ADVANCED GUIDANCE AND NAVIGATION OF SMALL UAVS UNDER GPS-DENIED ENVIRONMENT WITH EXPERIMENTAL VALIDATIONS

By

LIYANG WANG

A dissertation submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Doctor of Philosophy

Graduate Program in Mechanical and Aerospace Engineering

Written under the direction of

Xiaoli Bai

And approved by

_____

_____

_____

_____

New Brunswick, New Jersey

October, 2019

ABSTRACT OF THE DISSERTATION

Advanced Guidance and Navigation of Small UAVs under GPS-denied Environment with Experimental Validations

by Liyang Wang

Dissertation Director: Xiaoli Bai

Small vertical take-off and landing (VTOL) Unmanned Aerial Vehicles (UAVs) are playing more and more important role in recent years for a wide range of applications including wireless communication, topographic mapping, disaster relief, and military action. The size of the UAVs becomes smaller to enable challenging applications such as indoor exploration. With the reduced size of the structure, battery, and on-board sensors, the payload is constrained. As a result, the quality of the on-board sensors and the on-board computation ability of small UAVs are limited. Under these conditions, to complete complicated tasks, research on small VTOL UAVs are required in many aspects. This thesis mainly focuses on four aspects: 1) Object recognition and tracking for vessel deck landings; 2) Simultaneous localization and mapping (SLAM); 3) Wind disturbance estimation; and 4) UAV platform design.

Autonomous landing of a quadrotor UAV on a vessel deck is challenging due to the special sea environment. In this thesis, an on-board monocular vision-based solution that provides a quadrotor with the capability to autonomously track and land on a vessel deck platform with simulated high sea state conditions is presented. The whole landing process includes two stages: approaching from a long range and landing after hovering above the landing platform. Only on-board sensors are used in both stages, without external information input. This thesis use Parrot AR.Drone as the experimental quadrotor platform, and a self-designed vessel deck

emulator is constructed to evaluate the effectiveness of the proposed vessel deck landing solution. Experimental results demonstrate the accuracy and robustness of the developed landing algorithms.

Second, this thesis presents a novel solution to on-board sensor only feature based monocular 3-D SLAM algorithm for Small UAVs. The proposed SLAM algorithm can navigate autonomously in previously unknown environments without external positioning aid by only using one camera and an IMU. Main contribution of this work is the proposed method can generate absolute scale map without map initialization and optimization methods. These advantages make the presented SLAM algorithm more applicable to use than other existing methods for small UAVs. Applying the proposed method to a low-cost quadrotor, experimental results demonstrate that our system can generate map correctly with absolute scale. At the same time, the position estimation accuracy of the camera is also improved compared to using inertial navigation only.

Wind speed estimation for VTOL UAVs is challenging due to the low accuracy of airspeed sensors, which can be severely affected by the rotor's down wash effect. Different from the traditional aerodynamic modeling solutions, in this thesis, a K Nearest Neighborhood learning based method which does not require the details of the aerodynamic information is presented. The proposed method includes two stages: an off-line training stage and an on-line wind estimation stage. Only flight data is used for the on-line estimation stage, without direct airspeed measurements. This thesis uses Parrot AR.Drone as the testing quadrotor, and a commercial fan is used to generate wind disturbance. Experimental results demonstrate the accuracy and robustness of the developed wind estimation algorithms under hovering conditions.

For VTOL UAVs hardware system design, this thesis provides a quadrotor solution which can support most of the research of small VTOL UAVs. The quadrotor's structure is made by carbon fiber, which provides large freedom of weight for on-board computers and sensors. For the on-board computer, a high-level processor and a low-level processor work together as a two-level processor system. The low-level processor is responsible for sensor data acquiring, synchronization, and motor speed controls. Furthermore, the low-level processor also monitors the manual control signal during the flight and can switch the control source to manual control under emergency, which makes the platform safe to operate. For the high-level processor,

an Ubuntu operation system is equipped. At the same time, Robot Operating System (ROS) is also embedded to access on-line available ROS packages easily. Researches can also make their own packages which contain their own algorithms. Another function for high-level processor is to communicate with users in real-time through wireless communication. The on-board sensor system includes: 1) a downward looking camera, 2) an Inertial Measurement Unit (IMU) which contains a 3-axis gyro and a 3-axis accelerometer, 3) a 3-axis compass, and 4) a downward looking ultrasonic sensor. These sensors provide rich information. Combined with the powerful two-level processor system, the whole system can help many important UAV research.

Acknowledgements

First and foremost, I would like to thank my advisor Prof. Xiaoli Bai for providing me the opportunity to pursue challenging and interesting problems through out my Ph.D. Thank you for your patience, trust, and guidance especially when I struggled with research. I am grateful for your help. The scientific and rigorous attitude you treat for research set a great example to me. I believe this invaluable spirit will benefit for my whole life. Thank you!

I would like to thank my committee members: Prof. Haim Baruh, Prof. Qingze Zou, and Prof. Yang Cheng for taking time to review my dissertation and providing invaluable insights and feedback which makes the work better. I also would like to thank all the professors who share knowledge to me from various discipline. You give me the ability to think, study, design, and create. Thank you!

Thanks to all the past and present members of my research group, Gaurav Misra, Hongwei Yang, Hao Peng, Tianyu Gao for providing a fun and friendly working environment. I enjoyed collaborating on research with you and hope for more such alliances in the future. Thank you!

My friends have been extremely helpful and supportive, Pengcheng Wang, Yongbin Gong, Han Du, Hanxiong Wang, Zheyuan Ji, Jiaming Li, Yikun Xian, Zhihan Fang, Xiaoyang Xie, Qiaoying Huang. I have enjoyed the colorful life with you guys. Thank you!

Lastly, all of this would absolutely not be possible without the unconditional love and support of my parents and my wife. Thank you!

Dedication

To my mother, Yuehui Wang, and my wife, Bihan Xu, for their unwavering support and love.

Table of Contents

Chapter 1

Introduction

1.1 Growing Applications of small VTOL UAVs

Small vertical take-off and landing (VTOL) Unmanned Aerial Vehicles (UAVs) are playing more and more important role. In the past few years, small VTOL UAVs have been used for applications such as wireless communication [1] [2], topographic mapping [3] [4], disaster relief [5] [6], and military action [7] [8].

In [1], the authors provide an overview of UAV-aided wireless communications system. They introduce the basic networking architecture and main channel characteristics, highlight the key design considerations, and the new opportunities are exploited. In [2], the authors studied a multi-UAV enabled wireless communication system. Multiple UAV-mounted aerial base stations are employed to serve a group of users on the ground. To achieve fair performance among users, the authors maximize the minimum throughput over all ground users in the down-link communication by optimizing the multi-user communication scheduling and association jointly with the UAV's trajectory and power control.

In [3], using simulations of multi-image networks with near parallel viewing directions, the authors show that enabling camera self calibration as part of the bundle adjustment process inherently leads to erroneous radial distortion estimates and associated error. They provide practical flight plan solutions for fixed wing or rotor based UAVs. They proves that in the absence of control points, the method can reduce error by up to two orders of magnitude. In [4], the authors provide some examples for the use of topographic mapping techniques in structural geology and paleoseismology. In their work, Photogrammetric models serve to act as an ideal electronic repository for critical outcrops and observations, similar to the electronic lab book approach employed in the bio-sciences.

In [5], the paper provides a review of recent utilization of UAVs for imagery collection for disaster monitoring and management. In [6], the authors point out that optimal design of UAV system can provide great potential for relief and emergency action. At the same time, a case of civil UAV system for disaster relief to save lives is studied.

In [7], the authors provide guidance of multiple UAVs as sensor platforms in military helicopter missions. Their work is addressed by the development and integration of the Artificial Cognitive Units (ACUs). The ACUs have been given knowledge of the Manned-Unmanned Teaming (MUM-T) domain, which allows the guidance of UAVs based upon certain high-level tasks. In [8], the authors states mainly 4 aspects for the effectiveness of a drone strategy can be subdivided into four separate claims. First, drones are effective at killing terrorists with minimal civilian casualties. Second, drones have been successful at killing so called 'high value targets' (HVTs). Third, using drones puts such pressure on terrorist organizations that it degrades their organizational capacity and ability to strike. Lastly, the cost-benefit analysis of their use relative to other options-such as the deployment of ground troops-provides a compelling argument in their favor.

1.2 Current Small UAV Configurations and Relevant Technologies

Recently, the size of UAVs have become smaller to enable some challenging applications such as some indoor applications. Many manufactures have released their quadrotor solutions for research. For example. in 2017, DJI released Matrice serial UAVs for research [9]. In [10], Vikram et al use the DJI Matrice 100 platform for agriculture monitoring and observation for yielding better crop quality and preventing fields from any sort of damage. Another market available research drone is provided by Parrot, called AR.Drone serial [11]. Many researches use the AR.Drone as the drone platform, such as [12] [13] [14] [15] and [16].

In general, for current market available UAV platforms, the sensors system includes 1) a downward looking or a forward looking camera, 2) an Inertial Measurement Unit (IMU) which contains a 3-axis gyro and a 3-axis accelerometer, 3) a 3-axis compass, and 4) a downward looking ultrasonic sensor. The camera of the UAV can provide image flows which is able to

apply vision algorithms. The IMU sensor system can provide the angular velocity and acceleration information of the vehicle. The compass sensors measures the magnetic field of the earth, and that information can provide the heading angle of the vehicle which is able to inhibit the gyro drift [17]. The ultrasonic sensor can send ultrasonic wave and measure the time that the ultrasonic wave rebound. Since the speed of the ultrasonic wave is known as the sound speed, the distance between the sensor and the object can be calculated [18]. Usually, this kind of sensor is used to measure the altitude of the UAV.

For the processor on-board, it is usually low-cost ARM processor [19]. The function of the processor on-board current mainly include sensor data acquiring, wireless communication with ground station, and motor controls. The high level algorithms are usually programmed on the ground station. For example, the Parrot AR.Drone uses this type of configuration [20]. While some market available can run high level algorithms on-board, such as DJI Matrice serial [21]. However, the price is relevantly much higher.

Many techniques are involved for guidance and navigation of small UAVs. For applications based on visual information, pattern recognition [22], visual odometry [23], and visual Simultaneous localization and mapping (SLAM) [24] technologies are usually adopted. Pattern recognition is the automated recognition of patterns and regularities in data. Pattern recognition is closely related to artificial intelligence and machine learning [22], together with applications such as data mining and knowledge discovery in databases (KDD), and is often used interchangeably with these terms. In robotics and computer vision, visual odometry is the process of determining the position and orientation of a robot by analyzing the associated camera images. It has been used in a wide variety of robotic applications, such as on the Mars Exploration Rovers [25]. In navigation, robotic mapping and odometry for virtual reality or augmented reality, simultaneous localization and mapping (SLAM) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it [26] [27] [28]. For using the data from IMU, compass, and ultrasonic sensor, Strapdown inertial navigation [29] [30] is the typical technique to acquire attitude and speed of the vehicle. Strapdown inertial navigation is a navigation method that uses a computer, accelerometers and gyroscopes to continuously calculate by dead reckoning [31] the

position, the orientation, and the velocity of a moving object without the need for external references. Often the inertial sensors are supplemented by a barometric altimeter and occasionally by magnetic sensors or speed measuring devices.

1.3 Contributions and outline

The size of the structure, battery, and on-board sensors are reduced. So the payload of small UAVs is limited. As a results, the quality of the on-board sensors and the on-board computation ability are constrained. Under these conditions, to realize the complicated tasks, researches on small VTOL UAVs are growing in many aspects. This thesis mainly focus on three aspects: 1) Known object recognition and tracking for vessel deck landing application; 2) Simultaneous localization and mapping (SLAM) technique; 3) Wind disturbance estimation; and 4) UAV platform design.

The contributions of the thesis include four aspects.

(i) Using only on-board IMU, ultrasonic and camera sensor, a robust landing procedure is developed. The procedure include two stages: long range approaching and short range landing. At the long range approaching stage, the drone first recognize the red LEDs applied on the landing board from long range, then estimate the relevant distance and directions and drive automatically approach the landing board. At the short range landing stage, once the drone hover above the landing board, the bottom camera start to work and recognize the designed 'H' landing sign, adjust the position and land on the board. To test the developed algorithms, a self-designed landing platform, which can emulate the deck motion of a vessel 30 m wide under sea state 6 is used. Flight results demonstrate the landing success rate of 100% with a 9 cm average distance error.

(ii) A novel solution to on-board sensor only feature based monocular 3-D SLAM algorithm for Small UAVs is developed. The pin-hole camera model is derived in an alternative prospective and the proposed algorithm combines the IMU measurements inherently. The proposed SLAM algorithm can navigate autonomously in previously unknown environments without external positioning aid by just using one camera and an IMU. Since

the pin-hole camera model is derived in an alternative prospective and the proposed algorithm combines the IMU measurements inherently, the proposed method is able to generate absolute scale map without map initialization and optimization methods. These advantages make the presented SLAM algorithm more applicable to use than other existing methods for small UAVs. Applying the proposed method to a low-cost quadrotor, experimental results demonstrate that our system is able to generate map correctly with absolute scale. At the same time, the position estimation accuracy of the camera is also improved compares to use inertial navigation only.

(iii) Different from the traditional aerodynamic modeling solutions, in this thesis, a K Nearest Neighborhood learning based method which does not require the details of the aerodynamic information is developed. The proposed method uses the features generated by flight data as the learning model, and does not require the details of the aerodynamic information. Also, this method only uses flight data to generate the designed features for the training and wind estimation stages, and does not require direct wind speed measurements. Experiment results show that the proposed method can estimate the wind accurately, with a 0.074 m/s average estimation error, and a 3.3% average estimation relative error, while the average estimation error generated by AR.Drone's embedded system is 1.673 m/s.

(iv) A quadrotor hardware solution which is able to support majority of research of small VTOL UAVs is developed. The quadrotor research platform equipped two-level processor system which is able to process all algorithms on-board. With the sensor system embedded, rich flight information is provided to the researcher that can enable endless possibilities. Relied on the on-board wireless communication module, the flight state and important parameters can be monitored by researchers in real-time, which makes the platform humanized. Further more, the on-board processor also monitor the manual control signal during the flight, and is able to switch the control source to manual control under emergency situation, which makes the platform safe to operate.

Chapter 2

Visual Tracking and Navigation for Landings on Moving Vessel Decks

2.1 Introduction

Nowadays, unmanned aerial vehicles (UAVs) are playing critical roles in many areas [32]. Numerous applications exist that require autonomous maritime deployment and retrieval of a UAV, such as iceberg monitoring, coastal surveillance, and wildlife monitoring [33]. The vertical take-off and landing capability (VTOL) makes the quadrotor particularly suitable for maritime applications, where the runway for takeoff and landing is often very short. However, autonomous landing of quadrotors on a ship deck is a challenging task owing to limited payload for sensors [34], the unreliable GPS measurements [33], and especially ocean waves' disturbance [35].

Without using GPS measurements, for the moving platform autonomous landing problem, vision based solutions are widely used, and various approaches have been studied [36]. Compared with the on-ground vision landing system, such as [37] and [38], on-board vision landing system is typically adopted, especially for landing on a moving platform [39]. For onboard vision solutions, some researchers are interested in stereo vision systems, since stereo vision systems are convenient to acquire depth information [40]. Reuben et al. present a stereo based method for the interception of a static or moving target [41]. Target detection and relative position estimation are realized by using two fisheye cameras. Because stereo based method needs two cameras, which leads to more weight on-board and increased cost, many researchers propose monocular vision based approaches. For example, Xudong et al. describe a UAV implemented with vision and laser based localization algorithm to track and land on a moving platform [42]. In that work, a LiDAR scanning range finder is used to determine the altitude. In [43], Benini et al. present a real-time GPU-based pose estimation system for UAV

autonomously takeoff and landing. The monocular video stream is processed on-board, and the test results show that the proposed system is able to provide precise pose estimation with a frame rate of at least 30 fps and an image resolution of 640x480 pixels. About the camera type selection, some researchers use ordinary cameras. In [44], Francesco et al. committed to the ability of a vision algorithm to detect and track a given landing sign. Optimized adaptive thresholding technique is developed to ensure the robustness of the system in different illumination situations. Benini et al. present an indoor mini-UAV localization system in [45]. With on-board monocular vision and wireless sensor network, 10cm localization accuracy has been achieved. Some researchers use infrared (IR) camera, which is able to work at dark environments [46]. For example, Karl et al. use infrared (IR) camera to track a pattern of IR lights in conditions without direct sunlight [47].

Compared with other moving landing platforms, a vessel deck environment is even more challenging. It may have significant pose disturbances due to the sea waves and the wind. Most studies have not demonstrated their approaches can work when using vessel deck emulators, which include both pose and heave motion. In [48], Botao et al. have presented a control structure that achieves the fast and accurate landing of quadrotor onto a vertically sinusoidally oscillating platform. Their control structure consists of three modules: a motion estimation module, a trajectory generation module, and a tracking control module. However, the roll and pitch movements of the platform are not included. In contrast to the flat landing platform, John et al. present a Laser-based guidance of a quadrotor for precise landing on an inclined surface [49]. However, the pose angle of the landing platform is fixed in the experiments. PITM et al. present a landing algorithm that can land a quadrotor on a pose oscillating platform in [50]. However, in their experiments, the quadrotor is suspended from a string to restrict it from drifting away, which is not realistic in real landing. Furthermore, the movement of the platform is achieved by two people using their hands. The dynamics of the landing platform is not a good emulation of a vessel deck, and the quadrotor is too close to people and could be very dangerous. The dynamics of the ship-deck is introduced and emulated in [35]. The pose of the ship deck with respect to the vehicle is estimated by a vision system. However, that work didn't include UAV state estimation and motion control methods. A more complete and realistic solution is presented in [51]. The authors provide a complete approach which

lands a UAV on a kayak positioned 20 m from the shore. However, the indoor test did not take the pose change of the landing platform into consideration, and for the landing on the kayak, the water waves are 'mild' according to their description. More significantly, according to the experiment results, under the indoor test environment without any disturbance, the average landing accuracy is ± 12 cm from the center of the board, and the landing success rate is less than 90%, which is far away being satisfied for critical landing missions.

This work presents a monocular vision based system consisting of an autonomous long range approaching method and an accurate landing algorithm using only on-board cameras and other sensors provided by AR.Drone. The main contributions of this work include two aspects. First, simple but effective long range target detection and approach algorithms are developed. Second, we propose a novel system architecture that addresses realistic vessel deck landing using only low-cost on-board sensors. The designed landing process includes two stages: approaching from long range and landing after hovering above the landing platform. For the first stage, a robust method is designed to drive the done to an area above the landing platform. In this stage, the red color LEDs are used for landing platform tracking. The field of view (FOV) orientation of the front camera of the AR.Drone is redesigned to help searching the landing platform, and the state estimation is based on the new geometric relationship between the camera and the landing platform. We use a PID controller to stabilize the heading angle towards the landing platform. For the movingdeckond stage, the drone lands autonomously on the landing platform accurately. We accomplish the landing sign recognition using computer vision solutions, and 'H' landing sign will be tracked once it appears in the FOV of the bottom camera. The state estimations are performed using on-board low-cost IMU, ultrasonic altimeter, and the image informations. Kalman filters are used to further improve the estimation accuracy. Lastly, we use PID controllers to execute the flight controls. Additionally, a vessel deck emulator, which can emulate movements of a vessel deck in high sea state conditions, is designed and used to validate the reliability and robustness of the proposed architecture. This emulator includes an 'H' landing sign surrounded by 4 red LEDs, and it provides a realistic emulation of the pose and heave motion of a vessel deck.

The organization of this Chapter is as follows. In Section 2, the background information of the components of the system is presented, including AR.Drone, the self-designed landing

platform, and a Vicon Motion Capture System. Coordinate conventions are also introduced. Section 3 introduces how the landing platform is designed and how it works. The long range landing platform detection and tracking are presented in Section 4, including camera configurations, methods to recognize the landing sign, state estimation algorithms, and approaching procedure. Section 5 introduces the hovering and landing stage when the drone approaches to the landing platform. Image processing methods, state estimation algorithms, motion control, and the architecture of algorithms are presented. Indoor experiment results are presented in Section 6.

2.2 Back Ground Information

Here we present the hardware and system architecture we developed for the testbed. Because the on-board camera and sensors, the landing platform, and the motion capture system which is used as ground truth are working in different coordinate frames, we will also define the coordinate frames and introduce the transformation among them in this section.

2.2.1 Testbed Description

(i) Parrot AR.Drone

The micro UAV we used in this work is a Parrot AR.Drone. AR.Drone is a quadrotor helicopter that has a 6 degree of freedom internal IMU, an ultrasonic altimeter, a forward-looking (front) camera and a downward-looking (bottom) camera [52], with a 640×360 resolution for both cameras. For the front camera, we measured the horizontal FOV is about 60°, and the vertical FOV is about 30°. For the bottom camera, we measured the horizontal FOV is about 40°, and the vertical FOV is about 25°.

With the sensors equipped on-board, altitude z is determined by a ultrasonic altimeter, roll angle ϕ and pitch angle θ are calculated by using data from an IMU. Estimations of these states are directly available from the Parrot AR.Drone. Although yaw angle ψ is also provided, we don't use it due to its low accuracy – the yaw drift is about 12° per minute when flying and about 4° per minute when in standby [20]. For yaw angle ψ and relevant position estimation, our solution is using vision algorithms.

According to [20], there are 4 control channels to control AR.Drone: (1) Roll angle, (2) Pitch angle, (3) Angular velocity of yaw, and (4) velocity of the altitude direction. We denote them as:

$$u_1 = \phi \tag{2.1}$$

$$u_2 = \theta \tag{2.2}$$

$$u_3 = \dot{\psi} \tag{2.3}$$

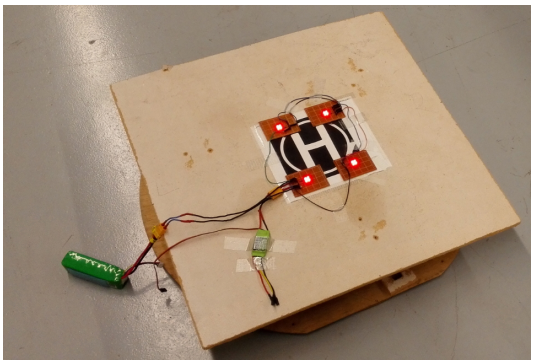
$$u_4 = \dot{z} \tag{2.4}$$

Because of the limited capability of the ARM processor equipped on AR.Drone [20], we run the system code on a ground computer and stream commands to the UAV over a Wi-Fi connection. The UAV streams IMU data and camera images to the ground computer also over the Wi-Fi connection. From our experiments, the delay of the Wi-Fi connection do not cause the failure of the state estimation as well as the control.

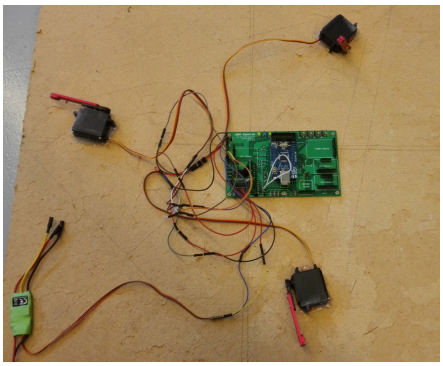
(ii) Vessel Deck Emulator

We emulate the movement of the ship deck in the sea, using our self-designed landing platform. See Fig. 2.1. The motivation to design this landing platform is to emulate pose change and heave motion of a vessel deck. The ship can be modelled as a rigid body moving in the sea [32], [35], such that the movement of a ship deck can be simulated by an attitude-programmable plate. The concept that 3 points define a plane is used in the design of the landing platform. The same idea is adopted in designing the landing platform in [53].

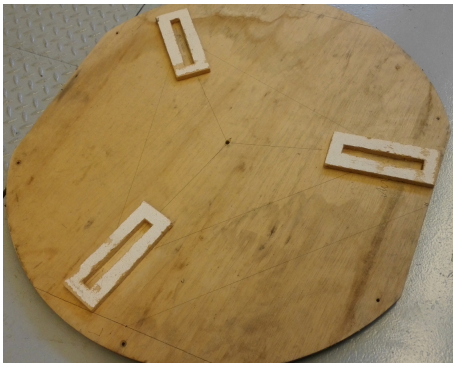
The landing platform includes 2 pieces: a square plate and a bottom base. The square plate is powered by 3 independent servos deployed on the back, and they are located at the 3 vertices of an equilateral triangle. Each servo is connected with a rigid arm at the joint, which is used to change the height of that vertex. The servos are controlled by a STM32 ARM processor, which sends PWM signals to change the angle of arms. On the top side of the square plate, an 'H' sign for landing is attached at the center. Besides that, 4 red LEDs are deployed at the 4 corners of the 'H' sign. These LEDs are used to guide the drone from long distances.



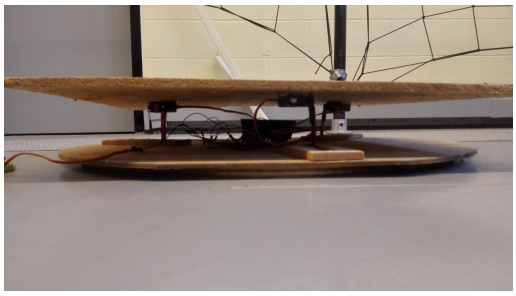
(a) Top view of the plate.



(b) Bottom view of the plate.



(c) Landing platform base.



(d) Side view of the platform.

Figure 2.1: The landing platform.

The function of the base is to hold the plate. There are 3 sliding grooves. We set the arms in those grooves and make sure the arms move in a limited area, so the plate will not move away from the base. The details of the landing platform is introduced in section 2.3.

(iii) Vicon Motion Capture System for Ground Truth

We also use a Vicon Motion Capture System (including 8 Bonita cameras capture up to 250 fps with one megapixel of resolution) to provide state measurements for the quadrotor, which is assumed as the ground truth. We emphasize that, the measurements from Vicon are only used for validation, and the drone only uses the information from on-board sensors to realize its autonomous flight.

2.2.2 Coordinate Frames and Transformations

Four coordinate frames are used in this work. The first 3 coordinate frames are 3D right-handed, and the unit is meter. The inertial or east-north-up (ENU) frame, denoted as N, is the basis for Vicon measurements and provides the inertia position. The origin is set to be the center of our experimental site. East, north, and up will be referred to as the axis x_N, y_N, and z_N, respectively. The quadrotor body frame B, with the origin to be the bottom camera's lens' center, x_B axis between rotors 1 and 2, pointing to the optical axis of the front camera, y_B between rotors 2 and 3 and z_B pointing to the optical axis of the bottom camera, is used to define the vehicle motion. Because the rigid connection between the body of quadrotor and the bottom camera, which we use as the image source, the body frame B also works as the camera frame. Local reference frame L works as the intermediate frame between the inertial frame and the body frame. The origin of local reference frame is same as the body frame, while x_L points to the north, y_L points to the east, and z_L points downward. See Fig. 2.2 and Fig. 2.3.

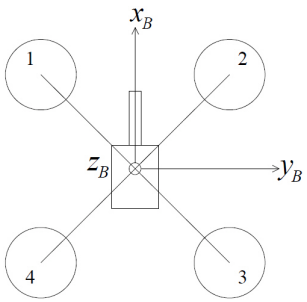


Figure 2.2: Body frame.

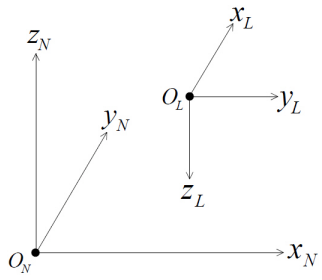


Figure 2.3: Inertial frame and local reference frame.

The last frame is the image frame I, used for both the front camera and the bottom camera. Frame I is a 2D frame, with the origin O_I to be the top-left point of the image, U axis along the length, and V axis along the width. The unit of image frame is pixel. See Fig. 2.4. The coordinate of an arbitrary point P is (u,v) with respect to the image frame.

