HKUST IARC Team Progress Report

YANG Shuo, YU Yun, ZHOU Zi, YING Jiahang, LIU Wenxin, WANG Mingxi

***Abstract*—This paper reports the current preparation progress of HKUST IARC Team. We treat Mission 7 as a final destination on a big road map. To achieve this final goal we must first solve some smaller problems including iRobot tracking, obstacle avoidance, localization without landmarks and so on. After these problems are solved, proper integration is needed to make the final system. At the beginning stage, we focus on solving iRobot tracking and UAV localization subproblems. We devise algorithms that use five cameras to do the tracking and localization work. Preliminary results showed the robustness of our algorithms.**

I. Introduction

After the initial release of the rules for Mission 7, HKUST IARC Team started to analysis the problems to be solved and finally divided Mission 7 into several subproblems.

The basic functionality of an aerial robot is attitude control. Though quadrotor control has already become a standard technique instead of a research topic, we still implement a new control algorithm rather than use existing ones. Our new algorithm adopts exponential mapping rotation parametrization and considers the dynamic model of the aerial robot. Therefore our aerial robot has faster action and more precision attitude control.

In parallel with the study of attitude control, we target on subproblems that are prerequisites of sophisticated behaviors required by Mission 7. These subproblems are: iRobot detection and tracking, obstacle avoidance, and localization without landmarks. We can only proceed when these three subproblems are solved efficiently and robustly.

Obstacle avoidance is the first subproblem we tackled and partially solved. Various obstacle avoidance algorithms are presented in literatures. The famous [3] is proved to be an efficient solution. It lead to VFH+ [23] and VFH* [24] where improved algorithms can handle more complicated situations. VFH algorithms are efficient for ultrasonic sensors and LIDAR, but they can only deal with 2D environment. In recent years, since inexpensive and on-board vision system becomes available, people started to use cameras to do obstacle navigation. A direct way is generate disparity image by stereo epipolar geometry [15]. Once the disparity image of cameras is obtained, the rest work is conceive vehicle control law according it [7], [15], [16]. Another vision-based method is detecting obstacles by image feature tracking using multiple view geometry [6], where obstacles are represented by features identified from multiple consecutive images. All the early methods only consider local information that obtained in a short period of time, such as one scan of LIDAR or a couple of sequential images. While in past years, as SLAM techniques becoming mature, obstacle avoidance with a given global map is possible [2], [13]. Information gathered from sensors will first be assimilated into a 3D map. And then the problem of generating dodging actions will become a path planning problem. Our method takes in the main ideas of existing methods. We need a global map for action generation, while the algorithm must be computational efficient to save resources for other tasks. Thus we choose LIDAR as detection sensor, and use map-based control law for obstacle avoidance.

iRobot detection and tracking is related to object tracking and visual servoing research. We not only need to detect the relative position of a iRobot to the aerial robot, but also get its orientation and status, namely whether it is moving forward or self rotating. To make the aerial robot able to touch the top plate of a iRobot, the key technique is visual servoing [8]. When hovering over the iRobot, the aerial robot must have instant and precise response to iRobot position changing. So the first task is get the exact position of the iRobot. Because the size of the top plate of iRobot is known, calculation of the position of iRobot in camera frame is a "Prospective n-point" (PnP) problem [9]. PnP problem was extensively studied for more than twenty years. There are many efficient algorithms been developed [10], [19]. Despite PnP problem itself is well studied, due to camera image distortion and noise, the result of PnP solution is not accurate enough to be directly used as true robot position. So we implemented a UKF-based position estimator to reduce measurement noise. Afther the position is robustly estimated, we have a cascade PID controller to let the aerial robot to track the iRobot.

Localization without landmarks, in the sepcial setting of Mission 7, is hard to find existing solutions in literature. The Mission 7 rules suggest using optical flow [14] to do localization. But to our knowledge, no known optical flow algorithm is able to meet the requirement of this mission. Optical flow is build on several assumptions that usually been violated easily in UAV working environment. Visual odometry [21] may be another feasible localization solution. However, after studied some visual odometry algorithms, we found that even if the local measurement error of the visual odometer is minimized, it will still accumulate and blow up when the UAV travels a long distance, so the visual odometer cannot be trusted globaly. To tackle this problem, we proposed a novel method to get global position in the arena. This method extract features from images of area boundaries as landmarks, and localize the aerial robot by solving geometric problem and estiamtion problem.

This paper is organized as follows. In section II, we discuss our aerial robot mechanical design. Section III describes our aerial robot attitude control algorithm. Section IV presents the work of tracking iRobot. Section V devotes to the localization algorithm. And in section VI we illustrate the whole software architecture and briefly talked about the obstacle avoidance algorithm. Finally a conclusion is given and future work will be discussed.

Fig. 1: 3D view of the aerial robot frame design

II. Mechanical Design

To design a good aerial robot frame, we first list several robot mechanical requirements.

Takeoff Weight	3kg
Flight Duration	10 minutes
Maximum Horizontal Speed	15 m/s
Rotor Shaft Distance	500 mm to 600 mm
Number Of Cameras	Five (One downward and the other four on sides of the robot)

These requirements are the result of balancing a number of factors. First of all, the robot must be able to carry more than 1000g payload, including LIDAR sensor, cameras, computing platforms and communication devices. Second, the robot should have high agility in order to cross the huge game arena in a few seconds and brake quickly. According to [17], the shorter the rotor shaft distance, the more agile the robot be. Taking the propeller blade size into account, we choose the shaft distance to be 500mm to 600mm. Third, the chassis have to be properly designed to contain the payload. Cameras must have clear field of view (FOV). Sensors must be protected from vibration and external hit. And since the shaft distance is small, the chassis must be compact.

Our preliminary design is satisfactory, as shown in Figure 1. The frame is made of carbon fiber, weighting only 600 gram. The robot, propeller and ESC system is DJI E600 Tuned Propulsion System. With a 8000mAh Li-Po battery, the aerial robot is able to hover 15 minutes with 3.2kg take off weight.

III. Attitude Control on SO(3)

In our attitude control algorithm, we try to balance model precision and engineering realizability. Dynamics of rotors and propellers are extremely difficult to be correctly modeled, thus we just use a measured speed-to-thrust function to convert electronic speed controller (ESC) signal to propeller torque. As for rotation representation, we choose the exponential map from $\mathfrak{so}(3)$ to $SO(3)$ instead of popular Euler Angle in order to eliminate model ambiguity, because Euler Angle has intrinsic singularity points but the exponential map can avoid singularities [11].

