Ard-μ-copter: A Simple Open Source Quadcopter Platform

Zhijian He, Yanming Chen, Zhaoyan Shen, Enyan Huang, Shuai Li, Zili Shao, Qixin Wang,
The Embedded Systems and CPS Lab, Dept. of Computing,
The Hong Kong Polytechnic Univ., HONG KONG SAR
{cszjhe, cszshen, csehuang, cssli, cszlshao, csqwang}@comp.polyu.edu.hk, yanming.chen@connect.polyu.hk

***Abstract*—With the emergence of many *commercial-off-the-shelf* (COTS) and/or open-source hardware and software, we can now build cheap *Unmanned Aerial Vehicles* (UAVs). This will enable a broad spectrum of potential UAV based mobile applications. In this work, we propose a simple open UAV platform: Ard-μ-copter. Ard-μ-copter is a quadcopter built upon the open source ArduPilot [1] infrastructure library and hardware. Comparing to the many existing commercial quadcopters platforms and open source quadcopter platforms, Ard-μ-copter platform is fully open source, simple (i.e. what the "μ" stands for), and with good documentations.**

I. Introduction

*Unmanned Aerial Vehicles (*UAVs) are becoming increasingly popular due to the sharp drop of UAV components' prices. They are wildly used for more and more mobile applications [2].

Quadcopters are probably the most popular type of UAVs, due to the wide availability of COTS quadcopter components, and simplicity of their control. Existing commercial quadcopters are mostly not open source. Most existing open source quadcopters, however, either are too complex, or have poor documentations. This motivate us to build an open quadcopter platform that is simple and with good documentations.

The following summarizes our contributions.

1) We build an open source quadcopter platform, Ard-μ-copter (see Fig. 1), upon the open source Ardupilot [1] low layer libraries and hardware.

2) Comparing to ArduPilot platform (which includes the upper layer, the low layer libraries, and hardware), Ard-μ-copter platform is much simpler (only involves 135KB of upper layer source code), and is better documented. Developers just need basic matrix knowledge and control law to understand the source code.

The rest of the paper is organized as follows. Section II discusses related work; Section III describes the hardware and software (control strategy) of Ard-μ-copter platform; Section IV evaluates Ard-μ-copter with experiments; and Section V concludes the paper.

II. Related Work

Although UAVs become commercially popular in recent years, theories to build UAVs have been developed for long time, including those for quadrotors [4] [5] [6], hexarotors [7] [8], and octorotors [9] [10]. In this paper, we are not to develop new theories for quadcopter controls, but to use existing theories to build a simple and well documented open source quadcopter platform.

Fig. 1. Our prototype Ard-μ-copter (image quoted from [3])

Table I compares Ard-μ-copter platform with other well-known existing quadcopter platforms: DJI [11], MicroPilot [12], and ArduPilot [1]. All of these are small to mid-sized quadcopters comparable to our quadcopter platform. Generally speaking, commercial quadcopter platforms (such as DJI and MicroPilot) operate at higher sampling/actuating rates, hence are capable of higher control performance. But commercial platforms are generally more expensive, and not fully open source (though some provide limited programming interfaces for application developers). Non-commercial quadcopter platforms (such ArduPilot) are open source, but the source code may be too complex and poorly documented, which result in steep learning curve and high maintenance cost.

In comparison, Ard-μ-copter platform is fully open source and simple. The source code size is only 135KB. Besides, Ard-μ-copter platform supports a wide range of onboard sensors and high sampling rate.

III. Platform Architecture

In this section, we elaborate the architecture of Ard-μ-copter. Fig. 2 gives the overall architecture of Ard-μ-copter. The overall architecture is layered. At the bottom lies the hardware layer, which involves various sensors and actuators. In the middle lies the driver layer. At the top lies the main program layer, where control strategies are carried out.

TABLE I
PRODUCTS COMPARISON

Product	DJI	MicroPilot	Ardupilot	Ard-μ-copter
Inertial sensors	djiIMU	MP2128	MPU6050	MPU6050
Magnetometers	djiIMU(integrate magnet)	MP2128(integrate magnet)	HMC5883	HMC5883
Barometer	djiIMU(integrate barometer)	MP2128(integrate barometer)	MS5611	MS5611
GPS	djiIMU(integrate GPS)	ublox-6M	ublox-6M or ublox-6H	ublox-6H
Wireless Module	Usmile Mini OSD	MP2128(integrate wireless)	MAVlink 433M	MAVlink 433M
Open Source	No	No	Yes(21.31MB)	Yes(135KB)