Fig. 2.5 shows the optical geometric relationship between the bottom camera and the drone's body. In the 3D chart of Fig. 2.5, O_B is the center of the camera's lens, and also is the origin of frame B. Distance between O_B and the center of the image plane is f, which is the camera's focal length. In the 2D chart of Fig. 2.5, O_I is the origin of image frame, and the coordinate of image center O_0 is (u_0, v_0) with respect to the image frame and $(0,0,f)$ with

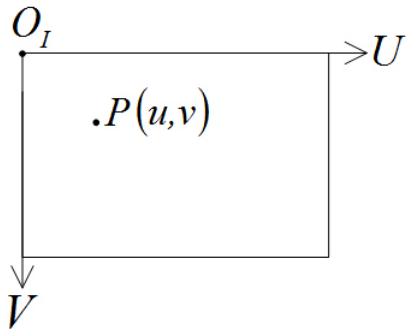


Figure 2.4: Image frame.

respect to the body frame. The coordinate of point P_0 is (u,v) with respect to the image frame and (x_0, y_0, f) with respect to the body frame.

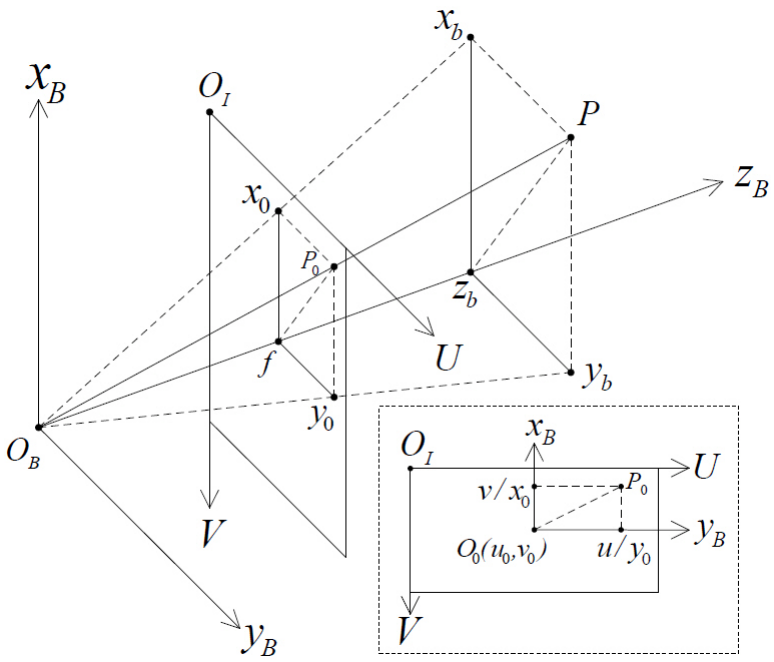


Figure 2.5: Relationship between image frame (respect to the bottom camera) and body frame.

Now define rotation matrices for the 3D frames. Let $\overrightarrow{p_B}$ be the vector respect to frame B, where $\overrightarrow{p_B} = \begin{bmatrix} x_b & y_b & z_b \end{bmatrix}^T$. And let C_B^L be the rotation matrix which transfers the vector respect to frame B to the vector respect to frame L. So $\overrightarrow{p_L} = C_B^L \overrightarrow{p_B}$. Also, by chain rule, $C_B^N = C_B^L C_L^N$.

From Fig. 2.3, we have the rotation matrix

$$C_L^N = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{2.5}$$

Rotation matrix C_B^L can be generated from sin and cos functions of Euler angles. We set ϕ be the roll angle, θ be the pitch angle, and ψ be the yaw angle. We also denote $s\theta$ for $\sin\theta$ and $c\theta$ for $\cos\theta$. Then we have

$$C_B^L = \begin{bmatrix} c\theta c\psi & -c\phi s\psi + s\phi s\theta c\psi & s\phi s\psi + c\phi s\theta c\psi \\ c\theta s\psi & c\phi c\psi + s\phi s\theta s\psi & -s\phi c\psi + c\phi s\theta s\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix} \tag{2.6}$$

From Fig. 2.5, we can derive the transformation from image frame I (respect to the bottom camera) to body frame B. Suppose a light source, located at point P, sends light to point O_B through point P_0, where P_0 is located at the image plane. We have

$$u = \frac{y_0}{k} + u_0 \tag{2.7}$$

$$v = -\frac{x_0}{k} + v_0 \tag{2.8}$$

where k is the size of the pixel, which is a constant parameter of the bottom camera, and the unit of k is meters. We also have

$$\frac{x_0}{x_b} = \frac{f}{z_b} \tag{2.9}$$

$$\frac{y_0}{y_b} = \frac{f}{z_b} \tag{2.10}$$

From equations (2.7) to (2.10), we have

$$x_b = -\frac{k}{f}\big(v - v_0\big)z_b \tag{2.11}$$

$$y_b = \frac{k}{f}\big(u - u_0\big)z_b \tag{2.12}$$

For convenience, we define $\lambda = \frac{k}{f}$, $\alpha = \frac{x_b}{z_b}$, and $\beta = \frac{y_b}{z_b}$. Then we can write equations (2.11) and (2.12) into matrix version, we have

$$\begin{bmatrix} \alpha \\ \beta \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -\lambda \\ \lambda & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} \lambda v_0 \\ -\lambda u_0 \\ 1 \end{bmatrix} \tag{2.13}$$

Apparently, we also have

$$\begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix} = z_b \begin{bmatrix} \alpha \\ \beta \\ 1 \end{bmatrix} \tag{2.14}$$

The constant λ can be determined by experiments using the least square method [54]. We choose known points, measure the corresponding α, β, u, and v, and then use equation 2.13 to calculate λ. From our result, we have obtained $\lambda = 0.00146$ for AR.Drone's bottom camera.

2.3 Vessel Deck Emulator

We have constructed a moving vessel deck emulator as the testing platform. The detailed design and the programmed movement of the vessel deck emulator are introduced in this section.

2.3.1 Vessel Deck Design

Fig. 2.6 shows the distribution of servos on the square plate for landing. The length of each side of the square is 60 cm. Servos are located at three vertices of an equilateral triangle, and the length of each edge is 40 cm.

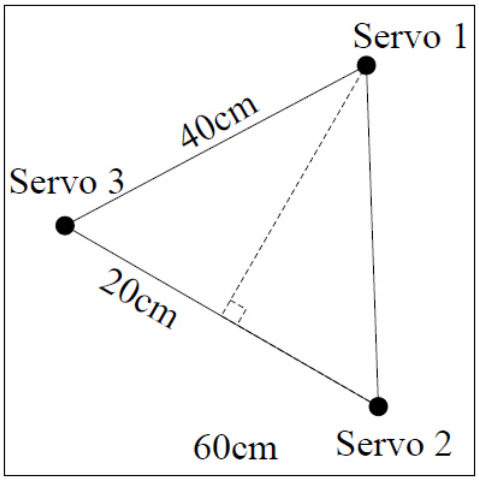


Figure 2.6: Servo distribution on the bottom of landing plate.

Since the three independent servos share the same configuration, Fig. 2.7 only shows one of them for illustration. The height of this servo point is defined as h. Distance from the joint to the bottom of the servo d is measured as 1 cm, and length of the rigid arm l is measured as 7

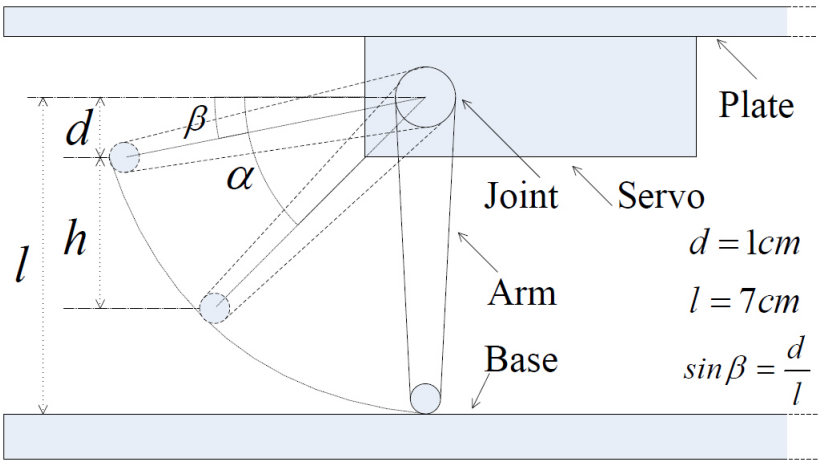


Figure 2.7: Servo point configuration.

cm. The angle between the landing plate and the arm α is programmable, and h changes when α changes. From the geometric relationship shown in Fig. 2.7, we have

$$h = l \sin \alpha - d \tag{2.15}$$

Since h reaches the maximum value of 6 cm when $\alpha = \frac{\pi}{2}$, and it reaches the minimum value of 0 cm when $\alpha = \beta$, α is constrained as $\beta \leq \alpha \leq \frac{\pi}{2}$.

Under this configuration, from calculation, the landing platform has the capability of a maximum 6 cm heave movement and a maximum 10° pose angle movement (pose angle is the angle between the plate plane and the bottom base plane). Suppose that the vessel is 30 meters wide, from the geometric relationship, if the slope of deck is 10°, the minimum wave height is 5.2 m ($\frac{5.2}{30} \approx \sin(10°)$). According to [35], this condition is under sea state 6, which is the "very rough" level.

2.3.2 Deck Control and Results

We program the landing platform to emulate the deck motion under unfavorable sea state condition. Sinusoidal function is commonly used for simulating the movement of a vessel deck [35]. For our landing platform, we program the height of each servo point with a sinusoidal function independently. See Fig. 2.8.

Fig. 2.9 is the actual motion of the landing platform under the control commands, and the data is an extracted 8 seconds' segment of a test from Vicon. The top plot is the position of the

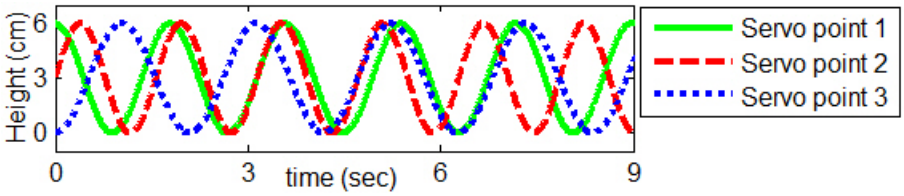


Figure 2.8: Height control to each servo point.

center of 'H' sign. The bottom plot is the attitude of the landing plane. From Fig. 2.9 we can see that the maximum altitude of the 'H' sign is 6cm, and the maximum absolute roll and pitch angles of the landing plane are 10°. According to the analysis in section 2.3.1, this movement emulates the deck motion of a vessel which is 30 meters wide under sea state 6 (very rough).

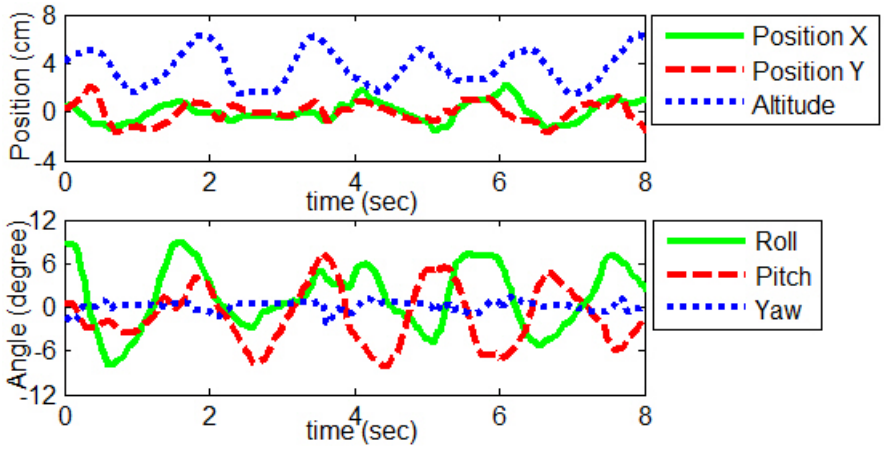


Figure 2.9: The motion of the landing platform.

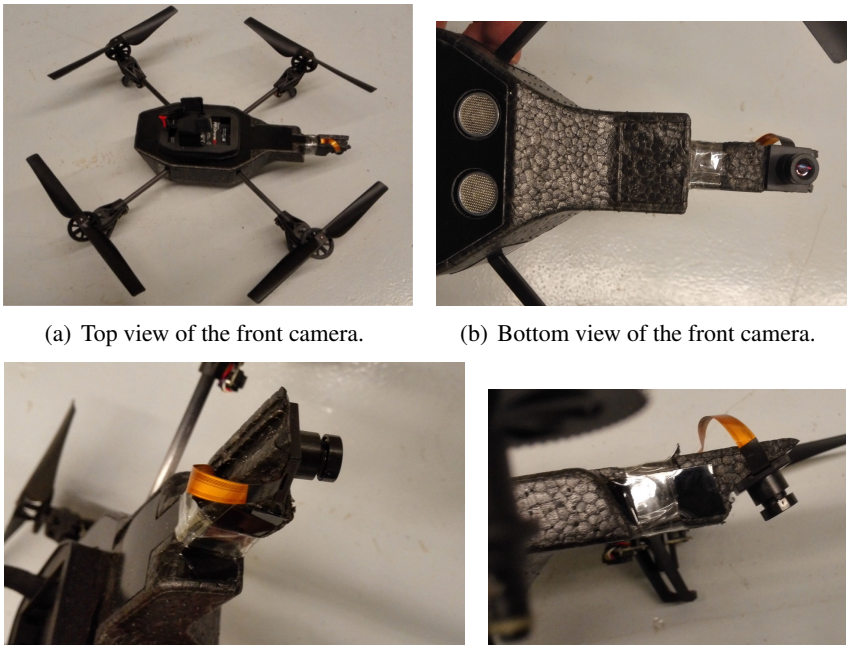
2.4 Long Range Landing Platform Detection and Tracking

For successful autonomous landing, the first task is to detect the landing platform. After detection, the drone should be able to track and approach to the platform autonomously. This section presents the method we used to achieve this goal.

2.4.1 Front Camera Configuration

The front camera of AR.Drone is originally set to look forward, and as we discussed in section 2.2.1, it has an approximate 60° horizontal FOV, and an approximate 30° vertical FOV. However, if this configuration is used without any adjustment, even if the algorithm can drive the

drone to approach the landing platform, there will be a blind zone since the landing platform will not show in neither the view of front camera nor the view of the bottom camera. To eliminate the blind zone, we did some modification on the front camera. First, we changed the fix angle of the camera. It points along the X axis of body frame B before. After the modification, it points a little downward, and the central view respect to B is in X-Z plane, with about 75° of angle to X axis. Second, we rotated the camera by 90°. So it has an approximate 60° vertical FOV, and an approximate 30° horizontal FOV. Fig. 2.10 shows the front camera configuration after the modification.



(a) Top view of the front camera. (b) Bottom view of the front camera.

(c) Front view of the front camera. (d) Side view of the front camera.

Figure 2.10: Front camera configuration after the modification.

Fig. 2.11 illustrates the entire camera configuration after the modification. In Fig. 2.11, α is the vertical FOV of the front camera, and β is the angle looking backward of the front camera. The distance between the drone and the landing platform is notated as l. The coordinate of the landing platform in the image frame is (u, v), and since it rotated by 90°, in side view, only u is shown there.

From this configuration, if the drone has enough altitude h, there will be an overlap FOV area between the front camera and the bottom camera. And once the landing platform appears in the overlap area, the system switches to bottom camera mode autonomously and it won't

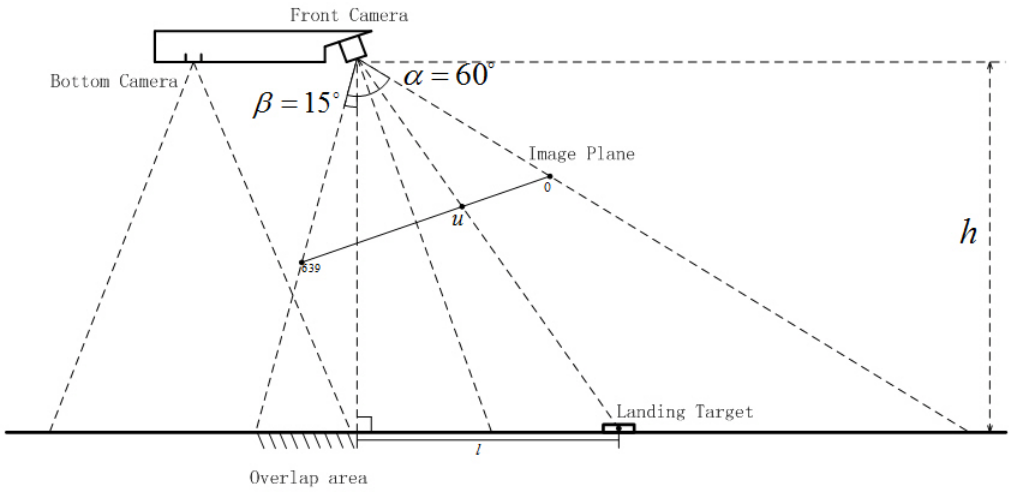


Figure 2.11: Camera Configuration.

have a blind zone.

2.4.2 Landing Platform Recognition

Four red LEDs are fixed at the 4 corners of the landing 'H' sign. As introduced in section 2.2.1, these LEDs are used for the UAV to search the landing platform from a long range. See Fig. 2.12. We note this is a simplified model of the reality. For long range shipboard detection, the ship can be seen as a single hot spot by the on-board IR camera [55]. Limited by the AR. Drone's cameras and indoor environment, we use red LEDs to simulate the hot spot and use the front camera of the AR. Drone to recognize and track the red LEDs.

In this part, red dots recognition algorithm is designed and used. The algorithm first converts the image into 3 matrices to store the RGB values. A simple but effectively algorithm is used. The dot which has the R value greater than 150, G and B values are smaller than 80 is considered as the red dot. After the algorithm searches over the entire image, a matrix is generated to store the coordinates in the image frame of the red dots. Then the average of the coordinates is calculated, which is considered as the coordinates of the landing platform. Fig. 2.13 shows the recognition results. The recognized red dots are circled with green circles, and the calculated average coordinate (landing platform coordinate (u, v)) is circled with a blue circle. We note the current algorithm can't distinguish between lights belonging to the landing pad and others, and we don't have other red light sources in the experiment area. This limitation

will be studied in future.

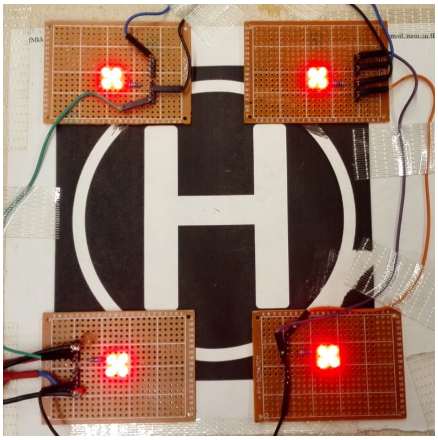


Figure 2.12: Red LEDs and 'H' landing sign configuration.

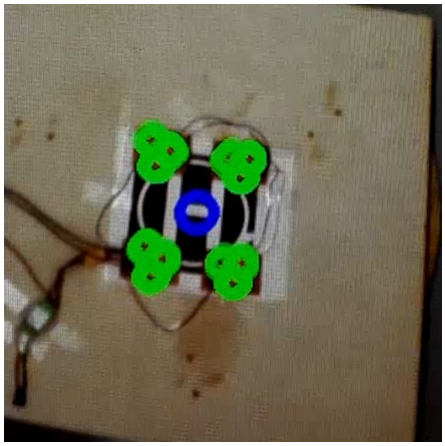


Figure 2.13: Red LEDs recognition and landing platform calculation.

2.4.3 States Estimation

After we get the coordinates of the landing platform in the image frame, the next step is to estimate the states of the drone. As introduced in section 2.2.1, the altitude z, roll angle ϕ and pitch angle θ are directly available from AR.Drone. The goal here is to estimate the heading angle and the horizontal distance between the drone and the landing platform.

(i) Heading Angel Estimation

The heading angle discussed here represents the angle between the line connecting the center of the landing platform and the projection of the drone on ground, and the projection of the x axis of the body frame B (x_B). Coordinate v of the landing platform in the image frame is used to determine the heading angle ψ. Because the horizontal FOV we measured is 30°, and the pixel number is 360 in that direction, we assume that $v = 359$ and $v = 0$ represent $\pm 15°$ heading angle, and $v = 180$ represents 0° heading angle. Furthermore, the relation between coordinate v and heading angle ψ is assumed to be linear. Then we get the heading angle estimation represented in Equation (2.16).

$$\psi = \frac{v - 180}{12} \times \frac{\pi}{180} \tag{2.16}$$

(ii) Horizontal Distance Estimation

Coordinate u of the landing platform in the image frame is used to determine the horizontal distance l (assuming the yaw angle $\psi = 0$). Refer to Fig. 2.11. The vertical FOV is 60° (α), and there is a 15° backward view (β). The pixel number in that direction is 640. Since we know altitude h and coordinate u, from geometric relationship, we can calculate l. First, we define angle θ to be the angle between the optical line connecting the lens' center and the landing platform, and the main axis of the lens. We get

$$\tan\theta = \frac{320-u}{w} \tag{2.17}$$

where w is equivalent pixel number of the distance between the center of the image plane and the center of lens.

We also have

$$\tan\frac{\pi}{6} = \frac{320}{w} \tag{2.18}$$

Combining equations (2.17) and (2.18), we have

$$\theta = \arctan\frac{(320-u)\times\tan\frac{\pi}{6}}{320} \tag{2.19}$$

If θ is greater than -15°, the landing platform is in front of the drone, and

$$l = h\times\tan(\theta + 15\times\frac{\pi}{180}) \tag{2.20}$$

If θ is smaller than -15°, the landing platform is in back of the drone, and

$$l = -h\times\tan(\theta + 15\times\frac{\pi}{180}) \tag{2.21}$$

2.4.4 Approach Procedure

In the stage of approaching from long range, the goal is to drive the drone to an area above the landing platform. Once the approach process completes, the drone will switch to the bottom camera autonomously, and the landing 'H' sign will be located in the FOV of the bottom camera. To achieve this goal, a robust approach procedure is designed and introduced in this section. Since we already have the state estimation, the control algorithm is also introduced.

(i) PID Control Law

A PID control method for discrete time systems is used in this work. In general, let $s(k)$ be the state measured in step k, $s^d(k)$ be the desired state of step k, then the error at step k is

$$s^e(k) = s^d(k) - s(k) \tag{2.22}$$

The summation of historical errors at step k is

$$s^{et}(k) = \sum_{i=0}^{k} s^e(i) \tag{2.23}$$

The difference of error at step k is

$$s^{ed}(k) = s^e(k) - s^e(k-1) \tag{2.24}$$

Then the control input u at step k is defined as

$$u(k) = Ps^e(k) + Is^{et}(k) + Ds^{ed}(k) \tag{2.25}$$

Where P, I, D are the gains of the PID controller.

For the control of heading angle ψ and roll angle ϕ, the goal is to maintain them close to 0°. The measured angles are the control error inputs, since the goal is to drive them to 0°. If the heading angle is equal to 0°, it means that the head of the drone is pointing to the landing platform. Under that condition, we can change the pitch angle θ to drive the drone to approach the landing platform. For the control of altitude z and pitch angle θ, the desired value changes during approach. The approach process is presented next.

(ii) Approach Process

First, the drone takes off from an arbitrary position in the lab, hovering at 2 meters. Then it turns around without position change to find the red dots. After the red dots appear in the view of the front camera, the algorithm locks the red dots, and calculates the relative heading angle. Heading angle PID controller adjusts the heading angle around 0°. Under this condition, the drone changes its pitch angle to approach the landing platform. Notice that the drone approaches the landing platform only when the heading angle is small. If

the angle is not small, the drone is programed to stop moving forward and hover at the current position, while the heading angle PID controller will drive the heading angle to 0°.

When approach, the red dots' position in image is also changing. To make sure the drone can always see the red dots, an altitude PID controller drives the drone to decrease its altitude if the coordinate u of the landing platform is too close to the image boundary during the approach process. Once the altitude reaches to 1 meter, the altitude PID controller stabilize the height to be 1 meter without further descending. This is because at 1 meter altitude, the view of the bottom camera is clear and wide enough for later landing. During this process, the horizontal distance l is calculated by using equations (2.20) and (2.21).

Finally, when l becomes negative, which means the landing 'H' sign is in the overlap FOV area between the front camera and the bottom camera, the algorithm switches to the bottom camera mode autonomously, and the landing stage algorithms start to work. Fig. 2.14 is the flow chart of the designed approach process.

2.5 Hovering and Landing

After the drone hovers above the landing area, the second stage starts and the goal is to land on the 'H' sign accurately. This section presents our methods to achieve this goal.

2.5.1 Image Processing

'H' landing sign is chosen as the landing beacon due to its widely used. The goal for the image processing is to find the 'H' sign in the image and locate the center of the sign in the image frame in real-time. Image moments technique is used for the 'H' sign tracking. In image processing, image moment usually contains some special characteristics of the contour, and image moment is generally used to recognize certain objects or patterns [36].

Before the flight, the contour of a standard 'H' is extracted from a reference image. During the flight, 'H' sign tracking algorithm keeps comparing the moment of each qualified contour derived from each image frame with the standard moment of 'H' contour. If the similarity of

the most similar contour is within a selected threshold, the contour will be recognized as the 'H' landing sign. In this part, OpenCV library functions are used for contour extraction and comparison. Details are presented next.

(i) Standard 'H' Contour Extraction - Before the Flight

Fig. 2.15 is the reference image we used for standard 'H' contour extraction. We also printed it out as the 'H' landing sign on the landing plate. The reference image is a binarized image. After we load it in OpenCV environment, function 'findContours' is called to extract all contours. We select and save the contour of 'H', and use red line to draw the 'H' in Fig. 2.16.

(ii) Real-time 'H' Sign Tracking - During the Flight

During the flight, the bottom camera will capture color images in 20Hz. For each image, it will first be converted to a gray image. Furthermore, the gray image will be converted to a binarized image by carefully selecting a threshold. We can do this because the 'H' sign has high contrast with the background. The binarized image will be used to extract contours. Before we extract the 'H' sign's contour, a de-noising process will be conducted. After that, we extract all contours detected in the image, and eliminate the contours which are too small to be the 'H' contour by calculating the contour area. Then the contours left will be compared with the standard 'H' contour saved before. OpenCV function 'matchShapes' is used for the comparison. The function first calculates the moment of contour, and then compares it with the standard 'H' contour moment. The result is given as 'comre', which represents the similarity of them. The smaller the 'comre' is, the higher similarity they have. A threshold for 'comre' is set to be 0.5. In each image frame, the contour which has the minimum 'comre' with the standard 'H' contour, which is also within the threshold, will be selected as the 'H' landing sign. Fig. 2.17 is the state machine of the image processing.

Fig. 2.18 shows the operation results during the image processing. Fig. 2.18(a) is the original color image taken from the bottom camera. Fig. 2.18(b) is the gray image converted from Fig. 2.18(a). Fig. 2.18(c) is the binarized image after de-noising. Binalize makes it easier to find contours, and de-noising eliminates the small dots, which

decreases the number of invalid contours and reduces the burden on comparing algorithms. Fig. 2.18(d) shows a case that 'H' landing sign is detected. Once the algorithm finds the 'H' landing sign, the contour of 'H' will be drawn. Then, OpenCV function 'minAreaRect' is used to bound the 'H' with a minimum area of rectangular, and also to provide the center coordinate in the image frame I. Because 'H' is symmetric, this coordinate is also the coordinate of the 'H' landing sign. After getting the center coordinate, we draw a small circle at the center.

Because the landing 'H' sign will move with the deck emulator, it will not always be parallel to the X-Y plane of the body frame. To further test the robustness of the proposed 'H' sign detection algorithm, we take pictures of the 'H' sign with different orientations and distances, using different cameras under different lighting conditions to test if the algorithm can still work. Fig. 2.19 shows some of the test results.

In Fig. 2.19, the 'Bright light condition' is obtained by using a computer screen to show the 'H' landing sign, the 'Medium light condition' is obtained in our lab with all lights turned on, the 'Dark light condition' is obtained in our lab with only half lights turned on. The 'Smart-phone camera' used has 1920×1080 resolution, while the AR.Drone bottom camera has 640×360 resolution. The 'Medium distance' in this test is 60cm, while the 'Short distance' in this test is 30cm. In Fig. 2.18(d), the distance from the sign and the camera is 1m.

From these experiments, we conclude that the 'H' sign detection algorithm is robust. It can work under different lighting conditions, different resolutions of camera, different distances, different orientations and different tilt angles. Additionally, we tested the maximum tilt angle of the 'H' when this algorithm works using another experiment, and found that the maximum tilt angle is 20°. Since the maximum pose angle of the designed deck emulator is 10°, the proposed algorithm can track the 'H' landing sign successfully when the landing platform is moving.