A. Model of Aerial Robot Kinematics & Dynamics

Our robot model is a standard quadrotor system contains four identical rotors and two sets of propellers. Each propeller

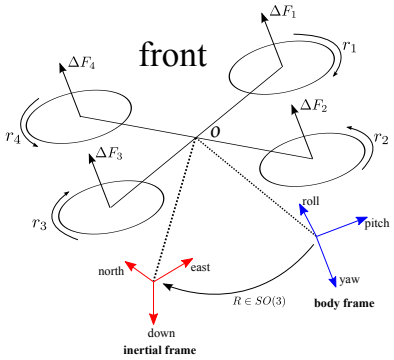


Fig. 2: The model of the aerial robot. Red frame is the inertial frame and blue frame is the body frame. Both frames ought to coincide with the origin O. They are separated here only to illustrate axis directions.

generates torque ΔF_i perpendicular to the plane on which the robot base resides. We choose the local north-east-down (NED) coordinate frame as the inertial frame, and the roll-pitch-yaw coordinate frame as the body frame. Since we do not consider translation in this case, both the inertial frame and the body frame have their origins located at the center of mass of the robot. Our aerial robot has X shape so roll axis equally divide the angle between the arm of motor 1 and the arm of motor 4. Directions of frame axes are shown in Figure 2.

Based on this model, we define:

$R \in SO(3)$, the rotation of the aerial robot expressed as frame transform from the body frame to the inertial frame;

$J \in \mathbb{R}^{3\times3}$, the inertia matrix expressed in the body frame;

$\omega^b \in \mathbb{R}^3$, the angular velocity in the body frame;

$\tau \in \mathbb{R}^3$, the control torque generated by the actuators of the robot, expressed in body frame as well.

With above definition, we present robot kinematic as a first order system

$$\dot{R} = R\hat{\omega}^b \tag{1}$$

which is deduced from the basic definitions and results of matrix Lie group $SO(3)$ [20].

Then by Newton-Euler equations we can write the robot dynamics as

$$J\dot{\omega}^b + \omega^b \times J\omega^b = \tau + \Delta \tag{2}$$

where $\Delta \in \mathbb{R}^3$ is a disturbance term.

B. Control of The Aerial Robot

The control goal of our algorithm is to move the current rotation R_c to a target rotation R_t. To explain the control law we need first introduce the logarithmic map from $SO(3)$ to $\mathfrak{so}(3)$.

For $R \in SO(3)$ satisfies $tr(R) \neq -1$, we define

$$log(R) = \frac{\phi}{2sin\phi}(R - R^T) \qquad \in \mathfrak{so}(3)$$

in which ϕ is $cos^{-1}(tr(R) - 1)$ and $|\phi| < \pi$. For a detailed treatment to lie group and lie algebra, we refer readers to [12].

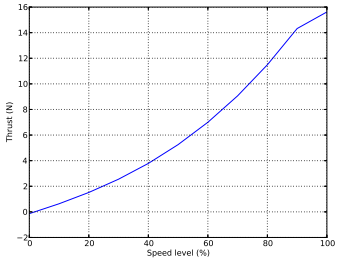


Fig. 3: Relationship between propeller speed and thrust force. The curve is neither linear nor quadratic.

According to [5], given Equation 1, the velocity control law for tracking the target rotation is

$$\hat{\omega}^b = k_p log(R_c^{-1} R_t) \tag{3}$$

with k_p be the propositional control gain.

If the target rotation is $I \in SO(3)$, then the tracking problem is reduced to an regulating problem with the velocity control law being

$$\hat{\omega}^b = k_p log(R_c^{-1})$$

The stability of the above control law is guaranteed if we choose a natural candidate Lyapunov function

$$W(R) = \frac{1}{2}\|R\|^2.$$

Since $\hat{\omega}^b$ is controllable, our control law results in

$$\dot{W}(R(t)) = -2k_p W \text{ [5]}$$

So the exponential stability is confirmed.

The control law only produce a matrix $\hat{\omega}^b$, from it we can extract desired angular velocity w_d^b. Then we employ three cascade PID controllers to control the angular velocity of the robot w^b to this desired angular velocity w_d^b. Each term of w_d, namely pitch velocity, roll velocity and yaw velocity, is controlled separately. In one cascade controller, on the first level we obtain desired angular acceleration using a PD controller, and then on the second level desired torque is generated by a PI controller with dynamics feedforward term. The controllers are presented as follows.

Angular velocity control

$$\omega_e^b = (\omega_d^b - \omega^b)$$

$$\dot{\omega}_e^b = k_p \omega_e^b + k_d \frac{\omega_e^b}{dt}$$

Angular acceleration control

$$\dot{\omega}_e^b = (\dot{\omega}_d^b - \dot{\omega}^b)$$

$$\tau = k_p \dot{\omega}_e^b + \int_0^t \dot{\omega}_e^b dt + \omega^b \times J\omega^b$$

Finally, torques on 3 dimensions are combined as $\overrightarrow{tau}$, from which rotational torques and lift thrust forces of the propellers are calculated. Forces and torques have to be converted to propeller rotational speeds and further to ESC signals.

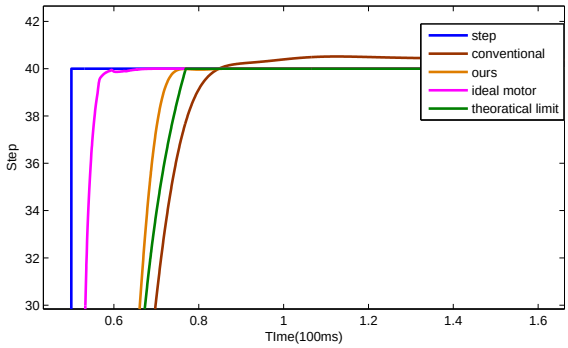


Fig. 5: Step response of our controller

In literature, conversion between propeller rotational speed and thrust force is not well treated. Some just ignore the dynamics of rotor and propellers [18], while some adopt complicated aerodynamics to rigorously analyze the relationship [4]. Either method is not precise enough to meet the practical requirement. To avoid over simplification of the problem and make conversion process efficient, we conducted experiments to measure a speed-force curve, as shown in Figure 3. The curve is stored as a look-up table in the algorithm implementation, with unknown points interpolated from neighboring existing points. For a desired thrust force, we check the corresponding speed level from the table, and apply ESC signal to let rotor change to that rotational speed.

A subtle problem is control output saturation. ESC signal is electronic signals that can change rapidly, while rotor rotational speed cannot change instantly. So the response of rotor speed must have a delay. Ideal rotor has no speed limitation, thus the system delay can be compensate by forward the control output to high value. However, real rotor does has maximum speed, so the system delay is inevitable. We will further elaborate this point in next section.

Figure 4 summarizes the entire control scheme.

