2021 International Conference on Unmanned Aircraft Systems (ICUAS)
Athens, Greece. June 15-18, 2021

RflySim: A Rapid Multicopter Development Platform for Education and Research Based on Pixhawk and MATLAB

Shuai Wang, Xunhua Dai, Chenxu Ke and Quan Quan

2021 International Conference on Unmanned Aircraft Systems (ICUAS) | 978-1-6654-1535-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICUAS51884.2021.9476786

Abstract— **In this paper, we propose and open a rapid development platform−−RflySim based on Pixhawk/PX4 and MATLAB/Simulink for UAV education and research. This platform adopts model-based development ideas and uses software-in-the-loop simulation and hardware-in-the-loop simulation to accelerate physical deployment. With that, beginners and developers can directly use MATLAB/Simulink to design low-level controllers (such as attitude control, position control) and high-level applications (such as decision-making, autonomous flight), and then deploy them into a multicopter autopilot system with no need to access the C/C++ underlying code. Three demonstrations are presented to verify the ease of use and the high efficiency of the proposed platform.**

- **Open source:** `https://github.com/RflySim/RflyExpCode`
- **Documentation:** `https://rflysim.com/en/index.html`

I. INTRODUCTION

Robotic technologies are widely used and growing rapidly in recent years, and the development of unmanned aerial vehicle (UAV) is particularly eye-catching. Under such a circumstance, many platforms related to UAV education and research have emerged. AR.Drone [1] and Tello [2] are educational platforms that provide application program interface (API) for position control, target tracking, etc., but the core functions are either proprietary or can be accessed only in part. These commercial platforms do not provide the simulation function for debugging and data inspecting, so it is inefficient to perform experiments on real aircraft. PiDrone [3] focuses on high-level applications, such as SLAM, and does not involve the development of low-level flight control. Some research institutions and universities propose many excellent hardware and software ideas for UAV research [4], [5], [6], and the proposed platforms are versatile but not open to use. Pixhawk/PX4 [7], [8] and APM [9] are popular open-source UAV platforms, where the low-level and high-level applications can be modified or used directly. They have a complete set of development systems, including software-in-the-loop (SIL) and hardware-in-the-loop (HIL) interfaces for simulation and debugging. But their system architectures are somewhat complex for beginners to get started with them. If beginners and developers (called users below) want to modify the source code, they should be familiar with the C/C++ programming language and Linux system knowledge.

*This work was supported by the National Natural Science Foundation of China under Grant 61973015 and Beijing Natural Science Foundation under Grant L182037.

Shuai Wang, Chenxu Ke and Quan Quan(Corresponding Author) are with the School of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, China. Email: qq_buaa@buaa.edu.cn.
Xunhua Dai is with Computer Science and Engineering, Central South University, Changsha 410083, China. Email: dai.xh@csu.edu.cn.

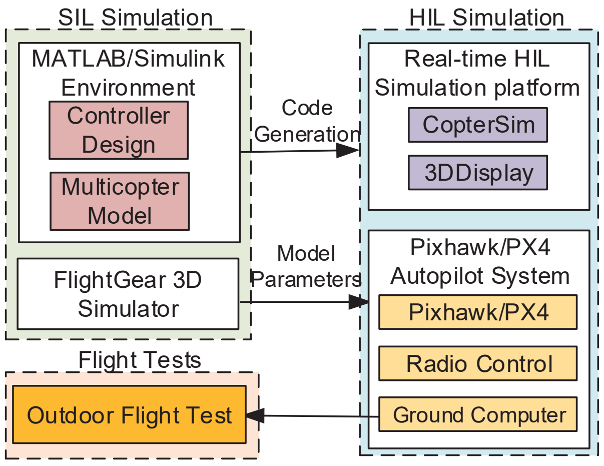

Fig. 1. System architecture of RflySim.

On the other hand, in recent years, many researchers have proposed excellent UAV algorithms, like low-level attitude control methods [10], [11], high performance attitude estimator [12], position control methods [13], mixers that can help achieve accurate tracking [14], and high maneuvering strategy [15]. How to efficiently realize these methods is a problem. The transplantation of algorithms from theory to a real aircraft is a complex process, which requires much programming work. Any minor error or omission in the programming process may cause unsatisfactory experimental results. However, in the field of robotics, experimental verification of the proposed method has become an increasingly common requirement and will consume a lot of time with a long trial-and-error cycle.

MATLAB/Simulink [16] is widely used because of its rich tools, which facilitates the development of the robot system. Furthermore, MATLAB/Simulink also supports the automatic code generation of C/C++ language for the deployment to embedded systems such as Pixhawk, which reduces the difficulty between simulation testing and physical deployment. With MATLAB/Simulink, UAV dynamic models, controllers, filters and decision-making logic can be designed efficiently.

To overcome the shortcomings of existing platforms and take advantage of existing tools, we design a rapid development platform based on Pixhawk/PX4 and MATLAB/Simulink. With that, users can directly use Simulink to design a low-level controller of multicopter such as attitude controller, position controller, and high-level applications such as decision-making, autonomous flight with no need to master C/C++ programming language. Besides, We provide an accurate quadcopter model to ensure that the results obtained by SIL simulation can be directly used in the real system [17]. Through the automatic code generation technology, the Simulink model can be automatically deployed to Pixhawk autopilots, and the real flight experiment can be performed indoor or outdoor. Moreover, due to rich debugging functions such as online parameter debugging, offline data recording, and online data inspecting, the process of UAV algorithm learning, development, and verification can be greatly accelerated.

