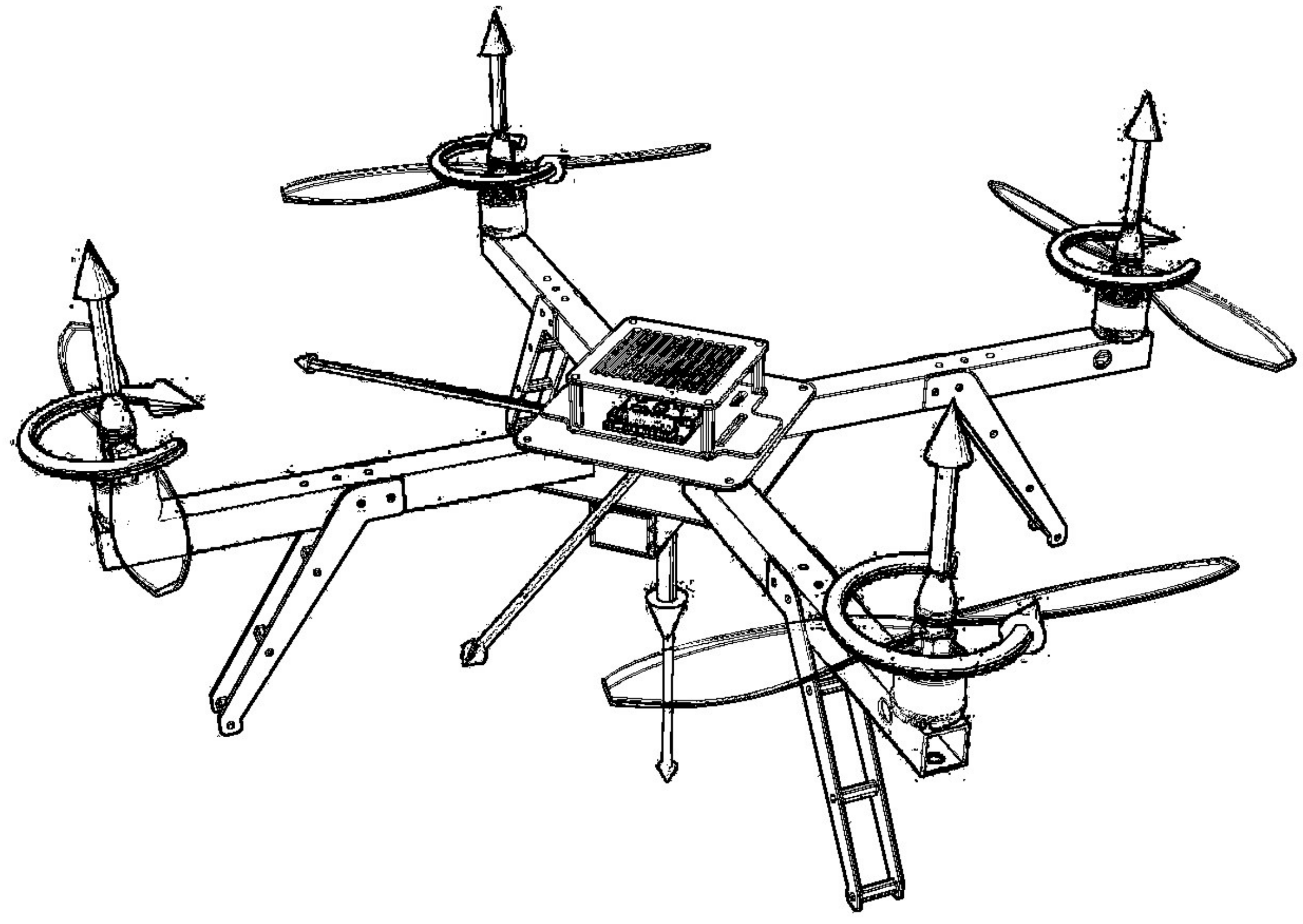
APM Multicopter Development Kit for Simulink