2.5.2 State Estimation

Only on board sensors are used for state estimation. Sensors provided by AR.Drone are a 6 degree of freedom internal IMU, an ultrasonic altimeter, and a forward-looking (front) camera, and a downward-looking (bottom) camera.

Six states are required for autonomous landing, including 3 components of position vector with respect to local reference frame $\left(x_l, y_l, z_l\right)$, and 3 Euler angles $\left(\phi, \theta, \psi\right)$. This section presents calculations of these states by using the sensors mentioned above. Notice that it only use the bottom camera during the landing stage.

Two Kalman filters are designed to improve the estimation accuracy. States for the first Kalman filter are z_l, ϕ, and θ, which are always available. And states for the second Kalman filter are x_l, y_l, and ψ, which are available only when 'H' landing sign is detected.

From AR. Drone system, users can directly acquire altitude z_l, roll angle ϕ, and pitch angle θ, where z_l is calculated using ultrasonic altimeter, and ϕ and θ are calculated using the IMU.

Although AR.Drone also provides users with the yaw angle ψ, the yaw angle drifts significantly. Since we don't use GPS or other external positioning systems, we can't acquire position information directly. Solutions for x_l, y_l, and ψ estimation are obtained by using images once the 'H'landing sign is recognized.

After we bound the 'H' sign with a minimum area of rectangular by calling the OpenCV library function 'minAreaRect', which is the last step when the 'H'landing sign is recognized in the image processing, the angle between this rectangular's long side and the image's long side is also provided in the output of the function 'minAreaRect'. According to the 2D chart shown in Fig. 2.5, this angle is between 'H'landing sign's vertical axis and the x_B axis of body frame. We denote the angle as γ, therefore γ is the relative heading angle between the drone and the landing platform. We denote the angle of 'H'landing sign's vertical axis and north direction (x_L axis) as δ, so δ is the setting angle of the landing platform respect to the inertial frame. Then we have

$$\psi = \gamma + \delta \tag{2.26}$$

In our experiments, we set $\delta = 0$, and then the relative heading angle γ is equal to the yaw angle ψ. We note that since the 'H' shape used for the landing pad is symmetric, the

heading estimation will have 180 degrees of uncertainty. In real applications, we shall use other information to eliminate that uncertainty. For example, when approaching from the long range, the heading direction of the ship could be determined, based on which a unique heading angle γ can be determined.

For position vector $\overrightarrow{p_L}(x_l, y_l, z_l)$ estimation, we have

$$\overrightarrow{p_L} = C_B^L \overrightarrow{p_B} \tag{2.27}$$

Once we obtain the values of ϕ, θ, and ψ, rotation matrix C_B^L is determined. Combining equation (2.27) with equations (2.11) and (2.12), we rewrite equation (2.27) as

$$\begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix} = \begin{bmatrix} C_{00} & C_{01} & C_{02} \\ C_{10} & C_{11} & C_{12} \\ C_{20} & C_{21} & C_{22} \end{bmatrix} \begin{bmatrix} -\frac{k}{f}(v - v_0) \\ \frac{k}{f}(u - u_0) \\ 1 \end{bmatrix} z_b \tag{2.28}$$

In equation (2.28), the unknown components are x_l, y_l, and z_b. By expanding equation (2.28), the third row becomes

$$z_l = \left\{ s\theta \left[\tfrac{k}{f}(v - v_0) \right] + s\phi c\theta \left[\tfrac{k}{f}(u - u_0) \right] + c\phi c\theta \right\} z_b \tag{2.29}$$

Once we acquire z_b from equation (2.29), $\overrightarrow{p_B}$ and $\overrightarrow{p_L}$ can be calculated.

2.5.3 Landing Controls

AR.Drone de-couples the controls of motors' speed into 4 independent controls of roll and pitch angles, yaw angle rate and altitude rate, while positions are controlled by controlling the roll and pitch angles. The PID controller introduced in section 2.4.4 is used.

For hovering and landing on the 'H' sign, the drone is required to stay at a position above the sign. The errors of positions are calculated respect to the body frame, and the desired position $({x_b}^d, {y_b}^d)$ is $(0,0)$. Thus the errors of positions are ${x_b}^e = -x_b$ and ${y_b}^e = -y_b$.

We also require the forward direction of the drone parallel to the 'H' sign when landing. So the desired value of γ is 0 and the error is $\gamma^e = -\gamma$.

The error of altitude is ${z_l}^e = {z_l}^d - z_l$, where the desired altitude ${z_l}^d$ is designed a prior. An altitude planner is designed to calculate ${z_l}^d$. During the hovering phase, ${z_l}^d$ is set as a constant c

(c is positive). During the landing phase, it's set to be a function of time t, such as ${z_l}^d = -qt + c$ (q is also positive, and used to control the descending rate).

Sensor data update rate of AR.Drone is 20 Hz. For each update, control commands are also sent to AR.Drone. Fig.2.20 shows the framework of the autonomous flight algorithms during the landing stage.

2.6 Experimental Results

2.6.1 Approaching From Long Range

To test the long range approach, we put the landing platform in the center of the flight area (coordinate of the landing platform in the inertial frame is $(0,0)$), and initialize the drone 2m from the landing platform with random initial heading directions. The length of our flight area is 5m, and the width is 3m. For each experiment, we choose different initial positions of the drone to take off to test the performance. Using the designed procedure presented in section 2.4.4, we did 20 independent experiments. The success rate of these experiments is 100%, and the average approach time is 20s.

Here we use one experiment to illustrate the approach stage. In this experiment, the drone takes off at the bottorm-right of the lab (Coordinate of the drone in the inertial frame is $(0.9,-2)$ when the drone starts to approach). The time $t = 0$ is the time when the red dots on the landing platform start to appear in the view of the front camera. At time $t = 16.5$, the approach completes, and the drone hovers above the landing platform.

(i) Heading Angle Estimation and Control

As discussed in section 2.4, the key factor for successful tracking and approaching to the landing is accurate heading angle estimation and control. Based on Section 2.4.3, the coordinate V in the image frame of the calculated landing sign is used to calculate the heading angle. See Equation (2.16).

Fig. 2.21 is the heading angle estimation and control results. At $t = 0$, the red dots on the landing platform start to appear in the view of the front camera. The estimated heading angle is about 15°, and the actual heading angle obtained by the Vicon system is 22.5°.

After that, the heading angle PID controller drives the heading angle to 0°, and tries to keep the heading around 0°. We notice that at 4 seconds and 11 seconds, there are two heading angle peaks. However, the heading angle drift will not cause approach failure. As presented in section 2.4.4, the drone only change its pitch angle to approach when the heading angle is small (within $\pm 5^\circ$). If the absolute heading angle is larger than 5°, the drone will first suspend approaching and correct the heading angle.

Fig. 2.22 shows the calculated coordinate V of the landing platform in the image frame during the approach. We noticed that the trend in Fig. 2.22 is similar to the estimated heading angle in Fig. 2.21, which is because the estimated heading angle is calculated using equation (2.16).

We define the estimation error as the difference between the ground truth and the estimated value during the approaching stage. In this experiment, the average estimation error during this flight is 1.03°, the standard deviation of the estimation error is 1.91°, and the maximum absolute error estimation is 8.35°. We conclude that our heading angle estimation algorithm is suitable. According to the ground truth, after the yaw angle becomes stable at 3s to the end of the test, the average heading angle is -1.29°, the standard deviation of the heading angle is 3.04°, and the maximum absolute heading angle is 6.49°. We conclude that our heading angle control algorithm is suitable.

(ii) Altitude Change During Approach

The altitude of the drone is designed to descend during the approach process. As presented in section 2.4.4, the initial hovering altitude is 2 meters, and for hovering above the landing sign, the altitude is 1 meter. The advantage of altitude descending is to keep the landing platform in the FOV of the front camera. Fig. 2.23 and Fig. 2.24 show the experimental results of the altitude change during the approaching.

From Fig. 2.23, we see that the drone starts to descend at 11s, and stops descending at 14s. After 14s the drone keeps its altitude at 1m, and the oscillation around 15s is the overshoot of the altitude PID controller. From Fig. 2.24, we see that when the drone finds the red dots sign, the initial calculated U coordinate of the sign in the image frame is 120. During the approach process, the U coordinate increases, and reaches

400 at 11s. If the altitude stays at 2 meters, and drone continues approaching, the U coordinate will continue increasing. Thus, the landing platform will be out of the FOV of the front camera before the drone hovers above it. Because at 11s, the altitude starts to descend, the U coordinate starts to decrease. The altitude stops decreasing at 14s, and the U coordinate also stops descending. After that, the U coordinate increases again, and reaches 570. At this moment, the sign can be seen in the bottom camera of AR.Drone, the algorithm switch to the bottom camera, the approaching stage ends, and the landing stage begins.

(iii) Approaching Trajectory

Fig. 2.25 and Fig. 2.26 show the trajectory of the approaching. The position is captured by Vicon system. In Fig. 2.25, we can see that the initial position of the AR.Drone is $(0.9,-2)$ in the inertial frame, and reaches $(0,0)$ when the approach process ends. We noticed that at $t = 11s$, the drone suspends to approach, which is because the heading angle is more than 5°. We can see that the coordinate keeps at $(0.2,-0.6)$ from 11s to 12s. A 2D trajectory is plotted in Fig. 2.26.

2.6.2 Experiment of Autonomous Hovering

(i) Hover Above a Stationary Platform

In this experiment, the landing platform is stationary and AR.Drone is required to hover above the 'H'landing sign autonomously. Fig. 2.27 shows the autonomous hovering experiment results. The green dash is the ground truth (data from Vicon motion capture system) and the red dot is the estimated results. Table 2.1 is the summarized statistical results of the estimation errors during 60 seconds. Positions are given in cm, and Euler angles are in degrees.

Table 2.1: Evaluation of estimation performance when landing platform is stationary.

	$x_l(cm)$	$y_l(cm)$	$z_l(cm)$	$\phi(^\circ)$	$\theta(^\circ)$	$\psi(^\circ)$
Average error	-0.05	-0.44	1.07	0.13	-0.02	-0.28
Error standard deviation	0.95	1.01	0.82	0.22	0.24	0.63
Max. absolute error	2.44	2.95	4.21	1.12	0.99	2.05

From the test results, we can see that the average error of the position estimation is less than 2cm, and the average error of the Euler angle is less than 0.3°. The standard deviation of the position estimation error is around 1cm, and the standard deviation of the Euler angle estimation error is less than 0.7°. The maximum absolute position estimation is less than 5cm, and the maximum absolute Euler angle estimation error is less than 3°. We can also see that the problem of yaw angle drift is eliminated once it is estimated from the image.

(ii) Hover Above a Moving Platform

In this experiment, the landing platform has pose and heave motion while the drone hovers above the 'H'landing sign autonomously. Fig. 2.28 shows the test results. The green dash is the ground truth (data from Vicon motion capture system) and the red dot is the estimated results. Table 2.2 is the summarized statistical results of the estimation errors. Positions are given in cm, and Euler angles are in degrees.

Table 2.2: Evaluation of estimation performance when landing platform is moving.

	$x_l(cm)$	$y_l(cm)$	$z_l(cm)$	$\phi(^\circ)$	$\theta(^\circ)$	$\psi(^\circ)$
Average error	2.75	-1.30	-1.74	-0.14	0.15	-2.14
Error standard deviation	1.34	1.21	3.15	0.23	0.35	1.82
Max. absolute error	7.43	4.72	8.55	1.73	1.56	2.73

From the test results, we can see that the average error of the position estimation is less than 3cm, and the average error of the Euler angle is less than 3°. The standard deviation of the position estimation error is less than 3cm, and the standard deviation of the Euler angle estimation error is less than 2°. The maximum absolute position estimation is less than 9cm, and the maximum absolute Euler angle estimation error is less than 3°.

Comparing with the results on the stationary landing platform, we observe two aspects. First, oscillation of the altitude is larger, which is due to the heave movement of the landing platform. The ultrasonic altimeter measures the distance from the drone to the surface of the platform. Second, because of the movement of the landing platform, the performance of hovering this time is slightly poorer compared to the test when the landing platform is stationary. The average error of both position estimation and Euler angle

estimation, the error standard deviation of both position estimation and Euler angle estimation, and the maximum absolute error of both position estimation and Euler angle estimation are slightly larger.

2.6.3 Experiments of Autonomous Landing

The difference between landing and hovering is that the desired altitude remains constant during hovering but changes during landing. In landing, an altitude planner will generate the sequence of desired altitudes, and the PID controller will drive the drone to track the reference. At the same time, position and yaw angle PID controllers are also taking actions. Fig. 2.29 shows the altitude when landing. In Fig. 2.29, blue dots represent the actual altitude, and red dots represent the desired altitude generated by the altitude planner. Before 25.4 s, the drone is hovering with oscillation due to the moving platform. The landing starts at 25.4 s and ends at 29.8 s. From 25.4 s to 29.4 s, the actual altitude is seen to follow the desired altitude. At 29.4 s, the actual altitude drops to 26 cm, and 'H' sign is no longer completely shown in the image, and the landing function of AR.Drone is called. The drone lands on the platform at 29.8 s, and after that it moves with the platform.

We further evaluate the landing accuracy through statistical tests. We first test on a stationary platform, and then test on a moving platform. For each situation, we did 20 independent experiments. In all experiments, the drone finds the landing platform and lands on it autonomously. Fig. 2.30 and Fig. 2.31 show the landing accuracy results.

Fig. 2.30 is the landing position results when the landing platform is stationary. The red circle is the center of 'H', and the blue stars are the landing positions from the trials. After AR.Drone lands on the platform, the distance from the body frame center to the center of 'H' sign is measured. The average distance error is 7.9 cm, and the standard deviation of the landing error is 2.64 cm. If we define success landing as the case that the drone completely lands on the moving platform, our landing success rate is 100%. Fig. 2.31 is the landing position results when the landing platform is moving. The average distance error is 9.0 cm, and the standard deviation of the landing error is 4.79 cm.

Compare with the accuracy when the landing platform is stationary, this result is slightly poorer because of the movement of the landing platform, but the landing success rate is still

100%. Compare with [51], which did not define clearly the success landing, the average distance error is 12 cm under no environment disturbance, and the maximum success landing rate is reported as 90%, our results are significantly better.

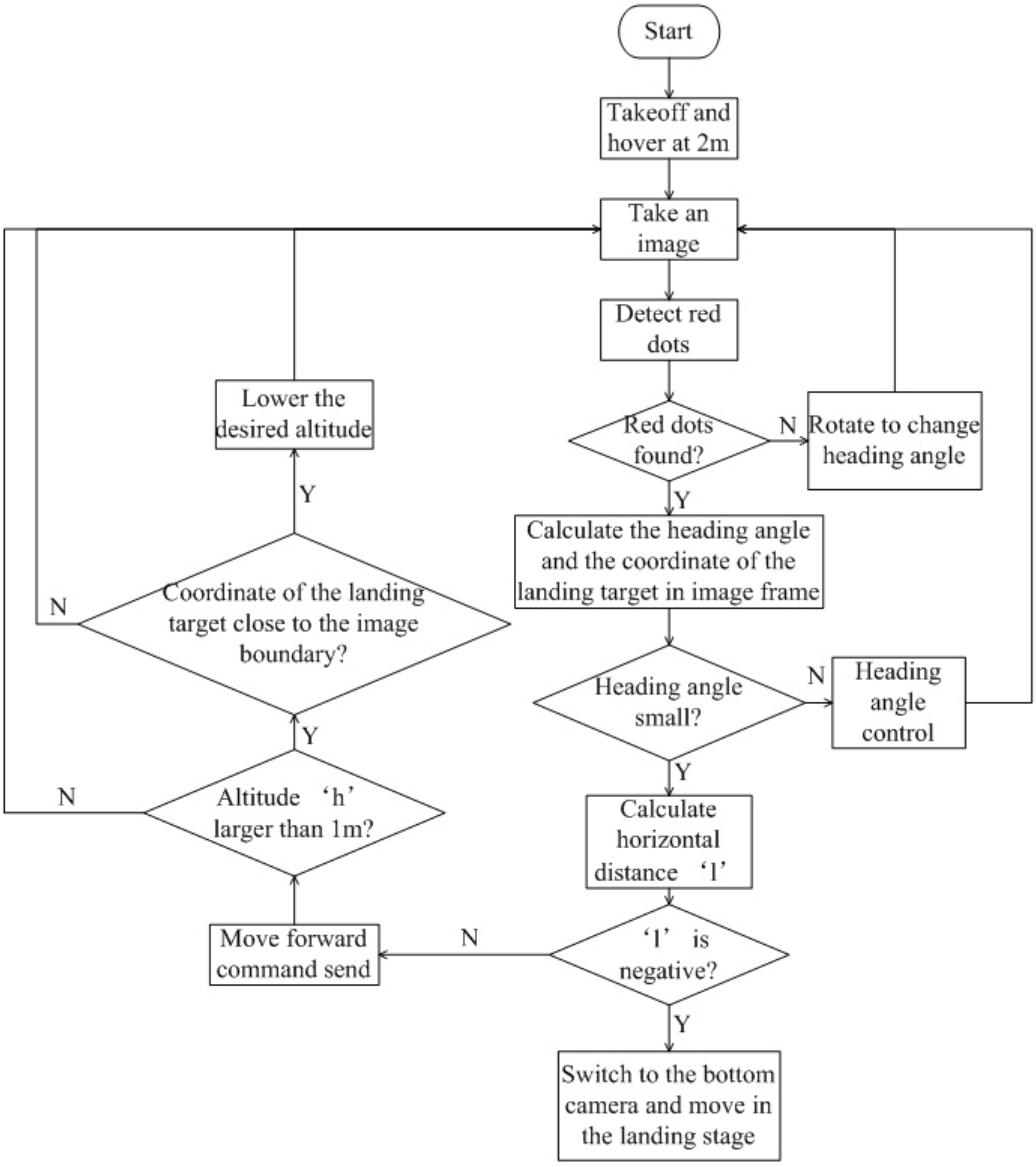


Figure 2.14: Flow Chart Illustration

Figure 2.15: Reference image.

Figure 2.16: Contour of a standard H.

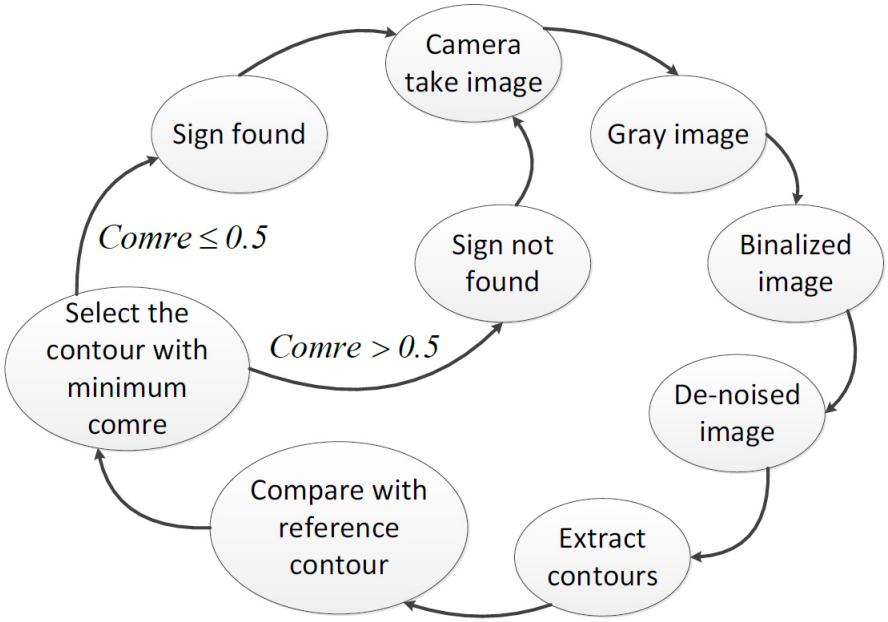


Figure 2.17: State machine of the image processing.

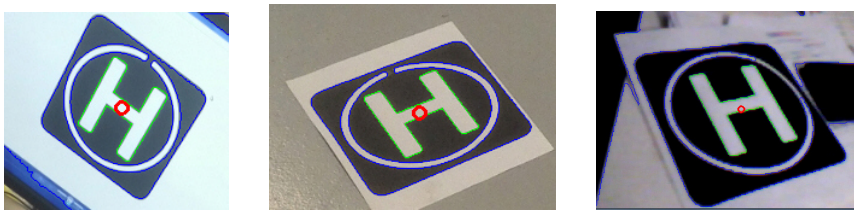
(a) Original color image from the bottom camera.

(b) Gray image.

(c) Binarized image with de-noising.

(d) 'H'landing sign detected.

Figure 2.18: Operation results during the image processing.



(a) Bright light condition; Smart-phone camera; Medium distance.

(b) Medium light condition; Smart-phone camera; Medium distance.

(c) Medium light condition; AR.Drone bottom camera; Short distance.

(d) Dark light condition; AR.Drone bottom camera; Medium distance.

(e) Dark light condition; AR.Drone bottom camera; Medium distance.

Figure 2.19: 'H' sign tracking algorithm robustness test.

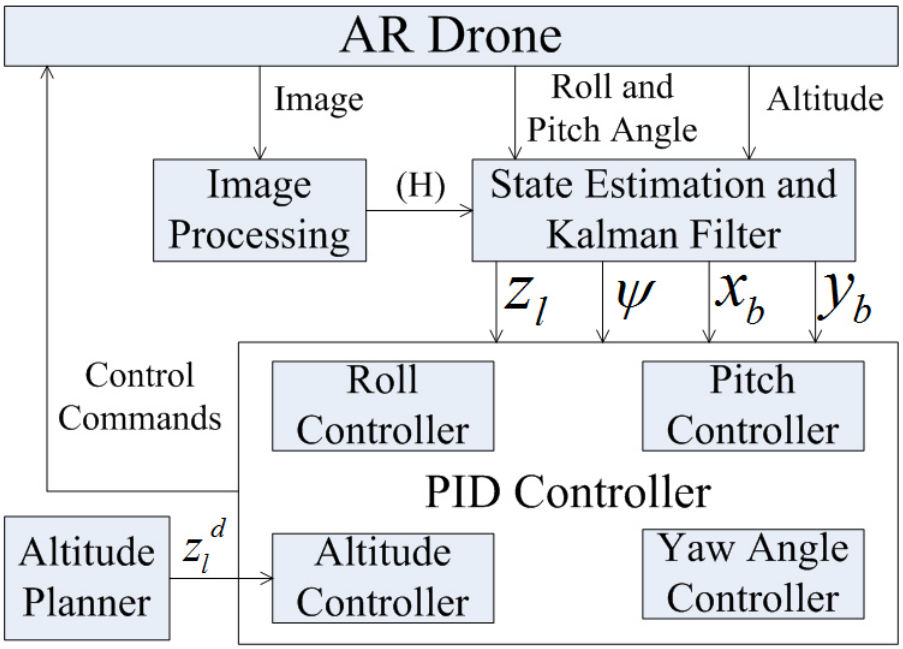


Figure 2.20: Framework of the autonomous landing algorithms.

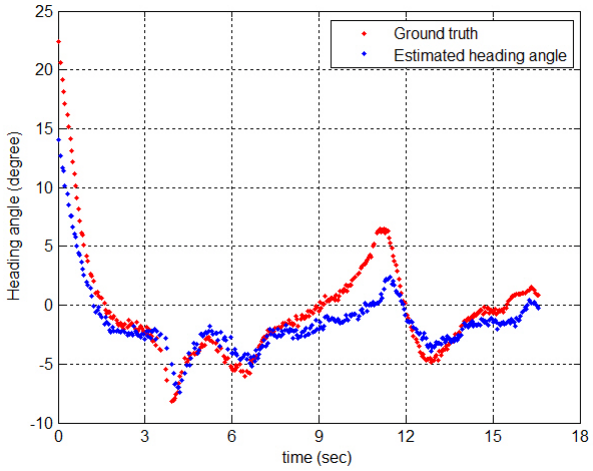


Figure 2.21: Heading angle estimation and control results.

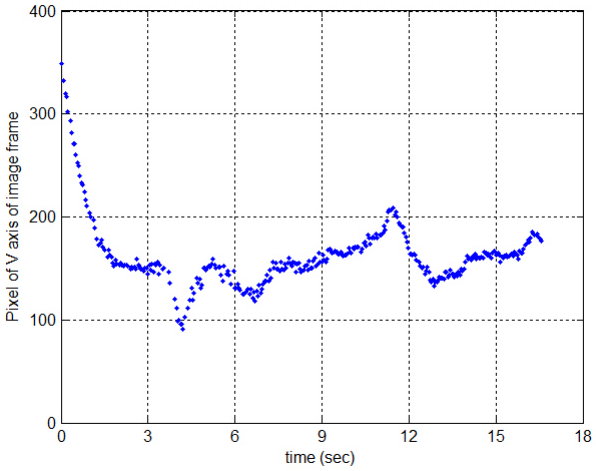


Figure 2.22: Coordinate 'V' of the landing platform in image frame.

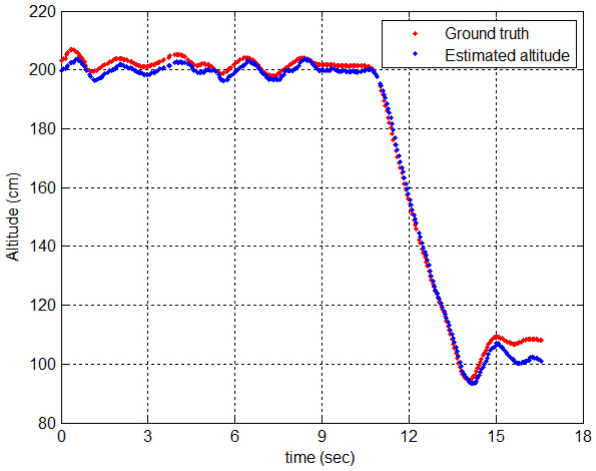


Figure 2.23: Altitude estimation and control results.

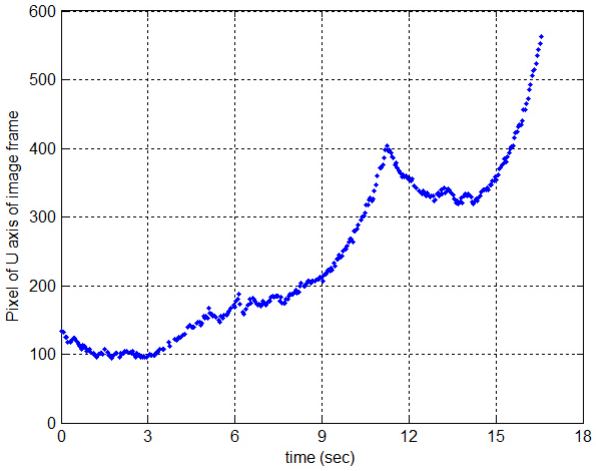


Figure 2.24: Coordinate 'U' of the landing sign in image frame.

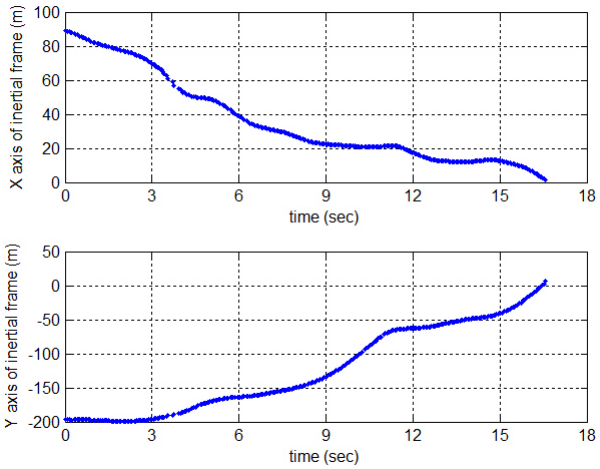


Figure 2.25: Position change during the approaching.