C. Controller Performance Evaluation

We implemented the algorithm on the hardware platform of DJI flight controller A2.

Its flight performance can be more accurately evaluated by step response. We simulate our controller in Matlab. A step response is generated to simulate a control command that make pitch angle reach 40 degree. In Figure 5, blue line is the step function, brown line is a conventional PID controller with fine tuned parameters. Green line is a theoretical limit if the system delay is known precisely. Purple line is the performance of ideal rotor with no speed saturation limit. Orange line is our controller. Our controller clearly outperforms conventional PID controller in terms of response speed and overshoot curbing.

IV. iRobot Tracking

The tracking problem can be divided into two parts. First part is an estimation problem. The aerial robot use its download looking camera to identify and track the iRobot in its

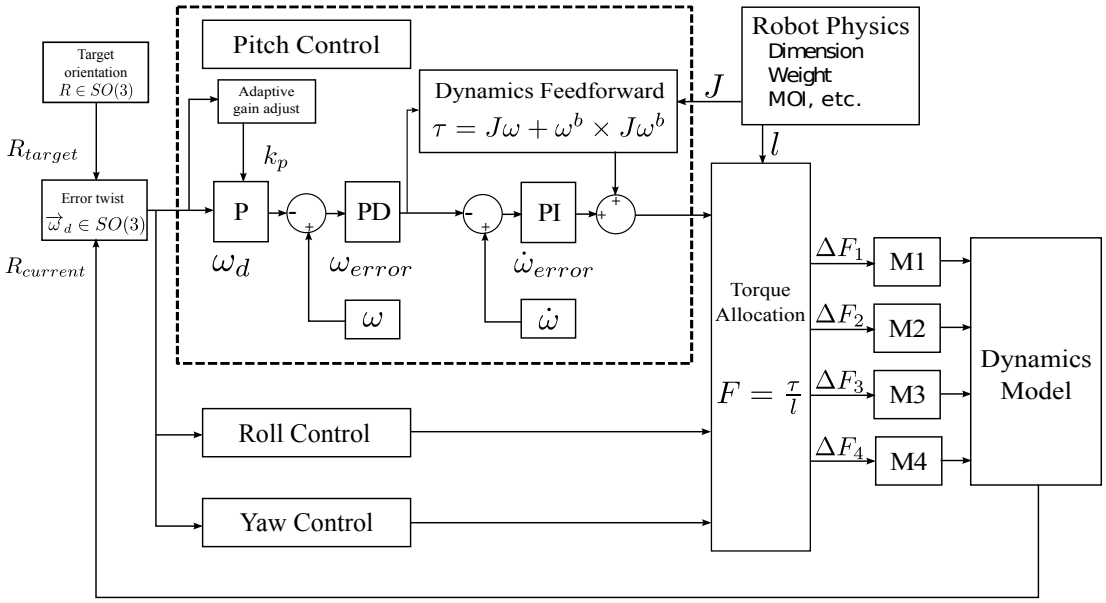


Fig. 4: The complete attitude control diagram. Pitch, roll and yaw control algorithms are the same, so the figure only describes pitch control detail.

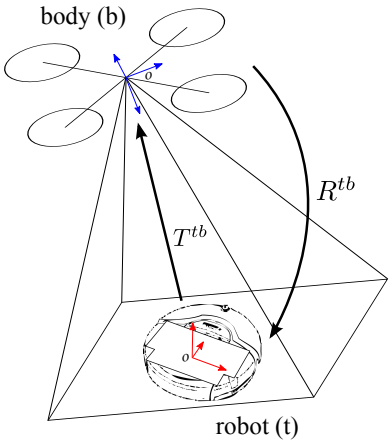


Fig. 6: Problem formation of the tracking problem.

view, and then estimate its position. The second is a position control problem. The aerial robot performs moving action according to its current status and position relative to the iRobot.

A. Problem Formation

As before, the aerial robot has a body frame attached to its center of mass, which is abbreviated as b. The top board of the iRobot has a coordinate frame too, which is t in short. The direction of axes of frame t is shown in Figure 6. Our task is to robustly estimate the rotation R^{tb} and translation T^{tb} between frame b and frame t.

We could have combined R and T as elements of $SE(3)$, and do tracking control on $SE(3)$, as [18] did. However, in our case we don't really need to control the rotation precisely. Rotation and translation can be decoupled to be controlled separately. And attitude control just realized rotation control. So the tracking algorithm here only need to tackle position tracking.

B. UKF Position Estimation

If the shape and the dimension of the iRobot top board is known, we can find correspondences between the board corner points and real points. The correspondences then form a PnP problem, from which R^{tb} and T^{tb} can be solved. However, due to image noise and motion blur, the solved translation is noisy. A filter considering robot movement can significantly reduce the noise. We use UKF to avoid the computation of Jacobian matrices of state transition function and measurement function. Although UKF is slower than EKF, for state has small size, the run time is roughly the same [25].

In this section, we first present our filter state definition, then describe the detail of PnP solution to introduce our filter measurement function. A filter performance evaluation is followed at the end.

1) State Definition: To estimate R^{tb} and T^{tb}, we employ a UKF to eliminate measurement noise. The state of the filter contains T^{tb}, R^{tb}, the velocity of the aerial robot v^t and the angular velocity of the aerial robot ω^b. Notice that the velocity is expressed in frame t while the angular velocity is in frame b.

Since R^{tb} is a 3-by-3 matrix, matrix representation is not suitable for filter state. A better approach is represent rotation as quaternion, denoted as q^{tb}. In the following text, we will write R^{tb} and q^{tb} interchangeably without ambiguity.

$$x_i = \begin{bmatrix} T^{tb} \\ q^{tb} \\ v^t \\ \omega^b \end{bmatrix} \tag{4}$$

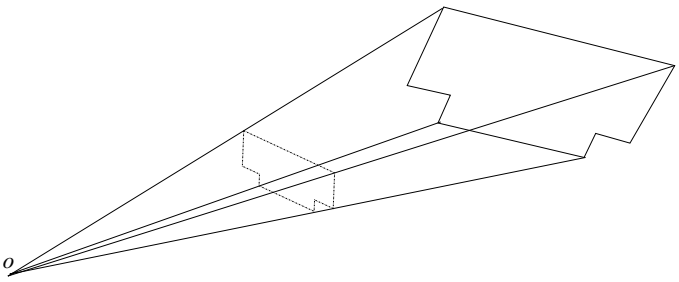


Fig. 7: Point correspondences in PnP problem

The state update equation of the UKF $x_i = f(x_{i-1})$ is

$$x_i = \begin{bmatrix} {T^{tb}}_i \\ {q^{tb}}_i \\ {v^t}_i \\ {\omega^b}_i \end{bmatrix} = \begin{bmatrix} {T^{tb}}_{i-1} + {v^t}_{i-1}\Delta t \\ {q^{tb}}_{i-1} \times q({\omega^b}_{i-1}\Delta t) \\ {v^t}_{i-1} + {a^t}_{i-1}\Delta t \\ {\omega^b}_{i-1} \end{bmatrix} \tag{5}$$

where function *q()* is a quaternion defined by the rotation vector ${\omega^b}_{i-1}\Delta t$. The symbol $\times$ means quaternion multiplication. ${a^t}_{i-1}$ is obtained from IMU measurement.

