and the specifics of the path in this XML node.

2. *drones*: The drone ID and type of each drone involved in the formation are configured with this XML node.

3. *communications*: The allowed information flows between the drones are configured through this XML node.

4. *kpvalues* & *kvvalues*: These XML nodes helps in configuring the K_p and K_v gains.

5. *sgain* & *pgain*: User can input the values for S and P matrices through this node. Only non-zero values need to be listed along with respective zero-based column and row indices.

On startup or on request, the server reads the XML configuration file and generates packets to configure the UAVs. The server needs to ensure that the configuration data reaches the UAVs. Thus, a packet delivery mechanism with acknowledgements of reception from UAV is necessary.

The server maintains a queue of configuration data packets for each drone. For any drone, the queue stores only the latest packet of each type. The *dispatcher thread* in server periodically sends packets in the queue to the drones. The packets in the queue are not removed until there is an acknowledgement of reception.

The client in Raspberry Pis on the reception of a configuration package uses the instance of class ClientPacketHandler to configure the UAV. Also, the client sends a packet of message type T_ACK to the server. The acknowledgement packet has three data, the type and CRC bytes of the received packet.

Upon reception of the acknowledgement packet from the UAV at the server, the server finds the packet in the queue which has the same type as that of the packet being acknowledged. If there is a packet in the queue with the same type, the server program matches the CRC with the CRC of the package being acknowledged. On a match, the packet is removed from the queue.

The server program further reads the keyboard inputs. On hit of a configured key, the server program sends the corresponding configuration to the client. For example, when key 'f' is pressed, the server program sends T_MODE message to activate the formation control.

Chapter 7

Simulations and Results

This chapter discusses the simulations done as part of this work and the results of the simulations. The chapter is organised as follows: Section 7.1 describes the simulation and results for the path planner node for straight line,loitering and waypoints mission. Section 7.2 discusses the simulations and results for the synchronization of a set of UAVs using the adaptive estimation laws.

7.1 Simulations for Path Planner Node

The path planner node executes the path mission input by the user and generates a dynamics which all other agents should synchronize to. The parameters and initial conditions for simulation of path planner node are given in Table 7.1.

Parameter	**Value**
UAV Type	Path Planer
Mass (kg)	1.0
Inertial Position X_e (m)	[-100,400,-50]
Linear Velocity V_b (m/s)	[15,0,0]
Euler Angles $[\phi,\theta,\psi]$ (rad)	[0.1,0.1,1]
Angular Velocity ω_b (rad/s)	[1, 1, 2]
Moment of Inertia (kgm^2)	$\begin{bmatrix} 0.02 & 0 & -0.01 \\ 0 & 0.026 & 0 \\ -0.01 & 0 & 0.053 \end{bmatrix}$

Table 7.1: Parameters and initial conditions for path planner node.

For a straight line mission $\mathcal{SL}_1 = (\mathcal{P}, \mathcal{S})$, where line origin $P = (100, 0)$ and slope $S = (1, 1)$, the plot obtained for path planner in xy plane of inertial frame is as shown in Figure 7.1. The red colored plot represents the mission the UAV is expected to execute, the blue plot is the inertial position of UAV in the xy plane of inertial frame, and green shows the xy characteristics if the reference generator. It is important to quote that the plot of inertial position is mostly coinciding with the plot of reference dynamics, thus hidden. From the plot it is clear that the in simulation path planner UAV is able to execute the straight line mission from a given initial condition.

