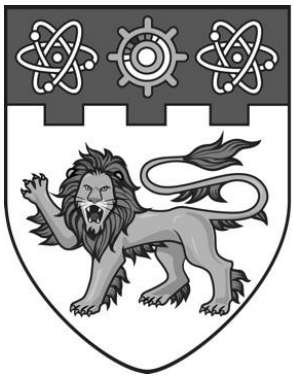


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Final Year Project

An Implementation of Multi-Agent System

Submitted by: Hoang Minh Chung*
Matriculation Number: U1320792H

Supervisor: Prof. Xie Lihua**
Co-supervisor: Asst Prof. Lau Gih Keong*

**School of Electrical & Electronic Engineering
*School of Mechanical & Aerospace Engineering

A final year project report presented to the Nanyang Technological University
in partial fulfilment of the requirements of the degree of Bachelor of Engineering

April, 2017

Contents

Abstract

Having multiple robots collaboratively tackling a particular task is sometimes more desirable than doing so with one complicated robot. That motivation has been aspiring countless researches on "swarm", also formally called Multi-agent System, for decades. This project is an attempt to implement a functional multi-agent system, based on Robot Operating System, tested on two heterogeneous robots. Various implementations in the past assume each vehicle is one agent which makes agents become hardware-specific. However, to achieve a more robust system, in this project, the concept of agent is constrained to a software node, separated from the concept of vehicle, which is a hardware platform. In this way, agents are not dependent of the type of vehicle or hardware capabilities, and thus can be generalized into three types: Command, Data and Action. Furthermore, with this concept, one platform now can carry more than one agents and agents could flexibly migrate from one hardware platform to another through the network in the case of hardware failures. In this project, two types of vehicle are used: unmanned ground vehicle (UGV) and unmanned aerial vehicle (UAV). To ensure that individual robots are ready for collaborative missions, they are equipped with various necessary hardwares and software controllers. The underlying theories, the designs and selections of hardware and the implementations and tests of operation are also discussed. Last but not least, precision landing experiments, which involves simultaneous independent control of multiple robots are conducted to verify the robustness of the implementation. During the experiments, the UAV attempts to search and land on the UGV. The results show a consistent accuracy of 10cm for both cases of static and moving UGV. However, the UAV took longer time to catchup and approach in the case of moving UGV.

Acknowledgement

First of all, I would like to express my heartfelt gratitude towards Professor Xie Lihua and Asst Professor Lau Gih Keong, for their kind support. Moreover, as an inter-school, multi-disciplinary project that requires extra administrative clearances, it would not have been possible without their trust and guidance.
I would also like to thank mom, dad and Nguyen for their care, their understanding and their tireless support throughout this tough time.
Last but not least, I would also like to thank Mr Wang Chen for his tremendous support with the vision perception, Dr Chen Chun-Lin for his guidances on ground robot control, Mr Thien-Minh Nguyen-Pham for his kind help on setting up ground truth system and Mr Hanif on advises on vision-imu fusion. Without their kind support and consideration, the experiments of this project would not have been possible.
I wish to acknowledge the funding support for this project from Nanyang Technological University under the Undergraduate Research Experience on CAmpus (URECA) programme.

To dad, mom and Nguyen, ...

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Multi-agent system (MAS) refers to a system of multiple robots set out for a common goal. MAS has been an active research topic for the past decades [5], with many promising future applications including Traffic Control, Formation Control [9], Cooperative Manipulation [20], Collaborative Obstacle Avoidance [4], Foraging [8] and many others. The MAS is an appealing research topic because an MAS system is usually a stochastic, non-linear one, which is challenging to pin down a mathematical model. Besides, practical attractiveness of MAS pertains when certain tasks are either inherently too complex for a single robot to accomplish or significantly beneficial to, instead, be carried out by many simpler/cheaper robots. Moreover, some of MAS research works could also shed lights into more complicated social and biological behaviors such as organization theory, cognitive psychology or animal ethology.
In many literatures, multi-agent systems (MAS) are also coined by the term "swarm" [8], which further emphasize its distributed characteristics. A swarm does not have a leader, or a centralized control agent, that coordinate the activities of its sub-individuals. This implies that the system is robust against weaknesses of individuals, flexible with changes and quickly scalable if necessary.

1.2 Objectives and scopes

Through literature survey, the author has come to conclude that a functional swarm would require three elements: a set of individual robots, a shared goal and a network between them. Firstly, a goal gives meanings to actions that follow it and a swarm is of no exception. In many cases, such as the Social Foraging problem, the goal is the benchmark to compare the utility of a distributed system against that of centralized ones. Secondly, a network between robots determines the mode of operation in a swarm. According to [5], three main modes of interaction are interaction via environment, interaction via sensing and interaction via communication. Last but not least, even though the complexity of swarm sometimes shadows the appeal of individual mobile robots, it is important to note that any swarm would not be possible without the individuals in it. Many a time, the utility of a MAS is driven or limited by the individual capabilities.
This project conveys an undergraduate student's attempt to implement a multi-agent system, bearing in mind the three important pillars that stand the concept of swarms: common goal, individual robots and inter-robot network. In this report, the terms 'multi-agent system (MAS)' and 'swarm' would be used interchangeably, even though other literatures might caution such practice.

1.3 Report organization

The structure of this report is illustrated in Figure 1. Firstly, for individual robots, the author attempts to revisit important concepts that build up current state-of-the-art capabilities of mobile robots. Most notable corner stones are Kalman Filter, Extended Kalman Filters, which allow robust attitude estimation and derive solutions to traditional problem of Simultaneous Localization and Mapping (SLAM). SLAM itself is also one of the most important capabilities that make robots truly autonomous [23], which is why tremendous works has been invested into various approaches of SLAM solutions, including traditional landmark-based SLAM, laserscan-based SLAM or vision-based SLAM. These concepts would be briefly revisited in Section 2.1 to 2.5 of this report.
Moreover, since this project involves non-homogeneous robots, including one ground mobile robot (UGV) and one aerial vehicle (UAV), the hardware capabilities, including actuators and sensors, are also briefly discussed in Section 3.2 and 3.3. In order to simulate and control these vehicles, their mathematical models are also visited in section 2.5 and 2.6.

The ultimate aim of this project is to arrive at a practical concepts and a functional implementation and a flexible testbed for MAS. The author believes that a scalable and robust implementation of MAS comes with a distinction between two usually interchangeably used concepts: agent and vehicle. In this project, vehicles refer to mobile hardware units, whereas agents refer to software units that perform a specific tasks. With this distinction, a vehicle could carry multiple agents and, because of being a software in a networked system, agents can migrate from one vehicle to another in case of vehicle failures. Moreover, the author classified agents into three types: Command agent, Data agent and Action agent. The implementation and deployment of these agents are vehicle independent.
To verify the performance of the software implementation and the scalability of the proposed MAS concept, various experiments were conducted. Individual tests, including velocity control, position control, SLAM are done on both omni-direction ground vehicle and aerial vehicles. Precision landing experiments, both on static landing pad and moving landing pad, were also conducted to verify performance of multiple robots and multiple agents when working together. Last but not least, as shown in Figure 1, this project still left a huge corner stone untouched, due to time constraint and limited resources. That important gap is the Intelligence of individual robots, which refers to the ability of individual robot to 'learn' from its sensory perception and obtain non-programmatic understandings of the environments that eventually lead to meaningful yet non-preprogrammed and logic-based reactions.

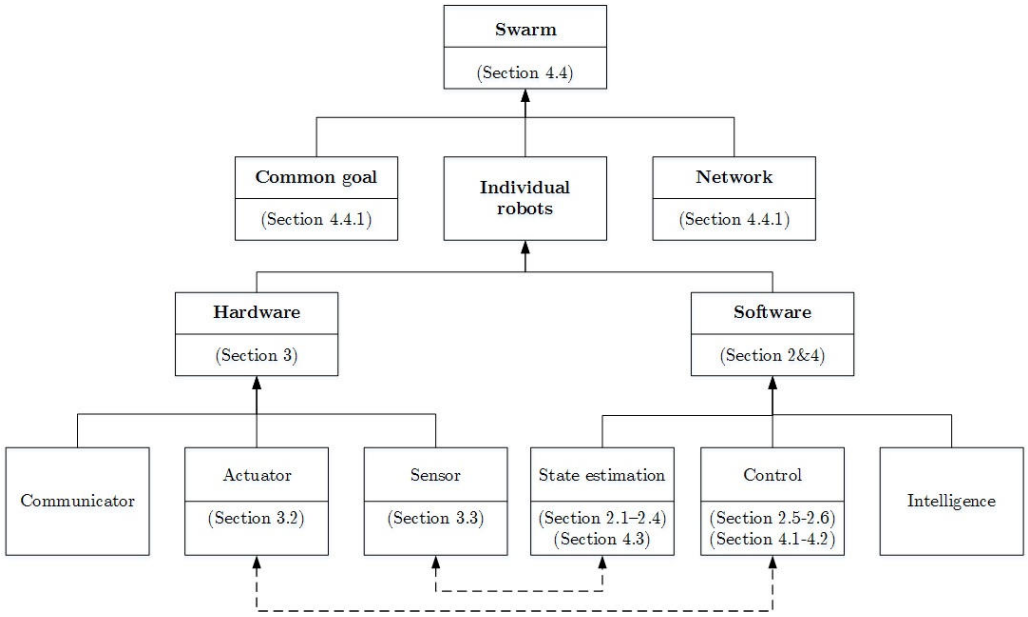


Figure 1: Report structure

2 Theoretical background

2.1 Kalman Filter (KF) and Extended Kalman Filter (EKF)

The KF and the EKF are one of the most important algorithms in modern robotics. Among wide range of applications, it provides the basis for vehicle state estimation and Simultaneous Localization and Mapping (SLAM), which will be discussed in later section of this article. In this section, for the ease of reading, time will be written in subscript. For example $x(t)$ will be written as x_t.
The KF assumes a robot model where the state of the robot evolves overtime from $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t+1}$ obeying equation (1), and state measurement $\boldsymbol{z}_{t+1}$ obeying equation (2):

$$\boldsymbol{x}_{t+1} = \boldsymbol{F}_{t+1}\boldsymbol{x}_t + \boldsymbol{B}_{t+1}\boldsymbol{u}_{t+1} + \boldsymbol{w}_{t+1} \tag{1}$$

$$\boldsymbol{z}_{t+1} = \boldsymbol{H}_{t+1}\boldsymbol{x}_{t+1} + \boldsymbol{v}_{t+1} \tag{2}$$

where:

$\boldsymbol{F}_{t+1}$ is state-transition matrix,
$\boldsymbol{B}_{t+1}$ is control input matrix,
$\boldsymbol{H}_{t+1}$ is measurement matrix,
$\boldsymbol{w}_{t+1}$ is process noise with covariance $\boldsymbol{Q}_{t+1}$,
$\boldsymbol{v}_{t+1}$ is measurement noise with covariance $\boldsymbol{R}_{t+1}$.
$\hat{\boldsymbol{x}}_{t+1}$ is state estimate with covariance $\boldsymbol{P}_{t+1}$

Provided with such model, the KF can recursively perform state estimation via two steps: state prediction (3) (4) and measurement update (5) (4).

$$\hat{\boldsymbol{x}}_{t+1|t} = \boldsymbol{F}_{t+1}\hat{\boldsymbol{x}}_{t|t} + \boldsymbol{B}_{t+1}\boldsymbol{u}_{t+1} \tag{3}$$

$$\boldsymbol{P}_{t+1|t} = \boldsymbol{F}_{t+1}\boldsymbol{P}_{t|t}\boldsymbol{F}_{t+1}^T + \boldsymbol{Q}_{t+1} \tag{4}$$

$$\hat{\boldsymbol{x}}_{t+1|t+1} = \hat{\boldsymbol{x}}_{t+1|t} + \boldsymbol{K}_{t+1}(\boldsymbol{z}_{t+1} - \boldsymbol{H}_{t+1}\hat{\boldsymbol{x}}_{t+1|t}) \tag{5}$$

$$\boldsymbol{P}_{t+1|t+1} = \boldsymbol{P}_{t+1|t} - \boldsymbol{K}_{t+1}\boldsymbol{H}_{t+1}\boldsymbol{P}_{t+1|t} \tag{6}$$

Where Kalman gain $\boldsymbol{K}_{t+1}$ is defined as

$$\boldsymbol{K}_{t+1} = \boldsymbol{P}_{t+1|t}\boldsymbol{H}_{t+1}^T(\boldsymbol{H}_{t+1}\boldsymbol{P}_{t+1|t}\boldsymbol{H}_{t+1}^T + \boldsymbol{R}_{t+1})^{-1} \tag{7}$$

The Extended Kalman Filter (EKF) possesses similar characteristics. It also requires a model comprising of state transition and measurement update, both accompanied by Gaussian noise, similar to equations (1) and (2). However, EKF can be used for non-linear systems by linearizing the system dynamics about its most recent estimate through updating $\hat{\boldsymbol{x}}_t$, $\boldsymbol{K}_t$ and $\boldsymbol{P}_t$ at each iteration [22].