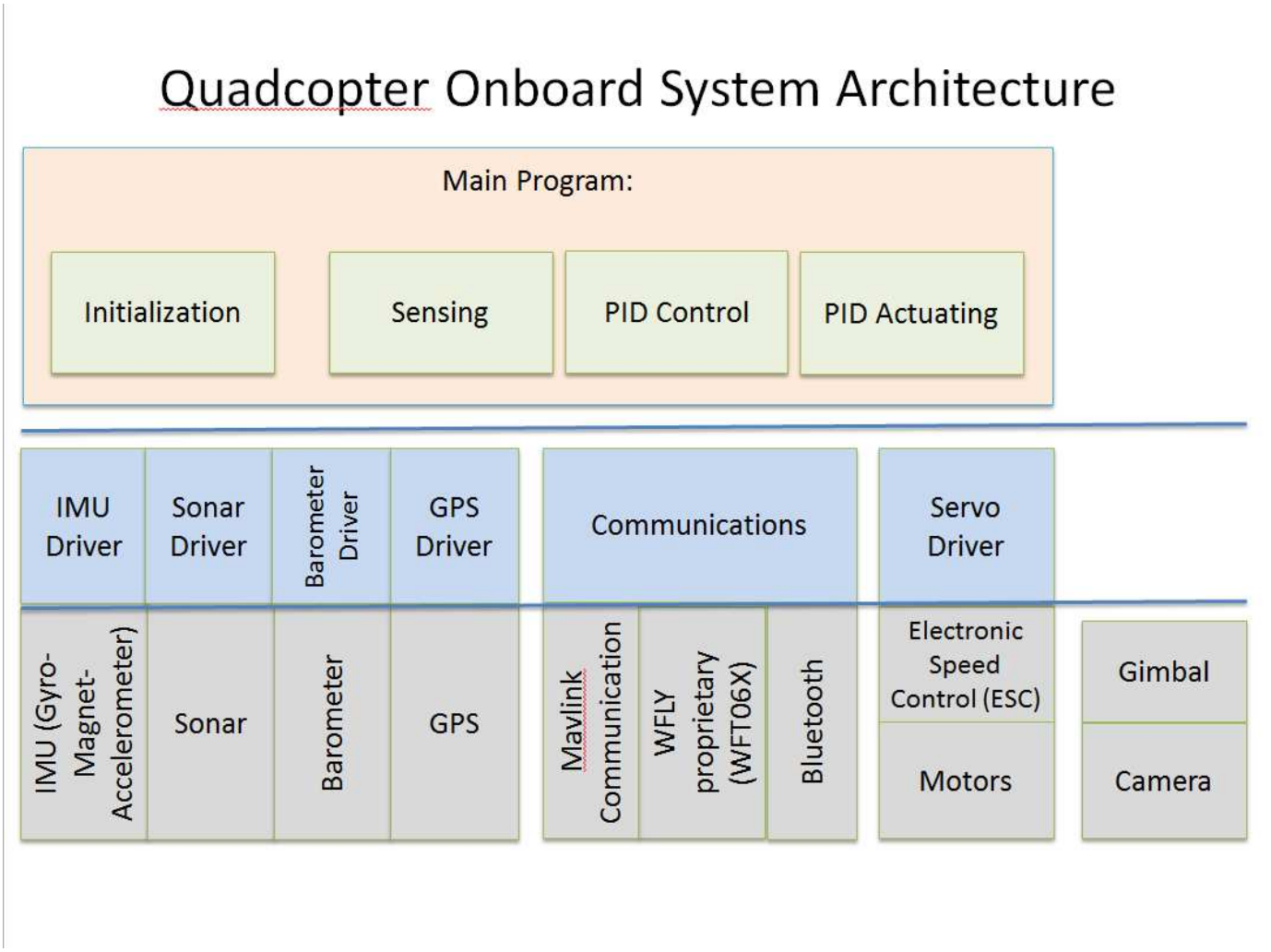

Fig. 2. Ard-μ-copter Overall Architecture

In the following, Section III-A will describe the hardware layer, and Section III-B will describe the main program layer. The driver layer is basically the open source ArduPilot infrastructure libraries [1], hence will not be elaborated in this paper.

A. Hardware Layer

The Ardu-μ-copter hardware layer reuses the open specification ArduPilot Mega 2.5.2 board (aka APM 2.5.2 board) [1]. This board integrates an ATmega 2560 [13] Micro-Controller Unit (MCU) with various sensors and actuators, as listed in Table II.

TABLE II
HARDWARE COMPONENT DETAILS

MCU	ATmega 2560
IMU	MPU-6050
Magnetometer	HMC5883
GPS	UBLOX-6H
Barometer	MS5611
Sonar	XL-MaxSonar EZ4
Wireless	Usmile Mini 433M

The following further elaborates each hardware component listed in Table II.

1) Inertial Measurement Unit (IMU): The MPU-6050 IMU includes a 3-axis gyroscope and a 3-axis accelerometer. As a result, the output of IMU has 6 degrees of freedom. It works under 2.375V $\sim$ 3.46V. MPU-6050 has an internal 16bit ADC to convert its gyroscope and accelerometer readings to 16-bit digital output. With its internal Digital Motion Processing (DMP) engine, MPU-6050 can speed up its angle calculation [2].