To demonstrate the effectiveness of our platform, we showcase three examples including attitude control, position control and mode switch. The first is related to the design of a low-level attitude controller. We use the proposed platform to quickly complete the active disturbance rejection control (ADRC) controller design, simulation, debugging, and real flight test. It has been shown that the ADRC method we design has a robust anti-disturbance ability. In the second example, the implementation of fault-tolerance control shows that our platform can support high-performance control design for an extreme situation with high control rate. The last example about a failsafe logic design involves the design of the high-level decision-making of flight control. In this example, the multicopter returns home and lands autonomously when the radio control (RC) is lost.

A general and comprehensive introduction to this platform has been summarized in the published book [18] you can also refer to the video https://youtu.be/RTkOHJ0NT0k for a brief introduction. This paper is a brief introduction and extension of the work in [18] to support a complete autopilot development. Specifically, this paper makes the following contributions: (1) the addition of three debugging functions to the platform to connect the ground control station, making the platform more practical in helping developers deploy new algorithms on real aircraft; (2) the design of a time-triggered model that ensures different control algorithms run at the stable frequency respectively; (3) new demonstration of attitude control by ADRC and fault-tolerance control to show that the platform can support some advanced controllers.

II. ARCHITECTURE

The overall architecture of the system is shown in Fig. 1. The entire development consists of three steps: SIL simulation based on MATLAB/Simulink, HIL simulation, and real flight test.

The SIL simulation system includes a multicopter control model, a multicopter dynamic model, and a 3D display model. All three parts are built in the Simulink environment. The multicopter model sends sensor data or state information to the controller such as attitude and velocity. The controller sends each motor's control signal back to the model to form a SIL simulation closed-loop system.

HIL simulation means that the multicopter controller runs in Pixhawk hardware. The sensor data of the multicopter is similar to that of the SIL simulation and is also generated by a mathematical model. Besides, to ensure the real-time simulation, we design an application–CopterSim. The multicopter control model used in the HIL simulation is consistent with the SIL simulation, and the PSP toolbox can automatically convert the Simulink control model into C/C ++ code and integrate it into the Pixhawk/PX4 autopilot. CopterSim sends sensor data such as accelerometer, barometer, magnetometer, GPS to the Pixhawk system via a USB serial port. The Pixhawk/PX4 autopilot will receive the sensor data for state estimation and send the estimated state information to the controller through the internal uORB message bus. The controller sends the control signal of each motor as the output back to CopterSim via the USB serial port. Thereby a closed loop is established in the HIL simulation. Compared with the SIL simulation, the control algorithm can be deployed and run in a real embedded system.

In the real flight test, CopterSim's simulation model is further replaced by a real multicopter. The sensor data is directly obtained by sensor chips, and the controller signal is directly delivered to the motor. It is worth noting that the control model used by three processes is the same, and only the output module of the model is changed to achieve different experimental purposes.

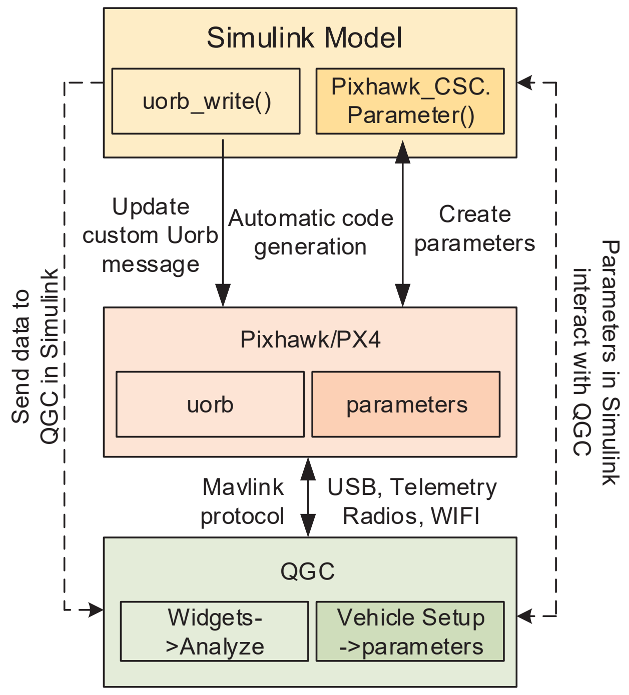

Fig. 2. The implementation process of online parameter tuning and real-time data inspecting.

A. *Model*

To improve the development efficiency of the flight controller further, we make full use of the functions of the Pixhawk/PX4 autopilot and add the following functions to the Simulink model we built.

- Online parameter tuning. Users can modify the parameters in the Simulink model online through a ground control station, namely QGroundControl (QGC) [19]. There is no need to recompile and upload the program after each parameter modification, and it improves the program debugging efficiency. The implementation process is shown in Fig. 2. First, some 'Simulink.Parameter' objects are created in Simulink, which can be used to share a value among multiple block parameters. Then, through automatic code generation, they can be converted to parameters in PX4, and these parameters can interact with QGC like parameters in PX4.
- Data recording. In order to facilitate experimental analysis, any data in the Simulink model can be recorded to the SD card. The number and frequency of the recorded data can be flexibly adjusted.
- Real-time data inspecting. If users need to observe the state of the aircraft or the intermediate variables of the model in real-time during the flight, it is also very convenient to add a data observation module to the Simulink model, and the data can be observed in QGC. What is more, QGC can record the data to a text file.The implementation process is shown in Fig. 2. The basic principle is to write data to the uORB message.

III. CORE COMPONENTS

The following describes the core components used in the entire system.