2.2 Simultaneous localization and mapping (SLAM)

Simultaneous localization and mapping (SLAM), as the name speaks for itself, is a problem where a robot is required to determine its own pose in an unknown environment, while at the same time trying to build a consistent map of the environment.
Figure 2 demonstrates a typical process of SLAM where a robot estimates its position based on one or multiple observed landmark in the environment and then subsequently estimates the position of surrounding landmark(s), or also called "mapping". Mathematically, the SLAM problem can be described as the relationship and co-evolution of:

- $\mathbf{x}_k$: the robot pose,
- $\mathbf{u}_k$: the robot control vector,
- $\mathbf{m}_i$: the landmark position and
- $\mathbf{z}_{ik}$: the landmark observation

where k represents the time index of the parameters. Putting the various parameters into its timeseries as:

- $\mathbf{X}_{0:k} = \{\ \mathbf{X}_{0:k-1}, \mathbf{x}_k\}$: set of vehicle states
- $\mathbf{U}_{0:k} = \{\mathbf{U}_{0:k-1}, \mathbf{u}_k\}$: set of control inputs

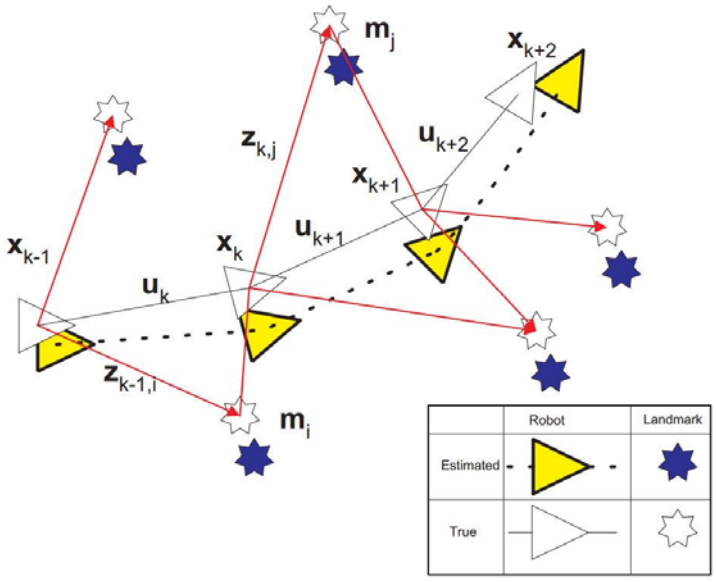


Figure 2: SLAM problem illustration [1]

- $\mathbf{m}_{0:n} = \{ \mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_n\}$: set of landmark locations
- $\mathbf{Z}_{0:k} = \{ \mathbf{Z}_{0:k-1}, \mathbf{z}_k\}$: set of landmark observations

If the SLAM problem is to be modeled in probabilistic form, it would be "to determine the probability distribution of the robot pose and the location(s) of landmark(s), given the history of control input, the history of landmark observations and initial assumingly known pose". Or:

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) \tag{8}$$

For mobile robot application, this problem is often solved recursively using Bayesian Theorem, where a state transition model and an observation model are required. Besides, an important assumption made, also known as Markov model assumption, is that the robot state $\mathbf{x}_k$ depends only on the immediate previous state $\mathbf{x}_{k-1}$ and the input $\mathbf{u}_k$ and independent of landmarks observation and map.
Firstly, the **state transition model** (9) represents the probabilistic distribution of the robot pose, provided the *priori* pose and the current motion input.

$$P(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k) \tag{9}$$

Secondly, the **observation model** (10) is the probabilistic distribution of the landmark position, provided a known robot pose and a set of landmark locations.

$$P(\mathbf{z}_k|\mathbf{x}_k, \mathbf{m}) \tag{10}$$

With the requirements satisfied, the SLAM problem can then be solved recursively with two steps **Prediction** (11) and **Correction** (12) [1].

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0) = \int P(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k) P(\mathbf{x}_{k-1}, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1}, \mathbf{x}_0) d\mathbf{x}_{k-1} \tag{11}$$

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) = \frac{P(\mathbf{z}_k|\mathbf{x}_k, \mathbf{m}) P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0)}{P(\mathbf{z}_k|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k})} \tag{12}$$

In general, there are three common approaches to solving the SLAM problem, namely Extended Kalman Filter (EKF), Rao-Blackwellised Particle Filter (FastSLAM) and Information Filter.
One of the key draw-backs of traditional SLAM methods, including EKF SLAM and FastSLAM,

is its demanding requirements on observation model and landmark model. Landmarks must be perceived and their poses must be available in order for traditional SLAM to work. However, in most practical applications, sensors only partially observe the landmark, which does not guarantee landmark detection, even if the detection is reliable.

2.3 Laser scan based SLAM

In practice, laser scanners, either 2D or 3D, are widely used for SLAM purposes even though most of the implementations are not based on traditional SLAM theory. One particular challenge in implementing stochastic SLAM with laser scan is that landmark recognition from point clouds is usually difficult. In stead, many applications make use of high scan data rate [11] and powerful computers to attempt direct scans alignment to determine the pose transform. The well-known algorithm that represents this approach is Iterative Closest Point (ICP) [3,13]. The essence of the algorithm is to try to determine the best correlation between two different scans, using a predefined cost function. The advantage of ICP is that it is theoretically proved to "always converge monotonically to a local minimum with respect to the mean-square distance objective function" [3]. The disadvantage of ICP is its complexity, which significantly limits its application on vehicles that have scarce computational resources.
A summary of ICP algorithm is presented here for readers' convenience. Let P_A be the scan obtained in frame A and P_B the scan in frame B. Let $T(X)$ represents a frame transformation on a point set X. The goal of ICP is to determine the transformation ${}_AT^B$ that satisfies

$$ {}_AT^B = arg\ min_T(error(T(P_A), P_B)) $$

Innovative variations of ICP also include the use of polar coordinates [6,7], which is natural with most laser scanners and the use of predefined laser scan templates for fast landmark detection, which relates ICP to traditional stochastic SLAM [16].

2.4 Vision based SLAM (vSLAM)

Another well-known SLAM method that utilizes camera images is called Visual SLAM (vSLAM). Many computer vision literatures also uses the term *Bundle Adjustment* (BA) to describe the same concept [24]. The word *bundle* refers to the beam of light originating from 3D objects, arriving at the sensor. The word *adjust* refers to the procedure of alignment, through which structures and sensor poses (and parameters) are *jointly* estimated. Similar to ICP, BA is essentially a problem of iterative error minimization.
Most BA algorithms requires features extraction from raw images. Notable researches on feature detection can be dated back as early as 1981 to the stereo corner matching of Moravec [15], which was later improved by Harris [10]. In 2004, Lowe [12] further made the detector scale invariant and more robust against image local distortion by introducing Scale-Invariant Feature Transform (SIFT). The usefulness of SIFT was verified by various applications including mobile robot localization [20] and image stitching [21]. Even though SIFT is robust, it is also computationally expensive, which is improved by Feature from Accelerated Segment Test (FAST) [18], Oriented FAST and rotated BRIEF [19] and Speeded Up Robust Feature (SURF) [2].

2.5 Quad-rotor unmanned aerial vehicle (UAV)

Assuming a quadcopter in steady state hovering flight condition, gravitational forces and thrusts from four motors produce its motion, in the inertial frame. Besides, the attitude estimation from the IMU, also in inertial frame, is used to control the vehicle. On the other hand, the thrust output is in copter body frame. Therefore, the dynamics of the quadcopter required transformation from the body to inertial frame and vice versa. The rotation matrix from inertial frame to body frame is :

$$ R = \begin{bmatrix} C_\phi C_\psi - C_\theta S_\phi S_\psi & -C_\psi S_\phi - C_\phi S_\theta S_\psi & S_\theta S_\psi \\ C_\theta C_\psi S_\phi + C_\phi S_\psi & C_\phi C_\theta C_\psi - S_\phi S_\psi & -C_\psi S_\theta \\ S_\phi S_\theta & C_\phi S_\theta & C_\theta \end{bmatrix} $$

where S_x and C_x represent $\sin(x)$ and $\cos(x)$ respectively and ϕ, θ, ψ are, in inertial frame, the roll, pitch and yaw angle of the quadcopter.

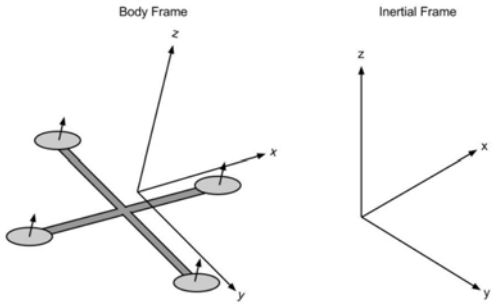


Figure 3: Quadcopter frames illustrations

Let the position, linear velocity, attitude and angular velocities in the inertial frame be denoted as $\boldsymbol{p}$, $\dot{\boldsymbol{p}}$, $\boldsymbol{\xi}$ and $\dot{\boldsymbol{\xi}}$ respectively. We have:

$$\boldsymbol{p} = (x, y, z)^T \qquad \dot{\boldsymbol{p}} = (\dot{x}, \dot{y}, \dot{z})^T$$
$$\boldsymbol{\xi} = (\phi, \theta, \psi)^T \qquad \dot{\boldsymbol{\xi}} = (\dot{\phi}, \dot{\theta}, \dot{\psi})^T$$

Let $\boldsymbol{\omega}$ be the angular velocities in the body frame. We have:

$$\boldsymbol{\omega} = \begin{bmatrix} 1 & 0 & -S_\theta \\ 0 & C_\phi & C_\theta S_\phi \\ 0 & -S_\phi & C_\theta C_\phi \end{bmatrix} \dot{\boldsymbol{\xi}} = \boldsymbol{W}\dot{\boldsymbol{\xi}}$$

Moreover, the contribution of the motors, in terms of both force and torque, are vital to the flight dynamics of any multirotor. Assuming negligible motor housing inertia and negligible rotational acceleration, the force and torque generated by a brushless motor i can be treated by a simplified model as:

$$T_i \approx k\Omega_i^2 \qquad \tau_i \approx b\Omega_i^2$$

where Ω is the RPM of the motor while k and b are constants. Thus, given distance L between CG and motor axis, the forces and torques generated by all the motors on the copter, in body frame, can be formulated as

$$\boldsymbol{T_B} = \begin{bmatrix} 0 \\ 0 \\ k\Sigma(\Omega_i^2) \end{bmatrix} \qquad \boldsymbol{\tau_B} = \begin{bmatrix} Lk(-\Omega_2^2 + \Omega_4^2) \\ Lk(-\Omega_1^2 + \Omega_3^2) \\ b(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \end{bmatrix}$$