MPU-6050 has to be initialized and programmed with proper sampling rate and desired scaling factor. It features an SPI interface and is connected to one of the MCU's SPI interface. MPU-6050 works in 400HZ frequency and data exposed on the SPI interface at the programmed sampling frequency. Besides providing angular velocity (from gyro) and linear acceleration readings (from accelerometer), MPU-6050 also provide angle readings using its DMP engine.

2) Magnetic Measurement Unit: Magnetometer HMC5883 is not used in our current implementation due to disturbances from electrical circuit and strong external electromagnetic interferences.

Nonetheless, HMC5883L is a sophisticated module. It is a triple axis compass magnetometer sensor module, accessible through a IIC Peripheral Interface. With proper timing parameter of IIC interface, three-axis magnetic field data can be polled. Similar to IMU, this unit has to be initialized and programmed with proper sampling rate and scaling factor. The measuring range is 1.3 $\sim$ 8 gauss. The package size is 1.8 x 1.3 cm. This module has to be initialized with calibration and used with yaw angle compensation due to possible board leaning.

3) GPS Unit: In order to get an accurate position of copter, a GPS module is needed. We choose Ublox NEO-6M High Precision GPS Module Built-in Compass with fast satellite searching speed and high precision. This module is compatible with APM serial port and I2C port with a package size of $15 \times 12 \times 2$cm and weight of 57g.

4) Barometer Unit: As we know, GPS module's height (i.e. altitude) readings are unreliable. Instead we use barometer (or sonar) to sense the height. The barometer sensor used is MS5611. MS5611 has a built-in 24bit AD converter, the communication interface is SPI.

5) Sonar Unit: In addition to barometer, we also use sonar to determine the height (altitude) of the quadcopter. The sonar sensor is attached to the bottom of the quadcopter. This sonar unit output analog voltages, and has a maximum sensing range of 7.65 meters.

6) Wireless Communication: In addition to the standard remote controller communication modules, we also install a Usmile Mini OSD MAVLink Flight Communication Module, to support digital communications between the quadcopter and ground computer station.

the MAVLink module works at 433MHZ. Its operating voltage is between 5V and 12V with standard 6-pin ISP header. It communicates with the quadcopter's microcontroller via standard UART interface.

7) Pulse Width Modulation (PWM) Inputs/Outputs: Motors are driven by PWM signals. The duty cycle of PWM on our quadcopter is 2.5ms (i.e. 400Hz) [2].

B. Main Program Layer: Control Strategies

Next, let us elaborate the control strategies implemented in the main program. The control strategies include location-angular control and height control, the control outputs of the two control sub-strategies are combined to drive the four propeller motors.

Location-Angular Control

The location control of a flying quadcopter is tightly coupled with the quadcopter's pitch, roll, yaw angular dynamics (sometimes the three angular dynamics are holistically called the "attitude" of the quadcopter) control. As shown in Fig. 3, a quadcopter moves forward/backward iff its pitch angle is non-zero; a quadcopter moves leftward/rightward iff its roll angle is non-zero.

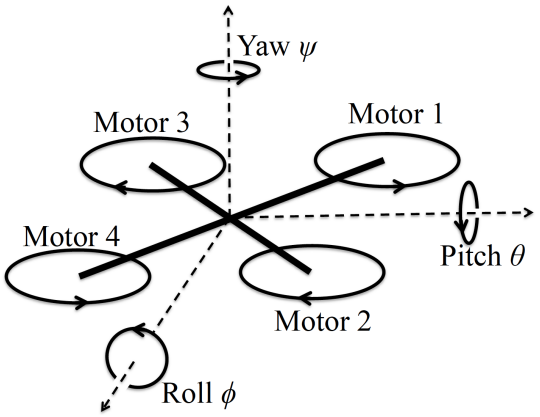

Fig. 3. Three Angular Movement Dimensions and Propeller Motor Numbering of a Quadcopter

Therefore, the location-angular control takes a nested outer-inner control loop form, as shown in Fig. 4. The outer control loop is the location control loop; and the inner control loop is the angular control loop.

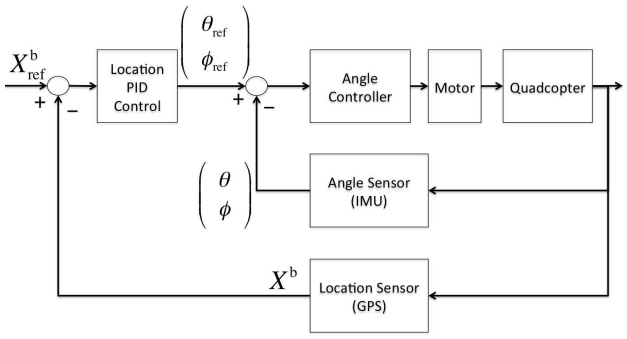

Fig. 4. Location-Angular Nested Control Loops

The input to the location (i.e. outer) control loop is the desired location coordinates (in terms of body-oriented (x, y)-coordinates, to be explained later) $X^{\mathrm{b}}_{\mathrm{ref}} = (x^{\mathrm{b}}_{\mathrm{ref}}, y^{\mathrm{b}}_{\mathrm{ref}})^{\mathsf{T}}$. The