A. *Pixhawk/PX4 Autopilot System*

We choose Pixhawk series as the hardware, PX4 as the software, RC for remote manual operation, and ground computer for data inspecting. Pixhawk is an independent open-source hardware project dedicated to providing easy-to-use, high-quality and low-cost autopilot hardware for education, enthusiasts and developers. PX4 [20] is an open-source flight control software system. It runs on the Pixhawk series autopilot hardware and constitutes the Pixhawk/PX4 autopilot software and hardware platform. It is a small drone autopilot platform that is widely used around the world.

B. *MATLAB/Simulink and Pixhawk Support Package(PSP)*

We choose MATLAB/Simulink as the programming environment because it is widely used in aerial vehicles, cars, and other applications. And it can be easily applied to develop a simulation system for dynamic system modeling, controller design, software and hardware simulation, and performance analysis through a modular programming language. With the Pixhawk support package (PSP) toolbox [21], the Simulink model can be deployed to Pixhawk automatically. We have made some updates and optimizations based on the official PSP toolbox to make it compatible with wider PX4 and MATLAB versions.

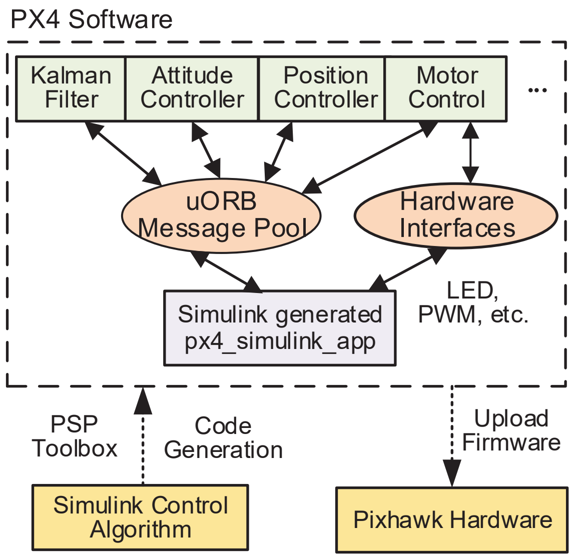

Fig. 3. The relationship among PSP toolbox, Pixhawk/PX4 autopilot and Pixhawk hardware system.

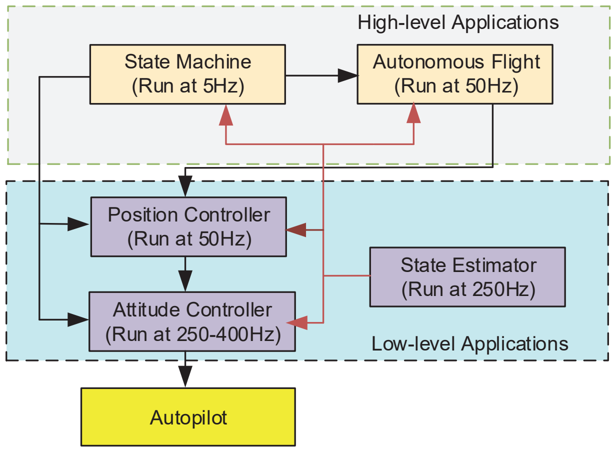
High-level Applications
State Machine (Run at 5Hz)
Autonomous Flight (Run at 50Hz)
Position Controller (Run at 50Hz)
State Estimator (Run at 250Hz)
Attitude Controller (Run at 250-400Hz)
Low-level Applications
Autopilot

Fig. 4. Flight control system structure of multicopter. Autonomous flight contains guidance and navigation. State machine is a decision-making module that controls the status transition in autopilot such as arming, disarming. The controllers make the vehicle move towards the setpoint. The estimator computes states of the multicopter such as attitude and position.

Fig. 3 shows the relationship among the PSP toolbox, the PX4 software system, and the Pixhawk hardware system. After the PSP toolbox generates the algorithm code, it is embedded into the Pixhawk/PX4 autopilot. A separate module (with independent thread) named "px4_simulink_app" will be created, which does not affect the operation of the native control modules in Pixhawk/PX4 autopilot. Instead, it runs in parallel with other modules. In this way, the native modules of the Pixhawk/PX4 autopilot such as filter, attitude controller can also be replaced.

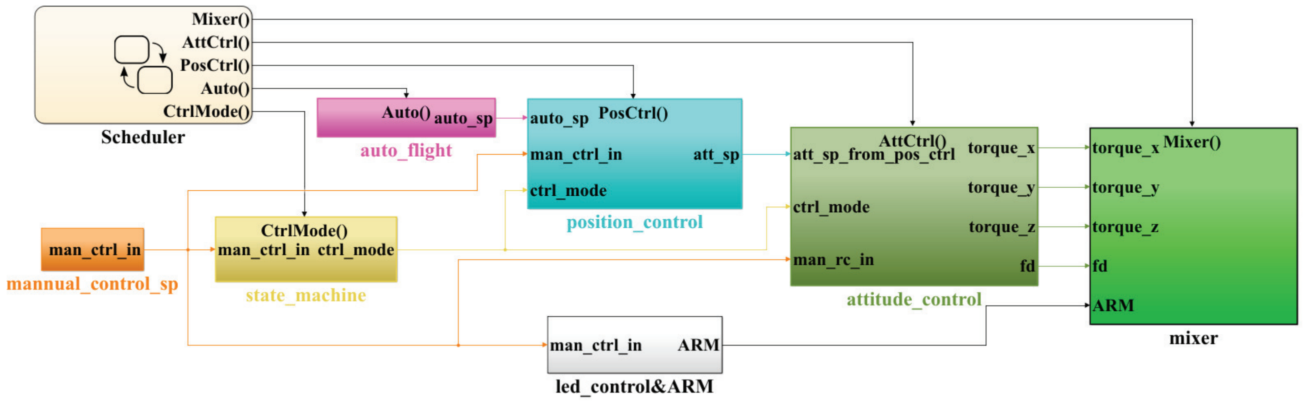

Fig. 5. Flight control model designed in Simulink.