2) Image PnP Problem: To detect and track iRobot top board from images. We use standard image processing techniques to find corners of the board. By analyzing the configuration of corner points, we can find the correspondence of image corner points and real points. This idea is illustrated in Figure 7.

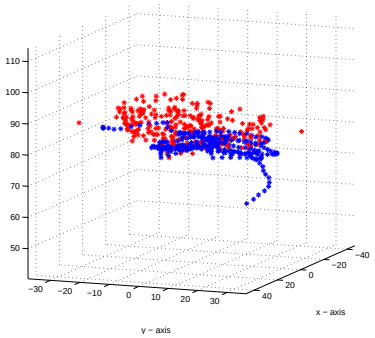
With point correspondences, we can solve the PnP problem. The method invented in [22] is utilized with modifications to reduce computation complexity. In the computation of local minima, the original method iterates through all possible pitch, roll and yaw angles. However, since the iRobot top board keeps parallel to the ground, IMU can provides pitch and roll angle measurement. Therefore, the algorithm only need to search yaw angle. With this modification, the algorithm complexity reduced from $O(n^3)$ to $O(n)$ where n is the size of search space. The solved R^{tb} and T^{tb} are then be used as filter measurement.

3) Measurement Function: Our measurement function $y_i = h(x_i)$ is simple:

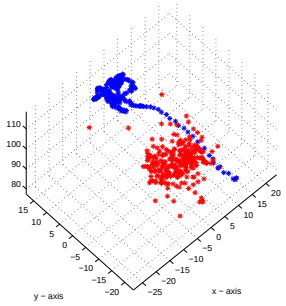
$$y_i = \begin{bmatrix} T^{tb} \\ q^{tb} \\ \frac{1}{w_1+w_2}[w_1({v^t}_i + a^t\Delta t) + w_2\frac{T^{tb}-{T^{tb}}_i}{\Delta t}] \\ \omega^b \end{bmatrix} \tag{6}$$

In the equation, T^{tb} and q^{tb} are the result of PnP problem. a^t and ω^b are IMU data. The only intricate measurement is velocity, because we have two sources of velocity measurement. ${v^t}_i + a_t\Delta t$ is the integration of IMU linear acceleration, while $\frac{T^{tb}-{T^{tb}}_i}{\Delta t}$ is the differentiation of position measurement. We use two adjustable weights w_1 and w_2 to give different trust to the two measurements.

4) Filter Evaluation: The performance of our filter can be seen from experiments. We let the iRobot stop on the ground, and then make the aerial robot to hover above the iRobot. We control the aerial robot to perform some movements and compare the measurement position and filtered position. Two sets of experiment record are depicted in Figure 8.



(a) Measurement record of a horizontal back-and-forth movement. The movement is only in pitch direction.



(b) The movement of this set of record is stationary hovering with large yaw rotation only. Blue dots are moved aside to have better view.

Fig. 8: Experiment records. Red dots are measurement position, and blue dots are filtered position.

In both records, red dots are the direct measurement position from PnP solutions, while blue dots are filtered position. In Subfigure 8a, a clear path is formed by blue dots, but red dots don't form any regular movement patterns. For the movement in Subfigure 8b, since the aerial robot is in hovering mode, then even if the robot has yaw rotation, its position should not change too much. We can see blue dots do show little horizontal movement noise, but red dots are too noisy to find the stationary point.

C. PID Position Tracking Control

To move the aerial robot, proper target rotation command must be send to A2 from Odroid XU via serial port. The attitude tracking control algorithm in Section III makes the robot tracks the given rotation command. The rotated aerial robot gains acceleration in the pitch-roll plane.

Therefore, from position information provided by UKF, we need to control velocity to reach a target position. To control the velocity, acceleration has to be controlled which is come from rotation angle. So a chain of controllers are involved to convert target position to target rotation. This idea is outlined in Figure 9. All of the position controller, the velocity controller and the acceleration controller are standard PID controllers.

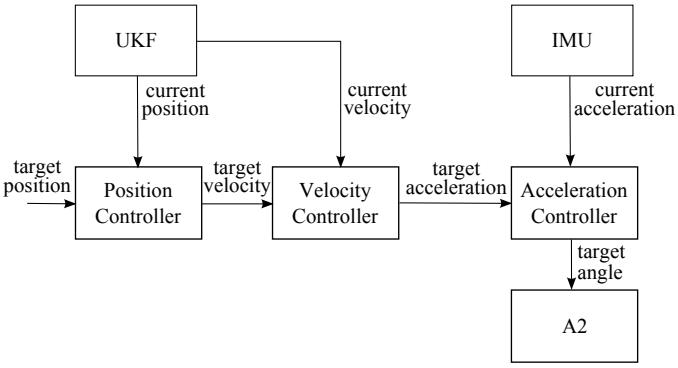


Fig. 9: The cascade position controller.

Fig. 10: Arena in IARC Mission 7.

The tracking controller performance can be viewed at https://www.youtube.com/watch?v=MkCJ_Tyyr9w.

V. Localization

In IARC Mission 7, global localization of UAV is an important step towards the realization of more complex iRobot tracking and overall game strategy. To attain UAV position in $\mathbb{R}^3$ Euclidean space over the game field, an algorithm which detects boundary lines in the field through camera image processing is used.

A. IARC Mission 7 Field Settings

In IARC Mission 7, an indoor GPS-free arena on the ground is provided. The field is 20×20 square meters with two white, one red and one green boundary line as shown in Figure 10.

B. Localization Algorithm

This algorithm finds UAV's position as well as its attitude. Two methods, *threelines* method or *corner* method can be used.

1. Preliminaries

According to the requirements of Mission 7, we model the game field as shown in Figure 11. (We only demonstrate the border lines in the picture. In the rest of the report the word "lines" refers to the boundary lines. The white lines are represented by thin black lines for illustration purposes.)