POLAKIUM ENGINEERING

APM Multicopter Development Kit for Simulink

Adam Polak[1]
Arizona State University, Tempe, Arizona, 85281

Many control systems are capable of being modeled on a computer; however, there exists a degree of uncertainty between the performance of these simulations and their intended applications. Small multirotor aircraft are becoming common and affordable test platforms for testing such systems. Using inexpensive and readily available components, a multirotor helicopter, or multicopter, can be constructed and used for safe testing of these algorithms using inexpensive flight hardware. A user friendly Simulink framework was developed for engineers to seamlessly integrate these control algorithms with the Ardupilot Mega flight controller on a multicopter.

Nomenclature

AHRS	=	attitude and heading reference system
APM	=	Ardupilot Mega
DoF	=	degree of freedom
ESC	=	electronic speed controller
IMU	=	inertial measurement unit
LED	=	light emitting diode
MEMS	=	micro-electro-mechanical systems
MIPS	=	millions of instructions per second
NED	=	North-East-Down coordinate system
PWM	=	pulse width modulation
ROTH	=	run on target hardware
w	=	angular velocity

I. Introduction

Control systems design and analysis can often be performed using computer simulations, however, many experimental control algorithms are cost prohibitive to test on full-scale flight hardware. To address this problem a multicopter was chosen as the most practical vehicle for experimental tests. Such a platform is suitable for a wide range of operational environments and easily reconfigurable for alternative rotor quantities and sizes. These features allow multicopters to safely and accurately simulate many test vehicles and carry diverse payloads.

A hobby-grade quadcopter was chosen as the primary test vehicle for the trials. This vehicle is capable of flying in both indoor and outdoor environments for ten to fifteen minutes. The onboard 16 MIPS Atmel-based flight controller contains a six DoF IMU, three DoF magnetometer, barometric pressure altimeter and GPS receiver.

In order to allow guidance, navigation and control systems engineers to easily test Simulink control algorithms onboard the multicopter, a Simulink block set was used to interface the flight controller with Simulink's Run on Target Hardware feature. Together with a common control structure and attitude estimator, engineers can drop their control systems into the Simulink multicopter framework and effectively flight test their control algorithms.

[1] Undergraduate Control Systems Intern, EV41, NASA Marshall Space Flight Center, apolak@asu.edu