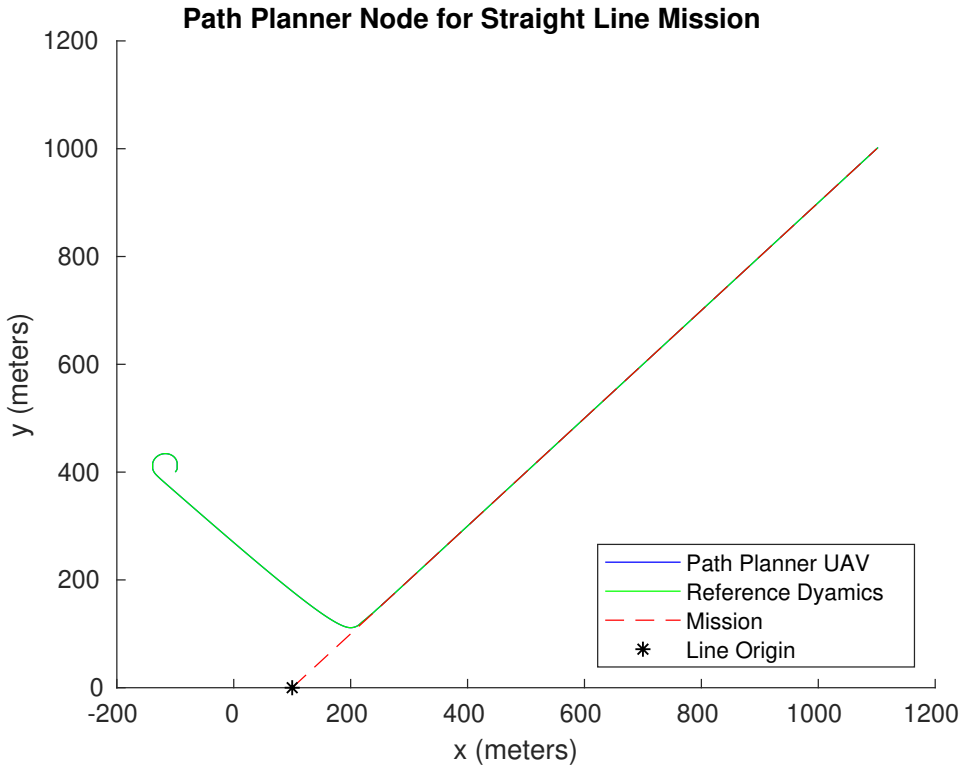


Figure 7.1: xy plot of path planner node executing straight line mission in inertial frame.

In the simulation for a loitering mission also, the UAV from the initial location moves gradually towards the orbit, smoothly merges into the orbit and starts loitering. The loitering mission used for simulation is $\mathcal{O}_1 = (\mathcal{D}_1, \mathcal{R}_1, \mathcal{C}_1)$, where radius $R_1 = 50m$, orbit centre $C_1 = (0, 0)$, and direction of rotation $\mathcal{D}_1 = 1$(clockwise). The plot obtained for path planner in xy plane of inertial frame is as shown in Figure 7.2. The colorings of the plot are the same as in plot in Figure 7.1. The plot is skewed and looks like an

ellipse because of the uneven scaling in horizontal and vertical axes.