C. Real-time HIL Simulation Platform

CopterSim and 3DDisplay together constitute a HIL simulation platform, both of which run on Windows operating system. CopterSim is a real-time motion simulation software developed for the Pixhawk/PX4 autopilot. Users can configure multicopter model parameters in the software, and connect it to the Pixhawk autopilot via a USB serial port to implement HIL simulation. 3DDisplay is a real-time 3D visual display software. It receives flight data from CopterSim via UDP to display the attitude and position of multicopter in real-time. Some simulation tools also support HIL simulation, such as AirSim [22]. However, it has a high entry for beginners to modify the model parameters of UAV. We have provided the tools to beginners and researchers for free, and they can be downloaded at https://rflysim.com/en/index.html.

D. Model

Using Simulink's graphical user interface, the hierarchical design of the flight control can be easily carried out, so that the whole system structure is clear and easy to maintain. The overall structure of the flight control system is shown in Fig. 4. In general, it can be divided into two levels: high-level applications include state machine and autonomous flight, and low-level applications include position control, attitude control, and state estimator. At the same time, to focus on the design, we do not need to design each part by ourselves from scratch. On the other hand, many functions of Pixhawk/PX4 autopilot have been successfully tested and applied. We can fully use the existing functions of this open-source system. For example, we only design the controller and state machine based on the state estimator from the current estimator in the Pixhawk/PX4 autopilot. The Simulink model is shown in Fig. 5. Users can open the module and replace it with their algorithms or add a new module such as filter for their purpose. It is easy to interact in the GUI interface.

IV. SYSTEM ANALYSIS

When deploying algorithms in an embedded system, real-time performance of the algorithm is a very noteworthy issue. For control algorithms, the real-time performance is

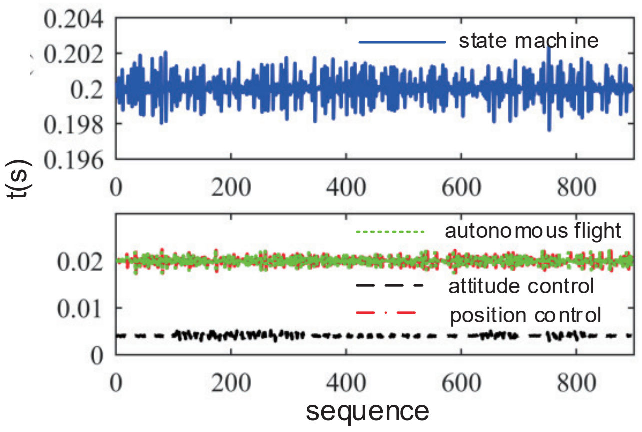

Fig. 6. The execution cycle of each module.

particularly important because it directly affects the bandwidth and robustness of the control system. We use the time-triggered method [23] to ensure the algorithm runs at a stable frequency, and it can be easily implemented in Simulink with Stateflow schedulers, as shown in Fig. 5. The underlying control logic of flight control generally adopts hierarchical control, as shown in Fig. 4. The attitude estimator update frequency in Pixhawk/PX4 autopilot is 250Hz, so we set the execution frequency of the attitude controller to 250Hz. The IMU has a refresh frequency of 1000 Hz, so the angular rate control frequency can be increased to a high level, such as 400Hz, with the refresh frequency of the electronic speed control (ESC) under consideration. The control frequency of the position control is generally lower than the attitude control, and the bandwidth of the attitude control should be 4 to 10 times of position control, so the execution frequency of the position control is set to 50Hz. The high-level controller does not need a high execution frequency, so the execution frequency of the state machine is set to 5 Hz, and the autonomous flight is set to 50 Hz. The estimator in Pixhawk/PX4 autopilot is directly used for state observation. This further demonstrates the superiority of our platform. Instead of building our flight control system from scratch, we focus mainly on the areas we are interested in, such as the design of attitude control, estimator, or state machine.

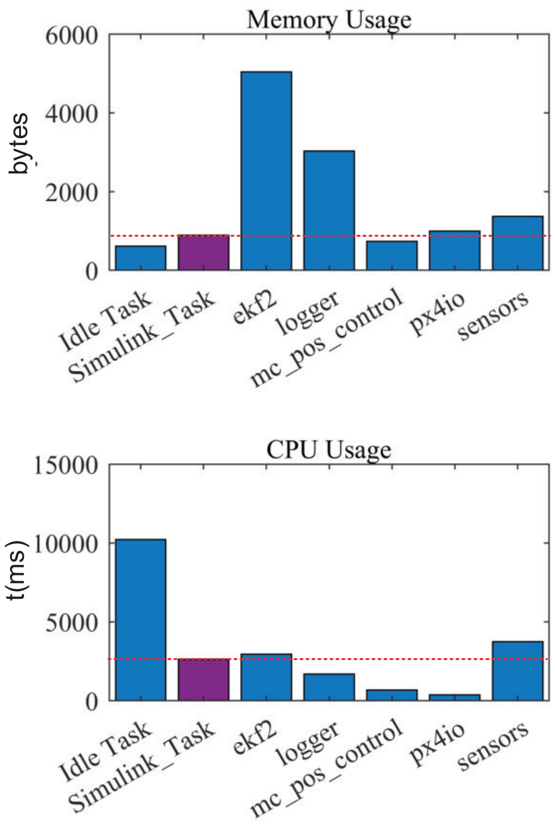

Fig. 7. Comparison of the hardware resource usage in the custom module and Pixhawk/PX4 autopilot.