control loop feedback $X^{\mathrm{b}} = (x^{\mathrm{b}}, y^{\mathrm{b}})^{\mathsf{T}}$ is the current quadcopter location coordinates (again, in body-oriented (x, y)-coordinates). The error $(X^{\mathrm{b}}_{\mathrm{ref}} - X^{\mathrm{b}})$ is fed to a PID controller to derive a desired attitude angle $(\theta_{\mathrm{ref}}, \phi_{\mathrm{ref}})^{\mathsf{T}}$, where θ_{ref} is the desired pitch angle and ϕ_{ref} is the desired roll angle. The desired pitch and roll angles serve as the input to the angular (i.e. inner) control loop, which will tilt the quadcopter's pitch angle θ and roll angle ϕ toward θ_{ref} and ϕ_{ref} respectively.

Formally, the control output of the outer control loop is described as follows.

$$\begin{aligned}
\theta_{\mathrm{ref}} &= k^{\mathrm{p}}_x(x^{\mathrm{b}}_{\mathrm{ref}} - x^{\mathrm{b}}) + k^{\mathrm{d}}_x(\dot{x}^{\mathrm{b}}_{\mathrm{ref}} - \dot{x}^{\mathrm{b}}) \\
&\quad + k^{\mathrm{i}}_x \int_0^t (x^{\mathrm{b}}_{\mathrm{ref}}(\tau) - x^{\mathrm{b}}(\tau))\mathrm{d}\tau, \\
\phi_{\mathrm{ref}} &= k^{\mathrm{p}}_y(y^{\mathrm{b}}_{\mathrm{ref}} - y^{\mathrm{b}}) + k^{\mathrm{d}}_y(\dot{y}^{\mathrm{b}}_{\mathrm{ref}} - \dot{y}^{\mathrm{b}}) \\
&\quad + k^{\mathrm{i}}_y \int_0^t (y^{\mathrm{b}}_{\mathrm{ref}}(\tau) - y^{\mathrm{b}}(\tau))\mathrm{d}\tau,
\end{aligned}$$

where k^{p}_x (also k^{p}_y), k^{i}_x (also k^{i}_y), and k^{d}_x (also k^{d}_y) are respectively the proportional, integral, and derivative control coefficients; t is the current time.

Note when specifying the (x, y)-coordinates of an aerial vehicle's location, usually two coordinate systems are used, as shown in Fig. 5. One is the *ground* (x, y)*-coordinate system*, where a fixed point on ground is chosen as the origin, the x-axis positive direction orients north, and the y-axis positive direction orients east. The other is the *body-oriented* (x, y)*-coordinate system*, which shares the same origin as the ground (x, y)-coordinate system, but rotates in the (x, y) plane by an angle ψ, such that the positive direction of x-axis aligns with the aerial vehicle's heading direction.

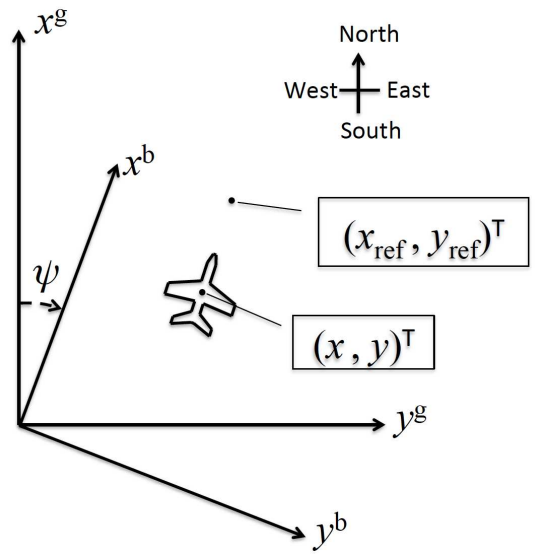

Fig. 5. Ground and Body-Oriented (x, y)-Coordinate Systems

The conversion formula between a ground (x, y)-coordinate $(x^{\mathrm{g}}, y^{\mathrm{g}})^{\mathsf{T}}$ and a body-oriented (x, y)-coordinate $(x^{\mathrm{b}}, y^{\mathrm{b}})^{\mathsf{T}}$ is

$$\begin{bmatrix} x^{\mathrm{b}} \\ y^{\mathrm{b}} \end{bmatrix} = \begin{bmatrix} sin\psi & cos\psi \\ cos\psi & -sin\psi \end{bmatrix} \begin{bmatrix} x^{\mathrm{g}} \\ y^{\mathrm{g}} \end{bmatrix}.$$

Next, let us discuss the angular (i.e. inner) control loop. The angular control loop is much more complex than the outer control loop, mainly because the angle sensor readings are unreliable. This is an inborn feature of MPU-6050 IMU sensors. However, thanks to the gyro inside of MPU-6050 IMU, we can get very reliable angular velocity readings. To fully exploit the above features, we adopt the well-known two-level PID angular control strategy [14] to implement Ard-μ-copter's angular control. This strategy is re-stated as follows.