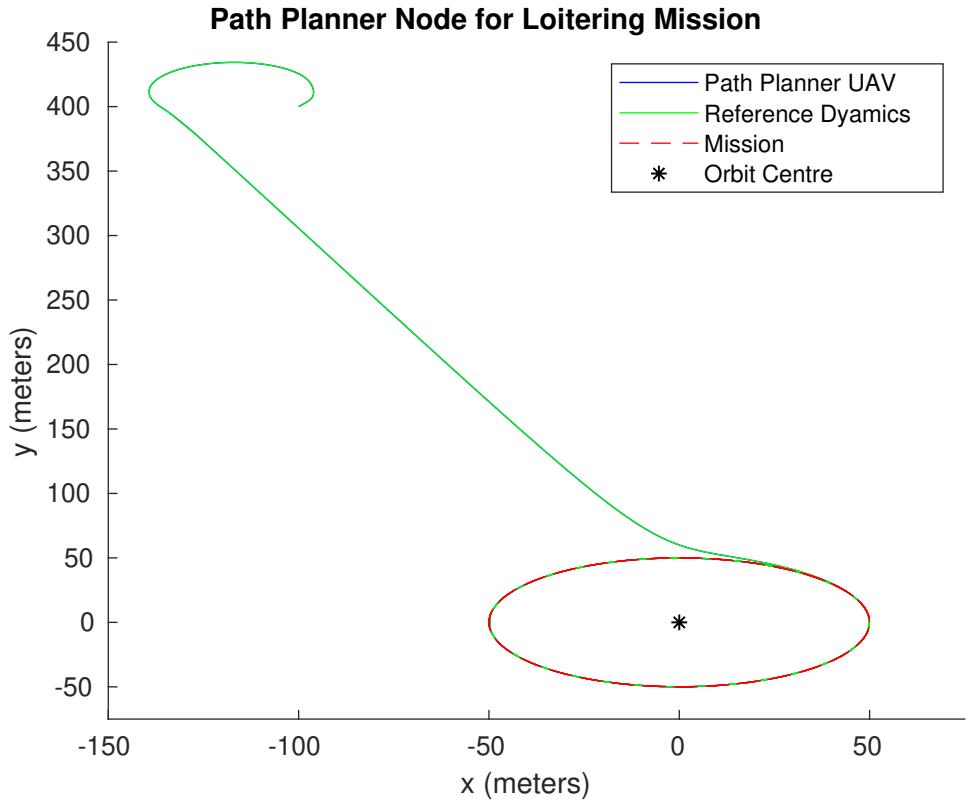


Figure 7.2: xy plot of path planner node executing loitering mission in inertial frame.

For waypoints mission, the simulations were done for a mission with parameters as summarised in Table 7.2. The mission has 5 straight line sections, 2 right and 2 left turns. The plot of location of UAV in xy inertial plane for this waypoint mission is in Figure 7.3. Since the UAV starts from a point away from the missions, the vector field approach for straight line missions gradually takes the UAV to the line connecting $[0, 0]$ and $[500, 500]$, and proceeds further along the path. From the plots, it is clear that complex missions can be split into primitive missions and executed thereon.

Radius (m)	**Waypoints (m)**					
	1	**2**	**3**	**4**	**5**	**6**
50	[0,0]	[500,500]	[500,0]	[0,500]	[0,0]	[500,500]

Table 7.2: Parameters of waypoint mission.

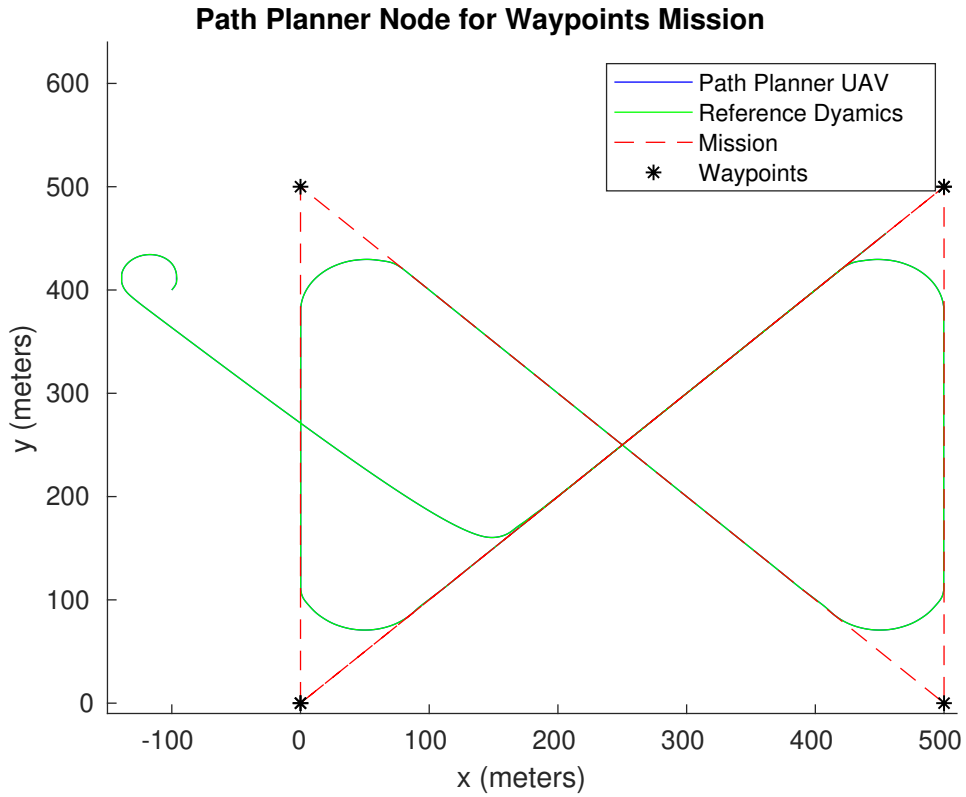


Figure 7.3: xy plot of path planner node executing waypoints mission in inertial frame.