Through the data recording function, the execution period of each module is obtained, as shown in Fig. 6. It can be seen that the execution cycle of each module in the system is controlled accurately. In addition, considering the hardware resource limitation of Pixhawk, the controller or state estimator we design should not be too complicated. Some online optimization methods such as MPC [24] cannot run in real-time with limited computing resources. The memory usage of main tasks running in Pixhawk/PX4 autopilot is shown in Fig. 7. It can be seen that the memory usage of the attitude control and position control we design is half of that the autopilot's. The CPU usage of our attitude control and position control is slightly higher than that in Pixhawk/PX4 autopilot, but the autopilot's CPU resource remains idle for 30.95% of the time.

V. CASES STUDY

Flight control involves dynamic system modeling, sensor calibration, state estimation, controller design, and decision logic design [25]. For all the knowledge, we have given step-by-step procedures and reference design examples based on our platform, and learners can refer to https://github.com/RflySim/RflyExpCode. Here, we first design the ADRC law for of low-level attitude control system as an example. After that, a fault-tolerance control example shows the fast response of the designed position controller by the proposed platform. Finally, we design a high-level failsafe logic in the form of a state machine for the RC signal being lost.

A. ADRC Controller Design

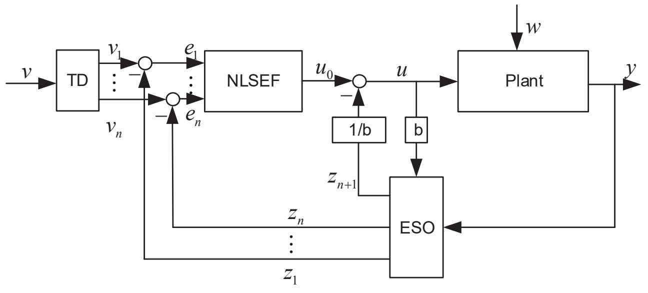

Fig. 8. Structure of the ADRC, which consists of a tracking differentiator(TD), a nonlinear state error feedback(NLSEF), and an expanded state observer(ESO).

ADRC law is a more and more popular method in practice [26], with the structure of ADRC shown in Fig. 8. Obviously, it is considered as an advanced version of PID controller but with a more complex structure. It is an appropriate testing case.

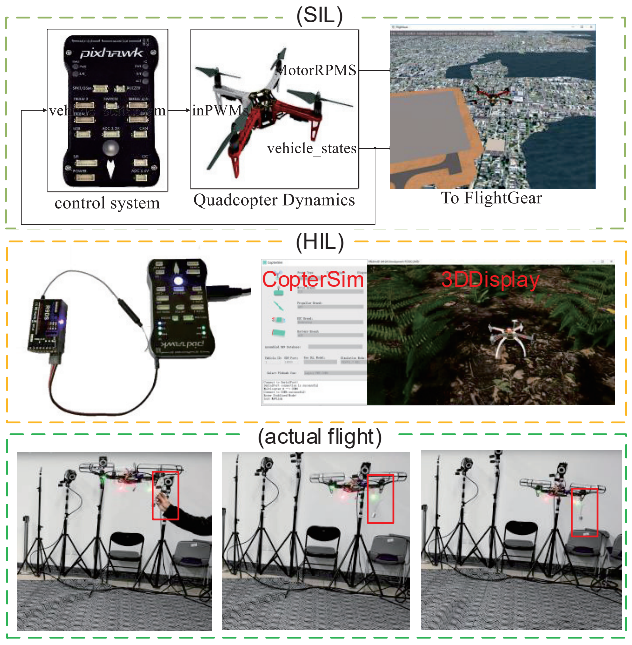

Fig. 9. Design procedure of ADRC including SIL simulation, HIL simulation and Real flight test.

The controller is designed in Simulink, and the SIL simulation is performed, as shown in Fig. 9(SIL). We can easily observe the interested states in Simulink and adjust the model to make it satisfied. As shown in Fig. 10(SIL), the controller

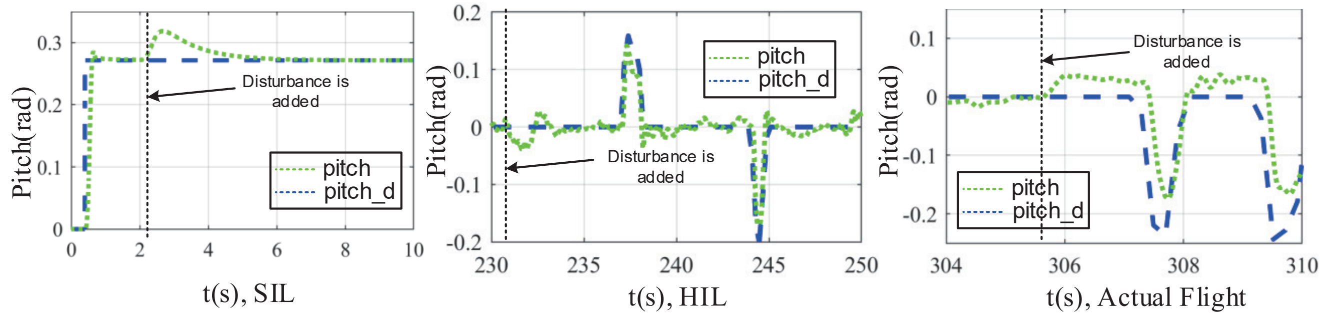

Fig. 10. Response of pitch angle of SIL simulation, HIL simulation and actual flight test with ADRC as a controller. Disturbance is added to each experiment. The blue dotted line denotes the desired pitch angle, and the green dotted line denotes the response of aircraft.