Moreover, assuming the copter is symmetrical about all axes, the moment of inertia can be formulated as:

$$\boldsymbol{I} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$$

Assuming that the aerodynamic drag is proportional to the velocity in the respective direction, it can be formulated as:

$$\boldsymbol{D} = \begin{bmatrix} C_{D_x} & 0 & 0 \\ 0 & C_{D_y} & 0 \\ 0 & 0 & C_{D_z} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} C_{D_x} & 0 & 0 \\ 0 & C_{D_y} & 0 \\ 0 & 0 & C_{D_z} \end{bmatrix} \dot{\boldsymbol{p}}$$

Using Newton's second law, the linear dynamics of the copter can be formulated as

$$m\ddot{\boldsymbol{p}} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + \boldsymbol{R}\boldsymbol{T}_{\boldsymbol{B}} + \boldsymbol{D}$$

Therefore, linear acceleration of the quadcopter in the inertial frame can be formulated as

$$\ddot{\boldsymbol{p}} = g \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \frac{k\Sigma(\Omega_i^2)}{m} \begin{bmatrix} C_\psi S_\theta C_\phi + S_\psi S_\theta \\ S_\psi S_\theta C_\phi - C_\psi S_\theta \\ C_\theta C_\phi \end{bmatrix} + \begin{bmatrix} C_{D_x} & 0 & 0 \\ 0 & C_{D_y} & 0 \\ 0 & 0 & C_{D_z} \end{bmatrix} \dot{\boldsymbol{p}} \tag{13}$$

The rotational dynamics has to be analyzed from the body frame due to centripetal forces and gyroscopic force Γ:

$$\boldsymbol{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times (\boldsymbol{I}\boldsymbol{\omega}) + \boldsymbol{\Gamma} = \boldsymbol{\tau}_{\boldsymbol{B}}$$

Therefore, the angular acceleration in body frame is

$$\dot{\boldsymbol{\omega}} = \boldsymbol{I}^{-1}(-\boldsymbol{\omega} \times (\boldsymbol{I}\boldsymbol{\omega}) - \boldsymbol{\Gamma} + \boldsymbol{\tau}_{\boldsymbol{B}})$$

Hence, the angular acceleration in inertial frame is

$$\begin{aligned} \ddot{\boldsymbol{\xi}} &= \frac{d}{dt}\dot{\boldsymbol{\xi}} = \frac{d}{dt}(\boldsymbol{W}^{-1}\boldsymbol{\omega}) \\ &= \frac{d}{dt}(\boldsymbol{W}^{-1})\boldsymbol{\omega} + \boldsymbol{W}^{-1}\dot{\boldsymbol{\omega}} \end{aligned} \tag{14}$$

With (13) and (14), it is possible to model the dynamics of the quadcopter. The model can then be used for quadcopter control simulation [14]. In order to control the attitude of the quadcopter, PID controller can be used because of its robustness in performance and simplicity in implementation [14]. The PID controller output $u(t)$ is based on the error $e(t)$ between actual state $\boldsymbol{\xi}_d(t)$ and desired state $\boldsymbol{\xi}(t)$.

$$\begin{aligned} \boldsymbol{e}(t) &= \boldsymbol{\xi}_d - \boldsymbol{\xi} \\ \boldsymbol{u}(t) &= k_P \boldsymbol{e}(t) + k_I \int \boldsymbol{e}(t)dt + k_D \frac{d}{dt}\boldsymbol{e}(t) \end{aligned}$$

The actual attitude is obtained by an on-board estimator, with input raw data from gyroscopes and accelerometers. The control gains can also be fine tuned from trial-and-error methods [14].

2.6 Omni-directional unmanned ground vehicle (UGV)

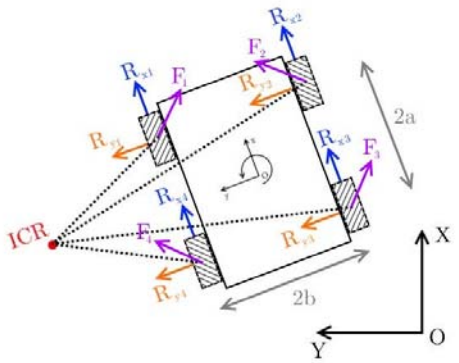


Figure 4: UGV Dynamics model

Firstly, the motion of a ground vehicle is the combined result from motor power, translated into traction force $\boldsymbol{F}$ and the wheel friction $\boldsymbol{R}$. Frictional force could be perpendicular to the wheel axis, which translates into rotation motion, or co-linear with wheel axis, which creates side slip. Even though conventional wheels are designed to avoid side slip, special ones, such as mecanum wheels, take advantage of this characteristics to increase the degree of freedom. In Figure 4, friction is denotes as R_x and R_y.
Secondly, at any instance, it is possible to determine an Instantaneous Center of Rotation (ISR), around which point, the motion of the vehicle is purely rotational.
Let the pose of the UGV in the inertial frame be defined as $\boldsymbol{p} = (X, Y, \theta)^T$. With heading angle defined, the rotation matrix ${}_B\boldsymbol{R}^E$ from body frame to earth inertial frame is defined as

$$ {}_B\boldsymbol{R}^E = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} $$

Let the torque from each motor be τ_i, with each wheel having diameter d_w, the circumferential force F_w, and the effective force F on the ground due to mecanum wheel effect on wheel i are

$$ F_{w_i} = \frac{2\tau_i}{d_w} $$

$$ F_i = F_{w_i}\cos(45) = \frac{\sqrt{2}\tau_i}{d_w} $$

Therefore, component forces in x and y directions of body frame are

$$ F_{x_i} = F_{y_i} = sign(i)F_i\cos(45) = sign(i)\frac{\tau_i}{d_w} $$

which combine into resultant forces and moments as

$$ F_x = \sum_{i=1}^{4} F_{x_i} = \frac{1}{d_w}(\tau_1 + \tau_2 + \tau_3 + \tau_4) \tag{15} $$

$$ F_y = \sum_{i=1}^{4} F_{y_i} = \frac{1}{d_w}(-\tau_1 + \tau_2 - \tau_3 + \tau_4) \tag{16} $$

$$ M_F = b(-F_{x_1} + F_{x_2} + F_{x_3} - F_{x_4}) + a(F_{y_1} - F_{y_2} - F_{y_3} + F_{y_4}) \tag{17} $$

$$ = \frac{a+b}{d_w}(-\tau_1 + \tau_2 + \tau_3 - \tau_4) \tag{18} $$

From Equation 15, it can be inferred that the UGV would move in longitudinal direction when wheels are rotated in the same direction. From Equation 16, UGV would move sideway if adjacent wheels rotate in the opposite direction. From Equation 18, UGV changes heading when wheels on the left and right are of inversed rotation. This interesting behavior is illustrated in Figure 5.
Assuming that the weight is evenly distributed on all 4 wheels and the friction is directly proportional to ground reaction force and wheel instantaneous contact velocity. Let f_x and f_y be the coefficients of rolling and slipping friction respectively, $\dot{x}_i$ be the longitudinal velocity and $\dot{y}_i$ lateral velocities in the body frame, the force $\boldsymbol{F}$ and moment $\boldsymbol{M}$ from friction acting on the vehicle body can be formulated as

$$ R_x = \sum_{i=1}^{4} R_{x_i} = \sum_{i=1}^{4} f_x \frac{mg}{4}\dot{x}_i $$

$$ R_y = \sum_{i=1}^{4} R_{y_i} = \sum_{i=1}^{4} f_y \frac{mg}{4}\dot{y}_i $$

$$ M_R = b(-R_{x_1} + R_{x_2} + R_{x_3} - R_{x_4}) + a(R_{y_1} - R_{y_2} - R_{y_3} + R_{y_4}) $$

$$ = bf_x\frac{mg}{4}(-\dot{x}_1 + \dot{x}_2 + \dot{x}_3 - \dot{x}_4) + af_y\frac{mg}{4}(\dot{y}_1 - \dot{y}_2 - \dot{y}_3 + \dot{y}_4) $$

Let $\dot{x}, \dot{y}$ be the vehicle longitudinal and lateral velocities in the body frame. Due to the geometric constraints between the wheels and vehicles, we have

$$\dot{x}_1 = \dot{x}_4 = \dot{x} - b\dot{\theta}$$
$$\dot{x}_2 = \dot{x}_3 = \dot{x} + b\dot{\theta}$$
$$\dot{y}_1 = \dot{y}_2 = \dot{y} + a\dot{\theta}$$
$$\dot{y}_3 = \dot{y}_4 = \dot{y} - a\dot{\theta}$$

Applying Newton's Second Law, we could arrive at

$$\begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} \ddot{X} \\ \ddot{Y} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} F_x + R_x \\ F_y + R_y \\ M_F + M_R \end{bmatrix} \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where m is the mass and I is the moment of inertia of heading rotation.

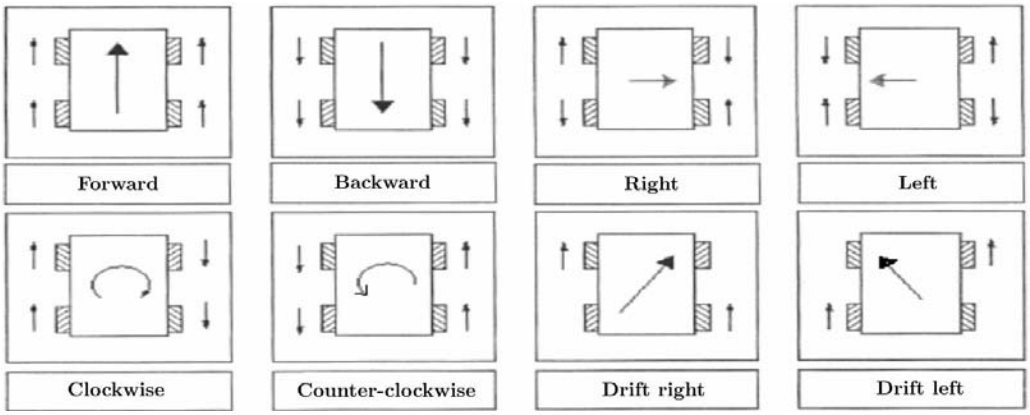


Figure 5: Omnidirectional drive mechanism with mecanum wheel [17]

3 Experimental setup

Each experiment comprises of three main parts: ground truth setup, vehicle setup and onboard sensor setup. For ground truth, tn this series of experiment, VICON system are deployed to obtain high quality ground truth data. Vehicle setup comprises of UAV setup and UGV setup. Onboard sensor setup depends of the planned mission and intended capability of the vehicle.

3.1 VICON system

VICON is a motion capturing system that makes use of video to determine motions of object in real-time. In this project, the VICON system consists of twelve individual IR cameras. Target objects, balls in this case, are coated with special material to reflect the IR rays emitted from the cameras. The IR footage of the balls, captured from multiple cameras could then be used to determine the position of the ball. Furthermore, by placing reflective balls on an object in a non symmetrical configuration, the pose of the object can be determined.
In this project, experiments are done with two vehicles, one UAV and one UGV, each of which is equipped with four reflective balls placed at proper positions to ensure non-symmetry configuration, clear camera exposure and convenient operation. One example setup in shown in Figure 9, with the corresponding VICON processed pose shown in Figure 8. With millimeter grade precision and upto 50Hz data rate, the VICON serves as a good ground truth for checking quality of vehicle onboard localization, detection and control. In this project, experiments involve one Unmanned Ground Vehicle (UGV) and one Unmanned Aerial Vehicle (UAV). These vehicle are setup so that they have the capabilities of self-navigation and inter-communication, while carrying mission-specific equipments and payloads.