A close-up image of UAV executing turn around (500,500) shown in Figure 7.4 reveal that there is a minute mismatch in following the path. The vector field approach only guarantees the lowest cross-tracking error. While following the arc of orbit path near (500,500), there is a small cross-track error. This implies that, the direction of velocity at the point of switch from orbit to line path, is not exactly coinciding with the straight line mission. The adaptive vector field approach for straight line mission gradually takes the UAV back to the straight line mission.

7.2 Synchronisation of Leader and Follower Node

Recall that the simulations for synchronisation in this work are done for the system of UAVs introduced in Figure 4.2. A normal laptop is used as the server, and Raspberry Pis as the UAV nodes. The nodes in the system are connected to the server through LAN. The configurations for the UAVs were done through the XML configuration file of the server program. Also,

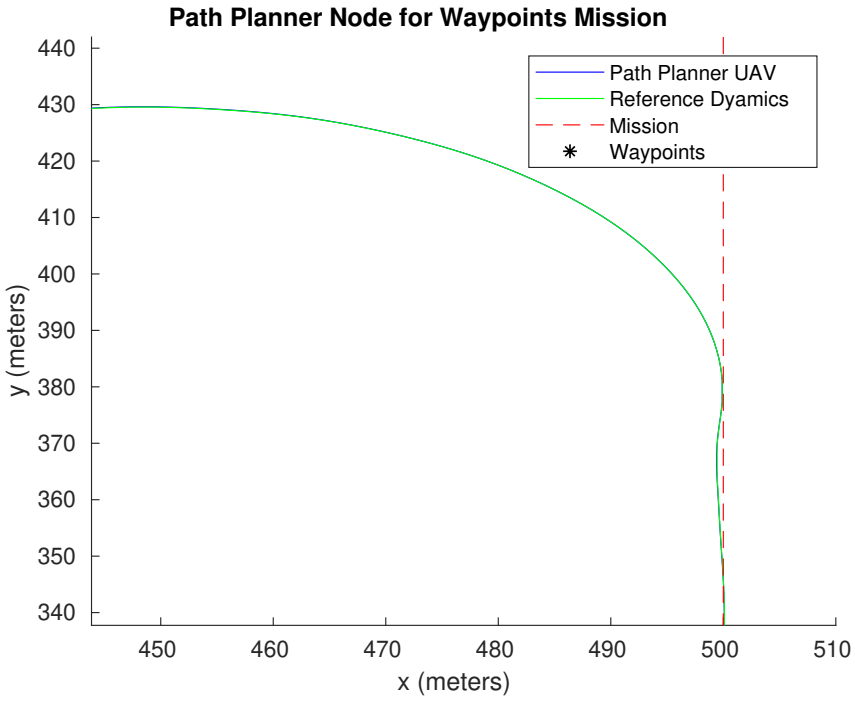


Figure 7.4: Close-up image of xy plot of path planner node executing waypoints mission near to waypoint (500,500).

the UAVs exchange real time data through the server. The real time data received in the server are written to a flat file and used for creating the plots using MATLAB.

The parameters for the leader and follower nodes used for simulations are as given in Table 7.3. Both the leader and follower nodes start from different inertial positions. The simulations were executed for cases of straight line and loitering mission. The simulations were tuned using the gains and matrices in adaptive estimation laws. The obtained values after tuning are summarized in Appendix **??**.

Figure 7.5 shows the plot for synchronization in a straight line mission. The blue coloured plots are the reference dynamics generated by the path planner node, the red by the leader node, and the yellow plot by the follower node. From the plots it can be concluded that the inertial positions of UAVs converges over time starting from their respective initial positions. The glitches in the plot could be caused due to inconsistencies in the network or the errors induced by the discretization of estimation laws using euler approximation.