II. Dynamic Model

A. Simulink Model

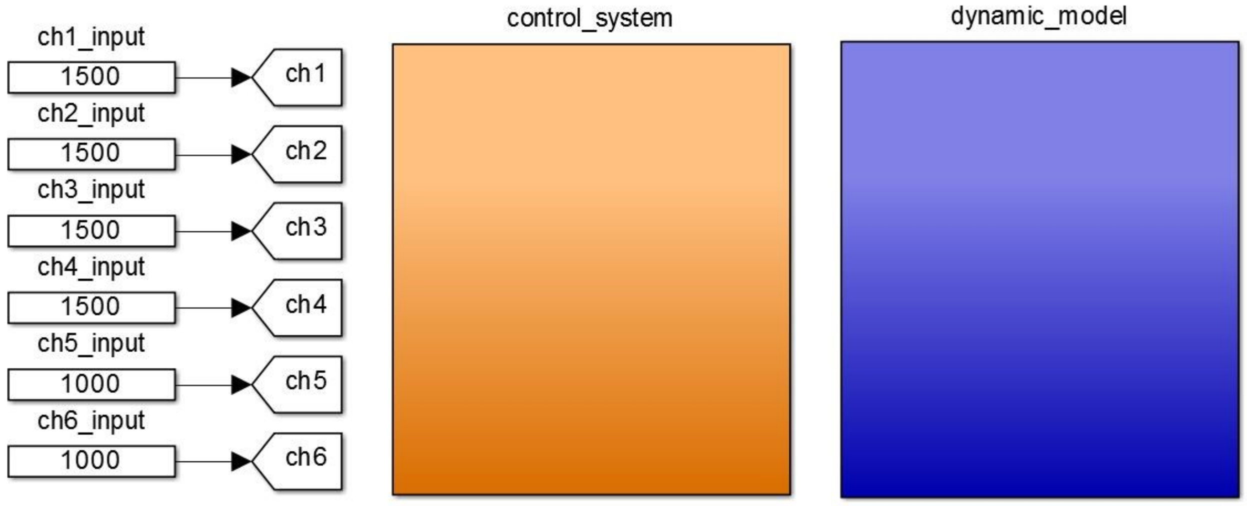


Figure 1. multi_model.slx Simulink model

The primary simulation component of the APM Multicopter Development Kit for Simulink is the Simulink model multi_model.slx (Figure 1). This model contains the dynamics simulation and the control system for the aircraft. From the model, six constants may be set for each of the user defined radio commands. These constants may be altered in real time from within the simulation or from the graphical user interface. The simulink model may be run independently of the other software components, such as the user interface. Data from any signal of the Simulink model can easily be monitored with a scope and saved to the MATLAB workspace for further analysis. The dynamic model simulates the rigid body dynamics of the aircraft as well as the rate transitions and noise characteristics of the sensors. The resulting simulation data allows accurate sensor feedback to be sent through the control system with respect to the aircraft's current state. The dynamic model is useful for verifying the stability of the control system prior to hardware implementation; however, generalizations and simplifications made to the dynamics diminish the accuracy of the simulation, and greater model fidelity is necessary to properly tune some filters and feedback control systems.

ii. Control System

The control system block is setup for a simple drag and drop procedure with different control systems. Within the block, each input and output signal uses a set of global tags to reference other parts of the Simulink model. By using similar tag names, any block can be dropped into the Simulink model without drawing signal paths between these blocks. Global tags exist for all of the IMU signals - ax, ay, az, p, q, r - as well as for the pressure and temperature signals. Adding a from tag to the control system will acquire the specified sensor signal from the dynamic model. All eight pwm outputs of the control system are connected to goto tags, m1, m2, m3, m4, m5, m6, m7 and m8. Provided that these tags exist within the control system block, any control system block can simply be dropped directly into the Simulink model for testing in simulation. It is possible to drop multiple control systems into the Simulink model; however, redundant outputs should be removed or commented out such that only one signal is sent to the eight motor outputs, m1-m8. The advantage of dropping multiple control systems into the simulation is that it allows the outputs from each system to be compared in order to measure the performance and accuracy of one system over another.