Figure 6: A typical reflective ball

Figure 7: VICON Bonita camera

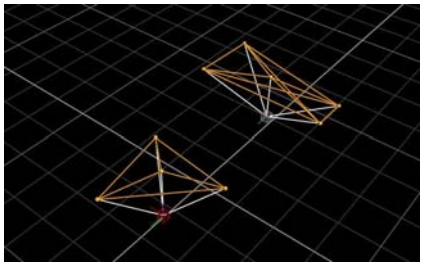


Figure 8: Vehicle pose from VICON

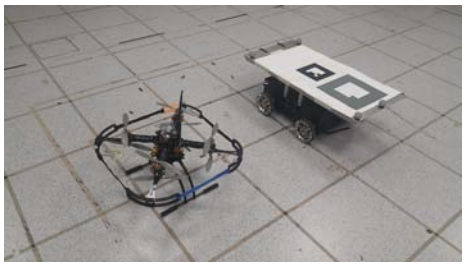


Figure 9: Actual real vehicle pose

3.2 Vehicle setup

3.2.1 UGV setup

UGV, as the name implied, is a vehicle that capable of autonomous navigation on the ground. For this series of experiment, to give the UGV greater degree of freedom, we equipped the UGV with four Swedish omni-directional wheel (omni-wheel). Driven by four independently controlled motors, such configuration allow the UGV to travel in three direction: longitudinal,

lateral and rotational. The UGV operates with one 24V Lithium polymer battery. Typical operation time ranges from 2 to 3 hours.
For some mission, the UGV serves as a static or dynamic target for the UAV to approach and land on. In order to aid the detection of UGV, the top surface of the UGV is also marked with two distinct patterns. From Figure 10b, it can also be seen that one of the markers is bigger in size. The reason is that bigger marker would make it easier for the UAV camera to register from far distance, while smaller marker would ensure that the camera is able to detect the marker at very close proximity. The too markers are reproduced in Figure 12 for reader's reference.

Front view Top view Omni-wheel

Figure 11: UGV setup

Marker 1 Marker 2

Figure 12: Landing markers

3.2.2 UAV setup

For this series of experiment, due to the potential heavy weight of on-board processors and sensors, the UAV is equipped with eight rotors, arranged four contra-rotating pairs configuration. Such configuration is also named X8, since it has eight motors while the overall appearance resembles a cross. The UAV runs on one 12V Lithium Polymer battery. Typical operation time ranges from 5 to 10 minutes. If the mission required extended flight time, second battery might be deployed.

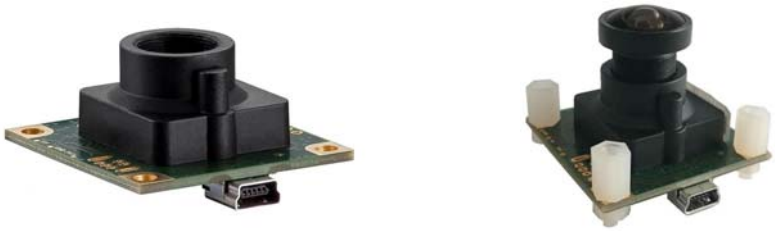
Front view Top view

Figure 14: UAV setup

3.3 Onboard sensor setup

3.3.1 RGB camera

One of the experiment in this project was to test the ability of the UAV to find a predefined marker which indicated the position of the UAV and subsequently attempted to land on the marker. In order to identify the marker, a RGB camera is installed on the UAV, facing downward. The camera that we selected for this purpose is produced by IDS, model UI-1221LE for its high frame rate and wide angle of view, hence allowing faster detection during operation. Some of the important parameters of the camera are extracted in Table 1 for reader's interests. Figure 15b shows a typical camera setup that is installed downward-facing on the UAV.



bare camera with lense and standoff

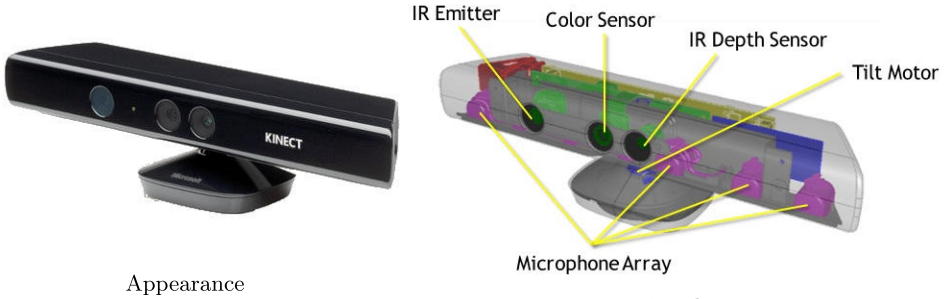
Figure 16: Camera IDS UI-1221LE setup

Specifications	**Data**
Sensor type	CMOS
Sensor size	1/3"
Resolution	752×480
Data rate	87.2 fps
Pixel size	6μm
Weight	16g

Table 1: Key parameters of camera IDS UI-1221LE

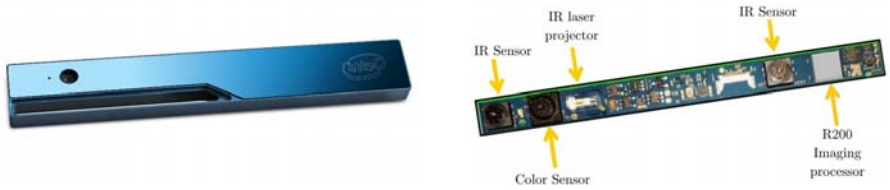
3.3.2 RGB-D camera

In order to allow the UGV to self-localize, vSLAM algorithms, as explained in the previous parts, could be deployed. The sensor for this purpose is selected to be Kinect 1, manufactured by Microsoft. The special feature of the Kinnect camera, as shown in Figure 17b, is that it has an IR emitter and an IR receiver, through which a depth point cloud can be obtained of the environment in front of the camera can be obtained. As discussed in the previous sections about vSLAM, the point cloud could aid in features extraction and features matching, thus could improve the vSLAM performance. Some key parameters of Kinect 1 are reproduced in Table 2 for readers' interests.
Even though Kinect is a good sensor for visual navigation, its relatively heavy weight limits its usefulness for aerial vehicles. Due to constrained capability of UAVs, both in terms of available lift force and computational resources, RealSense camera is selected to test vSLAM algorithms that are optimized for UAV on-board operation. From Figure 18b, it is noticeable that RealSense, made by Intel, has a similar operating mechanism compared to Kinnect 1, with one normal RGB sensor for image collection and one IR transmitter-receiver pair for depth measurement. From Table 3, Intel RealSense is much lighter, with only 8 grams, compared to Kinnect 1 with 430 grams. However, the disadvantage of RealSense is that its IR point cloud is more susceptible to noise compared to that of Kinect 1.



Appearance Inside

Figure 18: Microsoft Kinect 1



Appearance Inside

Figure 19: Intel RealSense

Specifications	Data
RGB resolution	640×480
Horizontal FOV	57 degrees
Vertical FOV	43 degrees
Depth resolution	320×240
Min depth distance	0.4m
Max depth distance	4.5m
Data rate	30 fps
Weight	430g

Table 2: Microsoft Kinect 1

Specifications	Data
RGB resolution	1920×1080
Horizontal FOV	70 degrees
Vertical FOV	43 degrees
Depth resolution	640×480
Depth vertical FOV	46 degrees
Depth horizontal FOV	59 degrees
Data rate	30-60 fps
Weight	8g

Table 3: Intel RealSense

3.3.3 2D laser scanner

Another method for the UGV to perform SLAM is point cloud matching. In this project the point cloud is obtained by 2D laser scanner, model UTM30LX produced by Hokuyo. The laser scanner has a straightforward principle of operation, with one stationary laser emitter and a rotating inclined mirror producing high frequency laser pulses that will be received at laser receiver and used to determine the range measurement by time-of-flight principle. This laser scanner has the advantage of high data rate, low noise and relatively high accuracy. The detailed specification of this sensor is reproduced in Figure 20 and Table 4 for reader's interest.

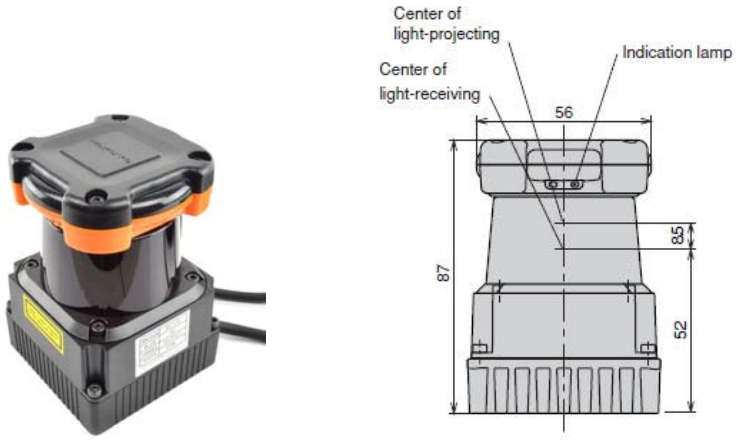


Figure 20: Hokuyo UTM-30LX Laser Range finder

Specifications	**Data**
Detection range	0.1m to 30m
Close range accuracy	30mm (range: 0.1 to 10m)
Long range accuracy	50mm (range: 10 to 30m)
Angular resolution	0.25 degree
Angular range	270 degrees
Data rate	25ms/scan
Weight	370g

Table 4: Key parameters of Hokuyo UTM-30LX

3.3.4 Inertial Measurement Unit (IMU)

In this project, due to limited computational resource, traditional methods of vSLAM, which requires iterative features extraction and matching, are not a practical approach. Therefore, Inertial Measurement Unit (IMU) is deployed in tandem with RGB-D camera. Such practice has proven to improve the overall performance of the vSLAM [25]. For this project, the IMU is fixed next to the camera by a customized 3D printed casing. The customized case ensure that the motions of the camera and the IMU are the same and the IMU is positioned and oriented at the most advantageous location. Furthermore, the whole sensor package is calibrated before use. Examples setup for Microsoft Kinect 1 and Intel RealSense is shown in Figure 23. Figure 22a shows the vSLAM module for UGV and Figure 22b for UAV.

Figure 21: WithRobot myAHRS+

myAHRS+ with Kinnect

myAHRS+ with RealSense

Figure 23: myAHRS+ setups with RGB-D cameras

Specifications	**Data**
Gyroscope	3-axis, ±2000dps
Accelerometer	3-axis, ±16g
Magnetometer	3-axis, ±1200μT
Data rate	100Hz (max)
Algorithm	EKF
Weight	3g

Table 5: Key parameters of WithRobot myAHRS+

4 Discussions, experiments and results

4.1 Ground vehicle (UGV) control

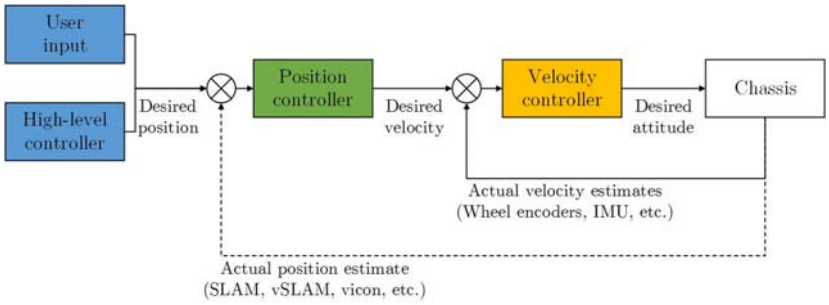


Figure 24: Cascaded control loops implemented on UGV

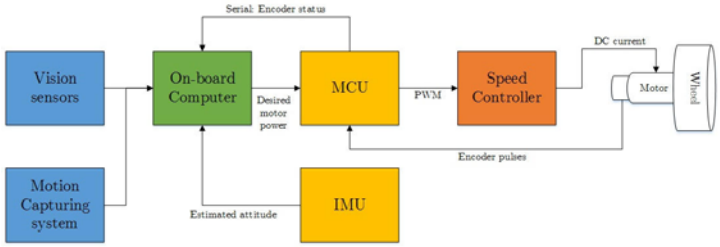


Figure 25: Components wiring on UGV

At the lowest level, the UGV is controlled by a PID velocity controller, as shown in Figure 24. The Velocity PID controller uses the difference between desired and estimated velocity to calculate outputs to four omnidirectional wheels. Due to high slip nature of omni-directional wheels, an EKF is deployed to fuse encoder data together with IMU data to produce better quality velocity estimates. In the case where encoder and IMU are not available, the controller will automatically change to blind-driving mode, where the output power to motor is directly proportional to input desired velocity.