In the figure, the world frame is denoted by ω, and the UAV body frame by b. We set the origin of the world frame at the corner where the y-axis lies on the green line as the model shows. The position of UAV in the world frame is represented

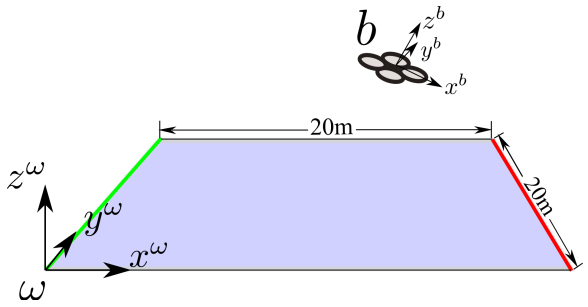


Fig. 11: Model of the gaming field.

by a 3-dimensional vector p^ω, where ${p_1}^\omega$, ${p_2}^\omega$, ${p_3}^\omega$ are the x, y and z coordinates relative to the world axis.

$$p^\omega = \begin{bmatrix} p_1^\omega & p_2^\omega & p_3^\omega \end{bmatrix}^T$$

Each point on the ground can be projected to UAV body frame through camera(s). (All points are represented in the same form as vectors starting from origin, and will be treated as vectors in algebra calculations.) We can only observe a point from camera images, which is on a 2-D pixel frame. There are four cameras on different sides of the UAV, so points on four different pixel frames will be transformed into four different camera frames separately. Since the cameras are rigidly attached to the UAV, the points in the camera frame will then be converted into body frame. Therefore, *every point appearing in the camera images can be transformed into a unit vector in the body frame, as long as we know the transformations.* These transformations depends on the intrinsic and extrinsic parameters of the camera, which are related to the characteristics of the cameras and the mechanical design of the UAV. We will jump over these topics and use vectors in the body frame directly since we already know how to convert points from images to body frame.

2. UAV Position Calculation

The key concept in this algorithm is finding the intersection line of the two planes containing different boundary lines. *threelines* method detects three different boundary lines, and *corner* method detects a corner. The *position* of UAV can be found by simple geometry based on the angles between the planes and the ground. The *attitude* can be found in the form of rotation matrix.

2.1. Threelines

In *threelines* method, we observe three lines out of four, which must include a pair of opposite lines. As can be observed from the model, opposite lines are parallel, which means the intersection of any two planes containing these two lines respectively is always a line parallel to them. Observing any two points on a line will give the plane. This gives enough information for obtaining $R^{\omega b}$, the rotation matrix from body frame to the world frame, since *this intersection line is parallel to an axis of the world frame, and is obtained in the body frame as well.* With this axis found and a gravity vector returned from the accelerometer inside UAV, $R^{\omega b}$ can be found.

Figure 12 is a rough sketch illustrating the idea. For demonstration, we assume the three lines observed are the

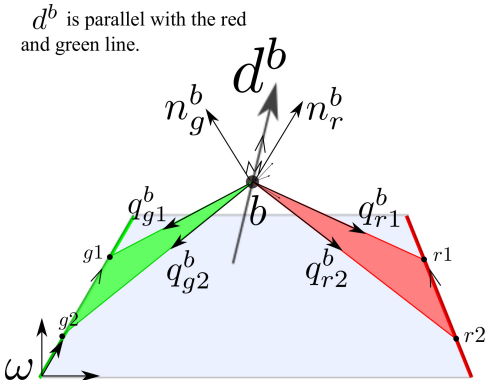


Fig. 12: Normal vectors on planes and intersection.

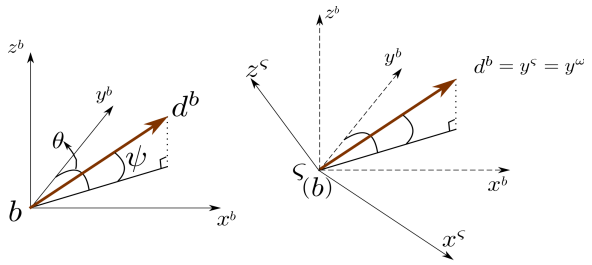


Fig. 13: Relationship between ς frame and body frame.

green, white and red line from left to right as shown in the figure. $g1$, $g2$, $r1$ and $r2$ are random sample points on the green and red line. q_{g1}^b, q_{g2}^b, q_{r1}^b and q_{r1}^b are the corresponding vectors in the body frame. n_g^b and n_r^b are the normal vectors perpendicular to the green and red planes correspondingly. d^b is the unit vector perpendicular to the plane consisting of n_g^b and n_r^b, and it is parallel to the y-axis in the world frame. We can calculate the unit intersection vector d^b by first finding n_g^b and n_r^b, then calculate their cross product:

$$\begin{cases} n_g^b = \frac{q_{g1}^b \times q_{g2}^b}{|q_{g1}^b \times q_{g2}^b|} \\ n_r^b = \frac{q_{r1}^b \times q_{r2}^b}{|q_{r1}^b \times q_{r2}^b|} \end{cases} \tag{7}$$

$$d^b = \frac{n_g^b \times n_r^b}{|n_g^b \times n_r^b|} \tag{8}$$

Now we have d^b, and assuming we have an accurate enough gravity measurement g^b, we can find the rotation matrix between the world frame and body frame. We define frame ς as an intermediate frame sharing the same origin as the body frame. Figure 13 illustrates the relation between these frames. ς frame has its y-axis pointing to the same direction as the world y-axis. Two angles θ and ψ can be derived from:

$$\begin{cases} \theta = \arctan \frac{d_1^b}{d_2^b} \\ \psi = \arctan \frac{d_3^b}{\sqrt{{d_1^b}^2 + {d_2^b}^2}} \end{cases} \tag{9}$$

where d_1^b, d_2^b and d_3^b represent xyz-coordinates of vector d^b in the body frame. Now we can get the rotation matrix from body frame to ς:

$$R^{\varsigma b} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{10}$$

We need also to find the rotation between ς and ω. Now we have their y-axis parallel, the only difference is an angle between either their x or z axis. With gravity measurement g^b, we can find the angle τ by:

$$\tau = \arccos\left((g^\varsigma)^T(-z^\varsigma)\right) = \arccos\left(\left(R^{\varsigma b} g^b\right)^T \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\right) \tag{11}$$

which comes from the inner product of g^ς and $-z^\varsigma$. Extra attention are needed here to assure the angle is of the right sign. Now we have the complete formula of the rotation matrix between UAV body frame and the world frame:

$$R^{\omega b} = \begin{bmatrix} \cos\tau & 0 & -\sin\tau \\ 0 & 1 & 0 \\ \sin\tau & 0 & \cos\tau \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{12}$$

After $R^{\omega b}$ is found, locating the UAV is very easy. The procedure has two steps. First we narrow our target range down to the line of intersection, then we use the third line (white line) detected to find the point. Figure 14 shows the geometric relationships between normal vectors and position of UAV.