Without loss of generality, the two-level PID pitch angular control diagram is shown in Fig. 6 (the roll and yaw angular control follows the same principles). Again this consists of an outer and an inner control loop. The outer control loop is the pitch angle control loop. The input is the desired pitch θ_{ref}, the feedback is the sensed current quadcopter pitch θ. The error $(\theta_{\mathrm{ref}} - \theta)$ is fed to a PID controller to generate a desired pitch angular velocity $\omega^{\mathrm{ref}}_{\theta}$. The desired pitch angular velocity $\omega^{\mathrm{ref}}_{\theta}$ serves as the input to the inner control loop: the pitch angular velocity control loop. The feed back of the pitch angular velocity control loop is the sensed pitch angular velocity ω_{θ} (sensed by the gyro in the MPU-6050 IMU). The error $(\omega^{\mathrm{ref}}_{\theta} - \omega_{\theta})$ is fed to a PID controller to generate the control signal u'_{θ} to be applied to quadcopter propeller motors (see Eq. (2) $\sim$ (5)). You can think of u'_{θ} as the component to adjust propeller motors to satisfy pitch angle control needs.

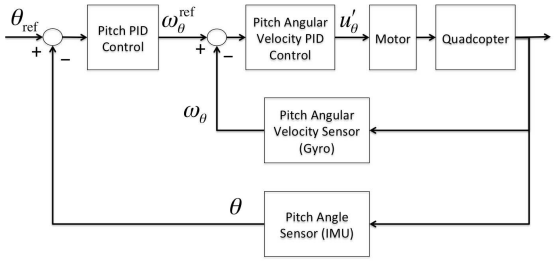

Fig. 6. Two-Level PID Pitch Angular Control

Formally, we have

$$\begin{aligned}
\omega^{\mathrm{ref}}_{\theta} &= k^{\mathrm{p}}_{\theta}(\theta_{\mathrm{ref}} - \theta) + k^{\mathrm{d}}_{\theta}(\dot{\theta}_{\mathrm{ref}} - \dot{\theta}) \\
&\quad + k^{\mathrm{i}}_{\theta} \int_0^t (\theta_{\mathrm{ref}}(\tau) - \theta(\tau))\mathrm{d}\tau, \\
u'_{\theta} &= k^{\mathrm{p}}_{\omega_\theta}(\omega^{\mathrm{ref}}_{\theta} - \omega_{\theta}) + k^{\mathrm{d}}_{\omega_\theta}(\dot{\omega}^{\mathrm{ref}}_{\theta} - \dot{\omega}_{\theta}) \\
&\quad + k^{\mathrm{i}}_{\omega_\theta} \int_0^t (\omega^{\mathrm{ref}}_{\theta}(\tau) - \omega_{\theta}(\tau))\mathrm{d}\tau,
\end{aligned}$$

where k^{p}_{θ} (also $k^{\mathrm{p}}_{\omega_\theta}$), k^{i}_{θ} (also $k^{\mathrm{i}}_{\omega_\theta}$), and k^{d}_{θ} (also $k^{\mathrm{d}}_{\omega_\theta}$) are respectively the proportional, integral, and derivative control coefficients.

Similarly, we can derive the control signal u'_{ϕ} and u'_{ψ} for roll and yaw angle control.

Height Control

The height control faces the similar challenge as angle control. The quadcopter height reading is provided by the barometer (MS5611) and/or sonar (XL-MaxSonar EZ4). This

reading is also unreliable. On the other hand, the vertical (i.e. height direction) acceleration readings a (provided by the accelerometer inside of the MPU-6050 IMU) is accurate. Therefore, the two-level PID angular control strategy of [14] also applies to height control [14]. We re-state the strategy as follows.

Fig. 7 describes the two-level height control loop. The outer loop is the height control loop. The input is the desired height h_{ref}. The feedback is the sensed quadcopter height h. The error $(h_{\text{ref}} - h)$ is fed to a PID controller to output a desired vertical acceleration a_{ref}. This a_{ref} together with gravitational acceleration g constitute the input to the inner control loop: the vertical acceleration control loop. The feedback of the vertical acceleration control loop is the sensed quadcopter's vertical acceleration a (sensed by the accelerometer inside of the MPU-6050 IMU). The error $(a_{\text{ref}} + g - a)$ is fed to a PID controller to create a control signal Δu_f to adjust the quadcopter propeller motors' throttle (see Eq. (1)(2) $\sim$ (5)). You can think of Δu_f as the component to adjust propeller motors to satisfy the vertical acceleration needs.