On top of the velocity controller is the position controller, which is also a PID type control loop. In this project, the position controller obtains position estimates from either vicon motion capturing system or from on-board SLAM computation. Due to noisy nature of velocity estimates which impedes the usefulness of the velocity controller, the position PID controller holds an important role in the performance of the UGV.

For this project, with the available computational resources, the onboard computer is in-charged of most of the demanding computation, including EKF velocity estimation, PID velocity control and PID position control. As shown in Figure 25, the computer receives IMU and encoder readings to estimate velocity and output necessary power to the Microcontroller Unit (MCU) to actuate the motors. Moreover, the computer, with data from on-board cameras, laser scanners or external vicon motion capturing system, could also independently perform SLAM and vSLAM, and navigate autonomously.

4.1.1 Experiment: UGV position control

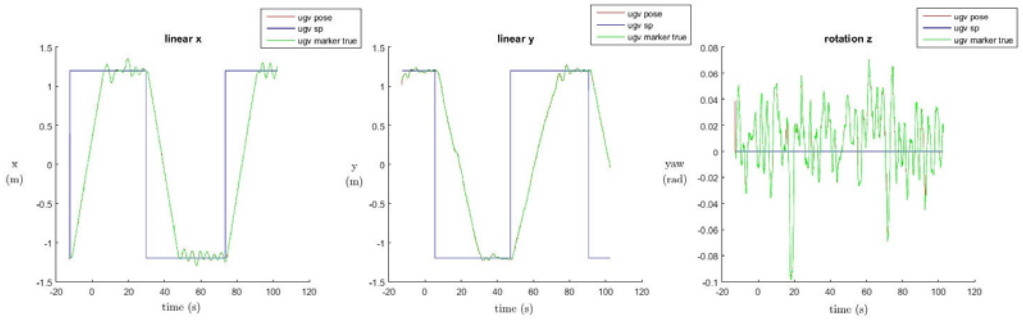


Figure 26: UGV 2DOF waypoint test

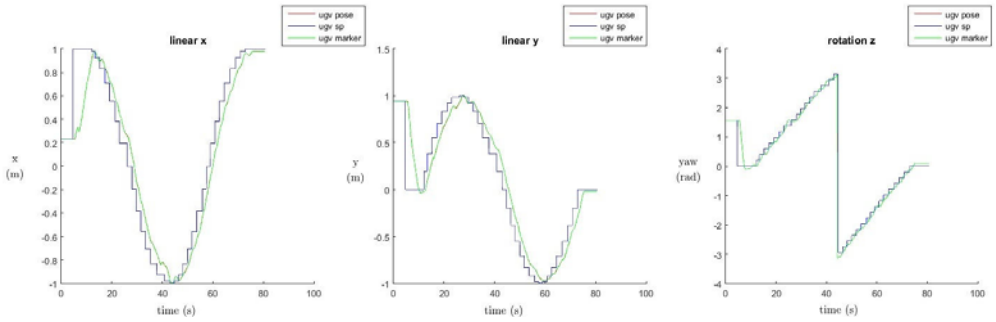


Figure 27: UGV 3DOF waypoint test

In order to make sure that the UGV is ready for advanced mission, the basic position control tests are conducted. In test 1, as shown in Figure 26, the UGV is tasked to follow a square trajectory with side 1(m). In this test, the position controller will output desired velocity, capped at 0.15(m/s). From Figure 26, it can be seen that the actual velocity, measured by vicon, is constant about 0.147(m/s), indicating that the velocity error is kept well below 5%. However, as the actual position approaches the desired position, the output velocities seems to be too small, which results in overshooting of upto 0.1(m). Firstly, one of the possible reason is due to practical limit of the motor, which makes the response curve non-linear toward the zero input power. Another likely reason could be attributed to the high slip nature of the omni-directional wheel that makes fine control hard to achieve. Therefore, it is observed that below 0.1m precision, the UGV tends to oscillate at about 0.5(Hz). Fortunately, such low frequency oscillation would not significantly affect SLAM and vSLAM performance.
The second test to verify the all 3 DOF controllability is a based on a circle trajectory. In this test, 35 waypoints are set uniformly around a circle of radius 1(m). Moreover, the UGV is also required to adjust orientation while changing position. According to the test results shown in Figure 27, the UGV is able to follow the trajectory with position error of less than 0.1(m) and heading error less than 0.1(radian). The linear velocity is kept well below 0.15(m/s) and yaw rate stable at 0.1 (rad/s). Such steady rate control would make good condition for SLAM and vSLAM experiments, which would be addressed in section 4.3.

4.2 Aerial vehicle (UAV) control

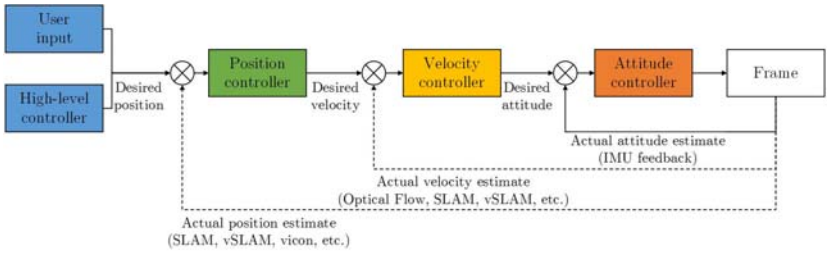


Figure 28: Cascaded control loops implemented on UAV

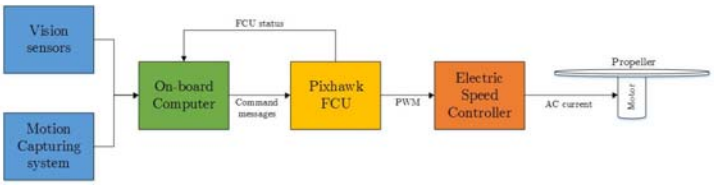


Figure 29: Components wiring on UAV

The control of the UAV requires multiple nested feedback control loops, as shown in Figure 28. Firstly, since the quadrotor UAV is inherently an unstable system, the most important loop is the Attitude controller. The attitude loop run at high frequency to allow responsive control of the UAV. An on-board Extended Kalman Filter fuses data from gyroscope, accelerometer and magnetometer to produce high frequency reliable attitude estimation. These attitude estimations, together with input desired attitudes, are then fed into a PID controller to produce appropriate motor outputs that would stabilize the UAV. Secondly, the velocity of the UAV is controlled by another PID controller that feed desired attitude to the attitude controller. For this project, in order to avoid reinventing the wheel, the PX4 firmware, with proven 32-state EKF and reliable PID implementation, is used for attitude, velocity and position control. Before flying mission, various PID parameters are tuned to ensure the response is gentle yet sufficiently precise. The PID tuning is performed with ground truth measurement from vicon system. The process is essentially trial-and-error, using Ziegler's method [26]. This method would ensure the optimal disturbance rejection behavior [26], which is desirable in this project. One draw-back of the PX4 PID controller is that it does not support I and D gains for position controller. One speculative reason is that such omit would practically prevent fast velocity output, which would protect novice users from aggressive UAVs. However, this would lead to steady-state error in position control. Therefore, another extra position controller is implemented in the on-board computer to practically add a ID controller for position control. This problem would be revisited in subsequent discussion with some actual flight data.

4.2.1 Experiment: UAV position control

In order to prepare the UAV for actual missions, the copter itself must be stable and controllable. Therefore, two types of test are conducted to observe the UAV behaviors: position hold test and way-point flight test.

Firstly, the position hold test is used to verify the precision of position control. Since the lab is an enclosed $10\times10\times3(m^3)$ room, most of the disturbances comes from the propeller's downwash and ground effect, which is random in both magnitude and direction. The copter is set to hover at $(x, y, z) = (0, 0, 1)(m)$, and the results captured from vicon is shown in Figure 30. As shown in Figure 30, during takeoff, the position error grows very quickly. This behaviour is due to the accumulation of I response in both position controller and velocity controller. However, after the transient takeoff behavior, the UAV could return and oscillate around its target position. The maximum error margin is found to be about 0.08(m) in x and y direction and upto 0.2(m)

in z direction. In terms of heading, the UAV is found to oscillate with amplitude within 0.06 (rad). The linear velocity is below 0.05(m/s). This tuned response is deemed satisfactory and the oscillation amplitudes are used to set the threshold radius for subsequent waypoint flights. Secondly, waypoint test is used to verify the response and precision of the UAV flight upon step input. The trajectory for this test is a square with side 1(m) and the test results are shown in Figure 31. The results show that the linear velocity for waypoint flight is kept well below 1 (m/s) while the linear precision margin is 0.1(m) and heading precision is below 0.1(rad). This tuned response is deemed satisfactory for mission flight, especially with SLAM and vSLAM.

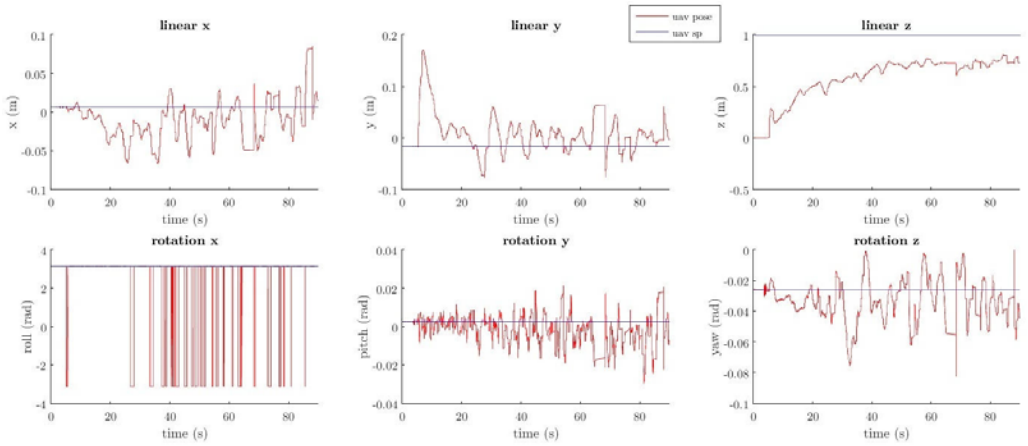


Figure 30: UAV position hold test

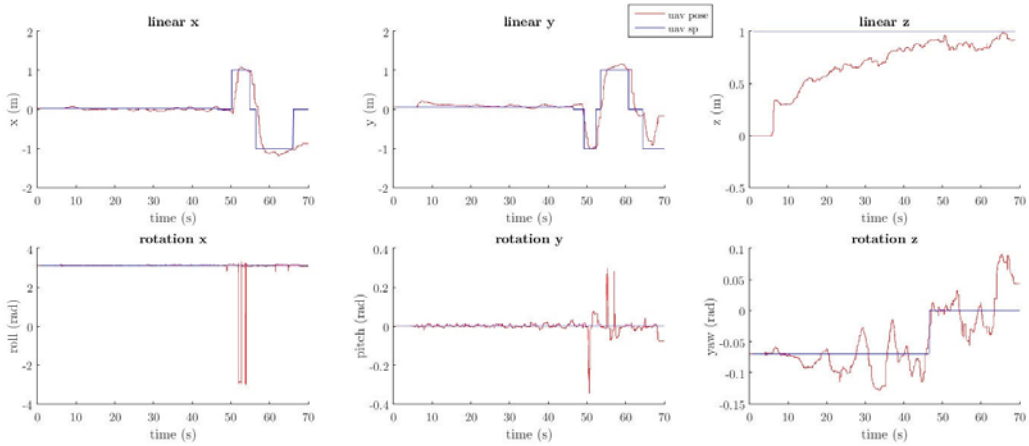


Figure 31: UAV square trajectory test

4.3 Simultaneous Localization and Mapping (SLAM)

4.3.1 Laser-scan-matching SLAM

In this project, due to the lack of accurate odometry feedback from both UAV and UGV, the method of laser SLAM must be implemented independently from odometry. Therefore, laser based SLAM has to rely almost entirely on Iterative Closest Point (ICP) algorithm. This requirement leads to the selection of Hector SLAM , an open-source ROS based package from Technical University of Munich (TUM). To test the reliability of Hector SLAM, a Hokuyo UTM-30LX, together with an Intel-i7 computer running Hector SLAM package, is deployed on the UGV. The UGV, monitored by the Vicon motion capture system, is set to travel with different trajectories to compare the accuracy of Hector SLAM against the ground truth from Vicon.