iii. Dynamic Model

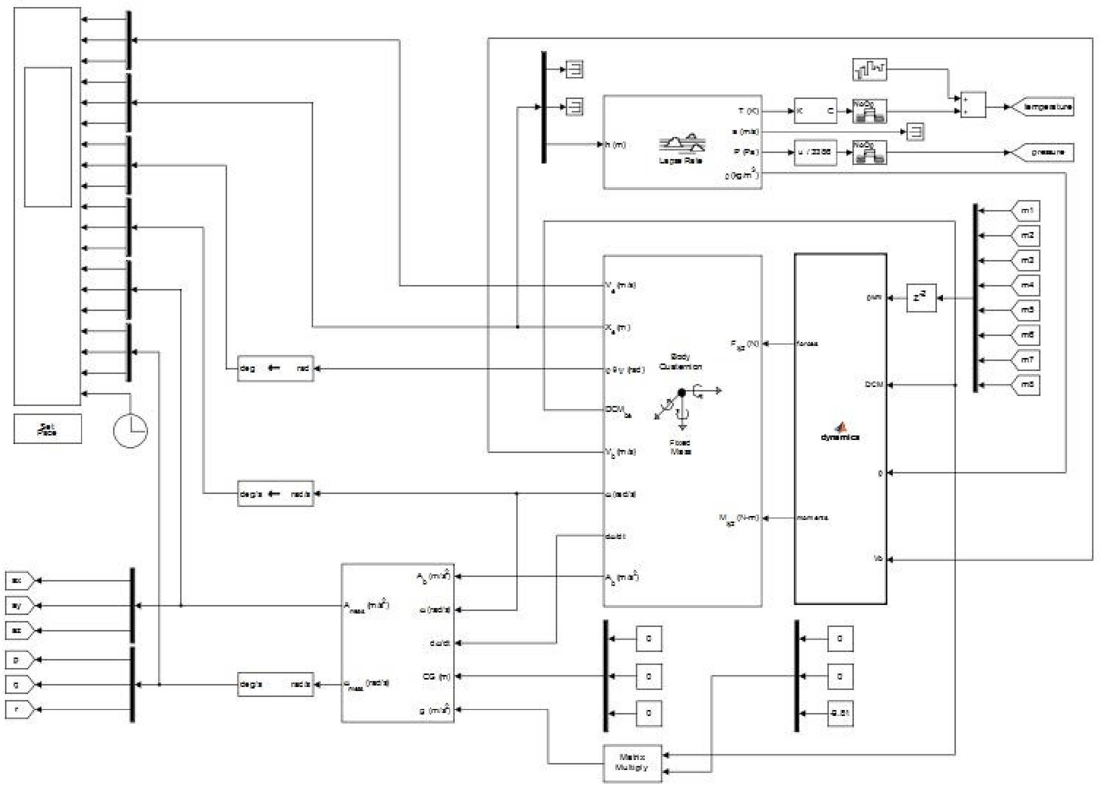
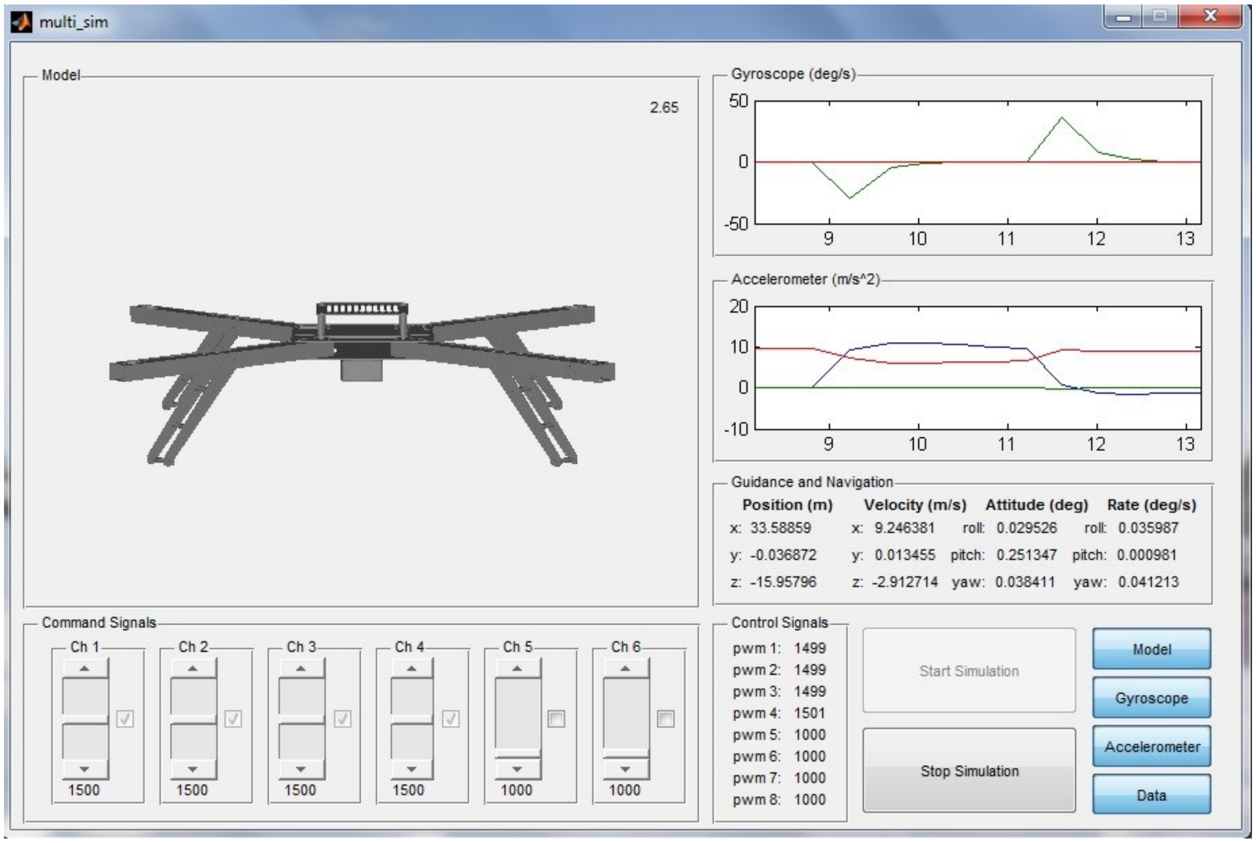