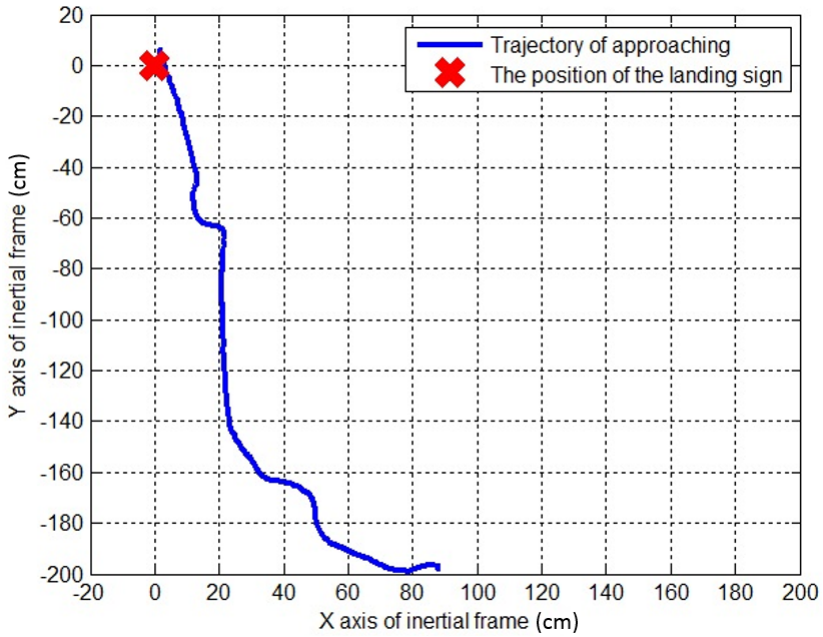


Figure 2.26: Trajectory of the approaching.

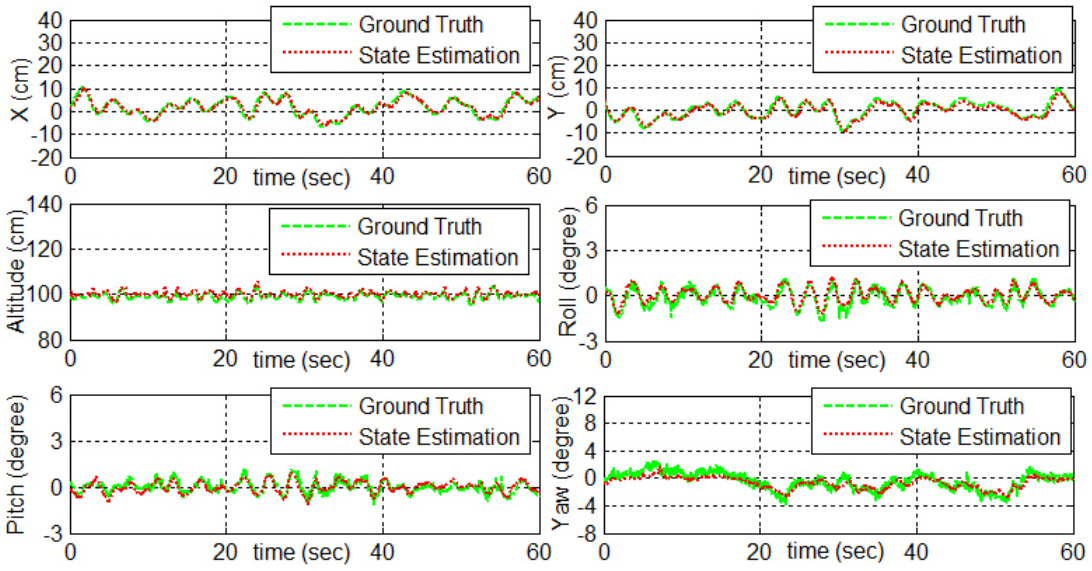


Figure 2.27: Hover test results when landing platform is stationary.

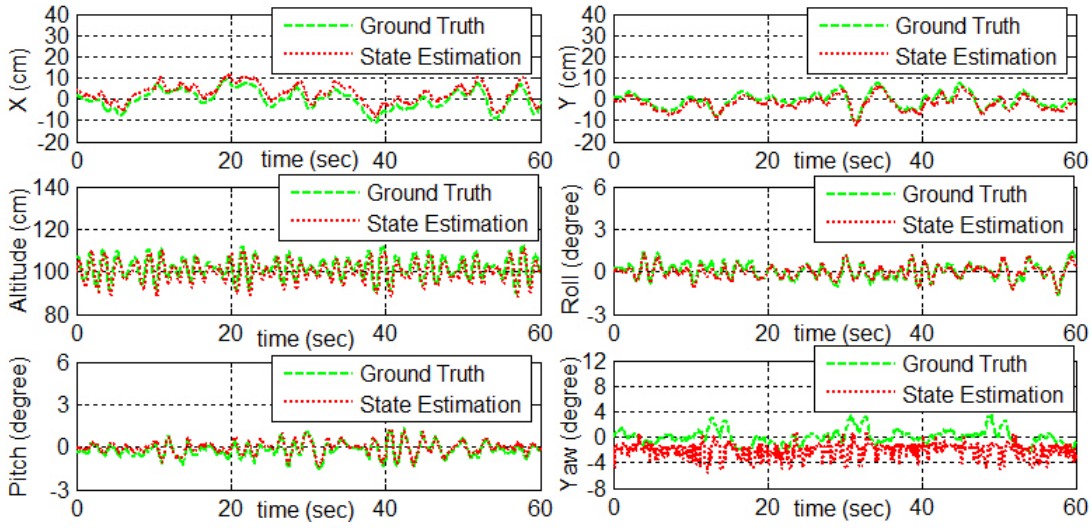


Figure 2.28: Hover test results when landing platform is moving.

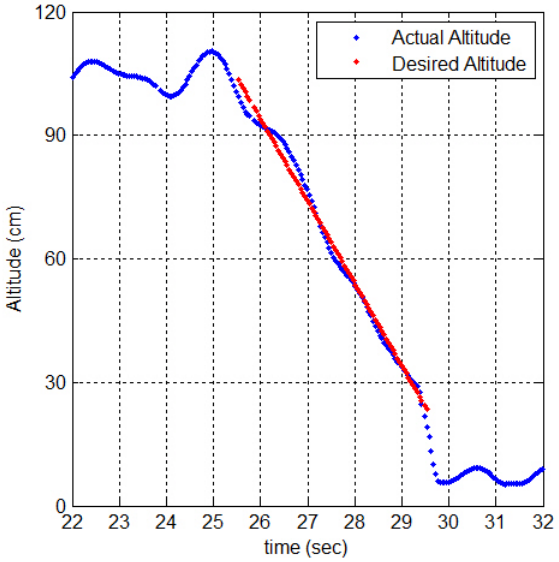


Figure 2.29: Altitude landing performance.

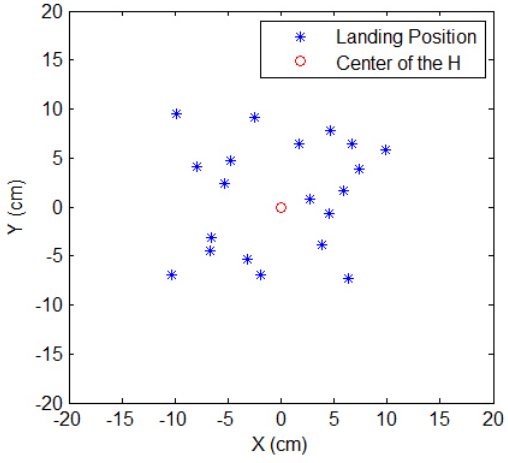


Figure 2.30: Position accuracy evaluation when the landing platform is stationary.

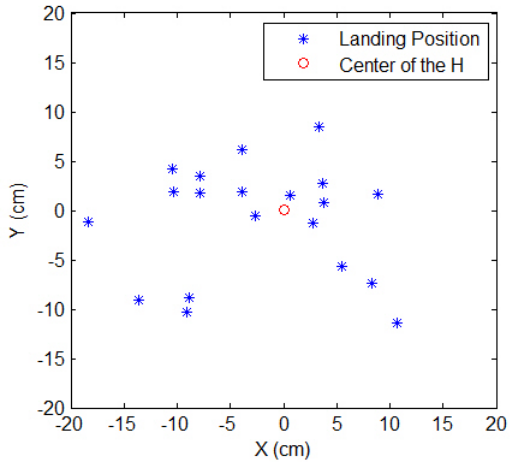


Figure 2.31: Position accuracy evaluation when the landing platform is moving.

Chapter 3

A New Approach for Visual Monocular 3-D SLAM for Small UAVs

3.1 Introduction

In Chapter 2, an on-board monocular vision based solution that provides a quadrotor with the capability to autonomously track and land on a vessel deck platform with simulated high sea state conditions is presented. To use this method, a known pattern, such as the 'H' sign, is required. However, for many VTOL UAV landing tasks such as earthquake rescue [56], the landing environment is unknown. So a navigation technique which requires no prior knowledge for environment is needed.

Over the past few years, the Simultaneous Localization and Mapping (SLAM) technique has received a lot of attention. In general, SLAM algorithm can be divided into appearance-based methods and feature-based methods. Appearance-base methods is also called direct methods. This type of method, such as LSD-SLAM [57], estimates motion and shape from each pixel of the image by minimizing an error measure such as brightness or brightness-based cross-correlation [58]. However, the performance of direct methods can be affected by lighting conditions or non-Lambertian surfaces. Compare to direct methods, feature-based methods is more robust since there is a wide choice of feature detectors.

Due to physical limitations, such as payload limited space of some applications, SLAM algorithms using only one camera as the vision source (monocular SLAM) is required. First monocular SLAM was developed in 2003 by Davison et al [59] and they named it MonoSLAM. MonoSLAM estimate the camera motion and generate map using an extended Kalman filter (EKF). The state vector of EKF includes 2 parts. The first part is the state of the camera such as position and attitude. The second part is the state of feature points. Since new feature points are added to the state vector depending on camera movement, the size of the EKF

could become very large and difficult to achieve real-time computation [60]. Another limitation of MonoSLAM is that the map needs to be initialized by using a known object. In 2007, G Klein et al split the tracking and the mapping algorithms into two different threads on processor [61]. They call this method Parallel Tracking and Mapping (PTAM). The frequency of tracking thread run faster than mapping thread, which significantly reduced computational cost and PTAM is able to handle large number of feature points. In PTAM, the initial map is reconstructed using five point algorithm [62]. Based on PTAM, R Mur-Artal et al developed ORB-SLAM in 2015 [63]. Compare to PTAM, ORB-SLAM includes multi-threaded tracking, mapping, and closed-loop detection, and the map is optimized using pose-graph optimization [60]. Since ORB-SLAM is an open source project, many applications are built based on it [61].

For all feature-based SLAM algorithms, pin-hole camera model is the basis to estimate camera movement and feature points. For bearing only monocular SLAM, when using only the camera, the scale factor of pin-hole camera model is not observable [64]. To restore the scale factor, measurements of two feature points or the distance from the camera to a feature point is required [24]. However, for applications working in unknown environment, none of the condition is satisfied. Many researchers use Inertial Measurement Unit (IMU) or other sensors which can provide metric measurements to help determine the the scale factor. There are 3 kinds of approaches are commonly used. The first method is an online spline fitting approach. Jung and Taylor adapted this method by fitting a second order spline into a set of several key frames obtained by the SLAM algorithm with monocular vision [65]. The second method directly sets the scale factor as a state in the state vector for EKF estimation. In [66], Gabriel NÃijtzi et al use this method and the result shows that the scale factor estimation converging time is 15s. The third method is called maximum likelihood approach, which is first proposed by Jakob Engel et al [67]. This method choose a likelihood function which represents the error of position estimation. By minimizing the function, global optimal solution of the scale factor can be calculated. In [68], Gehrig Daniel et al apply ORB-SLAM on a commercial quadrotor Parrot AR.Drone, and use the maximum likelihood approach to solve scale factor. In their work, the scale factor estimation converging time is also 15s.

In this Chapter, a new approach to on-board sensor only feature based monocular 3-D

SLAM algorithm is presented. It is able to navigate in previously unknown environments at absolute scale without requiring artificial markers or external positioning aid. Only a downward looking camera and an IMU on-board is required to complete this method. One of the key contributions of this Chapter is the proposed method apply the pin-hold camera model in an alternative prospective and derive a SLAM algorithm in close form. This SLAM algorithm inherently combine the data from the camera and the IMU together. The scale factor problem does not exist in this approach, which gives absolute estimation of the camera position as well as the map immediately when start to work. Another contribution of this Chapter is the proposed SLAM algorithm requires no EKF or other optimization methods adopted by PTAM and ORB-SLAM. So the computation cost of the proposed method is significantly lower than MonoSLAM, PTAM and ORB-SLAM. The third contribution of this work is that the map generation requires no initialization, which makes it easy to use. This work also use the AR.drone and Vicon system described in Section 2.2. The coordinate definitions are also defined in Section 2.2.

The organization of this Chapter is as follows. In Section 3.2, the proposed SLAM algorithm is derived. In Section 3.3, a practical realization of the proposed SLAM algorithm is introduced in detail. In Section 3.4, the experimental results is discussed.

3.2 A New SLAM Algorithm

Currently, pinhole camera model [69] is the basis for all monocular feature based SLAM algorithms. The equation of pinhole camera model can be present as

$$w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a & s & u_0 \\ 0 & b & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.1}$$

Also as

$$w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K\begin{bmatrix} R & t \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.2}$$

Where matrix K represents the parameters of the camera, matrix R represents the rotations of the camera, and vector t represents the position movements of the camera.

From equation 3.2 we can see that the model is a 3-D to 2-D transformation, which transfer a point in inertial frame N directly into image frame I. And we notice that the parameter w is a scale factor and it's not a constant. That parameter is unable to determine just use images from camera. So when mapping, current SLAM algorithms suffer from the dimensionless problem.

Compare equation 3.2 with equation 2.13, we can see that equation 2.13 is a 2-D to 2-D transformation, which transfer a point in image frame I into body frame B. The derivation of equation 2.13 is also based on the pinhole camera model, but it consider from inverse. In equation 2.13, all parameters are related to the camera itself, and all of them are constant. Because the definition of α and β, from equation 2.14, the scale factor is the z_b of that point. If we can determine z_b, then we can find the coordinate of a point in body frame B. By using matrix C_B^L and C_L^N, we can find the coordinate of a point in inertial frame N.

In this work, we describe the position of the drone and the position of the map points in inertial frame N. For the map, a feature's position in frame N, L, B, and I are represented as $\begin{bmatrix} x_n & y_n & z_n \end{bmatrix}^T$, $\begin{bmatrix} x_l & y_l & z_l \end{bmatrix}^T$, $\begin{bmatrix} x_b & y_b & z_b \end{bmatrix}^T$, and $\begin{bmatrix} u & v \end{bmatrix}^T$, respectively. The drone's position in N is is represented as $\begin{bmatrix} x & y & z \end{bmatrix}^T$. The drone's position located in frame L and B is at the origin. At the same time, there is no definition of the drone's position in image frame I.

From the definitions and Fig. 2.3, we have

$$\begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix} = C_L^N \begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.3}$$

We also have

$$\begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix} = C_B^L \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix} \tag{3.4}$$

If we combine equation (3.3) and (3.4), we have

$$\begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix} = C_B^N \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.5}$$

As other monocular feature based SLAM, we assume that the objects in the environment will stay still. So the features' coordinates in frame N does not change. Suppose there are two moments 1 and 2, and the drone get an image for both moments. The drone 's position of the two moments are $\begin{bmatrix} x_1 & y_1 & z_1 \end{bmatrix}^T$ and $\begin{bmatrix} x_2 & y_2 & z_2 \end{bmatrix}^T$. At the same time, a feature, which the coordinate in N is $\begin{bmatrix} x_n & y_n & z_n \end{bmatrix}^T$, shows up in both images, and the feature's coordinate in frame B are $\begin{bmatrix} x_{b1} & y_{b1} & z_{b1} \end{bmatrix}^T$ and $\begin{bmatrix} x_{b2} & y_{b2} & z_{b2} \end{bmatrix}^T$. From equation (3.5), we have:

$$C_{B\,1}^{N} \begin{bmatrix} x_{b1} \\ y_{b1} \\ z_{b1} \end{bmatrix} + \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = C_{B\,2}^{N} \begin{bmatrix} x_{b2} \\ y_{b2} \\ z_{b2} \end{bmatrix} + \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} \tag{3.6}$$

If we define the drone's position change $\begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{bmatrix} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix}$, combine with equation (2.14), we have:

$$C_{B\,1}^{N} z_{b1} \begin{bmatrix} \alpha_1 \\ \beta_1 \\ 1 \end{bmatrix} - C_{B\,2}^{N} z_{b2} \begin{bmatrix} \alpha_2 \\ \beta_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} \tag{3.7}$$

From the definitions of frame N, L and B, we also have:

$$c\phi_2 c\theta_2 z_{b2} - c\phi_1 c\theta_1 z_{b1} = \Delta z \tag{3.8}$$

Take equation (3.8) into equation (3.7), we have:

$$z_{b1} \left(C_{B\,1}^{N} \begin{bmatrix} \alpha_1 \\ \beta_1 \\ 1 \end{bmatrix} - \frac{c\phi_1 c\theta_1}{c\phi_2 c\theta_2} C_{B\,2}^{N} \begin{bmatrix} \alpha_2 \\ \beta_2 \\ 1 \end{bmatrix} \right) - \frac{\Delta z}{c\phi_2 c\theta_2} C_{B\,2}^{N} \begin{bmatrix} \alpha_2 \\ \beta_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} \tag{3.9}$$

In equation 3.9, Euler angles of moment 1 and moment 2 can be acquired from the IMU on-board, and $C_{B\,1}^{N}$ and $C_{B\,2}^{N}$ can be calculated. At the same time, $\begin{bmatrix} \alpha_1 & \beta_1 & 1 \end{bmatrix}^T$ and $\begin{bmatrix} \alpha_2 & \beta_2 & 1 \end{bmatrix}^T$ can be determined by feature matching algorithms and equation 2.13. If we know $\begin{bmatrix} \Delta x & \Delta y & \Delta z \end{bmatrix}^T$ by using velocity measurements in inertial frame, as:

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} \frac{\Delta t}{2}(v_{x1} + v_{x2}) \\ \frac{\Delta t}{2}(v_{y1} + v_{y2}) \\ \frac{\Delta t}{2}(v_{z1} + v_{z2}) \end{bmatrix} \tag{3.10}$$

where Δt is the time period from moment 1 to moment 2, then the only unknown term in equation (3.9) is z_{b1}, and it can be solved. Assume we already know $\begin{bmatrix} x_1 & y_1 & z_1 \end{bmatrix}^T$, using equation (3.5), we can calculate $\begin{bmatrix} x_n & y_n & z_n \end{bmatrix}^T$, which is the feature's coordinate in frame N. Using equation (3.5) again with the data of second moment, we can calculate $\begin{bmatrix} x_2 & y_2 & z_2 \end{bmatrix}^T$, which is the drone's position of the second moment. Fig. 3.1 shows the flow chart of the proposed SLAM algorithm.

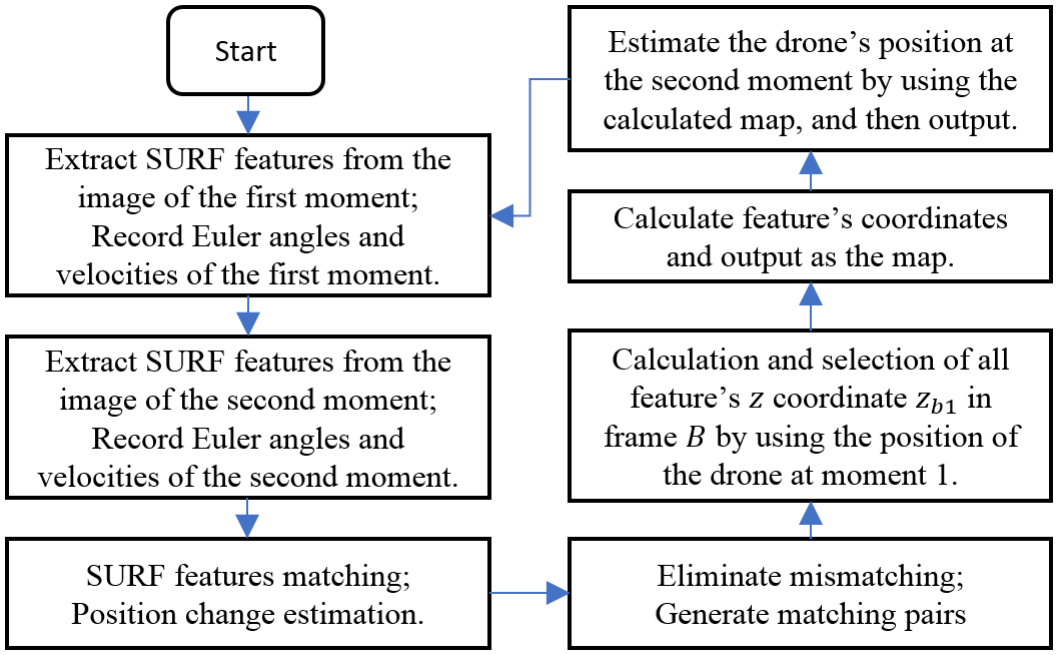


Figure 3.1: Flow chart of the proposed SLAM algorithm.

3.3 System Realization

3.3.1 ROS Packages

ROS (Robot Operating System) provides libraries and tools for robot applications. In this work we develop all algorithm using ROS environment. The version of ROS is 'kinetic', and it works under Ubuntu 16.04.

The main algorithms are distributed in 4 packages. The first package is a control package. We use PID controllers in experiments. The second package is a synchronization package. Because the frequency of image topic update from camera is 10 Hz, the frequency of Euler

angles and velocities topics update from IMU are 50 Hz, while frequency of the position and Euler angles estimations from VICON are 200 Hz, a ROS synchronization tool is used to align them. The third package is a image processing package. In this package, algorithms extract SURF features and match between adjacent images. Then, mismatching is also identified. The last package includes the main algorithms of the proposed SLAM algorithm. It gives the results of the drone's position estimation and the map points cloud.

3.3.2 PID controller

In this work, 4 PID controllers are used to control the AR.Drone. The controller design is described in Section 2.4.4. The 4 PID controllers are tuned under no wind condition. We use the Vicon data for feedback, set one point as the desired position, and tuned the gains P, I, and D to hover the drone at the point. The control cycle's frequency is 10Hz, and the unit of x, y and z error is in meters, the unit of yaw angular velocity is in degree per second. The result's of the tuned gains used are listed in table 3.1.

Table 3.1: PID gains used.

	P	I	D
For channel 1	0.6	0.0001	8.0
For channel 2	0.6	0.0001	8.0
For channel 3	0.08	0.0001	0.06
For channel 4	4.0	0.01	8.0

After the gains are tuned, the values are used for later experiments. The controls are sent to AR.Drone through the SDK provided by the manufacturer. The mode of AR.Drone is set as non-hover mode, so the controls to the drone are purely from the controllers designed.

3.3.3 Synchronization

To synchronize information from different sensors, 'message filters' library of ROS is used. The function takes in messages of different types from multiple sources, and outputs them only if it has received a message on each of those sources with the same time stamp [70], where time stamp is the internal time representation under ROS environment. In this work, the synchronized information include the image flow from camera, the Euler angles and velocity

in frame B estimation from IMU, and the Euler angles and drone's position in frame N from VICON. We name the synchronized data structure 'Sinfo', and list the details of Sinfo in table 3.2. After synchronize information from different sensors, the synchronized data structure (Sinfo) updates in 2 Hz.

Table 3.2: Members of Sinfo.

Name	symbol	ROS Massage Type
Time Stamp Header	header	std_msgs/Header
Image	image	sensor_msgs/Image
IMU Measurements	imu	sensor_msgs/Imu
Navigation Information	navdata	ardrone_autonomy/Navdata
Commands to AR.Drone	cmd	geometry_msgs/Twist
Position Data from VICON	vcnposi	geometry_msgs/Vector3
Euler angles from VICON	vcnangl	geometry_msgs/Vector3

3.3.4 Image processing

ROS kinetic has OpenCV library integrated and available to use in project. In this section, the OpenCV functions are used for SURF feature extraction and feature matching. We take [71] as our coding reference. From experimental observation, there are about thirty percent of mismatching. A matching selection algorithm is developed to eliminate mismatching.

(i) SURF Feature Extraction SURF feature is used for tracking in this work. For SURF feature extraction, a object of SURF handle is generated first, using 'SURF::create()' function. Then, a threshold is set by using 'setHessianThreshold()' method. We pick 60 as the minimum Hessian. After that, 'detect()' method is used to create key points, which is the SURF feature points. Usually, more than 100 SURF feature points will be created in this step. To reduce computation cost, we only pick the first 60 key points for later operation. If there are less than 60 SURF feature points created, the program will mark no valid image source for this step.

(ii) Feature Matching After SURF feature points created, 'compute()' method is used to calculate descriptors of the feature points. For convenience, we name the image at moment 1 as Base Image (BI), and name the image at moment 2 as Search Image (SI). At each step, the key points and corresponding descriptors of BI is copied from the data of SI at

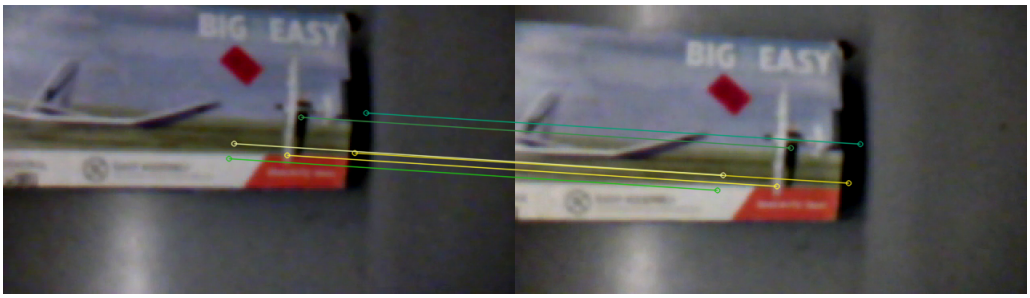
the previous step. At the same time, a feature calculation (including feature extraction and descriptors calculation) is conducted for SI. So there is only one feature calculation for each step.

When key points and the corresponding descriptors of both BI and SI are ready, a handle for matching is created and 'match()' function is used to generate one-to-one correspondence of the key points belong to BI and SI. When matching pairs are generated, the coordinates of a feature point in frame I for both BI and SI are stored with a quality parameter. The matching is based on the comparison of the descriptors' vector, and the quality parameter is the distance of two vectors. The smaller the distance, the better correspondent of the two key points. We set the threshold of the distance to be 0.2. If the distance less than 0.2, the matching pair is considered as a valid matching pair. From our experiment results, the number of the valid matching pairs is usually more than 30. If there are more than 30 valid matching pairs, we sort the results by the quality parameter, and pick 30 best matching pairs. If there are less than 30 valid matching pairs, we mark no valid image source for this step.

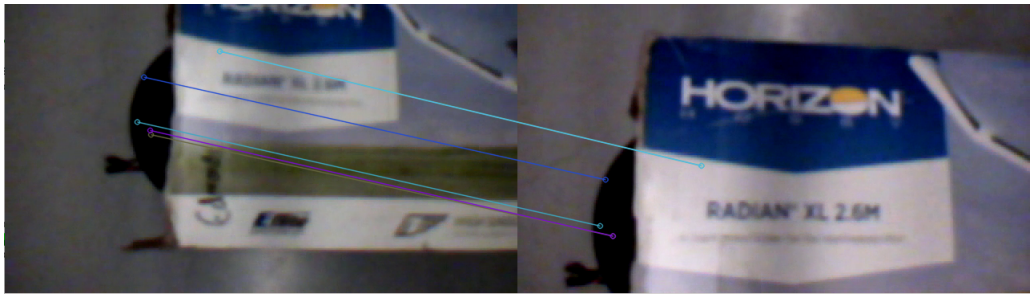
(iii) Elimination of Mismatching Feature tracking inevitably produces mismatching, and mismatching generate wrong measurements of the map and the drone's position. In this work, we design a matching filter to reduce the number of mismatching.

For a matching pair, we define u change as $\Delta u = u_2 - u1$, and v change as $\Delta v = v_2 - v1$. For correct matching, the Δu should be similar, and Δv should be similar as well. Based on this fact, the matching filter first sort the 30 matching pairs by Δu, and pick the middle 14 pairs. Then, sort the 14 matching pairs by Δv, and pick the middle 6 pairs. After filtering, mismatching rate is reduced significantly from about 40% to about 5%. By using the u and v information of the 6 selected matching pairs, we calculate vectors of $\begin{bmatrix}\alpha_1 & \beta_1 & 1\end{bmatrix}^T$ and $\begin{bmatrix}\alpha_2 & \beta_2 & 1\end{bmatrix}^T$ for each matching pair using equation 2.13. Fig. 3.2 shows two examples of feature matching. The image is taken from AR.Drone's bottom camera during flight. The BI is on the left, and the SI is on the right.