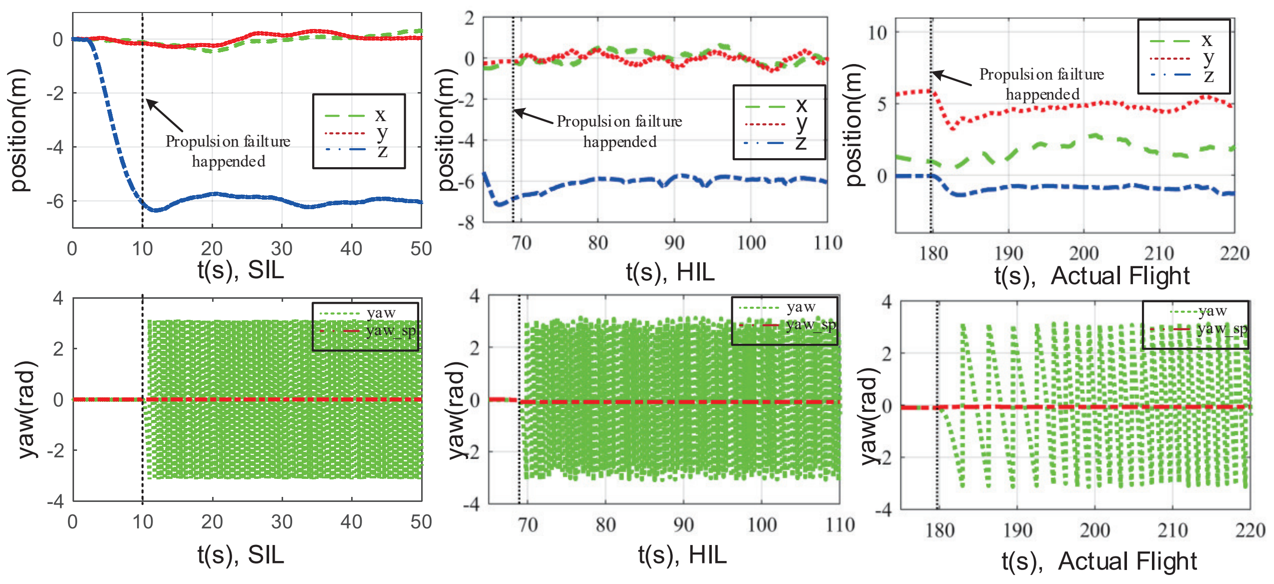

Fig. 11. SIL simulation, HIL simulation and outdoor flight tests of fault-tolerance control.

can eliminate the influence of disturbance very well. After that, in the HIL simulation, only a simple modification of the model's output is made for C/C++ code generation, but the controller will keep the same as the SIL simulation. The generated C/C++ code will be uploaded to Pixhawk automatically. Then, HIL simulation can be performed, as showed in Fig. 9(HIL). In this process, we can manually control the multicopter and observe the aircraft's response, as shown in Fig. 10(HIL). Controller parameters can be adjusted according to the HIL simulation performance. Finally, the actual flight test is performed, as shown in Fig. 9(Actual Flight). We add 0.2kg weight to a 1.4kg multicopter as an external disturbance. Under the influence of external disturbance, the multicopter tilts slightly and then begins to adjust its attitude. After the adjustment, the multicopter returns to the horizontal state and the disturbance is rejected, as shown in Fig. 10(Actual Flight). It should be noted that, based on the proposed platform, the designer only needs to design and improve the controller in Simulink. Basically, the controllers are almost the same in SIL simulation, HIL simulation and real flight.

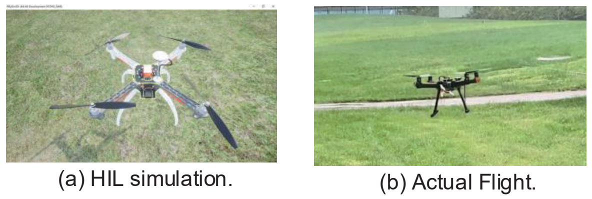

Fig. 12. HIL simulation and actual flight test scenes of fault-tolerance control.

B. Fault-tolerance Control

The growing interest in aerial robotics has prompted researchers to consider controllers for safety-critical systems. There are great demands on the ability to tolerate propulsion failure of quadcopter during the mission. After one rotor of a quadcopter fails, it is proved that the uncontrollability of the yaw will make the quadcopter spin around a certain axis at a high speed [27]. Fast response has to make in that the closed-loop stability of the fault-tolerance control system is very sensitive to control input and estimation phase lag. This is a perfect benchmark control problem to verify the high performance of the proposed platform.

To test the proposed platform, the fault-tolerance controller adopts that proposed in [28] and runs at 400Hz to ensure a fast response. In SIL, the state estimation directly uses the ground truth rather than from the estimator of PX4. In this phase, most controller problems in the algorithm are remedied. After SIL succeeds, as shown in Fig. 11(SIL), we turn to the HIL simulation, where the state estimation from PX4 and the controller downloaded in hardware are both in the closed-loop control system. During this phase, we found that commonly-used EKF estimator will fail because of the large estimation phase lag. Instead, we choose the complementary filter to satisfy the stability shown in Fig. 12(a) and Fig. 11(HIL). Finally, the same code and parameters are compiled and download to a real quadcopter, and the similar control performance is achieved in actual flight shown in Fig. 12(b) and 11(Actual Flight). It can be seen that, after one rotor fails, the quadcopter spins at a high speed, but it's position is still controllable.