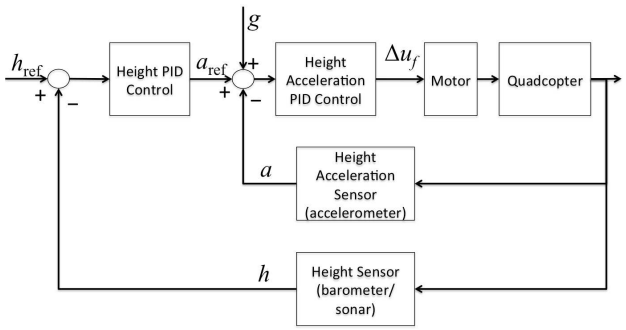

Fig. 7. Two-Level PID Height Control

Formally, we have

$$\begin{aligned} a_{\text{ref}} &= k_h^{\text{p}}(h_{\text{ref}} - h) + k_h^{\text{d}}(\dot{h}_{\text{ref}} - \dot{h}) \\ &\quad + k_h^{\text{i}} \int_0^t (h_{\text{ref}}(\tau) - h(\tau))\mathrm{d}\tau, \\ \Delta u_f &= k_a^{\text{p}}(a_{\text{ref}} + g - a) + k_a^{\text{d}}(\dot{a}_{\text{ref}} + \dot{g} - \dot{a}) \\ &\quad + k_a^{\text{i}} \int_0^t (a_{\text{ref}}(\tau) + g(\tau) - a(\tau))\mathrm{d}\tau, \end{aligned}$$

where k_h^{p} (also k_a^{p}), k_h^{i} (also k_a^{i}), and k_h^{d} (also k_a^{d}) are respectively the proportional, integral, and derivative control coefficients.

Total Control Output

Finally, all the above control outputs converge to become the control output toward quadcopter propeller motors.

The total control output consists of two high level components: throttle and angular control adjustments. The throttle component u_f is to control the vertical acceleration (ultimately, height) of the quadcopter. It is the same to each of the four propeller motors (see Fig. 3). The height control output Δu_f affects the throttle component: u_f is updated as per

$$u_f(t + \mathrm{d}t) = u_f(t) + \Delta u_f(t). \quad (1)$$

The angular control adjustments are different to each of the four propeller motors. Without loss of generality, suppose we are adjusting the pitch. Suppose we want to increase the pitch angle (see Fig. 3), then propeller 1 and 3's motors should speed up, while propeller 2 and 4's motors should slow down. Meanwhile, we cannot change the total throttle component. Therefore, the pitch angle control signal u'_θ should be applied positively to motor 1 and 3, but negatively to motor 2 and 4.

Combining all the above considerations, suppose U_1, U_2, U_3, and U_4 respectively represent the raw total control signal applied to propeller motor 1, 2, 3, and 4, then the update rules are

$$U_1(t + \mathrm{d}t) = u_f(t + \mathrm{d}t) + u'_\theta(t) + u'_\phi(t) - u'_\psi(t), \quad (2)$$
$$U_2(t + \mathrm{d}t) = u_f(t + \mathrm{d}t) - u'_\theta(t) + u'_\phi(t) + u'_\psi(t), \quad (3)$$
$$U_3(t + \mathrm{d}t) = u_f(t + \mathrm{d}t) + u'_\theta(t) - u'_\phi(t) + u'_\psi(t), \quad (4)$$
$$U_4(t + \mathrm{d}t) = u_f(t + \mathrm{d}t) - u'_\theta(t) - u'_\phi(t) - u'_\psi(t). \quad (5)$$

The above raw control signals U_1, U_2, U_3, U_4 are then range constrained and then normalized to create a PWM wave to drive the respective motors. A normalized value of 100% drives the motor to maximum speed, while 0% stops the motor.

Tuning PID Coefficients

PID controllers date back to 1890s. While proportional control provides fast response to reference set point changes and small disturbances, it cannot fully eliminate the impacts of steady disturbances, e.g. a stiff gale. To eliminate the steady disturbance impacts, we need integral control. Finally, derivative control constrains overshoot, hence improves control stability [15].

Generally, we first tune the proportional coefficients of the PID controllers to see if the quadcopter can achieve fast enough response time and acceptable overshoot. Then we increase the derivative coefficients to further reduce the overshoots. The integral control We will show the experiment result in the next section to prove our solution.

IV. Evaluation

We implemented the Ard-μ-copter architecture described in Section III (see Fig. 1). Next, we carry out various experiments to evaluate the performance of our implementation.

First, we fix the pitch axis (see Fig. 3) to a rig, which is in turn fixed on ground. We want to check whether the pitch angle control can keep the quadcopter horizontal on the pitch angular direction. We turn on the IMU sensor at time $t = 0$ (the IMU sensor needs to be powered on for at least 8 seconds before it can properly work); and turn on the motors at around $t = 15$ second. The pitch angle trace is shown in Fig. 8. According to the figure, we see the pitch angle control can effectively keep the quadcopter horizontal on the pitch angle dimension.