(iv) Organization of the Image Processor We draw a flow chart of the organization of the image processor in Fig. 3.3. We also name the data structure after image processing as



(a) Example 1.



(b) Example 2.

Figure 3.2: Two examples of feature matching.

'Iinfo', and list the details of Iinfo in table 3.3. The data structure (Iinfo) update rate is also 2 Hz.

3.3.5 Realization of the Proposed SLAM Algorithm

(i) Calculations for Features' z Coordinates in Body Frame After the image processor, SLAM module take over Iinfo topic and run the proposed SLAM algorithm. One critical challenge for SLAM module is to determine Features' z Coordinates z_{b1} in frame B. Basically, equation 3.9 can be written as

$$z_{b1}\begin{bmatrix} V_{11} \\ V_{12} \\ V_{13} \end{bmatrix} = \begin{bmatrix} V_{21} \\ V_{22} \\ V_{23} \end{bmatrix} \tag{3.11}$$

From math, we have

$$z_{b1} = \frac{V_{21}}{V_{11}} = \frac{V_{22}}{V_{12}} = \frac{V_{23}}{V_{13}} \tag{3.12}$$

However, in practice, we can not randomly take results from one of them, nor can we take average of the 3 results. Suppose a drone locate at point p_1 and p_2 at moment 1 and 2. A

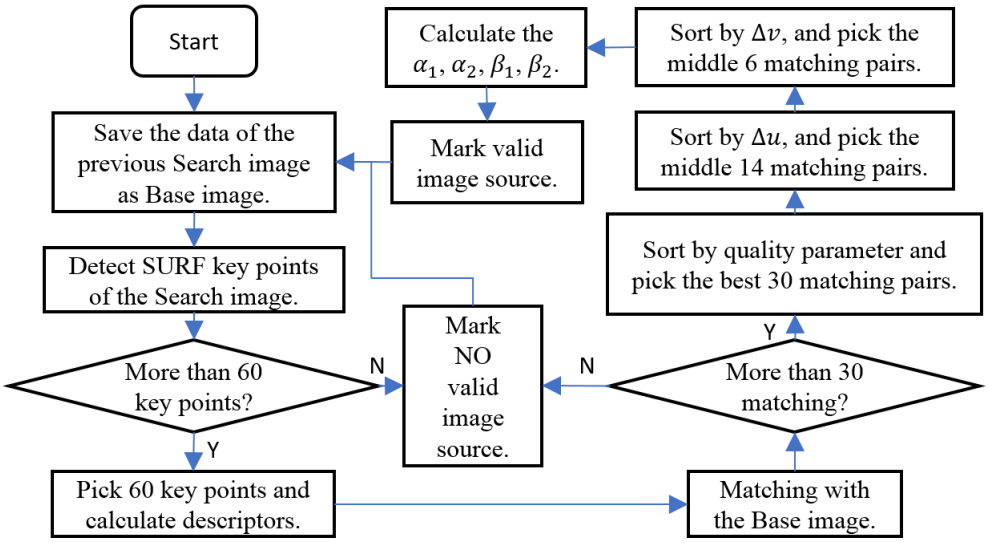


Figure 3.3: Flow chart of the organization of the image processor.

feature point locates at p_f and shows in BI and SI. At both moments, the drone's Euler angles are all zero. The movement from p_1 to p_2 is along $-x$ direction, and y and z axis of p_1 and p_2 are same. The moving distance is d. At moment 1, the drone is above the feature point and the feature's coordinate in image frame is at the image's center (u_0, v_0). The example shows in Fig. 3.4.

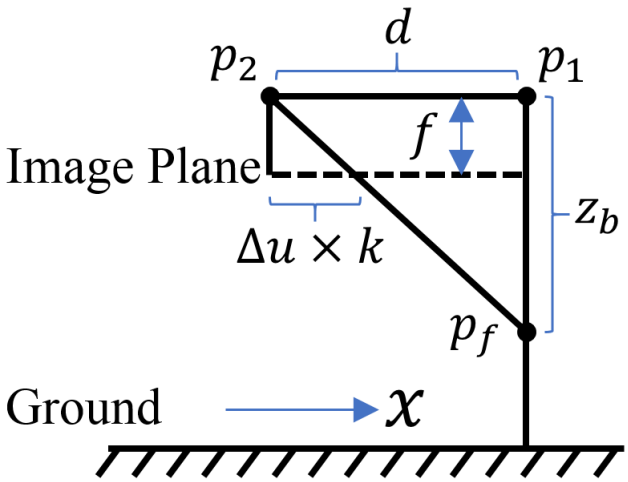


Figure 3.4: A example for calculation of z_{b1}.

Apparently, at moment 2, the feature's coordinate in image frame is $(u_0 + \Delta u, v_0)$. From equation 2.13, we have $\begin{bmatrix} \alpha_1 & \beta_1 & 1 \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$, and $\begin{bmatrix} \alpha_2 & \beta_2 & 1 \end{bmatrix}^T = \begin{bmatrix} 0 & \lambda\Delta u & 1 \end{bmatrix}^T$.

Table 3.3: Members of Iinfo.

Name	symbol	Massage Type/ROS
Time Stamp Header	header	std_msgs/Header
Valid Image Source	imgflag	bool
Calculated α_1	alpha_Base	float64[6]
Calculated α_2	alpha_Search	float64[6]
Calculated β_1	beta_Base	float64[6]
Calculated β_2	beta_Search	float64[6]
IMU data of moment 1	imudata_Base	sensor_msgs/Imu
IMU data of moment 2	imudata_Search	sensor_msgs/Imu
Nav. data of moment 1	navdata_Base	ardrone_autonomy/Navdata
Nav. data of moment 2	navdata_Search	ardrone_autonomy/Navdata
Commands of moment 1	cmd_Base	geometry_msgs/Twist
Commands of moment 2	cmd_Search	geometry_msgs/Twist
VICON Posi moment 1	vcnposi_Base	geometry_msgs/Vector3
VICON Posi moment 2	vcnposi_Search	geometry_msgs/Vector3
VICON Euler moment 1	vcnangl_Base	geometry_msgs/Vector3
VICON Euler moment 2	vcnangl_Search	geometry_msgs/Vector3

We also have $\begin{bmatrix}\Delta x & \Delta y & \Delta z\end{bmatrix}^T = \begin{bmatrix}-d & 0 & 0\end{bmatrix}^T$. From equation 3.9, we have: $z_{b1}\begin{bmatrix}-\lambda\Delta u & 0 & 0\end{bmatrix}^T = \begin{bmatrix}-d & 0 & 0\end{bmatrix}^T$. We can see that in this example, only $z_{b1} = V_{21}/V_{11}$ gives us the result of z_{b1}. That is because the movement of the drone only has information in one dimension. If the drone stay at the same point at both movement, then no moving information provided and z_{b1} can not be calculated.

Because of the above reason, we design an algorithm to calculate z_{b1} based on the drone's movement. First we define $z^a_{b1} = V_{21}/V_{11}$, $z^b_{b1} = V_{22}/V_{12}$, and $z^c_{b1} = V_{23}/V_{13}$. We also set a threshold (0.05) for V_{11}, V_{12}, V_{13}, V_{21}, V_{22}, and V_{23}. For example, if V_{11} or V_{21} less than the threshold, z^a_{b1} is not calculate. If z^a_{b1}, z^b_{b1}, and z^c_{b1} are all not qualified to calculate, then no information can be calculated from this feature matching pair. If there at least one of z^a_{b1}, z^b_{b1}, and z^c_{b1} can be calculate, then z_{b1} can be determined by taking average of the valid results. Once z_{b1} be determined, using the algorithm derived in Section 3.2, the feature's coordinates and the drone's position at moment 2 can be calculated.

(ii) Organization of the SLAM module For each step, 6 feature matching pairs are pushed into the SLAM module. For each matching pair, if the corresponding z_{b1} can be calculated, a feature's coordinates and an estimation of the drone's position at moment 2

can be estimated, which means this matching pair provides information. From our experiments, normally all 6 matching pairs provide information or non of them provides information. If non of the matching pairs provide information, we mark no SLAM sign for this step. At the same time, no map will be generated, and the estimation of the drone's position at moment 2 will be calculated purely by the drone's position at moment 1 and $\begin{bmatrix}\Delta x & \Delta y & \Delta z\end{bmatrix}^T$. When all 6 matching pairs provide information, for better results, we only take 5 of them. If there are no error, all 6 results of the drone's position at moment 2 should be same. The idea is to pick the points are close to each other. For the 6 calculated drone's position, we denote them as

$$\begin{pmatrix}\begin{bmatrix}p^1\end{bmatrix} & \begin{bmatrix}p^2\end{bmatrix} & \cdots & \begin{bmatrix}p^6\end{bmatrix}\end{pmatrix} \tag{3.13}$$

And $p^i = \begin{bmatrix}x^i & y^i & z^i\end{bmatrix}^T$. Then we define the distance of two points as

$$D\begin{pmatrix}p^i, & p^j\end{pmatrix} = \sqrt{(x^i - x^j)^2 + (y^i - y^j)^2 + (z^i - z^j)^2} \tag{3.14}$$

Then, we define total distance of a point p^i as

$$d^i = \sum_{j=1}^{6} D\begin{pmatrix}p^i, & p^j\end{pmatrix} \tag{3.15}$$

With these definitions, we calculate the total distance for all 6 points, then sort the total distance from small to large. After that, we discard the largest one and keep the rest 5 ones.

Table 3.4 shows the details of the map topic under ROS environment. For estimation of the drone's position at moment 2, we use least square to calculate based on the 5 results we keep. Fig. 3.5 shows the organization of the SLAM module.

Table 3.4: Map topic definition.

Name	symbol	Massage Type/ROS
Time Stamp Header	header	std_msgs/Header
Map Points	maps	geometry_msgs/Vector3[5]

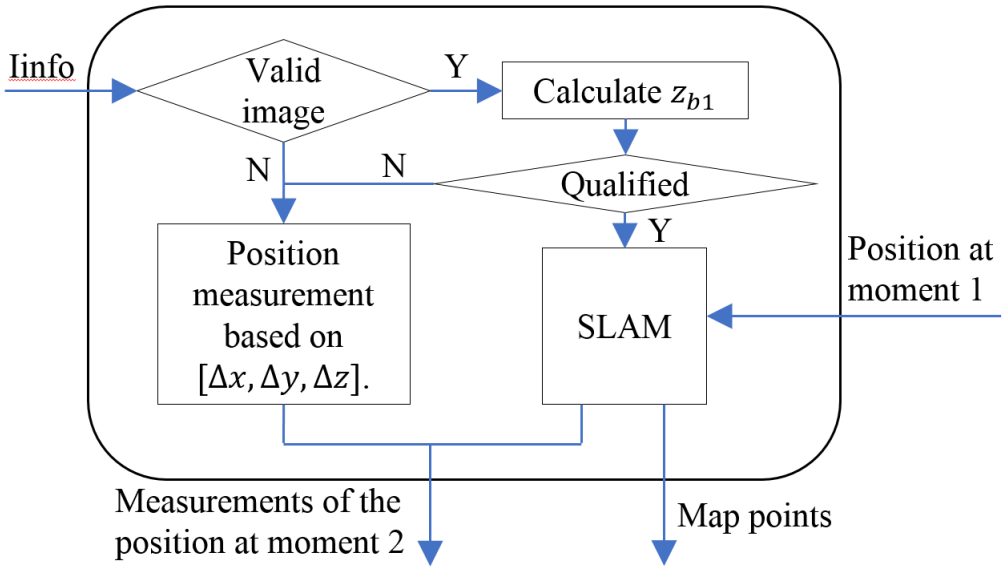


Figure 3.5: Organization of the SLAM module.

3.3.6 Structure of Entire System

(i) EKF design An EKF is used in this work. We define the state vector X as:

$$X = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \\ z \\ \dot{z} \\ \theta \\ \dot{\theta} \\ \phi \\ \dot{\phi} \\ \psi \\ \dot{\psi} \end{bmatrix} \tag{3.16}$$

Where x, y, z are the position of the drone respect to N frame, and θ, ϕ, ψ are Euler angles. The control vector U according to AR.drone is:

$$U = \begin{bmatrix} u_{\dot{z}} \\ u_{\theta} \\ u_{\phi} \\ u_{\psi} \end{bmatrix} \tag{3.17}$$

The drone have a drag force in frame B, and denote it as $\begin{bmatrix} 0 & 0 & -f \end{bmatrix}^T$, and have the gravity in frame N, and denote it as $\begin{bmatrix} 0 & 0 & -mg \end{bmatrix}^T$. Transfer the drag force to frame N, we have

$$C_B^N \begin{bmatrix} 0 \\ 0 \\ -f \end{bmatrix} = f \begin{bmatrix} s\phi c\psi - c\phi s\theta s\psi \\ -s\phi s\psi - c\phi s\theta c\psi \\ c\phi c\theta \end{bmatrix} \tag{3.18}$$

We can write $fc\phi c\theta = mg$, and $f = \frac{mg}{c\phi c\theta}$. So the total force can be present as

$$F = \begin{bmatrix} \frac{mg}{c\phi c\theta}(s\phi c\psi - c\phi s\theta s\psi) \\ \frac{mg}{c\phi c\theta}(-s\phi s\psi - c\phi s\theta c\psi) \\ 0 \end{bmatrix} \tag{3.19}$$

Let's denote the force in x direction Fx, and the force in y direction Fy. We have

$$Fx = \frac{mg}{c\phi c\theta}(s\phi c\psi - c\phi s\theta s\psi) \tag{3.20}$$

and

$$Fy = \frac{mg}{c\phi c\theta}(-s\phi s\psi - c\phi s\theta c\psi) \tag{3.21}$$

After that, we can model the system as:

$$\begin{cases} x_k = x_{k-1} + \Delta t \dot{x}_{k-1} \\ \dot{x}_k = \dot{x}_{k-1} + \Delta t \frac{Fx}{m} \\ y_k = y_{k-1} + \Delta t \dot{y}_{k-1} \\ \dot{y}_k = \dot{y}_{k-1} + \Delta t \frac{Fy}{m} \\ z_k = z_{k-1} + \Delta t \frac{1}{2}(\dot{z}_{k-1} + c_1 u^1_{k-1}) \\ \dot{z}_k = \frac{1}{2}(\dot{z}_{k-1} + c_1 u^1_{k-1}) \\ \theta_k = \frac{1}{2}(\theta_{k-1} + c_2 u^2_{k-1}) \\ \dot{\theta}_k = \dot{\theta}_{k-1} \\ \phi_k = \frac{1}{2}(\phi_{k-1} + c_3 u^3_{k-1}) \\ \dot{\phi}_k = \dot{\phi}_{k-1} \\ \psi_k = \psi_{k-1} + \Delta t \frac{1}{2}(\dot{\psi}_{k-1} + c_4 u^4_{k-1}) \\ \dot{\psi}_k = \frac{1}{2}(\dot{\psi}_{k-1} + c_4 u^4_{k-1}) \end{cases} \tag{3.22}$$

If we linearize it and write it in matrix form $X_k = AX_{k-1} + BU_{k-1}$, we have:

$$A = \begin{bmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & g\Delta t & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -g\Delta t & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \frac{\Delta t}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \frac{\Delta t}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \end{bmatrix} \tag{3.23}$$

and:

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{c_1 \Delta t}{2} & 0 & 0 & 0 \\ \frac{c_1}{2} & 0 & 0 & 0 \\ 0 & \frac{c_2}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{c_3}{2} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{c_4 \Delta t}{2} \\ 0 & 0 & 0 & \frac{c_4}{2} \end{bmatrix} \tag{3.24}$$

Kalman Filter method is provided by OpenCV. In this work, we realize the Kalman Filter using OpenCV function 'KF.predict()' and 'KF.correct()'. The measurement vector Z is equal to X for this system. The measurement of components x, y, z is from the proposed SLAM module. The measurement of components $\dot{x}, \dot{y}, \dot{z}$ can be directly read form the ROS topic "navdata". The measurement of components $\theta, \dot{\theta}, \phi, \dot{\phi}, \psi, \dot{\psi}$ can be directly read form the ROS topic "imudata".

(ii) Organization of the Entire System We draw the organization of the entire system in Fig. 3.6. We emphasize that the SLAM module and EKF module does not use information from Vicon state estimation. The Vicon data is synchronized just for comparison with the results from the proposed algorithm.

3.4 Experiment Results

This work is tested in the ROS environment, and 'RVIZ' package is used to show the estimated drone's position and the map points in real-time. RVIZ is a 3-D displaying tool to visualize information under ROS environment. At the same time, experimental data is also saved in files to replay and analyzed using MATLAB.

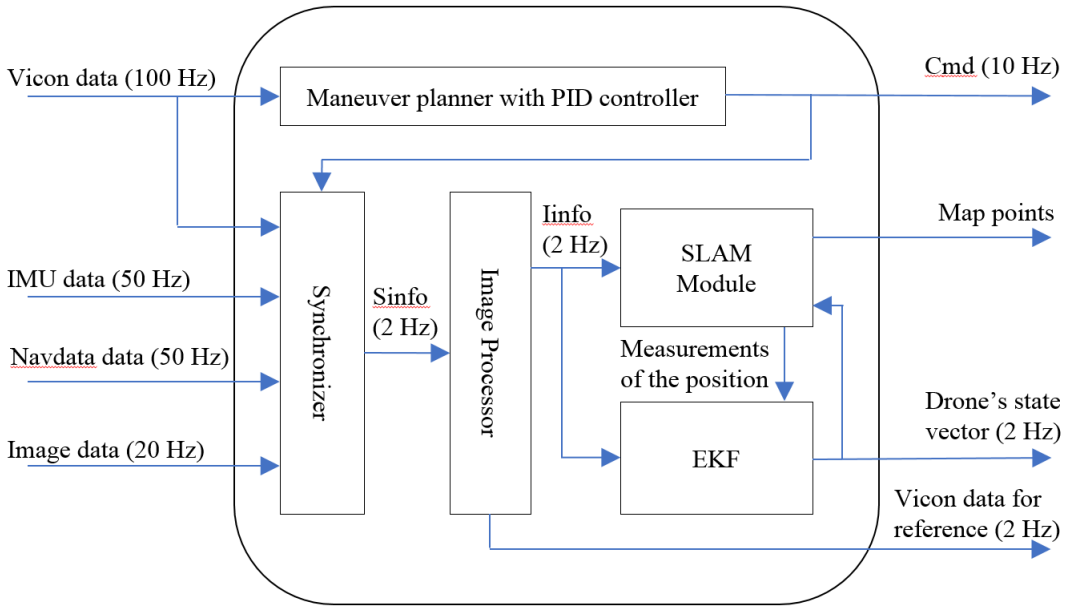


Figure 3.6: Organization of the entire system.

3.4.1 Hovering on Flat Surface Test

To test the drone's position estimation accuracy, we use Vicon data information to control the UAV hover at point $[0,0,1]$ in frame N. The flight data is saved to compare different algorithms. With the saved flight data, we first apply SLAM algorithm to calculate position measurements, apply EKF to estimate the drone's position and record. Then display the same flight data, but just use $\begin{bmatrix}\Delta x & \Delta y & \Delta z\end{bmatrix}^T$ from IMU readings to calculate position measurements, and use the same EKF setting to estimate the drone's position and record. Fig. 3.7 shows the test results. From Fig. 3.7, we can see that the drift of the position estimation is inhibited and the estimation accuracy has improved when using the proposed SLAM algorithm. If we define the estimation error as the absolute value of the difference of the estimation results and the Vicon results, we can list the statistics results Table 3.5 and Table 3.6. From Table 3.5 and Table 3.6, we can see that both the mean error and the standard deviation of the proposed SLAM algorithm are smaller than when SLAM is not used. The results proves that the proposed SLAM algorithm utilize the image information and the state estimation accuracy is improved.

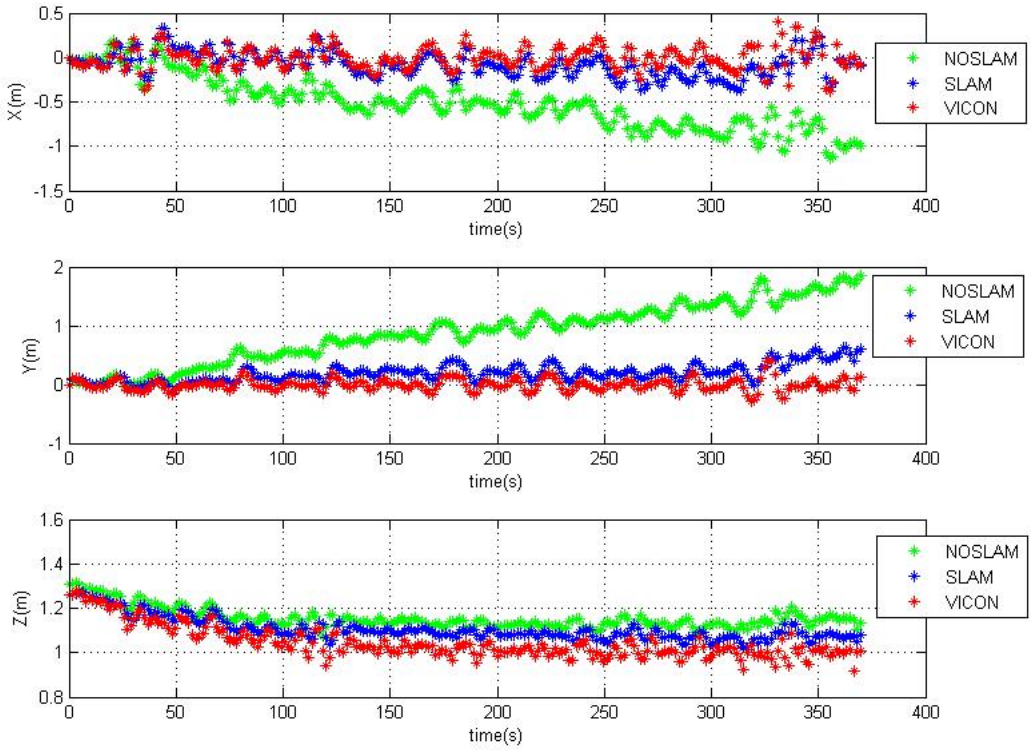


Figure 3.7: Drone's position estimation test.

Table 3.5: Mean error comparison.

	X axis	Y axis	Z axis	Distance
Mean error SLAM (m)	-0.0701	0.2032	0.0610	0.2514
Mean error NOSLAM (m)	-0.5081	0.8814	0.1091	1.0407

3.4.2 Maneuver on Unflattened Surface Test

To simulate in realistic and test the proposed 3-D SLAM mapping, we build a structure using a box. Fig. 3.9 shows the object deployed. The size of the box is $0.3m \times 0.5m \times 2.0m$. We align the long side with x axis, and hold up the left size to $0.6m$ to generate an inclined plane. We set the maneuver planner to hover the drone above the object and move around it for 120 seconds.

Table 3.6: Error standard deviation comparison.

	X axis	Y axis	Z axis	Distance
Standard deviation SLAM (m)	0.1106	0.1309	0.0320	0.1307
Standard deviation NOSLAM (m)	0.3092	0.5053	0.0429	0.5626

The reference hover position is defined as

$$\begin{cases} x_{ref} = 0.6\sin(0.2t) \\ y_{ref} = 0.2\cos(0.2t) \\ z_{ref} = 1.5 \end{cases} \tag{3.25}$$

During the flight, the positioning results is saved and draw in Fig. 3.8. At the same time, the map points is draw using RVIZ. Fig. 3.10 shows the map points accumulated from the beginning step using blue dots. The size of the grid show in image is $0.5m \times 0.5m$, which works as the distance reference. From Fig. 3.10, we can see the outline of the object is draw with correct physical size.

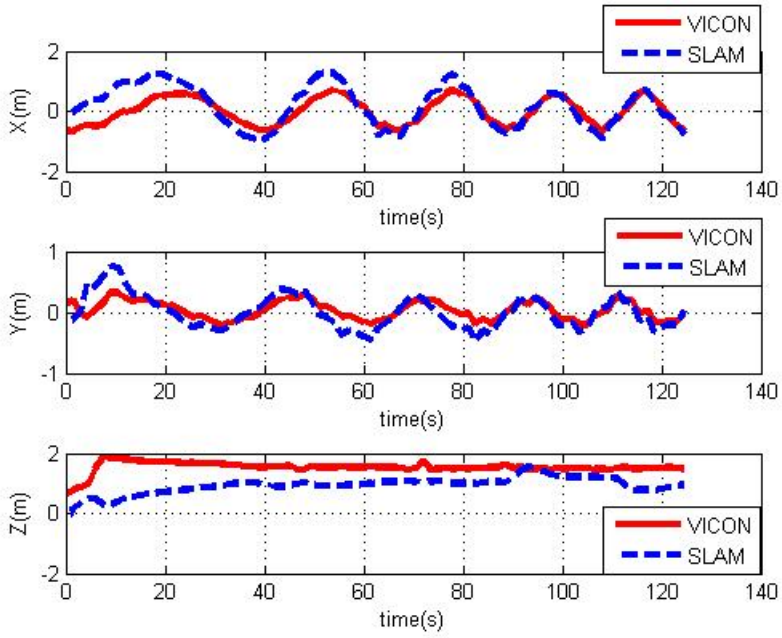
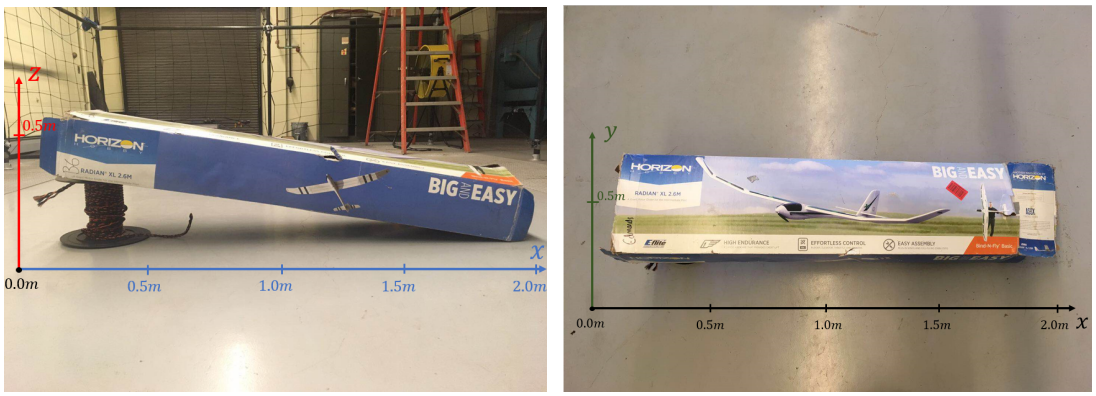


Figure 3.8: SLAM algorithm positioning results.

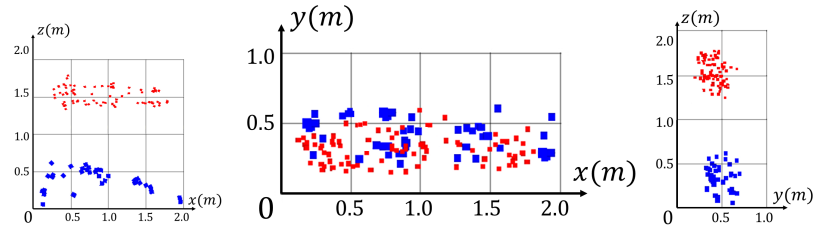
To compare with PTAM algorithm, we also test the 'tum_ardrone' package [72], with the same maneuver. The mapping and positioning results are draw in Fig.3.11 and Fig.3.12, respectively. In Fig.3.11, the green lines are the estimated trajectory of the drone, and the red dots are



(a) Side view of the object.

(b) Top view of the object.

Figure 3.9: Object for mapping testing.



(a) Side view from front (X-Z plane).