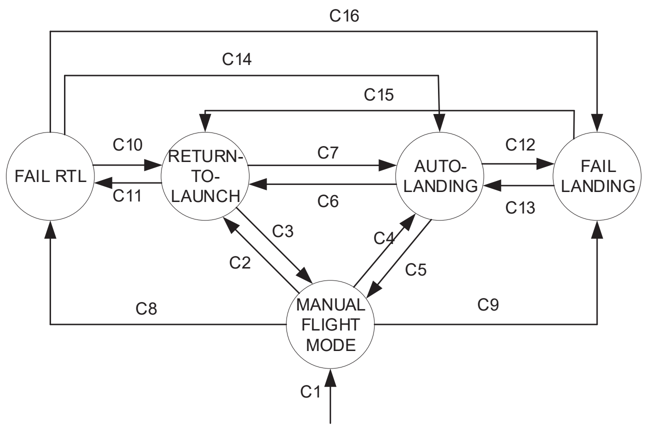

Fig. 13. State machine of failsafe.

C. Failsafe

For multicopters, several types of failure cannot be avoided, including communication breakdown, sensor failure, and propulsion system anomalies. To guarantee safety, the multicopter's decision-making module should prevent or mitigate the unsafe consequences of system failures. Here we only consider the failure when the RC is lost, and design failsafe logic for this case. To facilitate the failsafe logic design, five flight modes are defined here: 1) MANUAL FLIGHT MODE. This mode allows a remote pilot to control a multicopter manually. 2) RTL MODE. Under this mode, the multicopter will return to the home location and hover there. 3) AUTO-LANDING MODE. In this mode, the multicopter automatically lands. 4) FAIL LANDING MODE. The multicopter automatically lands when the RC is lost. 5) FAIL RTL MODE. The multicopter automatically returns to the initial horizontal position when the RC is lost.

Two kinds of events are defined: Manual Input Events (MIEs) and Automatic Trigger Events (ATEs). These events will cause the state or mode transition.

- MIES. MIEs are instructions from remote pilots sent through the RC transmitter, including the following: 1) MIE1: arm and disarm instructions; 2) MIE2: manual operation instruction (1: switch to MANUAL FLIGHT MODE; 2: switch to RTL MODE; 3: switch to AUTO-LANDING MODE).
- ATES. ATEs are independent of the remote pilot's operations but mainly generated by the status of components onboard and status of the multicopter. 1) ATE1: status of connections of RC (ATE1 = 1: normal; ATE1= 0: abnormal); 2) ATE3: (ATE3 = 0: multicopter's distance from the HOME point is above the threshold; ATE3 = 0: multicopter's distance from the HOME point is under the threshold).

The state machine is shown in Fig. 13, where Ci represents the corresponding transition condition. Where C1: MIE1 = 1. C2, C7, C10, C1: ATE1 = 1 and MIE2 = 2. C3, C5: ATE1 = 1 and MIE2 = 1. C4, C6, C12, C14: ATE1 = 1 and MIE2 = 3. C8: ATE1 = 0 and ATE3 = 0. C9: ATE1 = 0 and ATE3 = 1. C11, C13: ATE1 = 0. C16: ATE1 = 0 and ATE3 = 1.

The result in SIL simulation is shown in Fig. 14(SIL). It can be observed that, during 0-10s, the multicopter is in MANUAL FLIGHT MODE. After 10s, the RC is lost, and the altitude is less than the threshold. As the logic we designed, the multicopter first climbs up to the safe altitude we set, then enters RTL MODE. The return process is completed at 18s, with the horizontal position being (0, 0). Then, the multicopter enters AUTO-LANDING MODE and finally completes landing at 39s.

After the HIL simulation further verifies the correctness of the model, the real flight is performed. To ensure safety, the safe altitude set in the flight test is 5m. The final real flight curve is shown in Fig. 14(Actual Flight). At about 335s, the RC is lost, then the multicopter rises to altitude 5m, and returns to the HOME point horizontally. Finally, the multicopter lands with its altitude decreasing to 0. The real flight results show that multicopter can achieve failsafe. Based on the proposed platform, the designer only needs to design and improve the state machine in Simulink. What is more, Simulink can help to identify hidden errors in the designed state machine, like dead logic. This will facilitate the design.

VI. CONCLUSIONS

In this paper, we introduce a rapid multicopter development platform for education and research based on Pixhawk/PX4 and MATLAB/Simulink. This platform supports developers to design controllers, estimators, decision-making in MATLAB/Simulink for a conventional multicopter or new type UAVs such as vertical takeoff and landing (VTOL), which can run in Pixhawk. The entire development consists of three steps: SIL simulation, HIL simulation, and real flight test. With this step-by-step development, safety and efficiency can be ensured. About the real-time performance of