In the first test, the UGV traveled along a semi-circle with various changing heading. The results, as shown in Figure 32, indicated that errors could grow over time in both x and y directions. Moreover, at steady state, the error can persist and can be as much as 0.25(m).

In the second test, the UGV is set on a circular trajectory with various changing headings. The results, as shown in Figure 33 and Figure 34, indicated a consistent behavior with the first test with errors growing upto 0.25(m) at the furthest waypoint. However, the error is bounded and when the UGV returns to its original position, the accuracy is about 0.01(m). It is interesting to observe the same pattern in the second cycle. The error grows upto a maxima at the point of opposite heading and converges back when the UGV returns to original position. This behavior is confirmed with the second attempt at the same test, with slightly faster speed. This cyclic behavior could be due to the fact that Hector SLAM did not perform loop closure. Therefore, it could not correct the wrong map and reproduce the error when the cycle is repeated. This behaviors imply that errors could grow quickly and becomes impossible to recover in a large map which might limit the application of Hector SLAM to small environment.

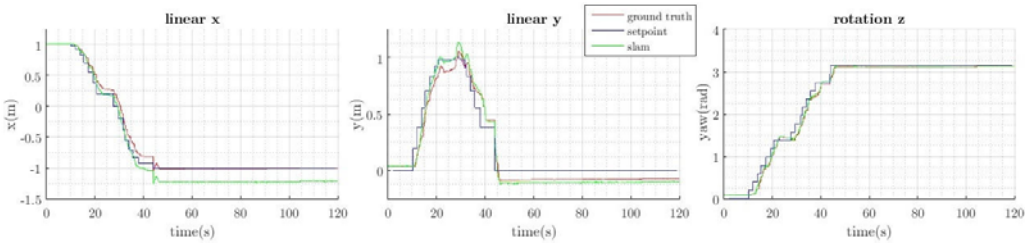


Figure 32: Half Cycle Test

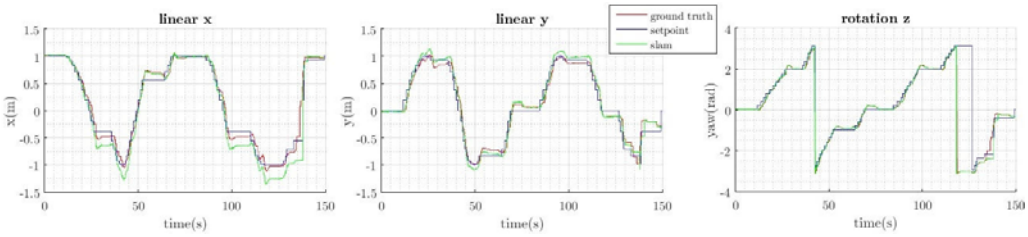


Figure 33: Full Cycle Test 1

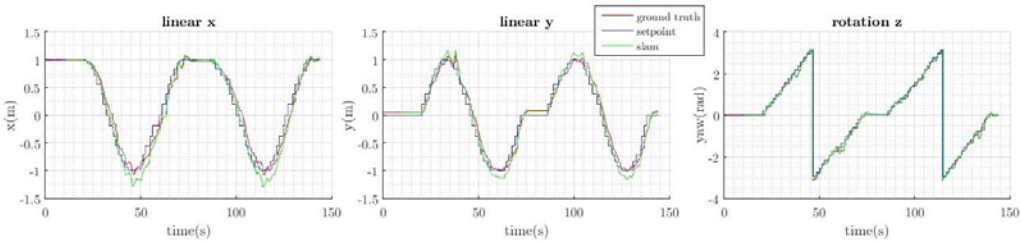


Figure 34: Full Cycle Test 2

4.3.2 Visual SLAM

Even though 2D Laser-based SLAM demonstrates good performance as shown in the previous section, one of its disadvantages is its assumption that the surrounding is mainly structured environment. This assumption could be tolerated when the vehicle moves in 2D plane like the case of UGV. However, it limits many applications in 3D operations, especially for UAV. Therefore, another SLAM method, based on correlation between RGB-D image features, is also tested in this project. The method is called General Kernelized Correlation Filter (GKCF).
For this experiment, the UGV, carrying a Kinect camera an AHRS IMU, is set to travel in a square trajectory with fixed heading. In the mean time, the on-board computer would receive RGB-D images together with IMU reading to compute the vSLAM algorithm. The vSLAM results, compared against ground truth, is shown in Figure 36 and Table 6. Moreover, the point dense cloud map of the environment is also computed and reproduced in Figure 35.
As shown in Table 6, the performance is comparable with Laser-based SLAM with maximum error at 0.3(m) and mean error about 0.11(m). With this level of precision and data rate of 30Hz, this vSLAM method is suitable for UAV flight missions.

Error type	Data
RMS	0.123045(m)
Mean	0.110183(m)
Median	0.096577(m)
STD	0.054770(m)
Min	0.046948(m)
Max	0.300279(m)

Table 6: Absolute Translational Errors

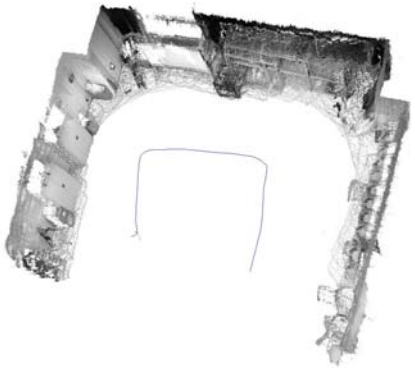


Figure 35: Dense vSLAM point cloud

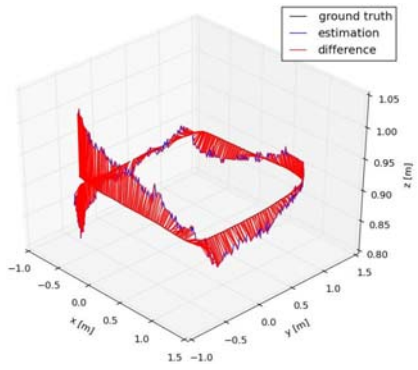


Figure 36: Trajectory

4.4 Multi-agent System (MAS)

4.4.1 Agent role classification

In order for multiple agents to collaborate toward a common goal, the group must have the capability to issue commands to individuals, store the collective information and perform certain actions or physical interactions with the environments. From these requirements , the agents should be classified into three types: Command (**type C**), Data (**type D**) and Action (**type A**).
Firstly, agent **type A** is in charged of performing actions based on a predefined goal. For example, it can collect data about surrounding environment or perform physical manipulations if the goal is defined so. The abstractness of the goal, in the context of operation, would imply the level of autonomy the robot operates. However, there must be a defined goal for agent A, for an agent without goal is an idle one. Therefore, there must be another agent capable of giving out commands or goals, which is agent type C.
Agent **type C** is in charged of distribution of goals. Since a robot, at this level could not define its own goal even for agent C, the singular ultimate goal for the whole system must be issued from a human operator. In this case, if the goal from human is sent directly to all agent, agent C will serve as an interface. However, agent C could use the ultimate goal as guidelines, especially when the ultimate goal is too abstract and unbounded, to come up with other more specific goals for other agents. In this case, agent C acts as a Commander and Monitor.
Many a times, situation required the team to combine all the data collected by individual agents. Examples of such situations include open-area-mapping or search-and-rescue or collaborative SLAM. Therefore, it is necessary to have agents specializing in storing and keeping track of all the data. This agent is called agent **type D**. It should be noted here that the data handled by agent D should focus on serving the whole network, rather than the individuals. In fact, in order to autonomously perform its own role, agent A also has its capability to store some data. However, agent D could request of these some individual data and make public in order to maximize the performance of the whole network.
In general, in a system that has all three types of agents: type A, C and D, we should see a network where agents type D are middle men, relaying information, whereas agents type C are leaders, and agents type A are workers. If the system is to be compared to a functioning biological system, agent C is the brain, agents D are the nerve fibers and agents A are the limbs. Together, they forms a system that could intelligently perceive and physically interact with the environment. An illustration of communication between different types of agents is presented in Figure 37.

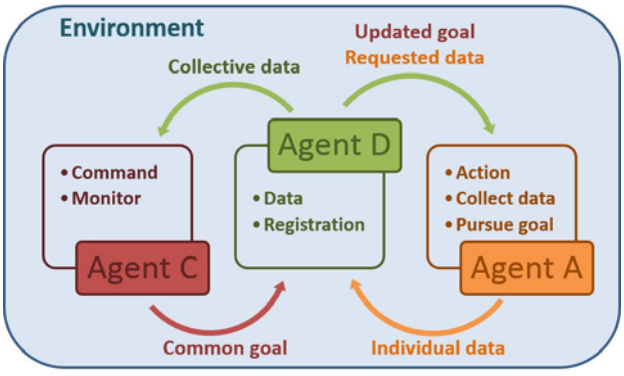


Figure 37: Inter-agents communication

4.4.2 Software implementation

As explained in Section 4.4.1, the concept of *agent* is classified into three types according to three different abstract roles: Command, Data and Action. In order to realize such concept, a software implementation is proposed as shown in Figure 38. The implementation is built on

Robot Operating System (ROS), with C++ language, to inherit the robust and tested node-based software concept of ROS. Moreover, this implementation is also made open-sourced on Github, at `http://github.com/hmchung/mas_ros`, in order to allow open scrutiny and public contribution.
Firstly, for agent C, its main role is to take command from human and distribute goals to the rest of the network. Therefore, since the mechanism of goal distribution could already inherit from existing ROS protocols, the software emphasis should be on the interface with human operator. Therefore, the major component in the software architecture of Agent C is its Graphical User Interface (GUI) that could allow good command distribution and visualization. The visualization includes environment perception, robot state and other data viewers. This part of the implementation is public on Github at mas_vis.
Secondly, for Agent D, its main task is to deal with data, including collection, distribution, storage, retrieval and pre-processing. Besides, agent D also registers the presence of all agents in the system, which would help agent C in the process of command distribution. In this project, due to the time limit and the constraint in project scale, software implementation of agent D was not possible. However, the author recognizes the importance of agent D in a larger scale MAS.