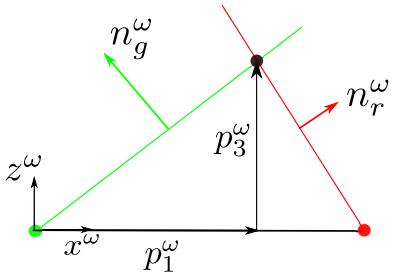


Fig. 14: x-z plane cross section in world frame.

We first transform normal vectors in body frame into world frame with $R^{\omega b}$. We use n_w^b as the normal vector of the plane relative to the white line, so we have:

$$\begin{cases} n_g^\omega = R^{\omega b} n_g^b \\ n_r^\omega = R^{\omega b} n_r^b \\ n_w^\omega = R^{\omega b} n_w^b \end{cases} \tag{13}$$

Based on the triangles, we find:

$$\begin{cases} \frac{p_3^\omega}{p_1^\omega} = \frac{-n_{g1}^\omega}{n_{g3}^\omega} \\ \frac{p_3^\omega}{20 - p_1^\omega} = \frac{-n_{r1}^\omega}{n_{r3}^\omega} \\ \frac{p_3^\omega}{20 - p_2^\omega} = \frac{n_{w2}^\omega}{n_{w3}^\omega} \end{cases} \tag{14}$$

where n_{g1}^ω, n_{g2}^ω and n_{g3}^ω represents the x, y and z coordinates of n_g^ω respectively, similar for n_r^ω and n_w^ω. With these three equations, we can easily deduce the result of the position vector p^ω of UAV:

$$\begin{cases} p_1^w = \frac{20 n_{r1}^\omega n_{g3}^\omega}{n_{g1}^\omega n_{r3}^\omega + n_{g3}^\omega n_{r1}^\omega} \\ p_2^w = 20 + \frac{20 n_{r1}^\omega n_{g1}^\omega n_{w3}^\omega}{n_{w2}^\omega (n_{g1}^\omega n_{r3}^\omega + n_{g3}^\omega n_{r1}^\omega)} \\ p_3^w = \frac{-20 n_{r1}^\omega n_{g1}^\omega}{n_{g1}^\omega n_{r3}^\omega + n_{g3}^\omega n_{r1}^\omega} \end{cases} \tag{15}$$

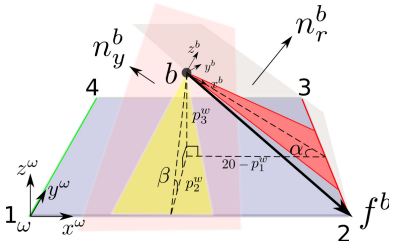


Fig. 15: Model for corner method.

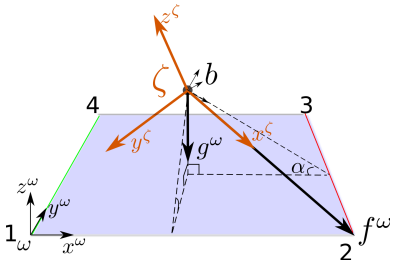


Fig. 16: Model for corner method - finding the attitude.

To calculate it faster and understand the way it works, solve the first and second equations in Equation 14 and solve for p_1^w and p_3^w first. p_3^w gives us the height of the UAV and then p_2^w can be easily solved from the third equation.

2.2. *Corner*

When the UAV cannot observe three different lines in the field, observing just one corner can also provide sufficient information to localize the UAV, assuming that the height is known. In this case, the attitude cannot be obtained as in *threelines* method since we don't have a parallel line pair, and we will calculate the position first and obtain attitude information afterwards.

The model for *corner* method is illustrated in Figure 15. Two planes crossing each other on a line that pass a corner point which in this figure is point 2.

Similarly, we first localize the UAV on this line. Then with the height known from UAV sensors, we can find the point. We can find the following geometric relations:

$$\begin{cases} \alpha = \arccos{(n_r^b)^T(-g^b)} \\ \tan\alpha = \frac{p_3^\omega}{20-p_1^\omega} \\ \beta = \arccos{(n_y^b)^T(-g^b)} \\ \tan\beta = \frac{p_3^\omega}{p_2^\omega} \end{cases} \tag{16}$$

The height p_3^ω is known to us, so we can get:

$$\begin{cases} p_1^\omega = 20 - p_3^\omega \cot\alpha \\ p_2^\omega = p_3^\omega \cot\beta \end{cases} \tag{17}$$

This is the complete p^ω position vector derived from *corner* method.

To find the attitude (rotation matrix) of the UAV relative to the world frame, we need to make use of the corner observed on camera. The model can be found in Figure 16. We define a new frame ζ as the intermediate frame to find the rotation matrix. Similar to the *threelines* method, we also have two pairs of known vectors, one is gravity g^ω and g^b, another pair is f^ω and f^b, which point towards the corner. We first find the rotation matrix $R^{\omega\zeta}$ by finding ζ axes in world frame:

$$\begin{cases} x_\zeta^\omega = \frac{f^\omega}{|f^\omega|} \\ y_\zeta^\omega = \frac{f^\omega \times g^\omega}{|f^\omega \times g^\omega|} \\ z_\zeta^\omega = \frac{x_\zeta^\omega \times y_\zeta^\omega}{|x_\zeta^\omega \times y_\zeta^\omega|} \end{cases} \tag{18}$$

where x_ζ^ω, y_ζ^ω and z_ζ^ω represents the unit ζ axes representation in ω frame. Rotation matrix $R^{\omega\zeta} = [x_\zeta^\omega \quad y_\zeta^\omega \quad z_\zeta^\omega]^T$

The rotation from ζ frame to body frame can be found in similar way:

$$\begin{cases} x_\zeta^b = \frac{f^b}{|f^b|} \\ y_\zeta^b = \frac{f^b \times g^b}{|f^b \times g^b|} \\ z_\zeta^b = \frac{x_\zeta^b \times y_\zeta^b}{|x_\zeta^b \times y_\zeta^b|} \end{cases} \tag{19}$$

Rotation matrix $R^{b\zeta} = [x_\zeta^b \quad y_\zeta^b \quad z_\zeta^b]^T$, and $R^{\zeta b} = (R^{b\zeta})^T$.

As a result, the rotation matrix from body frame to world frame is:

$$R^{\omega b} = R^{\omega\zeta} R^{\zeta b} \tag{20}$$

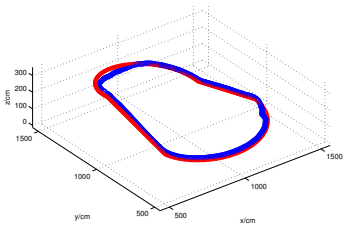


Fig. 17: Simulation result – red line is the real trajectory, blue line is the estimation.