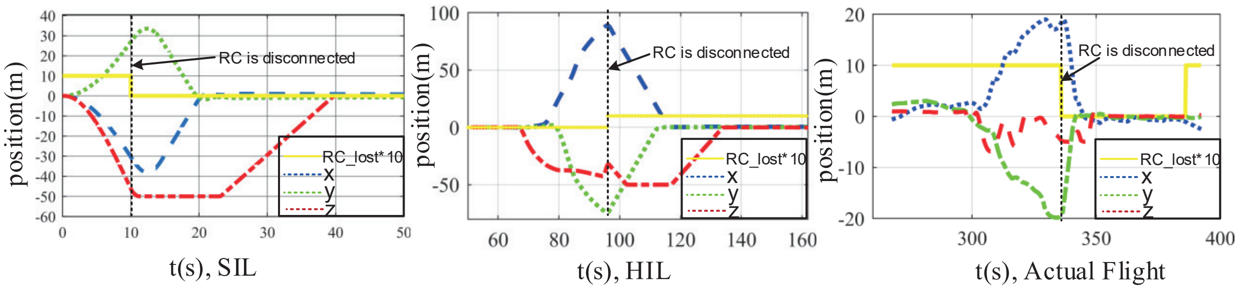

Fig. 14. SIL, HIL Simulation and outdoor flight tests of failsafe. The blue dotted line is the local position along x axis, the green dotted line is the local position along y axis, and the red dotted line is the local position along z axis.

the platform, the time-triggered method is adopted to ensure the designed model can execute at an accurate period. And debugging functions such as online parameter adjustment, data recording, and real-time data inspecting are provided to accelerate development and education for multicopter especially in real flight test. Finally, we provide three examples to demonstrate the effectiveness of the proposed platform.

References

[1] 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.
[2] dji.com, "Tello edu," https://store.dji.com/cn/product/tello?vid=38421, accessed October 1, 2020.
[3] L. Eller, T. Guerin, B. Huang, G. Warren, S. Yang, J. Roy, and S. Tellex, "Advanced autonomy on a low-cost educational drone platform," *arXiv preprint arXiv:1910.03516*, 2019.
[4] J. P. How, B. Behihke, A. Frank, D. Dale, and J. Vian, "Real-time indoor autonomous vehicle test environment," *IEEE Control Systems Magazine*, vol. 28, no. 2, pp. 51–64, 2008.
[5] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The grasp multiple micro-uav testbed," *IEEE Robotics &Automation Magazine*, vol. 17, no. 3, pp. 56–65, 2010.
[6] S. Lupashin, M. Hehn, M. W. Mueller, and A. P. Schoellig, "A platform for aerial robotics research and demonstration: The flying machine arena," *Mechatronics*, vol. 24, no. 1, pp. 41–54, 2014.
[7] L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, and M. Pollefeys, "Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision," *Autonomous Robots*, vol. 33, no. 1-2, pp. 21–39, 2012.
[8] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015.
[9] ardupilot.org, "Ardupilot," https://ardupilot.org/dev/index.html, accessed October 1, 2020.
[10] M. Hamandi, M. Tognon, and A. Franchi, "Direct acceleration feedback control of quadrotor aerial vehicles," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 5335–5341.
[11] Y. Yu, S. Yang, M. Wang, C. Li, and Z. Li, "High performance full attitude control of a quadrotor on so (3)," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015.
[12] C. W. Kang and C. G. Park, "Attitude estimation with accelerometers and gyros using fuzzy tuned kalman filter," in *2009 European Control Conference (ECC)*, 2009, pp. 3713–3718.
[13] M. H. Dhullipalla, R. Hamrah, and A. K. Sanyal, "Trajectory generation on se(3) with applications to a class of underactuated vehicles," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, 2017, pp. 2557–2562.
[14] M. Faessler, D. Falanga, and D. Scaramuzza, "Thrust mixing, saturation, and body-rate control for accurate aggressive quadrotor flight," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 476–482, 2016.
[15] B. Morrell, M. Rigter, G. Merewether, R. Reid, R. Thakker, T. Tzanetos, V. Rajur, and G. Chamitoff, "Differential flatness transformations for aggressive quadrotor flight," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018.
[16] Mathworks.com, "Simulink," https://www.mathworks.com/products/simulink.html, accessed October 1, 2020.
[17] X. Dai, C. Ke, Q. Quan, and K.-Y. Cai, "Simulation credibility assessment methodology with fpga-based hardware-in-the-loop platform," 2019.
[18] Q. Quan, X. Dai, and S. Wang, *Multicopter Design and Control Practice*. Springer, Singapore, 2020.
[19] Qgroundcontrol.com, "Qgroundcontrol," http://qgroundcontrol.com/, accessed October 1, 2020.
[20] px4.io, "Px4," http://dev.px4.io/v1.9.0/en/, accessed October 1, 2020.
[21] mathworks.com, "Pixhawk support package," https://www.mathworks.com/hardware-support/px4-autopilots.html, accessed October 1, 2020.
[22] microsoft.com, "Airsim," https://microsoft.github.io/AirSim/, accessed October 1, 2020.
[23] M. J. Pont, *Patterns for time-triggered embedded systems*. TTE System, Ltd, 2008.
[24] M. Brunner, K. Bodie, M. Kamel, M. Pantic, W. Zhang, J. Nieto, and R. Siegwart, "Trajectory tracking nonlinear model predictive control for an overactuated mav," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 5342–5348.
[25] Q. Quan, *Introduction to Multicopter Design and Control*. Springer, Singapore, 2017.
[26] J. Han, "From pid to active disturbance rejection control," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 3, pp. 900–906, 2009.
[27] G.-X. Du, Q. Quan, B. Yang, and K.-Y. Cai, "Controllability analysis for multirotor helicopter rotor degradation and failure," *Journal of Guidance, Control, and Dynamics*, vol. 38, no. 5, p. 978–985, May 2015. [Online]. Available: http://dx.doi.org/10.2514/1.G000731
[28] V. Lippiello, F. Ruggiero, and D. Serra, "Emergency landing for a quadrotor in case of a propeller failure: A pid based approach," in *2014 IEEE International Symposium on Safety, Security, and Rescue Robotics (2014)*, 2014, pp. 1–7.