The results for synchronization of UAVs for simulation of a loitering mis-

	Leader	**Follower**
Mass (kg)	1.0	1.0
Inertial Position X_e (m)	[500,500,-50]	[0,500,-50]
Linear Velocity V_b (m/s)	[15,0,0]	[15,0,0]
Euler Angles $[\phi,\theta,\psi]$ (rad)	[0.1,0.1,1]	[0.1,0.1,1]
Angular Velocity ω_b (rad/s)	[1, 1, 2]	[1, 1, 2]
Moment of Inertia (kgm^2)	$\begin{bmatrix} 0.02 & 0 & -0.01 \\ 0 & 0.026 & 0 \\ -0.01 & 0 & 0.053 \end{bmatrix}$	$\begin{bmatrix} 0.02 & 0 & -0.01 \\ 0 & 0.026 & 0 \\ -0.01 & 0 & 0.053 \end{bmatrix}$

Table 7.3: Parameters and initial conditions for leader and follower nodes.

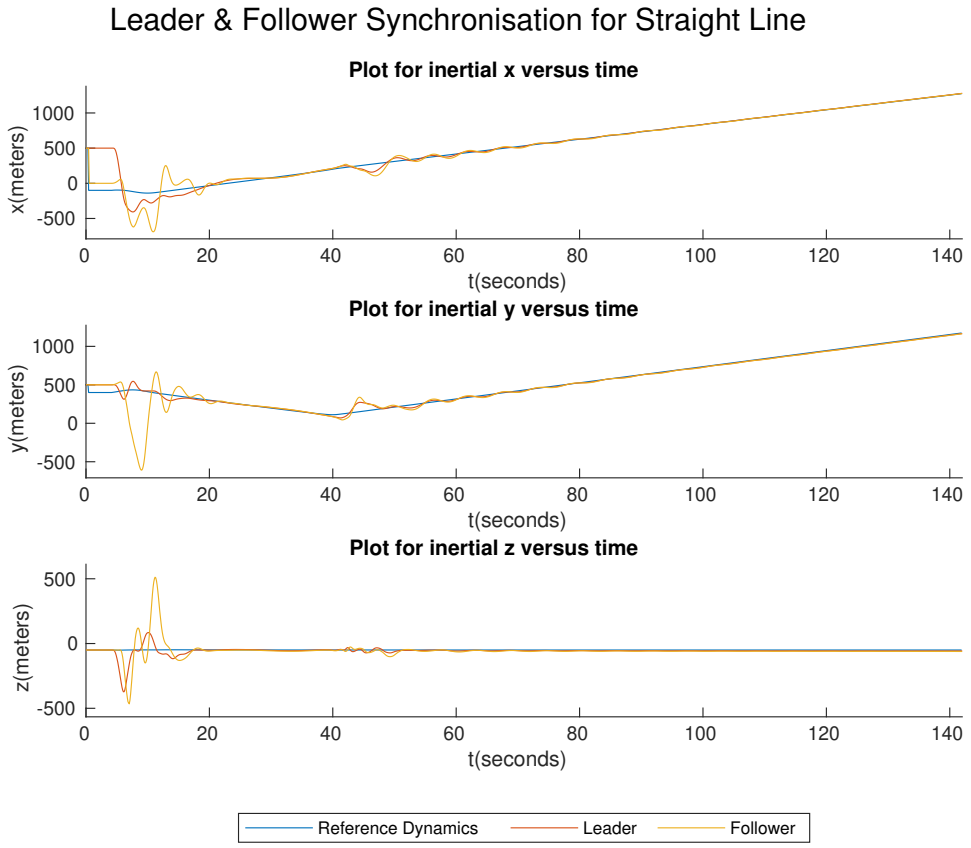


Figure 7.5: Plot for the location of leader and follower UAVs for synchronization in a straight line mission.

sion is given in Figure 7.6, and are very similar to the case of synchronization in straight line missions. The leader and follower UAVs starting from different inertial positions synchronises to the reference dynamics generated by the path planner node for the loitering mission.

Since the UAVs can be synchronised using the above approach, formation gaps can be provided to enable the UAVs to fly in a required formation.