Next, we test how well the pitch control can track manual control inputs. That is, instead of keeping a pitch setpoint of 0, we now manually change the setpoint. Fig. 9 shows the pitch angle trace tracking the setpoint trace. The red curve is the trace of the pitch setpoint (i.e. where the operator wants

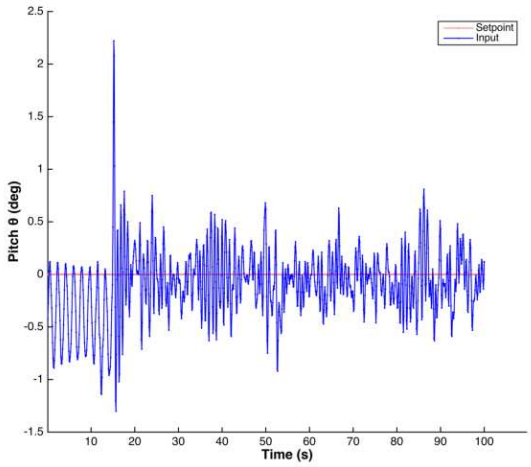

Fig. 8. Pitch Angle Horizontal Stabilization (IMU sensor powered on since $t = 0$; motors powered on since around $t = 15$ second; "Setpoint" means the desired pitch angle)

the pitch angle to be); while the blue curve is the trace of the controlled pitch angle. The IMU is powered on at $t = 0$; the motors are powered on around $t = 10$ second. We see the pitch trace very well tracks the setpoint trace.

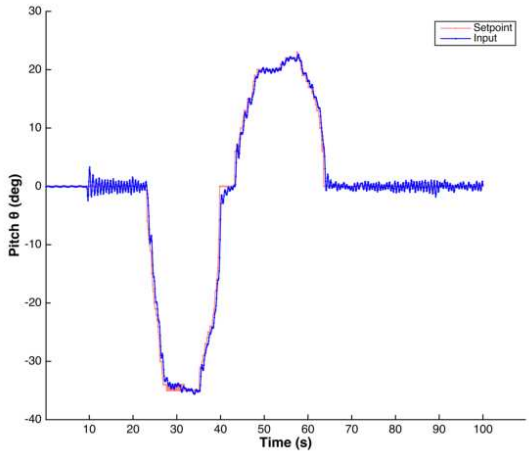

Fig. 9. Pitch Tracking (IMU sensor powered on since $t = 0$; motors powered on since around $t = 10$ second; "Setpoint" means the desired pitch angle)

Next, we do the same to test the roll angle control. The derived roll angle horizontal stabilization trace and tracking trace are shown in Fig. 10 and 11 respectively.

Next, we no longer fix the quadcopter to a rig, but let it hover in the air and test its height control. Fig. 12 shows the height control trace. The IMU and sonar are powered on at $t = 0$. At about $t = 17$ second, we power on the motor to a very high speed, so that the quadcopter takes off. At about $t = 19$ second, we change the height setpoint (i.e. the desired height) from 0cm to about 95cm. After that, we can see the height control takes effect: the quadcopter quickly stabilizes itself around the new height setpoint.

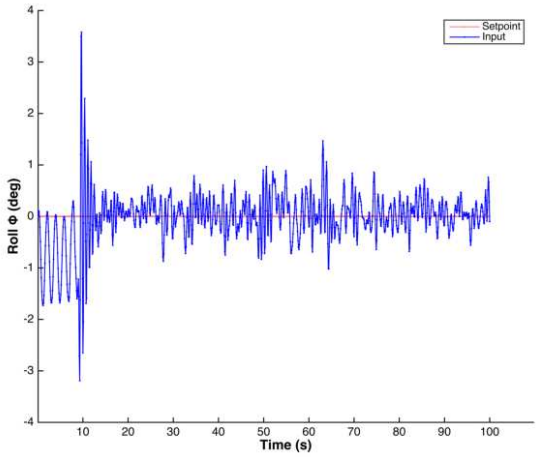

Fig. 10. Roll Angle Horizontal Stabilization (IMU sensor powered on since $t = 0$; motors powered on since around $t = 9$ second; "Setpoint" means the desired roll angle)

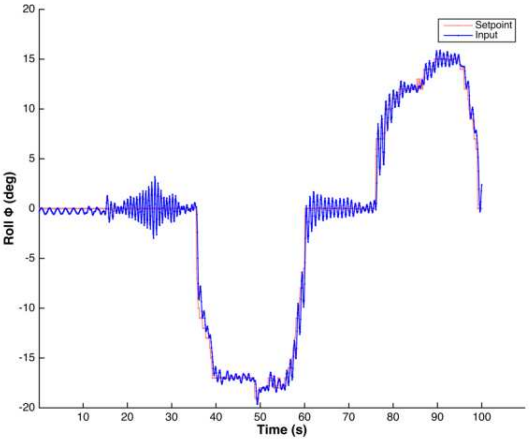

Fig. 11. Roll Tracking (IMU sensor powered on since $t = 0$; motors powered on since around $t = 15$ second; "Setpoint" means the desired roll angle)

V. Conclusion

In this paper, we build an open and simple quadcopter platform, Ard-μ-copter. Comparing to other open source quadcopter platforms, Ard-μ-copter has a very shall source code size (135KB), and is very well documented. Experiments show that the quadcopter can effectively control its flight.