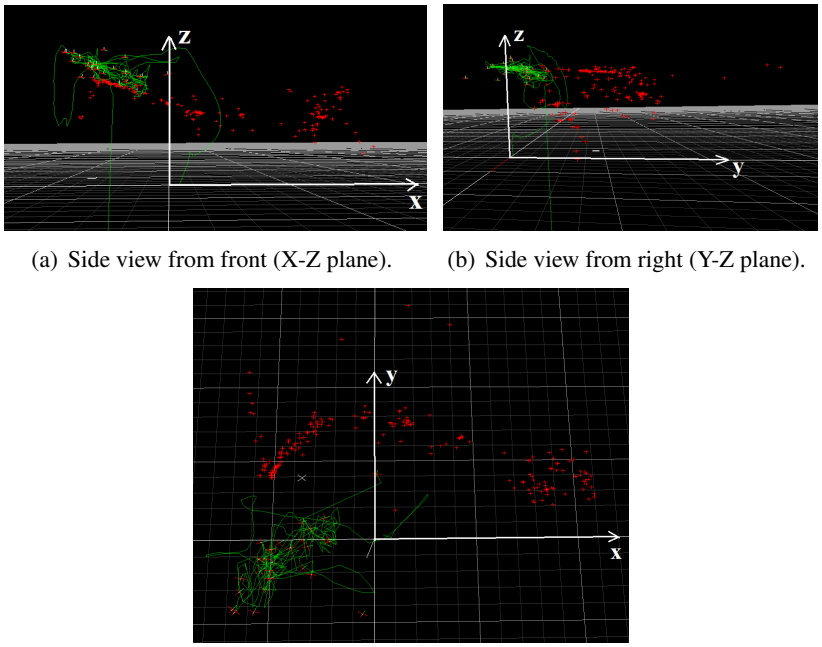
(b) Top view (X-Y plane).

(c) Side view from right (Y-Z plane).

Figure 3.10: SLAM mapping points.

the map points. At the same time, the PTAM state, scale, and the scale accuracy parameters provided by 'tum_ardrone' package is also plot in Fig.3.13. According to the 'tum_ardrone' package information, the state description is listed in table 3.7. The scale parameter is the estimated scale factor of the map. And the scale accuracy parameter represents the confidence of the estimated scale factor. The range of the scale accuracy parameter is from 0.5 to 1, and 0.5 represents no confidence for the estimated scale factor, 1 represents full confidence for the estimated scale factor.

From Fig. 3.13, we can see that the PTAM generates map tracking at 7s. After generates the initial tracking, the PTAM states stay 'PTAM_BEST' for most of the time, some times goes to 'PTAM_GOOD', and drop to 'PTAM_LOST' occasionally. To compare the positioning accuracy, we define the position estimation error to be the absolute value of the difference of the estimation results and the Vicon's results. We compare the error standard deviation of the proposed SLAM positioning data shown in Fig 3.8 with the PTAM positioning data shown in



(a) Side view from front (X-Z plane). (b) Side view from right (Y-Z plane).

(c) Top view (X-Y plane).

Figure 3.11: PTAM mapping points.

Table 3.7: PTAM state description.

Value	Name	Description
0	PTAM_IDLE	PTAM not running
1	PTAM_INITIALIZING	Initialization
2	PTAM_LOST	PTAM is running, but lost
3	PTAM_GOOD	Tracking quality OK
4	PTAM_BEST	Tracking quality best
5	PTAM_TOOKKF	Took a new key frame (equivalent to PTAM_BEST)
6	PTAM_FALSEPOSITIVE	PTAM thinks it is good, but its estimate is rejected

Fig. 3.12 from 7s to the end. Because the initial position selections of the proposed SLAM algorithm, PTAM, and Vicon system are all different, we do not compare the mean position error. The error standard deviation results are shown in table 3.8.

From table 3.8, we can see the the performance of the proposed SLAM algorithm is better than PTAM except for Z axis under the same experiment condition. The reason for the error standard deviation of Z axis of the proposed SLAM algorithm is not as good as other axis is because at 94s, an unusual altitude measurement from the ultrasonic sensor of AR.Drone has merged, which generates a disturbance. If we take the Z axis data to 94s, the error standard deviation is 0.1749, which is better than PTAM.

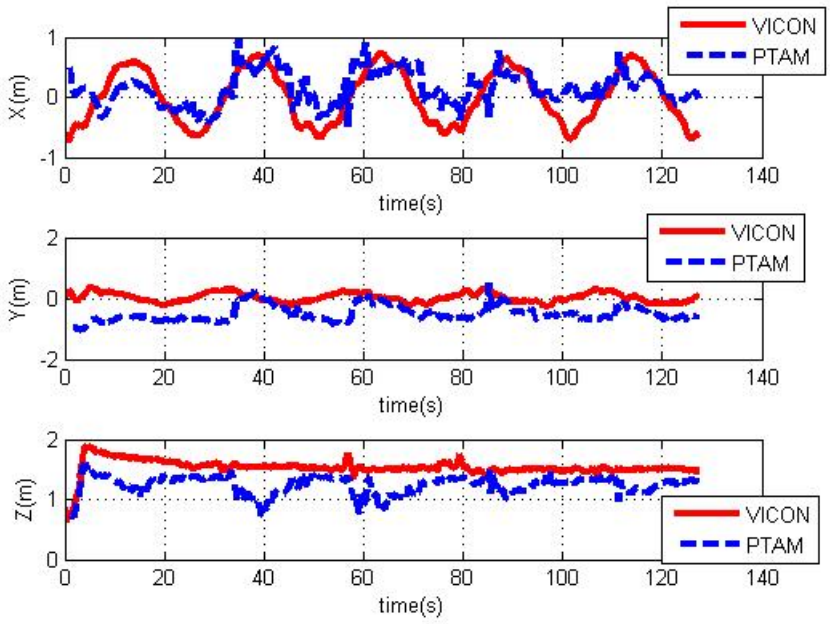


Figure 3.12: PTAM algorithm positioning results.

Table 3.8: Error standard deviation comparison.

	X axis	Y axis	Z axis	Distance
Standard deviation SLAM (m)	0.2284	0.1162	0.2302	0.2008
Standard deviation PTAM (m)	0.3267	0.2916	0.1841	0.2573

We should notice that the PTAM has to be combined with scale estimation algorithms. From Fig. 3.13, the estimated scale factor is changing, and the confidence parameter is low, which limits the practicality generated map. As a contrast, the proposed SLAM algorithm provides uniform absolute scale of the map. We should also notice that the PTAM results displayed is the best case we get. Due to the initializing process of PTAM, when start the algorithm, human guidance is required to generate the first two key frames. Because of the unflattened surface and the low quality of the image from the bottom camera of the AR.Drone, it is difficult to initialize PTAM. We did 20 test for PTAM, and this is the best case we have. Another PTAM state is shown in Fig. 3.14. From Fig. 3.14, we can see that the PTAM is mainly at state 'PTAM_LOST' during the flight, and that makes the both positioning and mapping results worse.

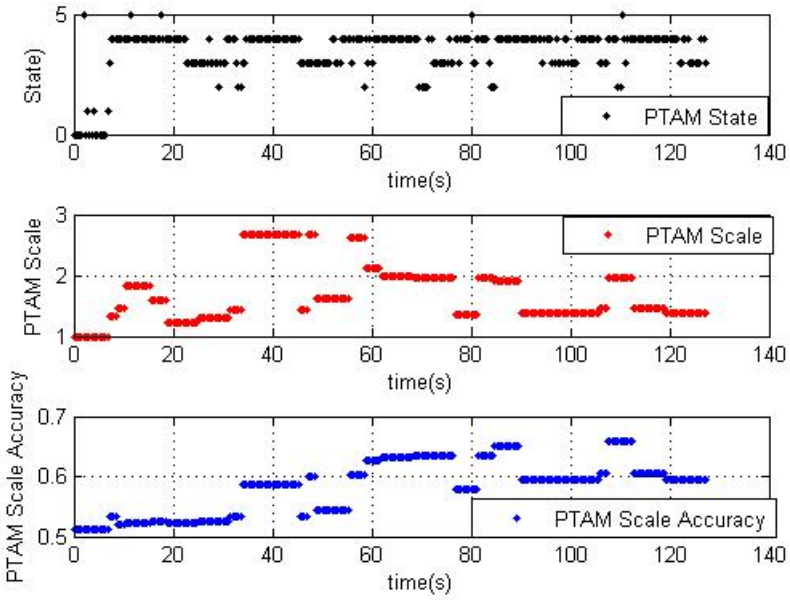


Figure 3.13: PTAM parameters.

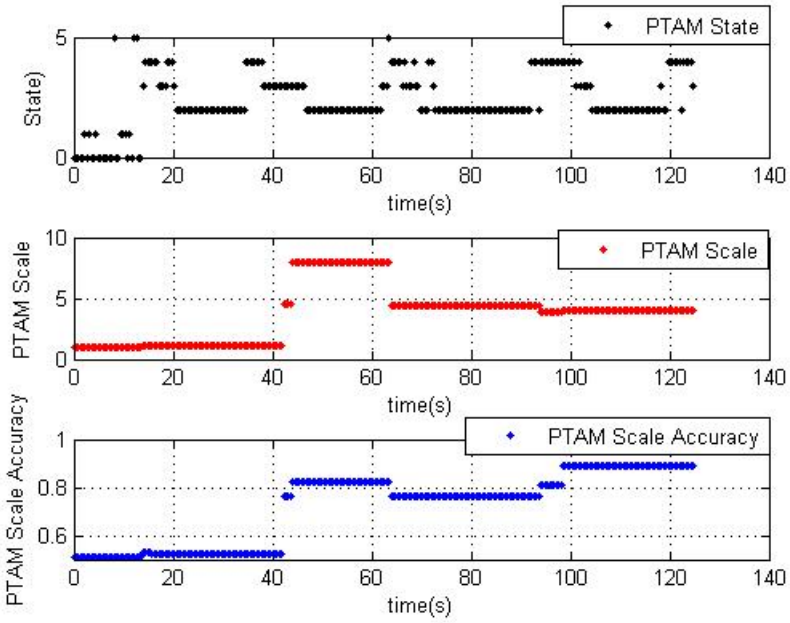


Figure 3.14: PTAM parameters in another case.

Chapter 4

A K Nearest Neighborhood Based Wind Estimation

4.1 Introduction

Wind disturbance is another challenge for small VTOL UAVs. One limitation of small UAVs is that they are sensitive to wind disturbance, leading to degraded flight accuracy and safety. If UAVs could measure or estimate wind during the flight, the information can be used to enhance the control for a more robust and safer flight. Such information can also be used to build a wind field map for a region of interest. For example, the wind field of a urban street canyon is generated by using UAV flight data, instead of using numerical models [73].

To measure the wind velocity, airspeed sensors such as pitot tubes are usually equipped on fixed-wing UAVs. However, due to the rotor down-wash effect [74], for rotary-wing VTOL UAVs, the measurement from airspeed sensors is not reliable if the airspeed sensors are deployed close to the propellers. Recently, to measure the wind speed by sensors, Watkins et al. mount a 3-D printed airspeed tube in front of a quadrotor by using a carbon fiber bar [75]. Since the carbon fiber bar is long enough, the rotor down-wash effect is avoided. Although the method shows that rotary-wing VTOL UAVs can be used as flexible wind sensing platforms, using the air tube and the bar increase the total size and weight. Furthermore, such UAVs are not likely to have aggressive maneuverability. Accurate wind velocity estimation method without airspeed sensors would be useful for the control compensation or wind field generation. To achieve this goal, the relationship of the wind disturbance and the UAV response should be generated.

Many researchers have solved this problem by building mathematical models. In general, system identification and Kalman filter techniques are widely used. For system identification methods, Lusardi et al develop a gust turbulence model through flight tests in [76]. The turbulence properties are characterized by equivalent control input and the method is derived using

measurements of control signals and inertial sensors from flight tests of a UH-60 helicopter. In [77], Nicola has developed a method to estimate the wind velocity from the measurements of a rotary-wing VTOL UAV flight in moving atmosphere. The wind velocity components are estimated by using a variational formulation. The proposed method uses air-frame and rotor model, and machine learning methods are used to determine the model parameters. In [78], a method for aerodynamic model identification of a micro air vehicle is proposed by J. Velasco et al. Without direct airspeed data measurement from sensors, the authors estimate the wind velocity based on the multi-objective optimization algorithms that use identification errors to estimate the wind-speed components that best fit the dynamic behavior observed. For Kalman filter methods, Venkatasubramani et al solve this problem by using Kalman filter based wind model identifications [79]. The approach uses only inertial, position, and control information, without airspeed measurements. The technique is demonstrated using flight tests of an off-the-shelf unmanned quad-rotor UAV. An accurate dynamic model of the aircraft is developed using system identification techniques, and the model is used in a Kalman filter to estimate the external wind disturbances. In [80], Sikkel et al. extend a drag-force enhanced quadrotor model by denoting the free stream air velocity as the difference between the ground speed and the wind speed. The model is used to create a nonlinear observer capable of accurately predicting the wind components. By using low-cost MEMS IMUs and GPS-velocity measurements, wind components can be estimated by EKF.

All methods discussed above solve the problem by using equations including aerodynamic models and wind field models. Such models need the details of the dynamics to ensure good estimation result. However, in many cases, a dynamical model may not be available which prevents use of model based wind estimation techniques. Also, the parameters in the dynamic equations sometimes can be difficult to be determined accurately. Human UAV pilots, when flying the UAV outdoor, they estimate the wind speed based on their experience, and they achieve this by watching the response of the UAV with respect to the wind and their controls. During this process, they do not use mathematical models and they do not have to determine the parameters.

Inspired by this fact, in this work we propose a learning based method for rotary-wing

VTOL UAVs to estimate the wind speed under hovering conditions, where the UAV is programmed to hover at a certain point. The method includes two stages: a training stage and a wind speed estimation stage. During the training stage, we design a single set of PID controllers to hover the UAV across a range wind speeds, and record the position, attitude, and controls. Then, we extract a set of designed features from the flight data, and save these features as well as the corresponding wind speed. During the on-line wind speed estimation stage, we use the K Nearest Neighborhood (KNN) strategy [81]. While several learning based regression methods including linear/polynomial regression are simple to implement, these methods often assign an unjustified underlying function to determine the input-output relationship. On the other hand, the KNN methodology is purely data driven, non-parametric, and often works well even in high dimensional spaces. When hovering in unknown wind speed, the UAV generates the features based on the real-time flight data by using the same method during the training stage. Afterwards, we compare the current features with the saved features, choose the most similar cases in the database, then generate the wind speed estimation.

The key advantage of the proposed technique is that it works in situations where the aerodynamic model cannot be developed accurately. Instead, we only use position, attitude, and control information to extract features. Direct air flow measurement from airspeed sensors is not required, so the payload of extra sensors is saved. We compare our results with those from the AR.Drone embedded system, and our estimation results are more robust and reliable.

The organization of this Chapter as follows. In Section 4.2, the background information including experiment setup, coordinate system definition, and system structure is introduced. In Section 4.3, the data curation of the learning method is described in detail. In Section 4.4, the KNN algorithm we used is presented, including the feature normalization, distance calculation, and the method to choose parameters. In Section 4.5, experiment results are demonstrated and analyzed.

4.2 Equipment and System Setup

The small VTOL UAV used for this work is Parrot AR.Drone, which introduced in 2.2. The global positioning system is Vicon, which is also introduced in 2.2.

4.2.1 Fan and Anemometer

In this work, a commercial fan made by MAXXAIR, model BF24TF2N1 is used to generate wind in the lab. It has two working modes: a high level mode and a normal level mode. The diameter of this fan is 60 cm, and the height of the center of the fan to the ground is set to be 1 m. To measure the wind speed, an anemometer made by Extech, model 45118 is used. According to the anemometer, the measured maximum wind speed of this fan is 6 m/s for the high level mode, and 2.7 m/s for the normal level mode, and the wind speed is reduced when the distance to the fan is increased.

4.2.2 Coordinate System

The coordinate system used in this work is shown in Figure 4.1. The origin O is defined at the projection of the center of the fan onto the ground. The Y axis points to the wind direction, the Z axis points upward, and the coordinate system follows the right-hand rule.

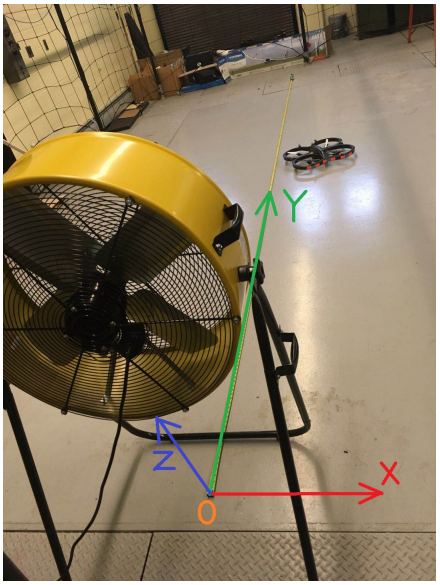
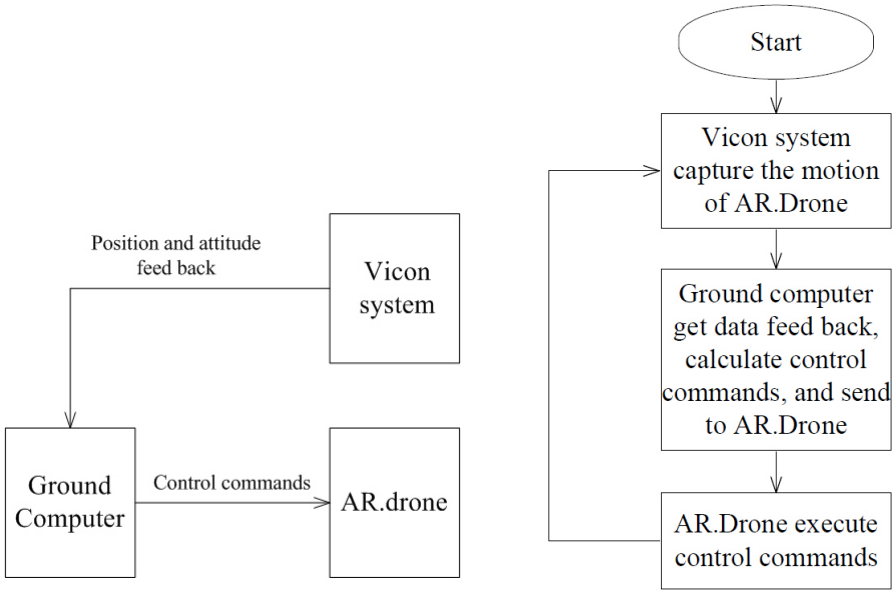


Figure 4.1: Coordinate system set up.

4.2.3 The System

In this work, the Vicon camera system captures the motion of the AR.Drone, calculates AR.Drone's positions x, y, z, attitude Euler angles roll, pitch, yaw (ϕ, θ, ψ), and sends these data to the

ground computer. Based on these data, the ground computer runs the designed controller, and sends the calculated control commands u_1, u_2, u_3, u_4 to the AR.Drone. Figure 4.2 shows the system structure and the system flow chart. In this work, the loop (show in Figure 4.2(b)) runs at 30 Hz. Since we keep sending the control commands from ground station at 30 Hz rate, the internal vision-based position hold using the bottom camera and sonar based altitude hold algorithms of AR.Drone are not activated.



(a) System structure. (b) System flow chart.

Figure 4.2: System description.

4.3 Data Curation

During this stage, we design a single set of PID controllers introduced in Section 2.4.4, to hover the UAV across a range wind speeds. From the recorded flight data, we extract features that are related to the wind speed. Then, the extracted features and the wind speed are saved in a database for wind estimation stage.

4.3.1 PID Controller

In this work, we design four independent PID controllers for the four control channels by using the PID control law. The control channel u_1 (roll angle) is calculated based on the x error.

The control channel u_2 (pitch angle) is calculated based on the y error. The control channel u_3 (yaw angular velocity) is calculated based on the yaw angle error. And the control channel u_4 (velocity of the altitude) is calculated based on the altitude error. For the control of yaw angle ψ, the goal is to maintain it equal to 0°, so the head of the drone will point along the y direction.

The 4 PID controllers are tuned under no wind condition. We use the Vicon data for feedback, set one point as the desired position, and tuned the gains P, I, and D to hover the drone at the point. The control cycle's frequency is 50Hz, and the unit of x, y and z error is in meters, the unit of yaw angular velocity is in degree per second. The result's of the tuned gains used are listed in table 4.1.

Table 4.1: PID gains used.

	P	I	D
For channel 1	0.6	0.0001	8.0
For channel 2	0.6	0.0001	8.0
For channel 3	0.08	0.0001	0.06
For channel 4	4.0	0.01	8.0

After the gains are tuned, the values are used for later experiments. The controls are sent to AR.Drone through the SDK provided by the manufacturer. The mode of AR.Drone is set as non-hover mode, so the controls to the drone are purely from the controllers designed.

4.3.2 Flight Data Collection

Based on the size of the flight area (7 m by 4 m), the maximum wind speed generated by the fan (6 m/s), and the flight safety clearance (set to be 0.6 m to the boundary of the flight zone), we select 8 wind speeds for training. The wind speed starts from 0 m/s and reaches maximum at 4.2 m/s, with 0.6 m/s increment.

For different wind speeds, we use the same PID controllers to hover the drone at the corresponding positions. For each flight, we command the drone hover for 30 seconds. Since the loop (Figure 4.2(b)) is run with 30 Hz, we collect 900 data points for each flight.

In each loop, we acquire 6 flight states from the Vicon system and 4 control commands from the PID controller. For the 6 flight states, we denote them as $\begin{bmatrix} x & y & z & \phi & \theta & \psi \end{bmatrix}$. And for the four control commands we denote them as $\begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}$. We denote the desired hover

position as x_d, y_d, and z_d. Then, we can define the position error as $x_e = |x_d - x|$, $y_e = |y_d - y|$, and $z_e = |z_d - z|$. Because the desired attitude angles are all zero, the true attitude is the attitude error in this work. In each loop k, 10 data points are recorded, and we denote them as:

$$X(k) = \begin{bmatrix} x_e(k) & y_e(k) & z_e(k) & \phi(k) & \theta(k) & \psi(k) & u_1(k) & u_2(k) & u_3(k) & u_4(k) \end{bmatrix} \quad (4.1)$$

4.3.3 Features

(i) Features Definition

This section illustrates how we extract features from the flight data under each wind situation. Figure 4.3 shows the comparison of the hover performance for 10 seconds using the same PID controllers. In these figures, the blue dash represents the case that the wind speed is zero, the red dot represents the case that the wind speed is 0.8 m/s, and the green dot dash represents the case that the wind speed is 1.6 m/s. From the comparison, we can see that the wind will significantly affect the hover flight. When the wind speed increases, the position and attitude errors are increased, and the control efforts are also increased in order to maintain the hover flight.

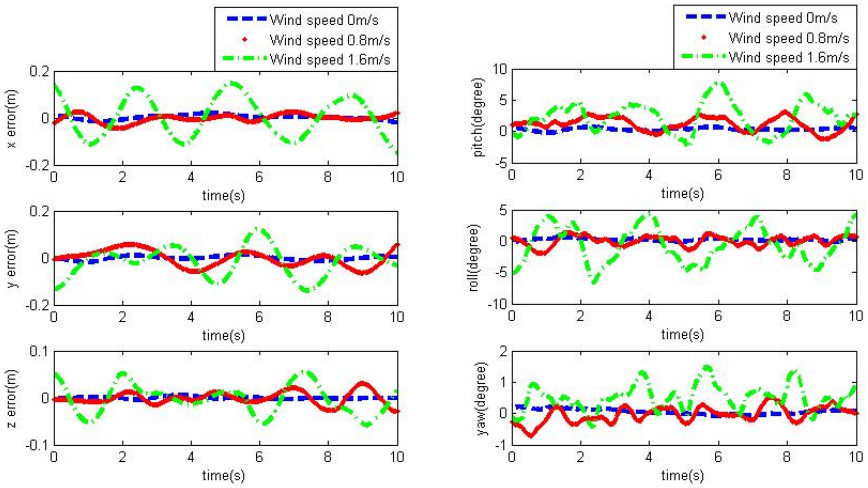
Based on our experience and observation, we choose 19 features which are highly related to the wind speed. The first 7 features are related to the position error. They are the mean and variance of x_e, y_e, z_e, and the mean of total position error. The total position error P_e is defined as:

$$P_e = \sqrt{x_e^2 + y_e^2 + z_e^2} \quad (4.2)$$

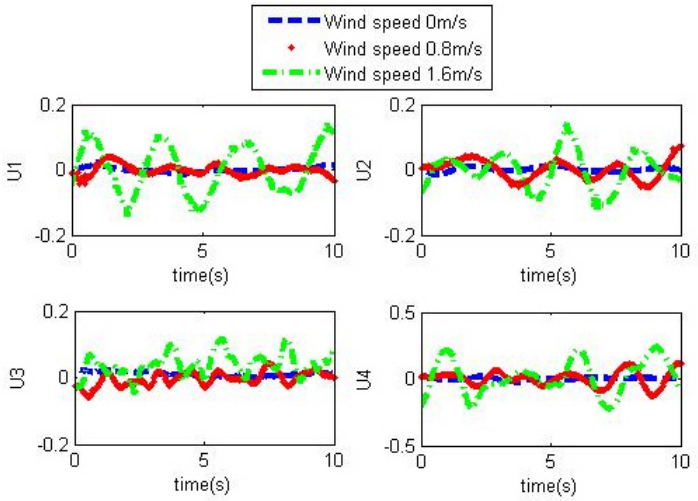
Similarly, the second 7 features are related to the attitude. They are the mean and variance of the absolute values of the ϕ, θ, ψ, and the mean of total attitude error. The total attitude error A_e is defined as:

$$A_e = \sqrt{\phi^2 + \theta^2 + \psi^2} \quad (4.3)$$

The last 5 features are related to the control effort. For the control, in programming, the scale is set from -1 to 1, where 0 means the control effort is zero, -1 and 1 means the maximum control efforts for opposite directions. In this work, the control effort is defined as the absolute value of the control commands, from 0 to 1. The 5 features we



(a) Position comparison. (b) Attitude comparison.



(c) Control comparison.

Figure 4.3: Hover performance comparison.

used are the mean control effort of u_1, u_2, u_3, u_4, and the total control effort. The total control effort C is defined as:

$$C = \sqrt{u_1^2 + u_2^2 + u_3^2 + u_4^2} \tag{4.4}$$

For each flight with the corresponding wind speed, we can analyze the flight data and calculate the feature vector. We denote the feature vector F^r as:

$$F^r = \begin{bmatrix} f_1 & f_2 & f_3 & \dots & f_{19} \end{bmatrix} \tag{4.5}$$

where:

$$\begin{cases} f_1 = mean(x_e) & f_2 = var(x_e) \\ f_3 = mean(y_e) & f_4 = var(y_e) \\ f_5 = mean(z_e) & f_6 = var(z_e) \\ f_7 = mean(P_e) & \\ f_8 = mean(|\phi|) & f_9 = var(|\phi|) \\ f_{10} = mean(|\theta|) & f_{11} = var(|\theta|) \\ f_{12} = mean(|\psi|) & f_{13} = var(|\psi|) \\ f_{14} = mean(A_e) & \\ f_{15} = mean(|u_1|) & f_{16} = mean(|u_2|) \\ f_{17} = mean(|u_3|) & f_{18} = mean(|u_4|) \\ f_{19} = mean(C) & \end{cases} \tag{4.6}$$

(ii) Features' Effectiveness Validation

To validate that the designed features are effective, we check the Pearson-r coefficient [82] by using Python's Seaborn library. Pearson-r coefficient indicates the correlation between two variables. It ranges from -1 to 1. The sign of Pearson-r coefficient shows the two variables are positive correlated or negative correlated. When two variables are irrelevant, the corresponding Pearson-r coefficient is 0, and when two variables are highly correlated, the absolute value of Pearson-r coefficient is close to 1. It is expected for the values of the features to increase as the wind speed increases. In the tests for all 19 features respect to the wind speed, Pearson-r coefficients are observed larger than 0.5, thus proving the positive correlation between the feature vectors and the wind speed, and it also indicates that the 19 selected features are effective. Table 4.2 shows the results of the Pearson-r coefficient respect to the features.

4.3.4 Training Database Construction

For each wind speed, we conduct 10 independent flights and acquire 10 feature vectors. In total, we have 8 wind speeds, so we have 80 feature vectors stored. For each feature vector, we

Table 4.2: Features and the tested Pearson-r coefficients.