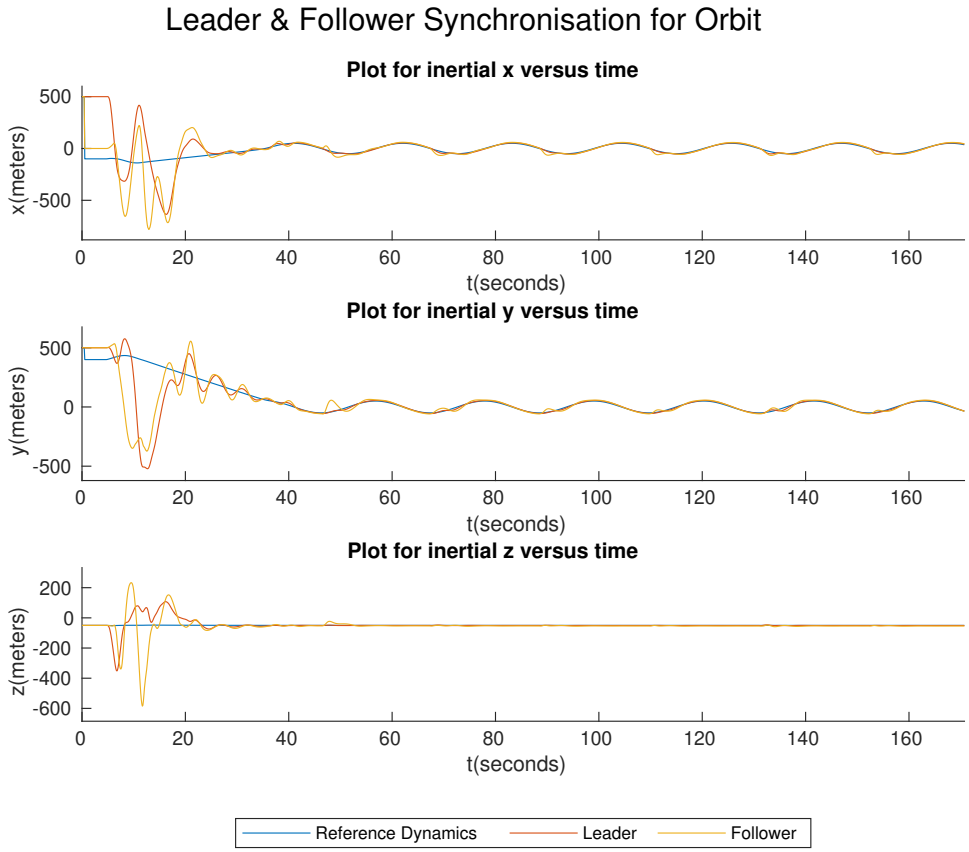


Figure 7.6: Plot for the location of leader and follower UAVs for synchronization in a loitering mission.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

In this work, we designed, implemented and simulated an autopilot component with emphasis on formation control of fixed-wing UAVs. The work is carried out in Raspberry Pis which can be extended to real UAV scenarios with ardupilot or PX4 autopilots as companion computers. The simulations are done in a distributed manner in with the help of a server designed and implemented for this work.

The work implements and validates the adaptive vector field approach for following a path, and is successful in executing straight line or loitering or waypoints mission. The way-point missions are split into primitive missions of straight line or orbit.

This work, also do the design, implementation and simulation of the MRAC synchronisation in a distributed environment which can be used for formation control in UAVs. The server configures all the nodes and handles the inter-node communications. The simulations confirm that the approach works over the distributed network, and the outputs converges for both the primitive missions - straight line and loitering.

The dynamics of the fixed-wing UAV is simulated with in C++ environment in the Raspberry Pis using a mathematical model of fixed-wing UAVs and odeint library. The above approach is advantageous to do simulations inside distributed computers itself, without depending on a central computer to simulate the dynamics for all the nodes. Since the simulator considers the aerodynamics, propulsion effect and low-level controller dynamics, using the model SITL for any adaptive laws can be done more realistically.