C. Simulation Results

Our simulation results in matlab is shown in Figure 17.

In this simulation we assumed the yaw angle of the UAV doesn't change, so we can always decide which line is which on camera images. We added noise to the gravity or height to simulate values returned from the sensors. We also applied an UKF Filter to smooth the trajectory. The result is satisfying and the estimation doesn't change much with the noise.

VI. System Integration

In this section we presents our hardware architecture and software architecture. Hardware part lists all major hardware components and communication protocols among them. As for software architecture, since all our programs are written in C++ or Python within the Robot Operating System (ROS) framework, we use mainly the terminology of ROS, such as node, topic, tf and so on through out the discussion.

A. Hardware Architecture

The core hardware components are autopilot system and high level computing platform. We use A2 autopilot set manufactured by DJI. A2 includes a main controller, a power management unit, a IMU and a GPS module. Both the IMU and the PS module are industrial precision. Our high level computing platform is Odroid XU from company Hardkernel. A2 and XU are linked by serial port, on which messages contain autopilot sensor data and control data are exchanged.

A2, ESCs, motors and propellers form the dynamic propulsion system. Among them are standard connections.

Odroid XU connects to two sensors via USB. One is a high speed camera, the other is LIDAR. LIDAR also need 12V power supply, which is extracted from the battery.

Images of four cameras are sent to ground station for further processing. The image transmission is handled by two sets of image transmitters. The transmitters send images via 5.8GHz radio signal to the ground station.

Odroid XU runs critical programs to monitor the UAV. Monitoring messages will be periodically sent to the ground station via WIFI connection. The ground station also send high level navigation information to Odroid XU through WIFI.

The model of hardware components are listed in the following table.

Autopilot	DJI A2
Computing platform	Hardkernel Odroid XU
Rotor	DJI E600 Propulsion System
Propeller	DJI E600 Propulsion System
ESC	DJI E600 Propulsion System
Camera	ZWO ASC120MC
LIDAR	Hokuyo UBG-04LX-F01
Image transmitter	DJI Lightbridge

B. Software Architecture

Our aerial robot has two computing devices where programs reside in. One is the microprocessor of A2 controller, the other is the system-on-chip (SOC) of Odroid XU. The program in A2 controller exactly implements the algorithm discussed in Section III. It is a single sequential program without any complex structure. While the programs in Odroid XU do have complicated relationships that need detailed explanation. So in our software architecture presentation we didn't show the structure of the low level control program on A2. It is only be indirectly presented as `serial_to_uav` in the architecture. `serial_to_uav` is a program on Odroid XU that communicate with A2 via serial port. Therefore, the architecture discussed here is only related to the programs on Odroid XU and the ground station.

Both of Odroid XU and the ground station run Ubuntu Linux and have ROS hydro installed. With the aid of ROS, Odroid XU and the ground station can be combined as one virtual machine once they are connected to the same LAN, which is achieved by setting the ground station as a WIFI access point and let Odroid XU connect to it. Therefore programs on Odroid XU and the ground station can share communication messages without worrying what intermediate channels they are using.

Using ROS terminology, a node refers to a program. Parameters mean arguments passed to a certain program, or node. Topic stands for a inter process communication (IPC) message between two node. A node that broadcast a topic is called publisher, while a node receives a topic is named as subscriber. Although a topic is sent as broadcast message, to simply description, we still say a topic is sent to a specific node if that node is the only subscriber. Package is a file management mechanism in ROS framework. Multiple nodes can be organized and managed in one package. More details can be found from ROS official tutorial [1].

In Figure 19, every rectangle represents a ROS node. Their names are self-explanatory.

`obstacle_avoid` reads in the LIDAR sensor topic. From sensor reading it detects nearby obstacles. Obstacle information is packed in an output topic which is sent out from this node. This node itself will also devise a control strategy from the analysis of obstacle positions. The control command topic sends to `uav_cmd_ctrl`.

`uav_cmd_ctrl` is the core control node in the whole system. This node subscribes various topics of control commands from four publishers. It acts as an arbiter to select the most appropriate command, and then send this selected command as a topic to `serial_to_uav`.

`iRobot_tracker` implements the algorithm in Section IV. Its control command is packed in a topic to `uav_cmd_ctrl`. Also, the position of the iRobot, packed in another topic, is reported to a strategy maker `strategy_simulate`.

`uvc_camera` is the node in ROS official repository that read camera using USB interface.

`serial_to_uav`. This node is crucial because it is the only bridge between all other nodes and low level controller. It reads sensor data and send control command to A2. Several nodes will subscribe sensor data for various purpose. The control command is obtained from node `uav_cmd_ctrl`. After read the command from ROS topic, this node send the command to serial port.

`urg_node` is a modified version of Hokuyo LIDAR sensor reading node in ROS repository. It not only read raw LIDAR data, but also use IMU data to compensate LIDAR angular deviation.

`strategy_simulate`. A mission process simulator is important for the aerial robot to have global planning ability. The principle is let simulated iRobots run in the arena constructed using robot simulation software Gazebo. The simulator keeps receiving real iRobot position and aerial robot position to correct its simulation. As long as the simulation resembles the real game process, a planner can make control decision base on the position of robots in the simulator. The planner control is published as ROS topic.

`uav_pose_filter` use UKF to estimate the pose of the aerial robot, as presented in Section V.

`field_localize` contains two sets of algorithm described in Section V. Together with `arena_detect` and `uav_pose_filter`, a relatively precise global aerial robot position measurement is computed.

`image_read` is a node read four images at one time from

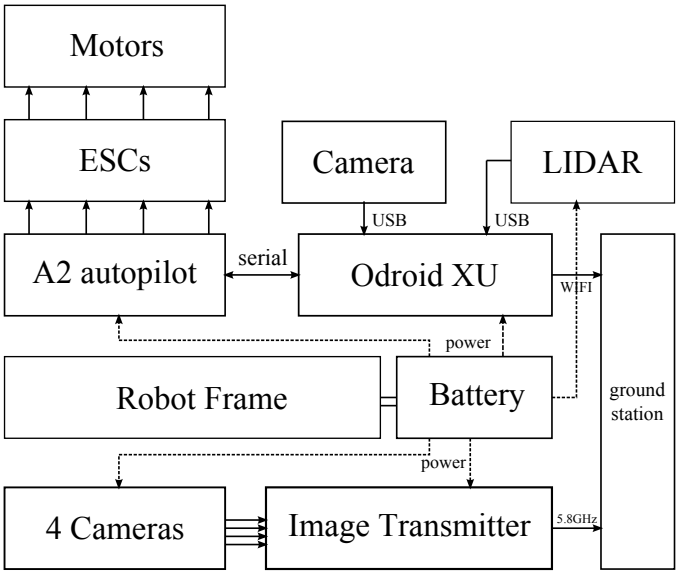


Fig. 18: Hardware architecture of our aerial robot system.