Features	Pearson-r coefficient
f_1	0.92
f_2	0.81
f_3	0.87
f_4	0.67
f_5	0.86
f_6	0.82
f_7	0.95
f_8	0.90
f_9	0.82
f_{10}	0.90
f_{11}	0.85
f_{12}	0.73
f_{13}	0.58
f_{14}	0.96
f_{15}	0.92
f_{16}	0.87
f_{17}	0.81
f_{18}	0.91
f_{19}	0.94

put the measured wind speed in the end. So we have a 80 by 20 matrix stored as the model. The row index i starts from 1 to 80, and the column index j starts from 1 to 20. When j is between 1 to 19, the data represents the feature, and when j is 20, the data represents the wind speed. Figure 4.4 illustrates the method to generate the database.

4.4 Wind Speed Estimation Stage

In this section, we introduce the KNN algorithm we used, including the feature normalization, distance calculation, and the wind speed calculation. Also, the strategy to choose the parameter K in KNN method is discussed.

4.4.1 KNN Algorithms

(i) Feature Normalization

As we can see, the feature vectors are of different magnitudes and units. To compare the two features, we first normalize the features. We take feature f_1 as an example. Suppose that we have n independent experiment for training, and we have n feature f_1. Now we

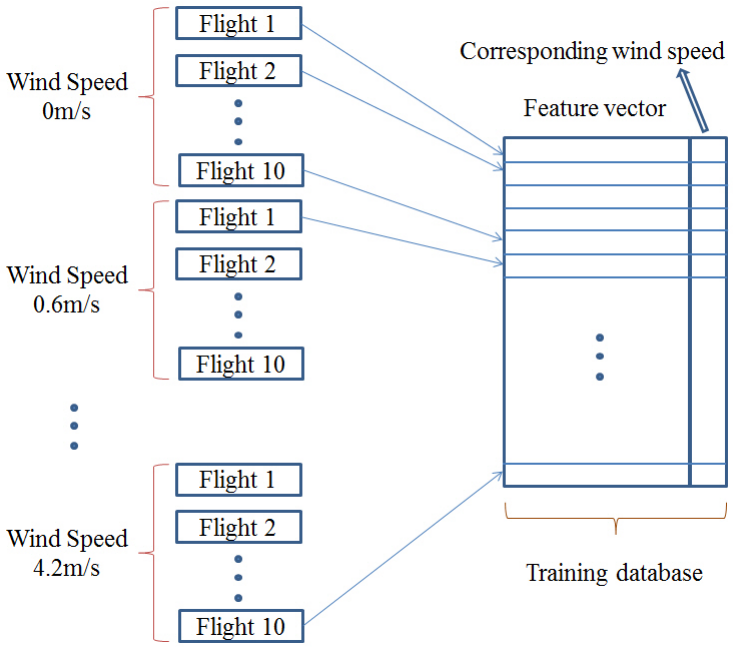


Figure 4.4: Illustration of the method to generate the training database.

have another flight experiment to estimate the wind for this flight, and have one more f_1. In total, we have $n+1$ features. In the $n+1$ feature, we find the largest one f_{max}, and the smallest one f_{min}. Take the difference, we have $f_{range} = f_{max} - f_{min}$. After that, each feature is divided by f_{range}, and the results are the normalized features. We denote the normalized feature vector as F.

(ii) Distance Calculation

To compare the distance d of two normalized feature vectors F_1 and F_2, we use the weighted Manhattan distance. The weighted Manhattan distance used in this work is defined as:

$$d = w_1 \cdot |F_1(1) - F_2(1)| + w_2 \cdot |F_1(2) - F_2(2)| + \cdots + w_{19} \cdot |F_1(19) - F_2(19)| \quad (4.7)$$

where w is the weight for each feature, which is equal to the Pearsonr coefficient which corresponds to the importance of the feature. The higher importance, the larger weight will be assigned.

(iii) Wind Speed Estimation

When there is a new flight, we can extract a 1x19 feature vector using the same method

as we used during model generation. Then, we compare the feature vector with the 80 feature vectors in the model, and record its distance to each one in the model. Among the 80 distances, we choose K smallest ones, and get their row index i. After that, we look at the column 20, add the wind speed together and divided by K. The results will be the wind speed estimation for this flight. Figure 4.5 illustrates the method to calculate the wind speed.

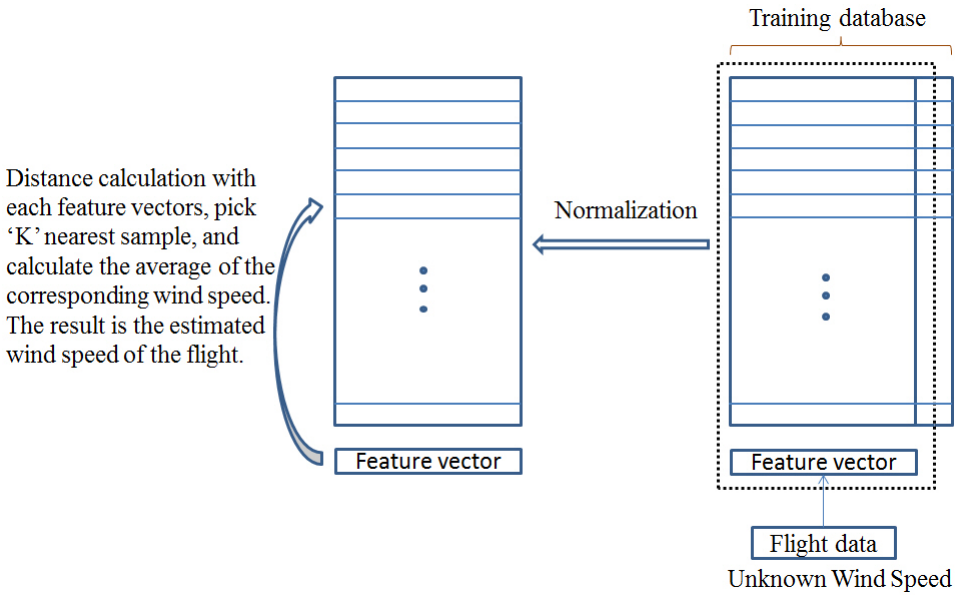


Figure 4.5: Illustration of the method to calculate the wind speed.

4.4.2 Choose the Value of K

The parameter K is important for the KNN algorithm to work properly. When K is either too large or too small, the estimation result will not be reliable. As K is reduced, the fitting is better but the function becomes more complex, while as K increases, the function is smoother but less flexible and the training error increases. To find the appropriate value of K, we first test the KNN algorithm by using the model data.

For each wind speed, we randomly choose 1 of the 10 experiments as the validation vectors, and the rest of the 9 remain as the model. In total, we will have 8 validation vectors, and 72 vectors as the model. For K value selection, we test it from 1 to 72, and find the best choice which leads to the minimum total estimation error of the 8 experiments. The estimation error

E is defined as:

$$E = |v_e - v| \tag{4.8}$$

where v_e is the estimated wind speed, v is the wind speed reference. And the total estimation error is $\sum_{i=1}^{8} E_i$.

Figure 4.6 shows the total estimation error using different K. We can see that when $K = 8$, the total estimation error reaches the minimum. Because the 10 validation data vectors are randomly chosen, the best value of K may not always be 8. We have repeated the experiment 10 times, and the average of the best K value is 11. So we choose 11 as the value of the K for later experiments.

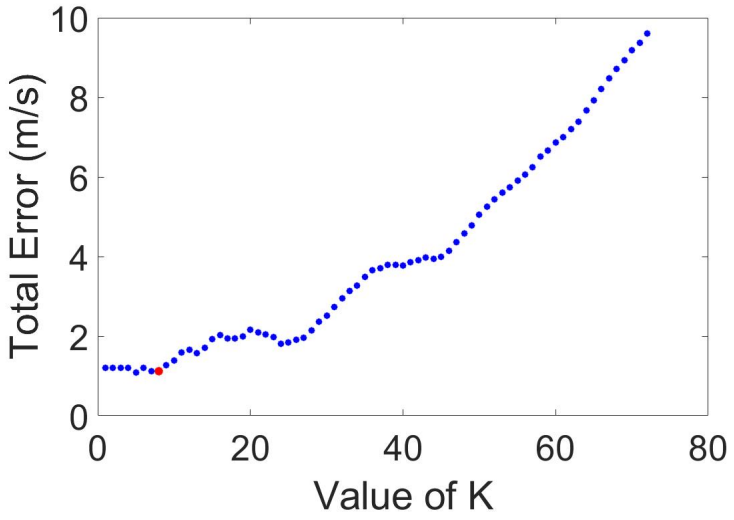


Figure 4.6: Totla error vs. parameter K.

4.5 Experiment Results

To test the performance, in this section, we first introduce the wind field of the fan, then we compare the wind field estimation of our proposed method and the result from the AR.Drone. Users are allowed to subscribe a wind estimation topic directly in ROS, which is provided by the AR.Drone.

4.5.1 Wind Field Generation

Based on the size of our lab, we choose 7 points along the wind direction, and measure the wind speed at these points. The coordinate of these points are $(0.0, 1.2, 1.0)$, $(0.0, 1.5, 1.0)$, $(0.0, 1.8, 1.0)$, $(0.0, 2.1, 1.0)$, $(0.0, 2.4, 1.0)$, $(0.0, 2.7, 1.0)$, and $(0.0, 3.0, 1.0)$. They are along the y axis and at the height of $1m$. At each position, we use the anemometer to measure the wind speed 5 times, and the average of the measurements is chosen as the reference wind speed at this point. Table 4.3 list the distance to the fan and the corresponding wind speed. Figure 4.7 shows the measured wind speed at these points.

Table 4.3: Wind speed measurements and their corresponding distance to the fan.

Distance to the fan (m)	1.2	1.5	1.8	2.1	2.4	2.7	3.0
Wind speed (m/s)	3.07	2.63	2.27	2.03	1.87	1.77	1.70

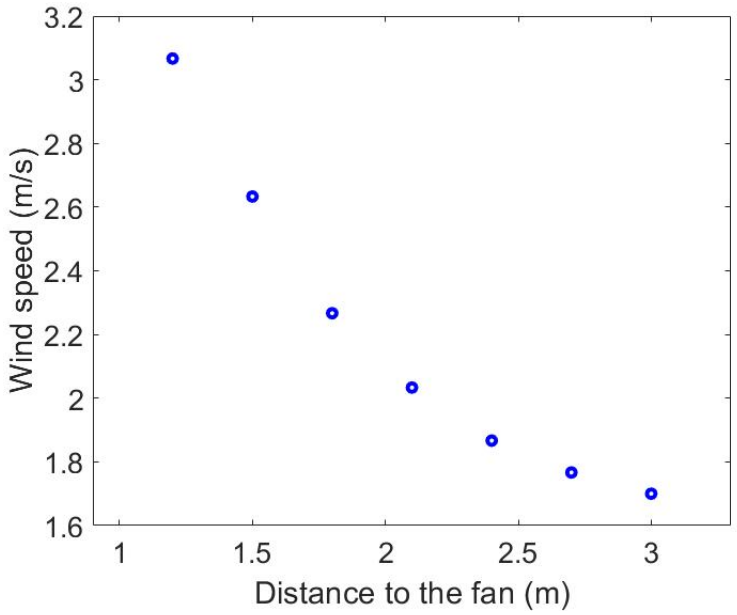


Figure 4.7: Wind speed change as the distance to the fan change.

4.5.2 Wind Speed Estimation Results

We assign the way point over the 7 positions, and hover the drone at each position for $20s$. Then, we use the saved database as the model, and choose $K = 11$ to calculate the wind speed

estimations. At the same time, the wind speed estimation results of from the AR.Drone embedded system are also recorded. We note that the wind estimation algorithms of AR.Drone are not available, so we only list the results as a reference. Table 4.4 and Figure 4.8 show the results of our proposed method and the result from the AR.Drone.

Table 4.4: The comparison of our proposed method and the AR.Drone.

Wind speed reference (m/s)	3.07	2.63	2.27	2.03	1.87	1.77	1.70
Estimation results (m/s)	2.99	2.41	2.34	2.15	1.89	1.74	1.71
AR.Drone results (m/s)	0.93	0.52	0.63	0.52	0.47	0.31	0.24

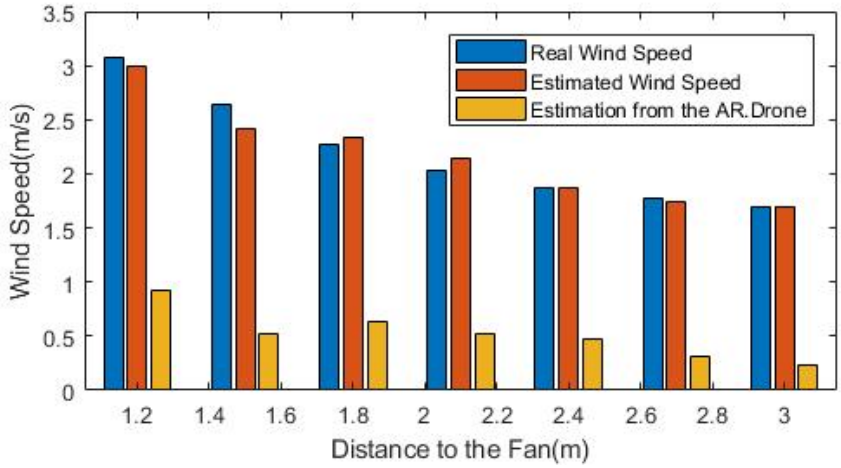


Figure 4.8: On-line wind estimation results.

From the results, for the 7 wind speeds, we calculate the estimation errors are 0.08 m/s, 0.22 m/s, 0.07 m/s, 0.12 m/s, 0.02 m/s, 0.03 m/s, and 0.01 m/s, respectively. The average estimation error is 0.074 m/s. And the estimation relative errors are 2.6%, 8.4%, 3.1%, 5.9%, 1.1%, 1.7%, and 0.6%, respectively. The average estimation relative error is 3.3%. At the same time, the average estimation error of the AR.Drone's embedded system is 1.673 m/s.

Chapter 5

A Low-cost Quadrotor System Design

5.1 Introduction

For VTOL UAVs hardware, this thesis develops a quadrotor solution which is able to support majority of research of small VTOL UAVs. The quadrotor's structure is made by carbon fiber, which provides large freedom of weight for on-board computers and sensors. For the on-board computer, a high level processor and a low level processor work together as a two-level processor system. The low level processor is responsible for sensor data acquiring, synchronization, and motor speed controls. Further more, the low-level processor also monitor the manual control signal during the flight, and is able to switch the control source to manual control under emergency situation, which makes the platform safe to operate. For the high level processor, a Ubuntu operation system is equipped. At the same time, Robot Operating System (ROS) is also embedded to access on-line available ROS packages easily. Researches can also make their own packages which contain their own algorithms. Another function for low level processor is to communicate with users in real-time through wireless communication. The on-board sensor system includes: 1) a downward looking camera, 2) an Inertial Measurement Unit (IMU) which contains a 3-axis gyro and a 3-axis accelerometer, 3) a 3-axis compass, and 4) a downward looking ultrasonic sensor. These sensors provide rich information. Combined with the powerful two-level processor system, the whole system can help many important UAV research.

5.2 Quadrotor Frame and Power System

5.2.1 Quadrotor Frame

The frame of the quadrotor we choose is S500 version with carbon fiber landing gear. S500 frame is mainly made of durable PCB version carbon fiber material and exquisite design, which

makes the frame strong enough and light as possible. The design of the 4 arms has 7 degree dihedral angle design which increase flight stability while minimizing power loss. Another outstanding feature of S500 frame is the landing gear design. The landing gear is made by 3K carbon fiber pipe with 16mm diameter. At the same time, the landing gear is covered by soft form, which make the landing soft to protect the devices on-board. Fig. 5.1 shows the S500 quadrotor frame components. From Fig. 5.1, we can see that the components are compact, which make it easy to assemble. Fig. 5.2 shows the S500 quadrotor frame after assemble. From Fig. 5.2, we can see that the free space of the frame is large for on-board devices.

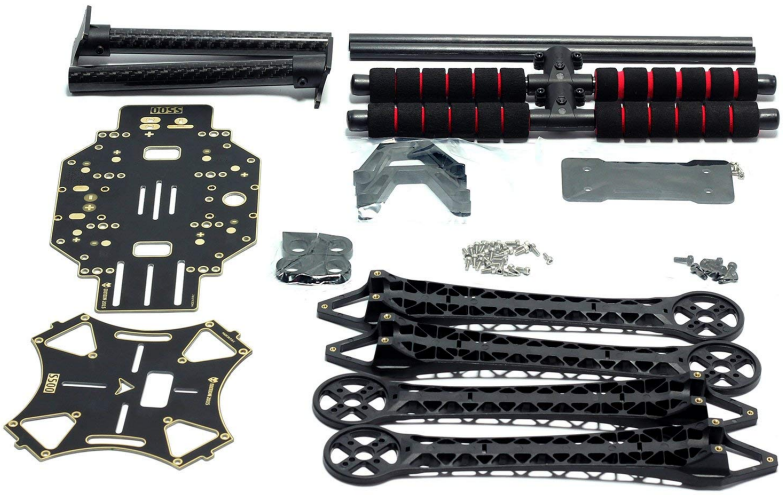


Figure 5.1: S500 quadrotor frame components.

5.2.2 Power System

The whole system is powered by Li-Po battery. Li-Po battery is a rechargeable battery of lithium-ion technology using a polymer electrolyte instead of a liquid electrolyte. High conductivity semisolid (gel) polymers form this electrolyte. These batteries provide higher specific energy than other lithium battery types and are used in applications where weight is a critical feature, like mobile devices and radio-controlled aircraft [83]. In this work, we choose Turnigy MultiStar 3 cells type. The full capacity is 5200 mAh. The voltage when fully charged is 12.6V. The maximum current provided is up to 50A. From our experiment, it can support 15min flight for the designed quadrotor platform. Fig. 5.3 shows the selected battery.

(a) Top view of S500 frame.

(b) Side view of S500 frame.

(c) Bottom view of S500 frame.

(d) Inside view of S500 frame.

Figure 5.2: S500 quadrotor frame.

Electronic Speed Control (ESC) is typical for control modern brush-less motor. The ESC can control the output voltage to the motor according to Pulse Width Modulation (PWM) [84] signal input. The ESC we choose for the designed quadrotor platform is Turnigy MultiStar 20A version. The maximum current tolerance is 20A. The 12.6V battery voltage is in the range of the ESC. Fig. 5.4 shows the selected ESC.

The motor with propeller provides the drag force for the drone. We choose 920KV, size 2212 brush-less motor with 9450 Self-Tightening propeller for the designed quadrotor platform. The 920 KV value of the motor means when the output voltage up to 1V, the rotations increase 920 per minute. The 2212 value is related to its physical size, which is able to install on the S500 frame. 9450 value for the propeller represents it's size. The length of the propeller is 9.4 inch long. Fig. 5.5 shows the selected motor and propeller.

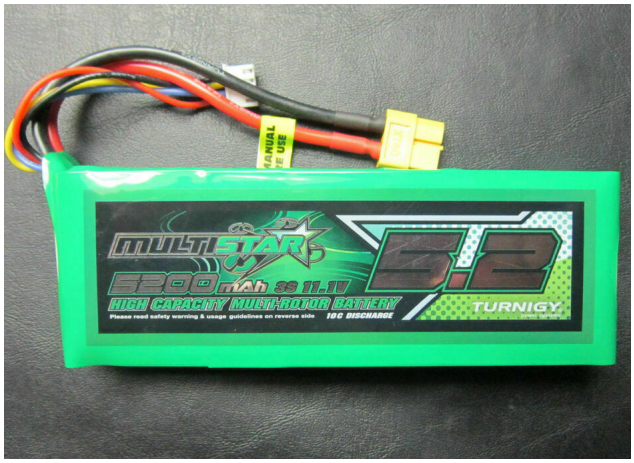


Figure 5.3: Turnigy MultiStar 3 cells Li-Po battery.

Figure 5.4: Turnigy MultiStar 20A ESC.

5.3 On-board Sensor System

The on-board sensor system includes: 1) a downward looking camera, 2) an Inertial Measurement Unit (IMU) which contains a 3-axis gyro and a 3-axis accelerometer, 3) a 3-axis compass, and 4) a downward looking ultrasonic sensor. The camera we select is ODROID USB-CAM 720P [85]. This camera provides real 720P HD resolution and 16:9 wide screen output. The interface with ODROID XU4 is USB2.0 High-speed and plug-n-play interface. The image flow rate is up to 30fps. The pixel size is 3.0um x 3.0um. The focal length is 3.5mm. The field of view (FOV) is 68 degree. For the IMU and compass, we choose LSM9DS0 [86]. The LSM9DS0 is one of only a handful of IC's that can measure three key properties of movement: angular velocity, acceleration, and heading in a single IC. The gyroscope can measure

(a) 920KV 2212 motor. (b) 9450 Self-Tightening propeller.

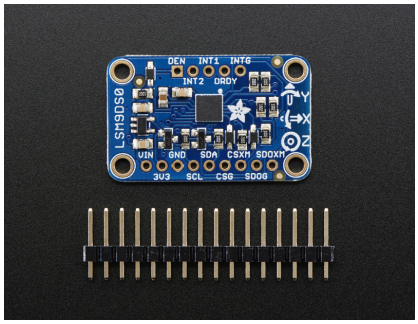
Figure 5.5: Motor and propeller combo.

angular velocity. Angular velocities are measured in degrees per second (DPS). The LSM9DS0 can measure up to $\pm$2000 DPS, though that scale can also be set to either 245 or 500 DPS to get a finer resolution. An accelerometer measures acceleration. The LSM9DS0 measures its acceleration in g's, and its scale can be set to either $\pm$ 2, 4, 6, 8, or 16g. There's the magnetometer, which measures the power and direction of magnetic fields. The LSM9DS0 measures magnetic fields in units of gauss (Gs), and can set its measurement scale to either $\pm$ 2, 4, 8, or 12 Gs. Ultrasonic sensors measure distance by using ultrasonic waves. The sensor head emits an ultrasonic wave and receives the wave reflected back from the target. Ultrasonic sensors measure the distance to the target by measuring the time between the emission and reception. The ultrasonic sensor we used for the quadrotor platform is HRLV-EZ1 [87]. This ultrasonic sensor detects objects from 1mm to 5meters, senses range to objects from 30cm to 5meters, with large objects closer than 30cm typically reported as 30cm. The interface output formats are pulse width, analog voltage, and serial digital in either RS232 or TTL. Fig. 5.6 shows the selected sensors.

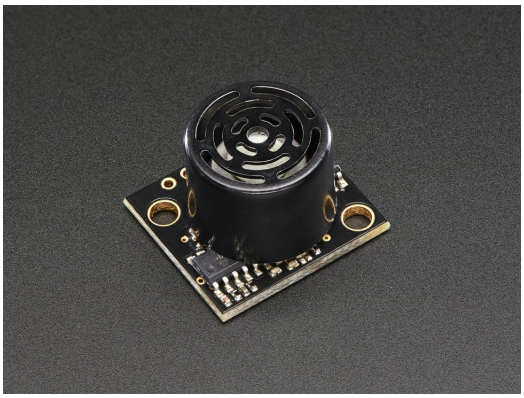
5.4 Manual Control System and Wireless Communication Module

The manual control system has a RC radio receiver on-board which can receive the manual control signal from ground. Although when doing experiments, the drone is controlled by on-board computers, is also monitoring the manual control signal for safety. In emergency situation, researchers can switch to manual control mode immediately to avoid damages to the

(a) ODROID USB-CAM 720P.



(b) LSM9DS0 module.



(c) HRLV-EZ1.

Figure 5.6: On-board sensor system.

platform or environment. The manual control system we choose for the quadrotor platform is DX18 by Spektrum which provides enough control channels for quadrotor. Fig. 5.7 shows the transmitter and the on-board receiver.

The wireless communication module can send interested flight data to the ground in real-time. The communication module we choose for the quadrotor platform is XBee Pro S1 [88]. XBee Pro S1 take the 802.15.4 stack (the basis for Zigbee) and wrap it into a simple to use serial command set. It allow a very reliable and simple communication between micro-controllers, computers, systems, really anything with a serial port. Point to point and multi-point networks are supported. Fig. 5.8 shows the XBee Pro S1 communication module.

(a) Transmitter. (b) On-board receiver.

Figure 5.7: Manual control system.

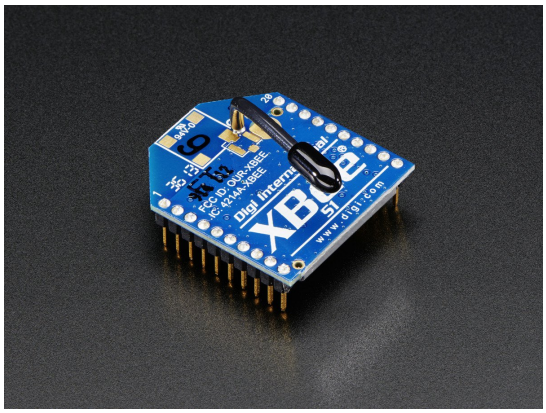


Figure 5.8: XBee Pro S1 communication module.

5.5 Two-level On-board Processor System

To optimize the on-board computation resource, the on-board processor system include a high-level processor and a low-level processor. We name it two-level processor system. The high-level processor in this design is ODROID-XU4 [89], while the low-level processor in this design is STM32F103ZET6 [90]. The high-level process run the algorithms for application. A Ubuntu operation system is equipped. At the same time, Robot Operating System (ROS) is also embedded to access on-line available ROS packages easily. Researches can also make their own packages which contain their own algorithms. Another function for high-level processor is to

communicate with users in real-time through wireless communication. The low level processor is responsible for sensor data acquiring, synchronization, and motor speed controls.

ODROID-XU4 is a new generation of computing device with more powerful, energy-efficient hardware and a smaller form factor. Offering open source support, the board can run various flavors of Linux, including the latest Ubuntu 16.04 and Android 4.4 KitKat and 7.1 Nougat. By implementing the eMMC 5.0, USB 3.0 and Gigabit Ethernet interfaces, the ODROID-XU4 boasts amazing data transfer speeds, a feature that is increasingly required to support advanced processing power on ARM devices. This allows users to truly experience an upgrade in computing, especially with faster booting, web browsing, networking and 3D games [89]. Fig. 5.9 shows the on-board ODROID-XU4 processor.

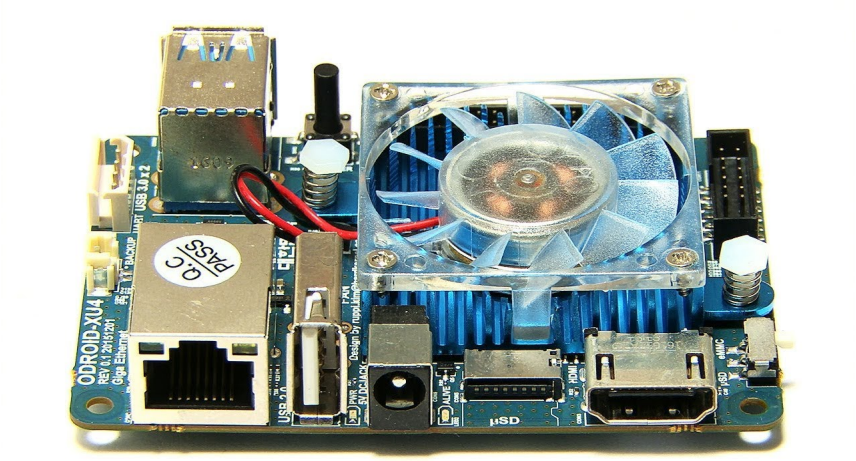


Figure 5.9: ODROID-XU4.

STM32 is a family of 32-bit micro-controller integrated circuits by STMicroelectronics. The STM32 chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M7F, Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each micro-controller consists of the processor core, static RAM, flash memory, debugging interface, and various peripherals [91]. Fig. 5.10 shows the on-board STM32F103ZET6 processor minimum system board.

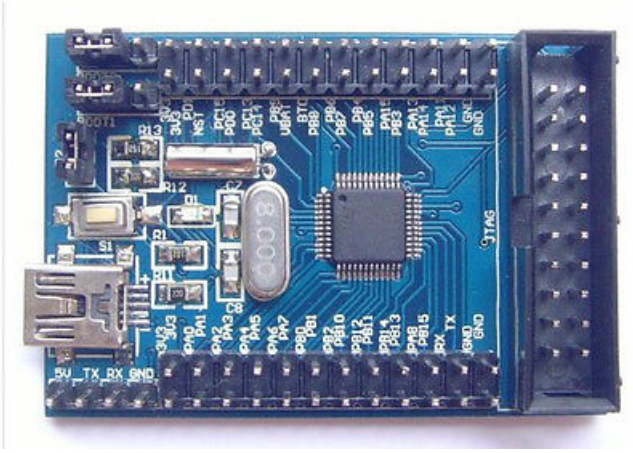


Figure 5.10: STM32F103ZET6 processor minimum system board.