8.2 Future Work

This work opens up a plethora of future works which can be based on the results, among which the important ones are,

- A hardware-in-the-loop simulation of the approaches in this work, using the C++ libraries developed as part of this work.
- Using technologies like ad-hoc Wi-Fi or Zigbee to have internode communications and comparing effectiveness of each on the adaptive laws.
- Design of a graphical user interface to configure the UAVs and visualize the real time data.
- Developing and testing discrete time estimation laws instead of the euler approximated adaptive estimation laws used.
- The MRAC multi-agent synchronization approach does not take into account of control allocation. Analysis need to be done to incorporate the effect of control allocation for fixed-wing UAVs.

Bibliography

[1] R. Yanushevsky, *Guidance of unmanned aerial vehicles*. CRC press, 2011.

[2] J. F. Guilmartin, "Unmanned aerial vehicle," Jul 2018.

[3] J. Gunnar Carlsson and S. Song, "Coordinated logistics with a truck and a drone," *Management Science*, vol. 64, 10 2017.

[4] R. G. L. Narayanan and O. C. Ibe, "6 - joint network for disaster relief and search and rescue network operations," in *Wireless Public Safety Networks 1* (D. Cmara and N. Nikaein, eds.), pp. 163 – 193, Elsevier, 2015.

[5] M. R. Haque, M. Muhammad, D. Swarnaker, and M. Arifuzzaman, "Autonomous quadcopter for product home delivery," in *2014 International Conference on Electrical Engineering and Information Communication Technology*, pp. 1–5, April 2014.

[6] G. Cai, J. Dias, and L. Seneviratne, "A survey of small-scale unmanned aerial vehicles: Recent advances and future development trends," *Unmanned Systems*, vol. 02, pp. 175–199, 04 2014.

[7] M. Boon, A. P. Drijfhout, and S. Tesfamichael, "Comparison of a fixed-wing and multi-rotor uav for environmental mapping applications: A case study," *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLII-2/W6, pp. 47–54, 08 2017.

[8] A. Filippone, *Flight Performance of Fixed and Rotary Wing Aircraft*. Elsevier Aerospace Engineering, Elsevier Science, 2006.

[9] K.-K. Oh, M.-C. Park, and H.-S. Ahn, "A survey of multi-agent formation control," *Automatica*, vol. 53, pp. 424 – 440, 2015.

[10] G. Weiss, ed., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999.

[11] G. Ellis, "Chapter 13 - model development and verification," in *Control System Design Guide (Fourth Edition)* (G. Ellis, ed.), pp. 261 – 282, Boston: Butterworth-Heinemann, fourth edition ed., 2012.

[12] F. Administration, *Advanced Avionics Handbook: FAA-H-8083-6*. Skyhorse Publishing, 2011.

[13] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6235–6240, 2015.

[14] "Ardupilot open source autopilot." [Online; accessed 02-July-2019].

[15] E. Anderson, R. Beard, and T. McLain, "Real-time dynamic trajectory smoothing for unmanned air vehicles," *Control Systems Technology, IEEE Transactions on*, vol. 13, pp. 471 – 477, 06 2005.

[16] S. P.B, S. Saripalli, and J. Borges Sousa, "Unmanned aerial vehicle path following: A survey and analysis of algorithms for fixed-wing unmanned aerial vehicless," *Control Systems, IEEE*, vol. 34, pp. 42–59, 02 2014.

[17] D. R. Nelson, D. B. Barber, T. W. McLain, and R. W. Beard, "Vector field path following for small unmanned air vehicles," in *2006 American Control Conference*, pp. 7 pp.–, June 2006.

[18] Bingyu Zhou, H. Satyavada, and S. Baldi, "Adaptive path following for unmanned aerial vehicles in time-varying unknown wind environments," in *2017 American Control Conference (ACC)*, pp. 1127–1132, May 2017.

[19] S. FARI, "Guidance and control for a fixed-wing uav," MSc thesis, Politecnico di Milano, 20162017.

[20] S. Fari, X. Wang, S. Roy, and S. Baldi, "Addressing unmodelled path-following dynamics via adaptive vector field: a uav test case," *IEEE Transactions on Aerospace and Electronic Systems*, pp. 1–1, 2019.

[21] W. Yu, G. Chen, and J. L, "On pinning synchronization of complex dynamical networks," *Automatica*, vol. 45, no. 2, pp. 429 – 435, 2009.