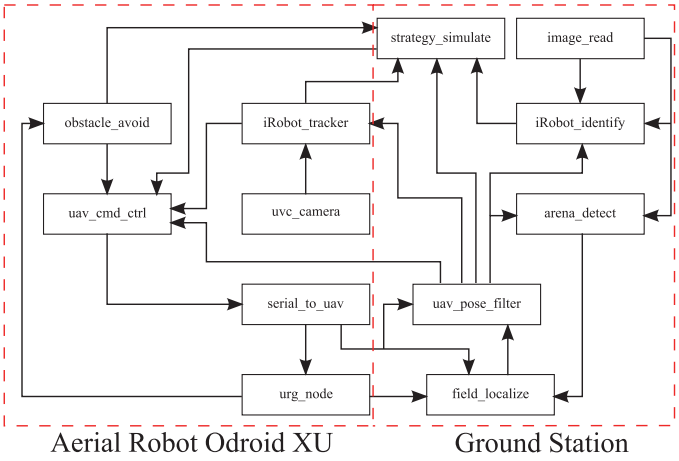


Fig. 19: Software architecture of the robot. Nodes on the left side run on the robot, while nodes on the right side run on the ground station.

Lightbridge. The images are published as a single topic to `arena_detect`.

`iRobot_identify` is one of the most significant nodes. It realize the algorithm illustrated in Section IV. The calculated aerial robot position and iRobot position will be transfered to `strategy_simulate` for simulation purpose. Also a control command is made for `uav_cmd_ctrl` to control the aerial robot.

`arena_detect` is a image processing node to detect the boundary of the arena.

Currently, not all nodes are ready to use, because we are still in early stage of software development. We anticipate to finish major nodes relevant to core functions before the competition of 2014. A complete software requires another year to finish.

VII. Conclusion & Future Work

In this report we explain the major components we designed and implemented in order to solve IARC Mission 7. Each part is proved, either by real experiments or simulations, to be a promising building block of a whole functional robot. We will further develop the algorithms and models of each part, transfer simulations to real systems, and refine the system integration continously.

Acknowledgment

HKUST IARC Team would like to thank DJI Innovations for their generous sponsorship. Thank HKUST Department of Electronic & Computer Engineering for providing various supports to our work.Professor Zexiang Li, Professor Ming Liu, Professor Kam-Tim Woo and HKUST student society Aeronautics Interest Group also helped a lot in due course.

References

[1] ROS Tutorial. http://wiki.ros.org/ROS/Tutorials, 2013.

[2] Franz Andert, Florian Adolf, Lukas Goormann, and Jörg Dittrich. Mapping and path planning in complex environments: An obstacle avoidance approach for an unmanned helicopter. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 745–750. IEEE, 2011.

[3] Johann Borenstein and Yoram Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *Robotics and Automation, IEEE Transactions on*, 7(3):278–288, 1991.

[4] Samir Bouabdallah and Roland Siegwart. Full control of a quadrotor. In *Intelligent robots and systems, 2007. IROS 2007. IEEE/RSJ international conference on*, pages 153–158. IEEE, 2007.

[5] Francesco Bullo. Proportional derivative (pd) control on the euclidean group. In *European Control Conference*, volume 2, pages 1091–1097.

[6] Brandon Call, Randy Beard, Clark Taylor, and Blake Barber. Obstacle avoidance for unmanned air vehicles using image feature tracking. In *AIAA Guidance, Navigation, and Control Conference*, pages 3406–3414, 2006.

[7] Tao Dong, XH Liao, Ran Zhang, Zhao Sun, and YD Song. Path tracking and obstacles avoidance of uavs-fuzzy logic approach. In *Fuzzy Systems, 2005. FUZZ'05. The 14th IEEE International Conference on*, pages 43–48. IEEE, 2005.

[8] Bernard Espiau, François Chaumette, and Patrick Rives. A new approach to visual servoing in robotics. *Robotics and Automation, IEEE Transactions on*, 8(3):313–326, 1992.

[9] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[10] Xiao-Shan Gao, Xiao-Rong Hou, Jianliang Tang, and Hang-Fei Cheng. Complete solution classification for the perspective-three-point problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(8):930–943, 2003.

[11] F Sebastian Grassia. Practical parameterization of rotations using the exponential map. *Journal of graphics tools*, 3(3):29–48, 1998.

[12] Brian Hall. *Lie groups, Lie algebras, and representations: an elementary introduction*, volume 222. Springer, 2003.

[13] Lionel Heng, Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys. Autonomous obstacle avoidance and maneuvering on a vision-guided mav using on-board processing. In *Robotics and automation (ICRA), 2011 IEEE international conference on*, pages 2472–2477. IEEE, 2011.

[14] Berthold K Horn and Brian G Schunck. Determining optical flow. In *1981 Technical Symposium East*, pages 319–331. International Society for Optics and Photonics, 1981.

[15] Stefan Hrabar. 3d path planning and stereo-based obstacle avoidance for rotorcraft uavs. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 807–814. IEEE, 2008.

[16] Stefan Hrabar, Gaurav Sukhatme, Peter Corke, Kane Usher, and Jonathan Roberts. Combined optic-flow and stereo-based navigation of urban canyons for a uav. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3309–3316. IEEE, 2005.

[17] Vijay Kumar and Nathan Michael. Opportunities and challenges with autonomous micro aerial vehicles. *The International Journal of Robotics Research*, 31(11):1279–1291, 2012.

[18] Taeyoung Lee, M Leoky, and N Harris McClamroch. Geometric tracking control of a quadrotor uav on se (3). In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 5420–5425. IEEE, 2010.

[19] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155–166, 2009.

[20] Richard M Murray, Zexiang Li, S Shankar Sastry, and S Shankara Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[21] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–652. IEEE, 2004.

[22] Gerald Schweighofer and Axel Pinz. Robust pose estimation from a planar target. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(12):2024–2030, 2006.

[23] Iwan Ulrich and Johann Borenstein. Vfh+: Reliable obstacle avoidance for fast mobile robots. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2, pages 1572–1577. IEEE, 1998.

[24] Iwan Ulrich and Johann Borenstein. Vfh*: Local obstacle avoidance with look-ahead verification. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 3, pages 2505–2511. IEEE, 2000.

[25] K Xiong, HY Zhang, and CW Chan. Performance evaluation of ukf-based nonlinear filtering. *Automatica*, 42(2):261–270, 2006.