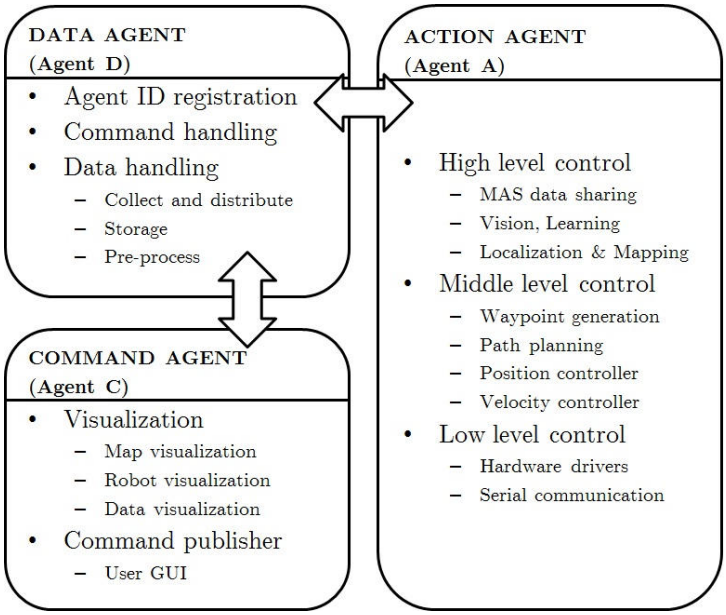


Figure 38: Agents's role software implementation

Last but not least, the interaction with environment, which is the main practical purpose of most robotic systems, would not be possible without agent A. Due to the complicated nature of its operation, the software architecture of agent A are divided into layers. From the bottom up, we have Low-level control, Middle-level control and High-level control. Low-level control is mainly concerned with hardware compatibility to ensure that various electro-mechanical devices on the robots properly functions upon commands from higher levels. On top of the Low-level is the Middle-level control, which is concerned with orchestrating various hardware components in a meaningful manner. For example, on a wheeled UGV platform, the Middle-level control would ensure that the wheels are rotated in the right direction so that the platform moves in the intended direction. On a multi-rotor UAV platform, the middle-level control would spin the motors at appropriate velocities to hover the platform at desired attitudes. Above the Middle-level is the High-level control, which is focusing on environment perception, localization, navigation and various complicated decision makings. The task of Middle-level could range from SLAM to Computer Vision to Machine Learning. The main distinction between Middle and

High layers is that the earlier control by rule-based logical response, whereas the later relies on more advanced algorithms. The software implementation of agent A can also be found on Github at mas_low_drivers, mas_mid_controller and mas_high_intelligence.

4.4.3 Hardware implementation

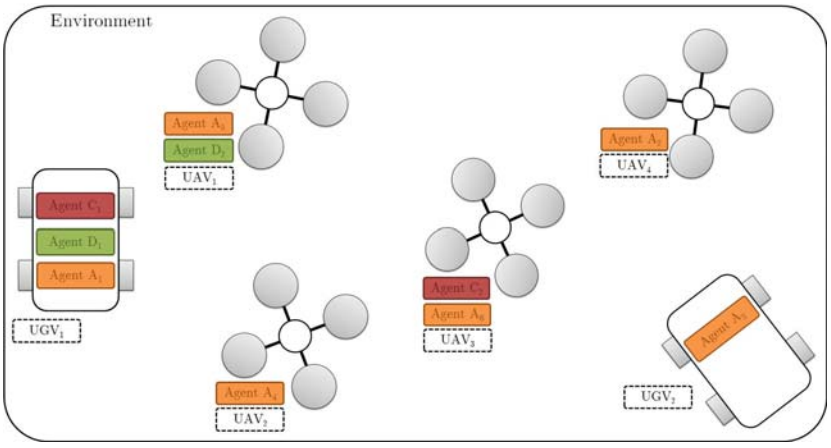


Figure 39: An example practical implementation of agents

Even though agents are conceptually classified into three types, such distinction should not be a limit for hardware implementation. In fact, the terms *agent* in this context refers to a software notion. In practice, each hardware platform, depending on its nature, could carry more than one type of agent. For example, one plat form could carry one agent A for platform control, one agent D for data storage and one agent C for distributing command to the whole network. A hardware platform that carries only agent C would likely be a ground control station. A platform that carries only agent D would be a data center. A platform with only agent A is a command-obey worker. An example of a swarm system is illustrated in Figure 39, where four quadrotor UAVs and two wheeled UGVs are deployed simultaneously. Firstly, it is noticeable that each robot has one agent type A, for platform control purpose. Secondly, agents could reside in various platforms, which demonstrates that the deployment of agents should be dependent only on the availability of computational resources, instead of platform types. In case of hardware failure, agents, especially C and D, could migrate to a different available platform to continue operation. Such a design would allows better hardwares integration and improve overall system robustness.

4.4.4 Experiments - An overview

In order to verify the feasibility of the proposed MAS implementation, some experiments are designed to observe the behaviors of individual robots when performing a common mission. In this project, the test experiment is UAV precision landing on UGV. Moreover, the UGV is set to patrol in an unknown pattern, therefore the UAV needs to actively search for the target before approaching and landing.
To formalize the experiment by the MAS concepts that are introduced in the previous discussion, the experiment involves four agents, including one agent A on the UAV platform named A_1, one agent A on the UGV platform named A_2, one agent C and one agent D at the operator workstation. Upon the start of each experiment, agent A_1 received the command to search for and land on the marker, agent A_2 to simply patrol around the environment while agent D collects the data published by A_1 and A_2. To materialize the abstract command, each agent has to use its own sensor data and software capabilities to properly control its attitude, velocities and positions. As already discussed in Section 4.2 and 4.1, the UAV is capable of autonomous flight including takeoff, waypoint navigation and landing, while the UGV is capable

of autonomous waypoint following.
In order for the UAV to successfully land on the UGV, especially when the UGV is moving, the UAV should be equipped with the hardware and software capabilities, not only to detect but also to estimate the marker future position. In this project, the UAV is equipped with a downward facing wide-view camera, as discussed in 3.3.1, which could register the marker image when the UAV hovers above the UGV. The camera would be calibrated and the digital image would then be rectified to amend the large degree of distortion due the wide view lens. To detect the marker from digital image, the ROS *ar_pose* package, which is based on *ARToolKit* library, is selected because of its tested high frequency and reliable robustness, testified by the open-source community.
In order to catchup with the moving UGV, it is necessary for the UAV to predict its near future position. To do so, the controller first estimates the velocity of the UGV based on the current and past position estimates. One of the simplest way is to take an average derivative of latests position estimates, which demonstrated to work at low speed moving UGV. With velocity estimates, it is possible to derive position estimates, which determine the x and y coordinates and heading for UAV setpoint. The z coordinate, or altitude, is calculated based on desired approach gradient and UAV to UGV distance. This is to ensure that the approach slope is sufficiently gentle and the landing marker is within view of camera. The whole process is iterated at every arrival of marker position estimates to send setpoint to UAV position controller at high frequency. An illustrated summary is demonstrated in Figure 40 for readers' ease of understanding.

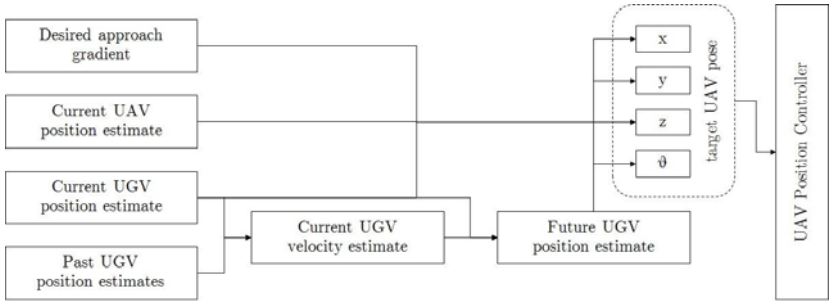
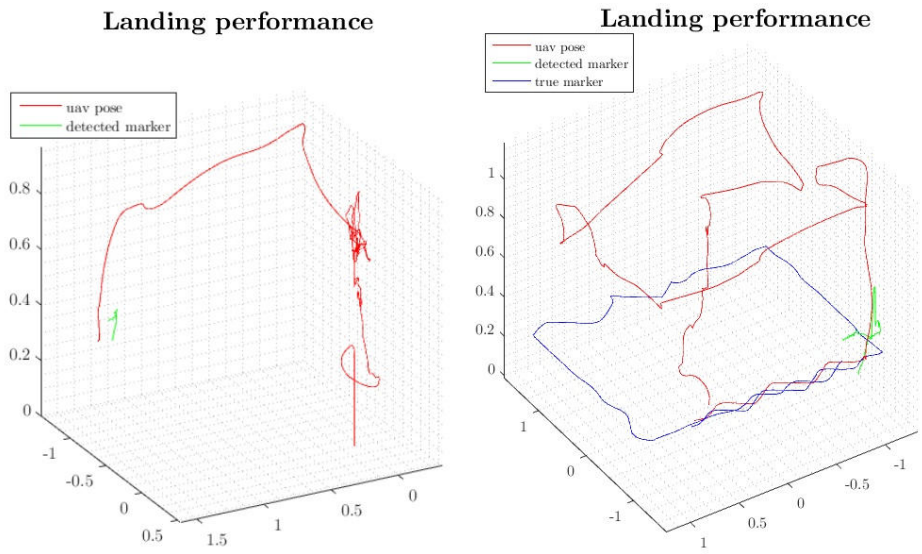


Figure 40: Landing position controller

4.4.5 Experiments: Precision landing on static and moving platform

With the marker position estimator and landing controller designed, as explained in the previous part, the first set of experiment is to verify its performance in the most ideal condition: static marker. In this experiment, the UAV is set to patrol in a square trajectory and the UGV is purposely positioned near the UAV path for ease of detection. The result of the test is shown in Figure 43. From Figure 43, it can be seen that, while hovering at 0.8m averaged altitude, the UAV starts to detect the marker from about 0.3m ground distance away. Within 4 seconds, the UAV quickly approach and successfully landed on the target. The precision in x and y direction is about 0.05(m), which makes the radial accuracy below 0.1(m). The trajectory of the whole test flight, from takeoff to landing, is shown in Figure 41a.



Static platform Moving platform

Figure 42: UAV precision landing on UAV

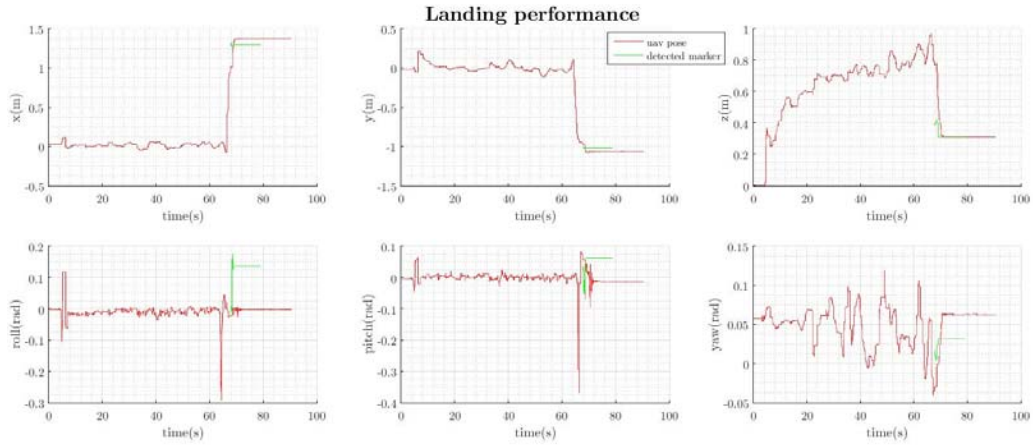


Figure 43: Static landing test results

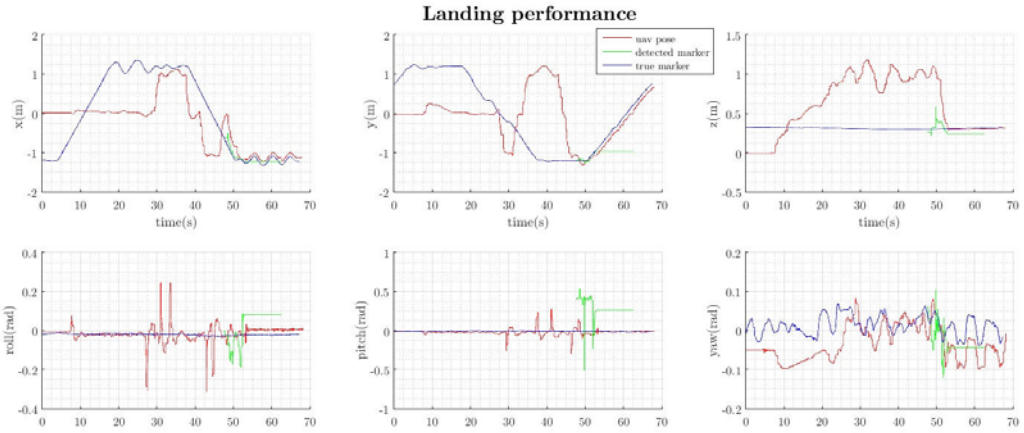


Figure 44: Dynamic landing test results, UGV 0.13(m/s)