Figure 2. dynamic_model block

The dynamic model block depicted in Figure 2 contains all of the necessary blocks to compute the rigid body dynamics of the aircraft using the acquired pwm signals m1–m8. The state of the rigid body is then used to develop an accurate representation of the sensor signals and noise. All of the sensor and navigation data is scoped to a simulation_data block so that this data may be acquired in real time by a MATLAB script or graphical user interface. The signals of this scope may also be set to save to the MATLAB workspace for further analysis.

B. Simulation

To begin the simulation, run the MATLAB script multi_sim.m. The Simulink model, multi_model.slx, will open in the background, and all controls signals are set to their default values. The initial control signals for the system can be set using the six sliders of the GUI located in the command signals window (Figure 3). Checking the box beside each slider will automatically center the command value at its midpoint. Four boxes along the bottom right side of the GUI allow the user to enable or disable features of the simulation to improve performance. The current refresh rate for the data will be displayed in the top right corner of the model window during simulation. Selecting the start button runs the Simulink model, multi_model.slx, in the background and sets conditions for the six control signals from the current slider values. True rotation data is used to determine the aircraft's attitude, and the CAD model is rotated in the model window. Raw gyroscope and accelerometer measurements are scoped in their respective windows, and true navigation values are populated to the fields of the guidance and navigation window. All values of the GUI are scoped by the simulation_data block within multi_model.slx, and these values can be saved to the MATLAB workspace. Each of the six control signals can be altered by the sliders at any time during the simulation in order to monitor the reaction of the system. Selecting the stop button will stop the simulation and freeze all simulation data in the GUI.



Guidance and Navigation

Position (m)	Velocity (m/s)	Attitude (deg)	Rate (deg/s)
x: 33.58859	x: 9.246381	roll: 0.029526	roll: 0.035987
y: -0.036872	y: 0.013455	pitch: 0.251347	pitch: 0.000981
z: -15.95796	z: -2.912714	yaw: 0.038411	yaw: 0.041213

Command Signals

Ch 1	Ch 2	Ch 3	Ch 4	Ch 5	Ch 6
1500	1500	1500	1500	1000	1000

Control Signals

pwm 1:	1499
pwm 2:	1499
pwm 3:	1499
pwm 4:	1501
pwm 5:	1000
pwm 6:	1000
pwm 7:	1000
pwm 8:	1000

Figure 3. multi_sim Graphical User Interface

i. Dynamics Configuration

Prior to simulation, the dynamics of the aircraft must be configured in the Simulink model. The aircraft properties may be altered from the MATLAB function dynamics within the dynamic_model block (Figure 4). Generic pwm latency is introduced to roughly account for the transfer function from pwm signal to force. Alternatively, the pwm latency and dynamics function may be removed and replaced by improved transfer functions for the forces and moments. This would greatly improve the accuracy of the model.

The 6DoF (Quaternion) equations of motion block must also be modified to suit the dynamics of the aircraft being simulated. Set the initial mass to match the mass specified with the dynamics function and use the inertia tensor of the body in place of the inertia field. The default values of the system are set to work best with a standard Arducopter with a mass of approximately 1kg.