Acknowledgement

The research project related to this paper is supported in part by Hong Kong RGC General Research Fund (GRF) PolyU 152164/14E, RGC Early Career Scheme (ECS) PolyU 5328/12E, ECS 25214015, The Hong Kong Polytechnic University internal fund A-PJ80, A-PK46, A-PL82, G-YN37, G-UA7L Department of Computing start up fund; and by China NSFC61401385, National 863 Program 2013AA013202. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of sponsors. The authors thank

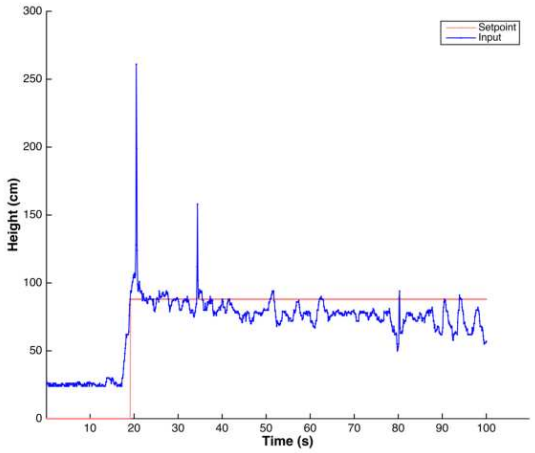

Fig. 12. Height Control Trace (IMU and sonar powered on at $t = 0$; motors powered on around $t = 17$ second; "Setpoint" means the desired height)

anonymous reviewers for their efforts/advice on improving this paper.

REFERENCES

[1] "Ardupilot - home," *http://ardupilot.com*, 2015.
[2] R. Mancuso, O. D. Dantsker, M. Caccamo, and M. S. Selig, "A low-power architecture for high frequency sensor acquisition in many-dof uavs," *2014 ACM/IEEE International Conference on Cyber-Physical Systems(ICCPS)*, pp. 103–114, 2014.
[3] Z. He, S. Li, Z. Shen, M. U. Khan, Z. Shao, and Q. Wang, "Wip abstract: A quadcopter swarm for active monitoring of smog propagation," *Proc. of IEEE/ACM ICCPS*, 2015.
[4] R. Mahony, V. Kumar, and P. Corke, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor," *Robotics Automation Magazine*, 2012.
[5] L. Taeyoung, M. Leoky, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se(3)," *In Decision and Control (CDC), 2010 49th IEEE Conference on*, pp. 5420–5425, 2010.
[6] P. Pounds, R. Mahony, and P. Corke, "Modelling and control of a large quadrotor robot," *Control Engineering Practice*, vol. 18, no. 7, pp. 691–699, 2010.
[7] S. Omari, M.-D. Hua, G. Ducard, and T. Hamel, "Hardware and software architecture for nonlinear control of multirotor helicopters," *Mechatronics, IEEE/ASME Transactions on*, pp. (99)1–13, 2013.
[8] G. Ducard and M.-D. Hua, "Discussion and practical aspects on control allocation for a multi-rotor helicopter," *In UAV-g 2011, Conference on Unmanned Aerial Vehicle in Geomatics*, pp. 1–6, 2011.
[9] V. G. Adir, A. M. Stoica, and J. F. Whidborne, "Modelling, stabilization and single motor failure recovery of a 4y octorotor," *In Proc. 13th IASTED International Conference on Intelligent Systems and Control (ISC 2011)*, pp. 82–87, 2011.
[10] R. Mahony and T. Hamel., "Robust trajectory tracking for a scale model autonomous helicopter," *International Journal of Robust and Nonlinear Control*, vol. 14, no. 12, pp. 1035–1059, 2004.
[11] "Dji - home," *http://www.dji.com*, 2015.
[12] "Micropilot - home," *http://www.micropilot.com*, 2015.
[13] "Atmega2560 - home," *http://www.atmel.com*, 2015.
[14] M. W. N. J. Ming, "Principles of quadcopter and two-level pid control/si zhou fei xing qi fei xing yuan li yu shuang bi huan pid kong zhi (in chinese)," *http://bbs.ednchina.com/BLOG_ARTICLE_3027342.HTM*, 2015.
[15] "Pid controller wiki," *https://en.wikipedia.org/wiki/PIDcontroller*.