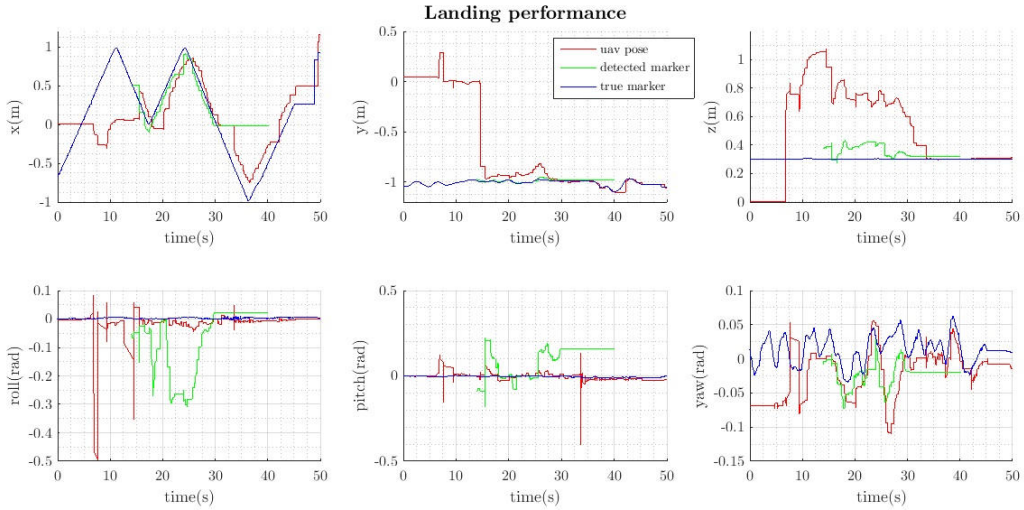


Figure 45: Dynamic landing test results, UGV 0.17(m/s)

Following the success of static landing, dynamic landing can now be tested. Since the main aim to the experiment is to verify if the UAV can follow, approach and land on a moving platform while not focusing on robustness of detection, the UGV should move near the ground trajectory of the UAV to increase likelihood of detection. Therefore, the UGV is set to move with a square trajectory in clockwise direction, while the UAV would takeoff from the center of the square and attempt to search for the marker by a counter-clockwise square trajectory. The trajectories of both vehicle are plot in Figure 41b and the experiment results are shown in Figure 44.

From Figure 44, the UGV is patrolling at about 0.13(m/s). The UAV took off at 8th second and detected the marker at about 47th second, from about 0.5m away from the UGV. On the other hand, it is noticeable that at 32nd second, the UAV did not detect the UGV when two vehices were only 0.2m away from each other. This miss could be explained by the fact that the camera view is rectangular, which is wide in lateral direction but narrow in longitudinal direction. It is also noticeable that the precision of detection is about 0.1-0.2(m), which would eventually affect the precision of landing.

Upon detection at 47th second, the copter was still on patrol mode, in forward direction at about 0.5m/s. Therefore, it took the UAV about 3 seconds to change its course to quickly

approach the UGV, descend from 50th second and eventually land at about 53rd second according to Figure 44. According to Figure 44, the final precision of landing is about 0.1(m) in both x and y direction. However, this level of precision is limited by the accuracy of marker detection. As can be seen from Figure 44, at the instant when the UAV land, its actual position is within 0.01(m) from the marker position estimates. Therefore, the landing controller and position controller has done its job, while the accuracy marker detection needs some further accuracy refinements.
Furthermore, more challenging tests are also conducted where the UGV moves slightly faster at about 0.17(m/s), compared to about 0.13(m/s) in the previous test. The test results in shown in Figure 45, indicating interesting behaviors. Firstly, the UAV takes significantly more time to approach the target. Eventhough landing marker is detected at 15th second, the UAV takes about 12 seconds, until 27th second, to start descending. Moreover, the decend process also takes 5 seconds, upto 37th second, for the UAV to be able to land. Compared with the previous test, the UAV is apparently more struggling to achieve its goal. The possible reason is that the faster the speed of the UGV, the harder for the UAV to catchup and as long as the UAV is out of the desired descending gradient, it will only attempt following in order to maintain safe visual contact with the marker.

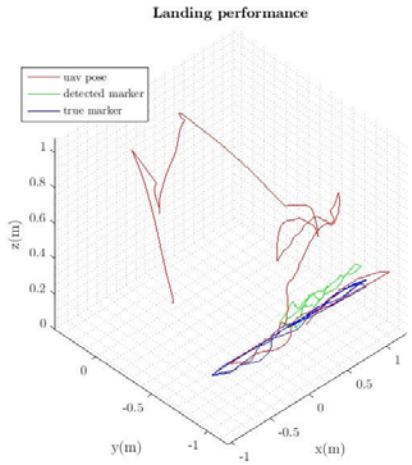


Figure 46: Dynamic landing test results, UGV 0.17(m/s)

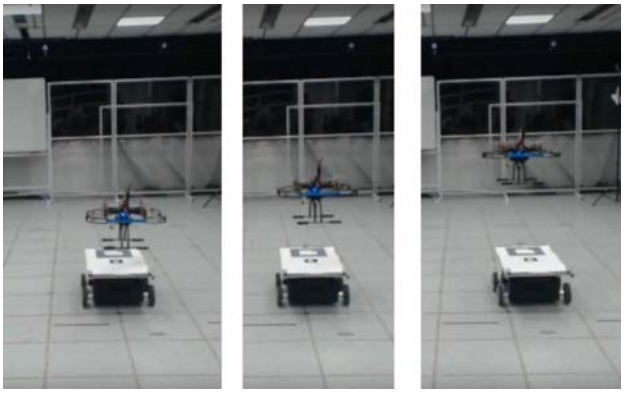


Figure 47: UAV approaching mobile UGV for landing

5 Conclusion

In conclusion, this projects is an attempt to implement Multi-agent System (MAS). By drawing a distinction between 'vehicle' and 'agent', such that 'vehicle' referring to hardware platform and 'agent' referring to a software entity, the author argues that such implementation would be more robust, flexible and scalable. The agents are classified into three types: Command agent, Data agent and Action agent. Command agent distributes goal to the whole system, Data agents collects and re-distribute information within the network, while Action agents interact with the physical environment through sensing, actuating or manipulating.
To ensure that individual robots has sufficient capabilities to perform meaningful missions, various experiments has been conducted. Velocity control and position control are tested on both UAV and UGV and the controllers are tuned to achieve precision upto 0.1(m) and 0.1(rad). To prepare for autonomous missions, two SLAM methods were also implemented and tested on the UGV. Firstly, Laser-based SLAM experiment, based on ROS hector_slam package, was conducted on the UGV and the errors of localization within indoor enclosed environment can be as much as 0.25(m). Secondly, visual SLAM experiments, with Kinect RGB-D camera and IMU, based on General Kernelized Correlation Filter (GKCF), is also tested and the accuracy is bounded at 0.3(m). Moreover, the GKCF vSLAM is more suitable for UAV mission because it overcomes the structured-environment assumption of 2D Laser-based SLAM.
Last but not least, new MAS software framework was implemented and experiments were conducted to verify robots's performance in joint mission. To test the MAS software implementation, two heterogeneous robots were deployed together. One of the major tests was precision landing, where UAV would attempt to search and land on the UGV. The results show that, when the UGV is static, the precision of the landing is 0.1(m) and depends on the precision of the accuracy of vision detection, rather than control. Moreover, when the UGV is moving, experiment results show that the UAV also can approach and land with 0.1(m) precision. However, the time required to follow, approach, descend and land on targets depends on the speed of the moving platform.

5.1 Future work

Due to time constraint and limited student experiences, this project still leaves a lot of room for future developments.
Firstly, even though experiments in this project involved multiple robots, they have yet to test collaborative behaviors, which is essential for swarm systems. Future experiments could further explore options such as collaborative SLAM, collaborative search or social foraging with homogeneous or heterogeneous robots. Secondly, the robots motion controllers and localization systems could further be tested to its limit. The SLAM system should be further tested in non-structured, sparse features environment and the controller could be tested when the feedback data is less accurate such as outdoor fields. Such experiments would further improve the robustness of the system against real-life application challenges. Last but not least, as the system scales up in number of individuals and the inter-robot interactions become complicated, it is important for the robot to have its own capability of learning rather than strictly following preprogrammed logics. The author hopes that these promising features could be studied and implemented in the future.

References

[1] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (slam): Part ii," *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006.

[2] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," *Computer vision–ECCV 2006*, pp. 404–417, 2006.

[3] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Robotics-DL tentative*. International Society for Optics and Photonics, 1992, pp. 586–606.

[4] R. Brooks, "A robust layered control system for a mobile robot," *IEEE journal on robotics and automation*, vol. 2, no. 1, pp. 14–23, 1986.

[5] Y. U. Cao, A. S. Fukunaga, and A. Kahng, "Cooperative mobile robotics: Antecedents and directions," *Autonomous robots*, vol. 4, no. 1, pp. 7–27, 1997.

[6] A. Diosi and L. Kleeman, "Laser scan matching in polar coordinates with application to slam," in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. IEEE, 2005, pp. 3317–3322.

[7] ——, "Fast laser scan matching using polar coordinates," *The International Journal of Robotics Research*, vol. 26, no. 10, pp. 1125–1153, 2007.

[8] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes, "A taxonomy for swarm robots," in *Intelligent Robots and Systems' 93, IROS'93. Proceedings of the 1993 IEEE/RSJ International Conference on*, vol. 1. IEEE, 1993, pp. 441–447.

[9] J. Fredslund and M. J. Mataric, "A general algorithm for robot formations using local sensing and minimal communication," *IEEE transactions on robotics and automation*, vol. 18, no. 5, pp. 837–846, 2002.

[10] C. Harris and M. Stephens, "A combined corner and edge detector." in *Alvey vision conference*, vol. 15, no. 50. Citeseer, 1988, pp. 10–5244.

[11] S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 155–160.

[12] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[13] F. Lu *et al.*, "Robot pose estimation in unknown environments by matching 2d range scans," in *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*. IEEE, 1994, pp. 935–938.

[14] T. Luukkonen, "Modelling and control of quadcopter," *Independent research project in applied mathematics, Espoo*, 2011.

[15] H. P. Moravec, "Rover visual obstacle avoidance." in *IJCAI*, 1981, pp. 785–790.

[16] J. Nieto, T. Bailey, and E. Nebot, "Recursive scan-matching slam," *Robotics and Autonomous systems*, vol. 55, no. 1, pp. 39–49, 2007.

[17] J. G. Phillips, "Mechatronic design and construction of an intelligent mobile robot for educational purposes," *Master of Technology Thesis, Massey University, Palmerston North, New Zealand*, vol. 150, 2000.

[18] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," *Computer vision–ECCV 2006*, pp. 430–443, 2006.

[19] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2564–2571.

[20] S. Se, D. Lowe, and J. Little, "Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks," *The international Journal of robotics Research*, vol. 21, no. 8, pp. 735–758, 2002.

[21] N. Snavely, S. M. Seitz, and R. Szeliski, "Skeletal graphs for efficient structure from motion." in *CVPR*, vol. 1, 2008, p. 2.

[22] G. A. Terejanu, "Extended kalman filter tutorial," *University at Buffalo*, 2008.

[23] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust monte carlo localization for mobile robots," *Artificial intelligence*, vol. 128, no. 1-2, pp. 99–141, 2001.

[24] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment—a modern synthesis," in *International workshop on vision algorithms*. Springer, 1999, pp. 298–372.

[25] C. Wang, J. Yuan, and L. Xie, "Non-iterative SLAM: A fast dense method for inertial-visual SLAM," *CoRR*, vol. abs/1701.05294, 2017. [Online]. Available: http://arxiv.org/abs/1701.05294

[26] J. G. Ziegler and N. B. Nichols, "Optimum settings for automatic controllers," *trans. ASME*, vol. 64, no. 11, 1942.