5.6 System Block Diagram

We draw system block diagram in Fig. 5.11. And Fig. 5.12 shows the complete quadrotor platform.

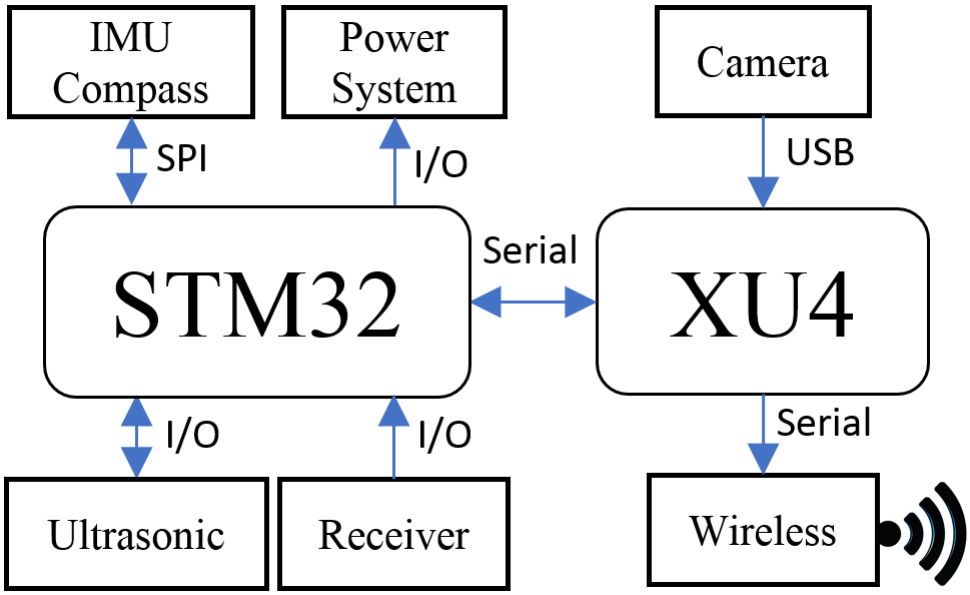
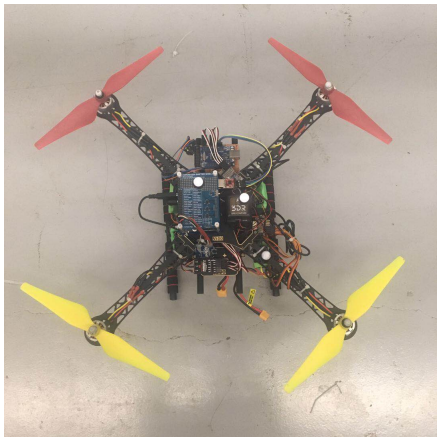


Figure 5.11: System block diagram.

5.7 Lesson Learned from the Design

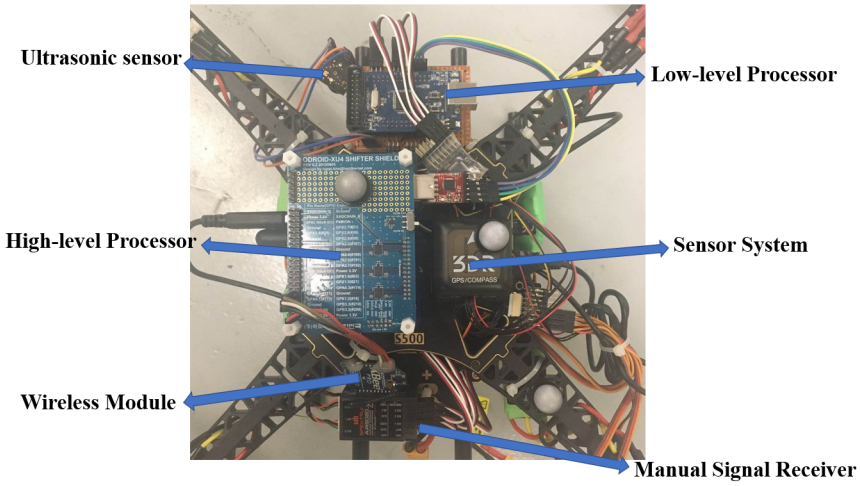
When the system is built, an important step before use the sensor data and image information is synchronization. The image from camera is acquired by the high-level processor, while the other sensor data is acquired by the low-level processor. Since the clock signals for the high-level and low-level processor are different, with the serial data port delay, the time stamps for these information are different. After synchronization, the information update rate will be slower than the slowest updated sensor. For comparison purpose, in this thesis, besides the sensors data and the image flow, Vicon data is also synchronized for reference. While the Vicon data is sent by wireless, the delay is inconsistent due to unstable wireless disturbance.

Currently, the synchronization tool this thesis used the ROS package 'message_filter' presented in Chapter 3. A problem not solved yet is because wireless delay of Vicon data sending, the synchronized information does not output with a constant frequency, which cause problem when use.

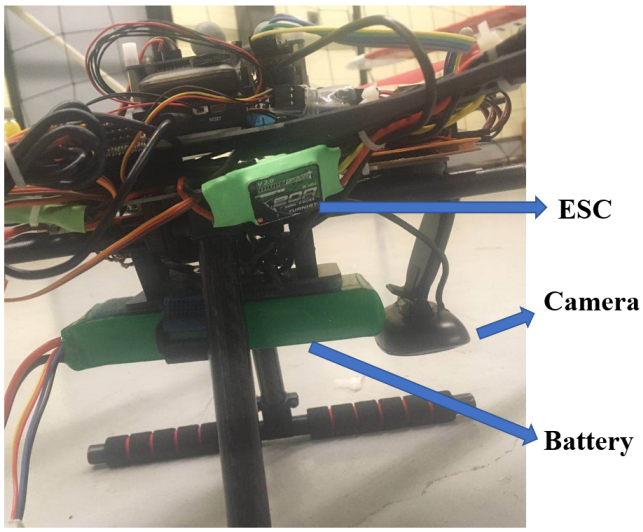


(a) Top view.

(b) Side view.



(c) Top view in detail.



(d) Side view in detail.

Figure 5.12: Quadrotor research platform.

Chapter 6

Conclusions and Future Work

6.1 Summary and conclusions

Motivated by more and more small VTOL UAV applications under GPS-Denied environment with on-board sensors only, this thesis studies relevant advanced state estimation and navigation techniques. First, a robust on-board monocular vision based pattern tracking algorithm that provides a quadrotor with the capability to autonomously track and land on a vessel deck platform is presented. Second, a novel solution to on-board sensor only feature-based monocular 3-D SLAM algorithm is developed. At last, a K Nearest Neighborhood learning based wind estimation method which does not require the details of the aerodynamic information is provided.

In the work presented in Chapter 2, an autonomous approach for a quadrotor to land on a vessel deck from long range is presented. On-board IMU and altimeter, image processing, and Kalman filters are used to provide the state estimation. Experiments are implemented on a self-designed landing platform, which can emulate the deck motion of a vessel 30 m wide under sea state 6. The drone first recognize the red LEDs applied on the landing board from long range, then estimate the relevant distance and directions and drive automatically approach the landing board. Once the drone hover above the landing board, the bottom camera start to work and recognize the designed 'H' landing sign, adjust the position and land on the board. Flight results demonstrate the landing success rate of 100% with a 9 cm average distance error.

In the work presented in Chapter 3, this thesis provide an new approach to on-board sensor only feature-based monocular 3-D SLAM algorithm. The pin-hole camera model is derived in an alternative prospective and the proposed algorithm combines the IMU measurements inherently. Experimental results shows that the proposed method can generate the absolute map immediately without artificial landmarks nor prior knowledge about the environment. With the

positions of the feature points calculated, the camera's position estimation is also improved compares to using only IMU, which proves the effeteness of the proposed SLAM algorithm. The mapping and positioning results are sensitive to feature matching and distance movement measurements. This thesis use low cost Parrot AR.Drone as our VTOL UAV fro experiments. Experimental results shows that the proposed SLAM algorithm is able to reconstruct absolute 3-D map of the environment and provide the drone's position in real-time. At the same time, the positioning accuracy is better than PTAM using same experimental condition.

In the work presented in Chapter 4, a KNN-based wind estimation method for a rotary-wing VTOL UAV is presented. The proposed method uses the features generated by flight data as the learning model, and does not require the details of the aerodynamic information. Also, this method only uses flight data to generate the designed features for the training and wind estimation stages, and does not require direct wind speed measurements. Experiment results show that the proposed method can estimate the wind accurately, with a 0.074 m/s average estimation error, and a 3.3% average estimation relative error, while the average estimation error generated by AR.Drone's embedded system is 1.673 m/s.

In the work presented in Chapter 5, a quadrotor hardware solution which is able to support majority of research of small VTOL UAVs is developed. The quadrotor research platform equipped two-level processor system which is able to process all algorithms on-board. With the sensor system embedded, rich flight information is provided to the researcher that can enable endless possibilities. Relied on the on-board wireless communication module, the flight state and important parameters can be monitored by researchers in real-time, which makes the platform humanized.

6.2 Recommendations for future work

Based on the studies of guidance and navigation of small VTOL UAVs to solve using on-board sensor system under GPS-Denied environment, this thesis opens several avenues for improvement.

(i) In Chapter 2, the moving deck simulates the waves of the sea, but it does not have horizontal movements. To better simulate a ship motion in the sea, it is useful to resident

the landing platform on a ground vehicle which provides additional horizontal motion of the deck. Then the landing problem will be more challenging. Additionally, wind disturbance shall be considered to make the algorithms more robust.

(ii) For visual SLAM algorithms, all feature-based visual SLAM algorithms require accurate feature matching. If the matching is wrong, there will be error measurements for both positioning and mapping. To make the feature-based visual SLAM algorithm more robust, feature mismatching recognition and elimination algorithms are important to be studied. Another constraint for feature-based visual SLAM algorithm is the movements of the camera. If the camera stays stationary, the image provides no information. The program should recognize this situation and handle properly to avoid mapping error.

Additionally, in the work presented in Chapter 3, the proposed SLAM algorithm requires accurate estimation of the feature's depth at moment 1 (z_{b1}). This requires accurate estimation for the position change between two steps. In future, visual odometer or other advanced position change estimation technique can be integrated to estimate z_{b1} more accurately and improve the robustness.

Another future direction is to track the map continuously. When the map generated, the algorithm directly output the results without tracking. Because of that, although the drift of positioning has been inhibited significantly, it still exist. If the map is tracking continuously, the positioning drift problem can be solved.

(iii) For the wind estimation experiments, the direction of the wind can also be considered to make the research more realistic. Secondly, optimal selection of the features related to the wind with directions shall be studied. At last, experiments in outdoor environments shall be considered to validate the results.

(iv) For small VTOL UAV design, currently, the camera image information and other sensor information are output separately. Researchers should synchronize the information before use. The on-board sensor data acquiring system shall be synchronized when output. An important research direction is embed the image acquiring on the low-level processor. Another promising research direction is to develop more powerful battery or other power source to make the fight time longer. At the same time, the weight of power system shall

be reduced to give more freedom for on-board sensors and processors. Another direction is try to make the UAV smaller size to enable more challenged application such as indoor tasks.

References

[1] Y. Zeng, R. Zhang, and T. J. Lim, "Wireless communications with unmanned aerial vehicles: Opportunities and challenges," *IEEE Communications Magazine*, vol. 54, no. 5, pp. 36–42, 2016.

[2] Q. Wu, Y. Zeng, and R. Zhang, "Joint trajectory and communication design for multi-uav enabled wireless networks," *IEEE Transactions on Wireless Communications*, vol. 17, no. 3, pp. 2109–2121, 2018.

[3] M. R. James and S. Robson, "Mitigating systematic error in topographic models derived from uav and ground-based image networks," *Earth Surface Processes and Landforms*, vol. 39, no. 10, pp. 1413–1420, 2014.

[4] S. P. Bemis, S. Micklethwaite, D. Turner, M. R. James, S. Akciz, S. T. Thiele, and H. A. Bangash, "Ground-based and uav-based photogrammetry: A multi-scale, high-resolution mapping tool for structural geology and paleoseismology," *Journal of Structural Geology*, vol. 69, pp. 163–178, 2014.

[5] S. M. Adams and C. J. Friedland, "A survey of unmanned aerial vehicle (uav) usage for imagery collection in disaster research and management," in *9th International Workshop on Remote Sensing for Disaster Response*, vol. 8, 2011.

[6] W. DeBusk, "Unmanned aerial vehicle systems for disaster relief: Tornado alley," in *AIAA Infotech@ Aerospace 2010*, 2010, p. 3506.

[7] J. Uhrmann, R. Strenzke, and A. Schulte, "Task-based guidance of multiple detached unmanned sensor platforms in military helicopter operations," *COGIS, Crawley*, 2010.

[8] G. B. A. Ronconi, T. J. Batista, and V. Merola, "The utilization of unmanned aerial vehicles (uav) for military action in foreign airspace," *UFRGSMUN: UFRGS Model United Nations Journal*, pp. 137–182, 2014.

[9] D. Matrice, "100," *On-line: www. dji. com/matrice100, Accessed*, vol. 19, p. 12, 2017.

[10] V. Puri, A. Nayyar, and L. Raja, "Agriculture drones: A modern breakthrough in precision agriculture," *Journal of Statistics and Management Systems*, vol. 20, no. 4, pp. 507–518, 2017.

[11] "Ar.drone 2.0 elite edition," https://www.parrot.com/global/drones/parrot-ardrone-20-elite-editionparrot-ardrone-20-elite-edition, accessed April 20, 2019.

[12] P.-J. Bristeau, F. Callou, D. Vissiere, and N. Petit, "The navigation and control technology inside the ar. drone micro uav," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 1477–1484, 2011.

[13] J. J. Lugo and A. Zell, "Framework for autonomous on-board navigation with the ar. drone," *Journal of Intelligent & Robotic Systems*, vol. 73, no. 1-4, pp. 401–412, 2014.

[14] L. V. Santana, A. S. Brandao, M. Sarcinelli-Filho, and R. Carelli, "A trajectory tracking and 3d positioning controller for the ar. drone quadrotor," in *2014 international conference on unmanned aircraft systems (ICUAS)*. IEEE, 2014, pp. 756–767.

[15] N. Dijkshoorn, "Simultaneous localization and mapping with the ar. drone," *PhD diss., Masters thesis, Universiteit van Amsterdam*, 2012.

[16] J.-S. Pleban, R. Band, and R. Creutzburg, "Hacking and securing the ar. drone 2.0 quadcopter: investigations for improving the security of a toy," in *Mobile Devices and Multimedia: Enabling Technologies, Algorithms, and Applications 2014*, vol. 9030. International Society for Optics and Photonics, 2014, p. 90300L.

[17] J. Borenstein, L. Ojeda, and S. Kwanmuang, "Heuristic reduction of gyro drift for personnel tracking systems," *The Journal of Navigation*, vol. 62, no. 1, pp. 41–58, 2009.

[18] A. Carullo and M. Parvis, "An ultrasonic sensor for distance measurement in automotive applications," *IEEE Sensors journal*, vol. 1, no. 2, p. 143, 2001.

[19] D. Jaggar, "Arm architecture and systems," *IEEE micro*, vol. 17, no. 4, pp. 9–11, 1997.

[20] T. Krajník, V. Vonásek, D. Fišer, and J. Faigl, "Ar-drone as a platform for robotic research and education," in *International Conference on Research and Education in Robotics*. Springer, 2011, pp. 172–186.

[21] I. Sa, M. Kamel, R. Khanna, M. Popovic, J. Nieto, and R. Siegwart, "Dynamic system identification, and control for a cost effective open-source vtol mav," *arXiv preprint arXiv:1701.08623*, 2017.

[22] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[23] D. Nistér, O. Naroditsky, and J. Bergen, "Visual odometry," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 1. Ieee, 2004, pp. I–I.

[24] J. Artieda, J. M. Sebastian, P. Campoy, J. F. Correa, I. F. Mondragón, C. Martínez, and M. Olivares, "Visual 3-d slam from uavs," *Journal of Intelligent and Robotic Systems*, vol. 55, no. 4-5, p. 299, 2009.

[25] M. Maimone, Y. Cheng, and L. Matthies, "Two years of visual odometry on the mars exploration rovers," *Journal of Field Robotics*, vol. 24, no. 3, pp. 169–186, 2007.

[26] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[27] 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.

[28] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[29] P. G. Savage, "Strapdown inertial navigation integration algorithm design part 1: Attitude algorithms," *Journal of guidance, control, and dynamics*, vol. 21, no. 1, pp. 19–28, 1998.

[30] ——, "Strapdown inertial navigation integration algorithm design part 2: Velocity and position algorithms," *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 208–221, 1998.

[31] R. W. Levi and T. Judd, "Dead reckoning navigational system using accelerometer to measure foot impacts," Dec. 10 1996, uS Patent 5,583,776.

[32] S. Jin, J. Zhang, L. Shen, and T. Li, "On-board vision autonomous landing techniques for quadrotor: A survey," in *Control Conference (CCC), 2016 35th Chinese*. IEEE, 2016, pp. 10 284–10 289.

[33] K. Ling, D. Chow, A. Das, and S. L. Waslander, "Autonomous maritime landings for low-cost vtol aerial vehicles," in *Computer and Robot Vision (CRV), 2014 Canadian Conference on*. IEEE, 2014, pp. 32–39.

[34] J. M. Daly, Y. Ma, and S. L. Waslander, "Coordinated landing of a quadrotor on a skid-steered ground vehicle in the presence of time delays," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 4961–4966.

[35] J. L. Sanchez-Lopez, J. Pestana, S. Saripalli, and P. Campoy, "An approach toward visual autonomous ship board landing of a vtol uav," *Journal of Intelligent & Robotic Systems*, vol. 74, no. 1-2, pp. 113–127, 2014.

[36] A. Gautam, P. Sujit, and S. Saripalli, "A survey of autonomous landing techniques for uavs," in *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*. IEEE, 2014, pp. 1210–1218.

[37] D. Pebrianti, F. Kendoul, S. Azrad, W. Wei, and K. Nonami, "Autonomous hovering and landing of a quad-rotor micro aerial vehicle by means of on ground stereo vision system," *Journal of System Design and Dynamics*, vol. 4, no. 2, pp. 269–284, 2010.

[38] W. Kong, D. Zhang, X. Wang, Z. Xian, and J. Zhang, "Autonomous landing of an uav with a ground-based actuated infrared stereo vision system," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 2963–2970.

[39] W. Kong, D. Zhou, D. Zhang, and J. Zhang, "Vision-based autonomous landing system for unmanned aerial vehicle: A survey," in *Multisensor Fusion and Information Integration for Intelligent Systems (MFI), 2014 International Conference on*. IEEE, 2014, pp. 1–8.

[40] S. T. Barnard and M. A. Fischler, "Computational stereo," *ACM Computing Surveys (CSUR)*, vol. 14, no. 4, pp. 553–572, 1982.

[41] R. Strydom, S. Thurrowgood, A. Denuelle, and M. V. Srinivasan, "Uav guidance: A stereo-based technique for interception of stationary or moving targets," in *Conference Towards Autonomous Robotic Systems*. Springer, 2015, pp. 258–269.

[42] X. Chen, S. K. Phang, M. Shan, and B. M. Chen, "System integration of a vision-guided uav for autonomous landing on moving platform," in *Control and Automation (ICCA), 2016 12th IEEE International Conference on*. IEEE, 2016, pp. 761–766.

[43] A. Benini, M. J. Rutherford, and K. P. Valavanis, "Real-time, gpu-based pose estimation of a uav for autonomous takeoff and landing," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 3463–3470.

[44] F. Cocchioni, E. Frontoni, G. Ippoliti, S. Longhi, A. Mancini, and P. Zingaretti, "Visual based landing for an unmanned quadrotor," *Journal of Intelligent & Robotic Systems*, vol. 84, no. 1-4, pp. 511–528, 2016.

[45] A. Benini, A. Mancini, and S. Longhi, "An imu/uwb/vision-based extended kalman filter for mini-uav localization in indoor environment using 802.15. 4a wireless sensor network," *Journal of Intelligent & Robotic Systems*, vol. 70, no. 1-4, pp. 461–476, 2013.

[46] J.-i. Meguro, T. Murata, J.-i. Takiguchi, Y. Amano, and T. Hashizume, "Gps multipath mitigation for urban area using omnidirectional infrared camera," *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 1, pp. 22–30, 2009.

[47] K. E. Wenzel, A. Masselli, and A. Zell, "Automatic take off, tracking and landing of a miniature uav on a moving carrier vehicle," *Journal of intelligent & robotic systems*, vol. 61, no. 1, pp. 221–238, 2011.

[48] B. Hu, L. Lu, and S. Mishra, "Fast, safe and precise landing of a quadrotor on an oscillating platform," in *American Control Conference (ACC), 2015*. IEEE, 2015, pp. 3836–3841.

[49] J. Dougherty, D. Lee, and T. Lee, "Laser-based guidance of a quadrotor uav for precise landing on an inclined surface," in *American Control Conference (ACC), 2014*. IEEE, 2014, pp. 1210–1215.

[50] P. I. T. M. Das, S. Swami, and J. M. Conrad, "An algorithm for landing a quadrotor unmanned aerial vehicle on an oscillating surface," in *Southeastcon, 2012 Proceedings of IEEE*. IEEE, 2012, pp. 1–4.

[51] T. Venugopalan, T. Taher, and G. Barbastathis, "Autonomous landing of an unmanned aerial vehicle on an autonomous marine vehicle," in *Oceans, 2012*. IEEE, 2012, pp. 1–9.

[52] S. M. Chaves, R. W. Wolcott, and R. M. Eustice, "Neec research: Toward gps-denied landing of unmanned aerial vehicles on ships at sea," *Naval Engineers Journal*, vol. 127, no. 1, pp. 23–35, 2015.

[53] M. Garratt, H. Pota, A. Lambert, S. ECKERSLEY-MASLIN, and C. Farabet, "Visual tracking and lidar relative positioning for automated launch and recovery of an unmanned rotorcraft from ships at sea," *Naval Engineers Journal*, vol. 121, no. 2, pp. 99–110, 2009.

[54] Å. Björck, *Numerical methods for least squares problems*. SIAM, 1996.

[55] O. A. Yakimenko, I. I. Kaminer, W. J. Lentz, and P. Ghyzel, "Unmanned aircraft navigation for shipboard landing using infrared vision," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 38, no. 4, pp. 1181–1200, 2002.

[56] N. Michael, S. Shen, K. Mohta, Y. Mulgaonkar, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida *et al.*, "Collaborative mapping of an earthquake-damaged building via ground and aerial robots," *Journal of Field Robotics*, vol. 29, no. 5, pp. 832–841, 2012.

[57] L. von Stumberg, V. Usenko, J. Engel, J. Stückler, and D. Cremers, "From monocular slam to autonomous drone exploration," in *2017 European Conference on Mobile Robots (ECMR)*. IEEE, 2017, pp. 1–8.

[58] M. Irani and P. Anandan, "About direct methods," in *International Workshop on Vision Algorithms*. Springer, 1999, pp. 267–277.

[59] A. J. Davison, "Real-time simultaneous localisation and mapping with a single camera." in *Iccv*, vol. 3, 2003, pp. 1403–1410.

[60] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual slam algorithms: A survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, no. 1, p. 16, 2017.

[61] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*. IEEE Computer Society, 2007, pp. 1–10.

[62] D. Nistér, "An efficient solution to the five-point relative pose problem," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 6, pp. 0756–777, 2004.

[63] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[64] P. Piniés, T. Lupton, S. Sukkarieh, and J. D. Tardós, "Inertial aiding of inverse depth slam using a monocular camera," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 2797–2802.

[65] S.-H. Jung and C. J. Taylor, "Camera trajectory estimation using inertial sensor measurements and structure from motion results," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 2. IEEE, 2001, pp. II–II.

[66] G. Nützi, S. Weiss, D. Scaramuzza, and R. Siegwart, "Fusion of imu and vision for absolute scale estimation in monocular slam," *Journal of intelligent & robotic systems*, vol. 61, no. 1-4, pp. 287–299, 2011.

[67] J. Engel, J. Sturm, and D. Cremers, "Camera-based navigation of a low-cost quadrocopter," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 2815–2821.

[68] D. Gehrig, M. Göttgens, B. Paden, and E. Frazzoli, "Scale-corrected monocular-slam for the ar. drone 2.0," 2017.

[69] P. Sturm, "Pinhole camera model," *Computer Vision: A Reference Guide*, pp. 610–613, 2014.

[70] M. Pecka, "Message filters overview and summary," http://wiki.ros.org/message_filters/, accessed April 4, 2019.

[71] opencv dev team, "Feature matching with flann," https://docs.opencv.org/3.0-beta/doc/tutorials/features2d/feature_flann_matcher/feature_flann_matcher.htmlfeature-flann-matcher, accessed April 5, 2019.

[72] JakobEngel, "The tum_ardrone package summary," http://wiki.ros.org/tum_ardrone, accessed April 18, 2019.

[73] Z. Tan, J. Dong, Y. Xiao, and J. Tu, "A numerical study of diurnally varying surface temperature on flow patterns and pollutant dispersion in street canyons," *Atmospheric Environment*, vol. 104, pp. 217–227, 2015.

[74] C.-x. Pan, J.-z. Zhang, L.-f. Ren, and Y. Shan, "Effects of rotor downwash on exhaust plume flow and helicopter infrared signature," *Applied thermal engineering*, vol. 65, no. 1-2, pp. 135–149, 2014.

[75] S. Prudden, A. Fisher, M. Marino, A. Mohamed, S. Watkins, and G. Wild, "Measuring wind with small unmanned aircraft systems," *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 176, pp. 197–210, 2018.

[76] J. A. Lusardi, M. B. Tischler, C. L. Blanken, and S. J. Labows, "Empirically derived helicopter response model and control system requirements for flight in turbulence," *Journal of the American Helicopter Society*, vol. 49, no. 3, pp. 340–349, 2004.

[77] N. d. Divitiis, "Wind estimation on a lightweight vertical-takeoff-and-landing uninhabited vehicle," *Journal of aircraft*, vol. 40, no. 4, pp. 759–767, 2003.

[78] J. Velasco-Carrau, S. García-Nieto, J. Salcedo, and R. H. Bishop, "Multi-objective optimization for wind estimation and aircraft model identification," *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 2, pp. 372–389, 2015.

[79] S. Pappu, Y. Liu, J. F. Horn, and J. Copper, "Wind gust estimation on a small vtol uav," in *AHS Technical Meeting on VTOL Unmanned Aircraft Systems and Autonomy*, 2017.

[80] L. Sikkel, G. de Croon, C. De Wagter, and Q. Chu, "A novel online model-based wind estimation approach for quadrotor micro air vehicles using low cost mems imus," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 2141–2146.

[81] D. T. Larose, "k-nearest neighbor algorithm," *Discovering knowledge in data: An introduction to data mining*, pp. 90–106, 2005.

[82] K. O. McGraw and S. P. Wong, "Forming inferences about some intraclass correlation coefficients." *Psychological methods*, vol. 1, no. 1, p. 30, 1996.

[83] B. Scrosati, K. Abraham, W. A. van Schalkwijk, and J. Hassoun, *Lithium batteries: advanced technologies and applications*. John Wiley & Sons, 2013, vol. 58.

[84] K. Shimohara, "Pulse width modulation signal generating circuit," Mar. 29 1994, uS Patent 5,298,871.

[85] "Odroid usb-cam 720p," https://www.hardkernel.com/shop/odroid-usb-cam-720p/, accessed April 20, 2019.

[86] "Lsm9ds0 datasheet," https://cdn-shop.adafruit.com/datasheets/LSM9DS0.pdf, accessed April 20, 2019.

[87] "Hrlv-ez1 description," https://www.adafruit.com/product/984, accessed April 20, 2019.

[88] “Xbee datasheet,” https://www.digi.com/resources/documentation/digidocs/.../90000982.pdf, accessed April 20, 2019.

[89] “Odroid-xu4,” https://wiki.odroid.com/odroid-xu4/odroid-xu4, accessed April 20, 2019.

[90] “Stm32 arm processor,” https://en.wikipedia.org/wiki/STM32, accessed April 20, 2019.

[91] “Stm32 overview,” https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html, accessed April 20, 2019.