[22] W. Yu, P. Delellis, G. Chen, M. Di Bernardo, and J. Kurths, "Distributed adaptive control of synchronization in complex networks," *Automatic Control, IEEE Transactions on*, vol. 57, pp. 2153 – 2158, 01 2012.

[23] M. R. Rosa, S. Baldi, X. Wang, M. Lv, and W. Yu, "Adaptive hierarchical formation control for uncertain euler–lagrange systems using distributed inverse dynamics," *European Journal of Control*, 2018.

[24] I. Azzollini, S. Baldi, and E. Kosmatopoulos, "Adaptive synchronization in networks with heterogeneous uncertain kuramoto-like units," in *17th European Control Conference (ECC)*, pp. 2417–2422, 06 2018.

[25] S. Baldi, I. Azzollini, and E. Kosmatopoulos, "A distributed disagreement-based protocol for synchronization of uncertain heterogeneous agents," in *17th European Control Conference (ECC)*, pp. 2411–2416, 06 2018.

[26] S. Baldi and P. Frasca, "Leaderless synchronization of heterogeneous oscillators by adaptively learning the group model," *IEEE Transactions on Automatic Control*, pp. 1–1, 2019.

[27] R. Chapa-Garcia, M. Jimenez-Lizarraga, O. Garcia, and T. Espinoza-Fraire, "Formation flight of fixed-wing uavs based on linear quadratic affine game," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 736–741, June 2016.

[28] Y. Nagao and K. Uchiyama, "Formation flight of fixed-wing uavs using artificial potential field," in *29th Congress of the International Council of the Aeronautical Sciences*, 2014.

[29] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. von Stryk, "Comprehensive simulation of quadrotor uavs using ros and gazebo," in *Simulation, Modeling, and Programming for Autonomous Robots* (I. Noda, N. Ando, D. Brugali, and J. J. Kuffner, eds.), (Berlin, Heidelberg), pp. 400–411, Springer Berlin Heidelberg, 2012.

[30] W. Rigon Silva, A. da Silva, and H. Ablio Grndling, "Modelling, simulation and control of a fixed-wing unmanned aerial vehicle (uav)," in *24th ABCM International Congress of Mechanical Engineering*, 12 2017.

[31] H. Goldstein, *Classical Mechanics*. Pearson Education, 2002.

[32] T. R. Kane and D. A. Levinson, *Dynamics, theory and applications*. McGraw Hill, 1985.

[33] P. H. Zipfel, *Modeling and Simulation of Aerospace Vehicle Dynamics (2nd Edition)*. American Institute of Aeronautics and Astronautics, 2007.

[34] J. Roskam and C. Lan, *Airplane Aerodynamics and Performance*. Airplane design and analysis, DARcorporation, 1997.

[35] PX4 Dev Team, "Controller diagrams."

[36] A. Lambregts, *Vertical flight path and speed control autopilot design using total energy principles*.

[37] R. Stengel, *Flight Dynamics*. Princeton University Press, 2015.

[38] K. Ahnert and M. Mulansky, "Odeint solving ordinary differential equations in c++," *AIP Conference Proceedings*, vol. 1389, 10 2011.

[39] E. W. Frew, D. A. Lawrence, C. Dixon, J. Elston, and W. J. Pisano, "Lyapunov guidance vector fields for unmanned aircraft applications," in *2007 American Control Conference*, pp. 371–376, July 2007.

[40] J. Bang-Jensen and G. Gutin, *Digraphs: Theory, Algorithms and Applications*. Springer Monographs in Mathematics, Springer London, 2008.

[41] K. Ogata, *Modern Control Engineering*. Instrumentation and controls series, Prentice Hall, 2010.

[42] PX4 Dev Team, "Cube flight controller."

[43] The Editors of Encyclopaedia Britannica, *Wi-Fi*. By Example Series, Encyclopdia Britannica, inc., 2017. [Online; accessed 06-July-2019].

[44] S. Dasgupta, *Encyclopedia of virtual communities and technologies*. Gale virtual reference library, Idea Group Reference, 2006.

[45] W. Gay, *Linux Socket Programming by Example*. By Example Series, Que, 2000.