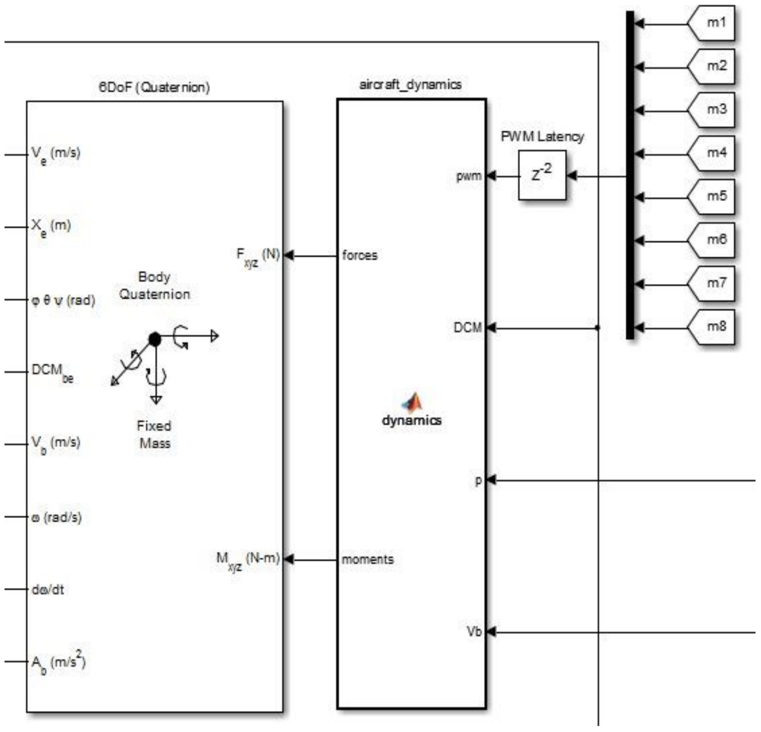


Figure 4. dynamics function

ii. CAD Model

Included within the APM Multicopter Development Kit for Simulink is a stereo lithography file, or stl file, of the Arducopter. This model will be displayed in the model window of the GUI. It may be replaced with any other binary stl file with units set to meters; however, the more complex the model, the less performance should be expected from the simulation. The provided stl model will render at approximately 5 to10 frames per second on most computers. Rendering of the model may be disabled from the GUI in order to improve the performance of the simulation.

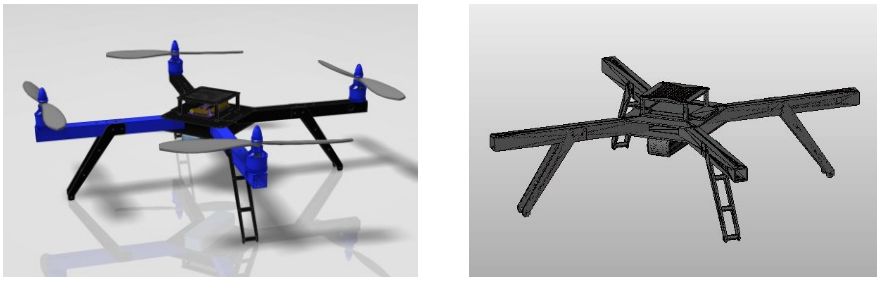


Figure 5. Arducopter CAD assembly (left) and simplified stl model (right)

III. APM ROTH Simulink Framework

A. Initial Setup

To properly access all of the sensors on the flight controller and compile the firmware, the control system framework makes use of a Simulink block set for the Ardupilot Mega 2.0. This block set contains the open-source Arducopter libraries necessary to integrate the hardware with Simulink and to compile and write the firmware directly to the flight controller. It is necessary that the APM2 Simulink Blockset[1] be downloaded from MATLAB Central prior to installation. Documentation is contained within the archive to explain in further detail the proper setup process to use the APM2 Simulink Blockset.

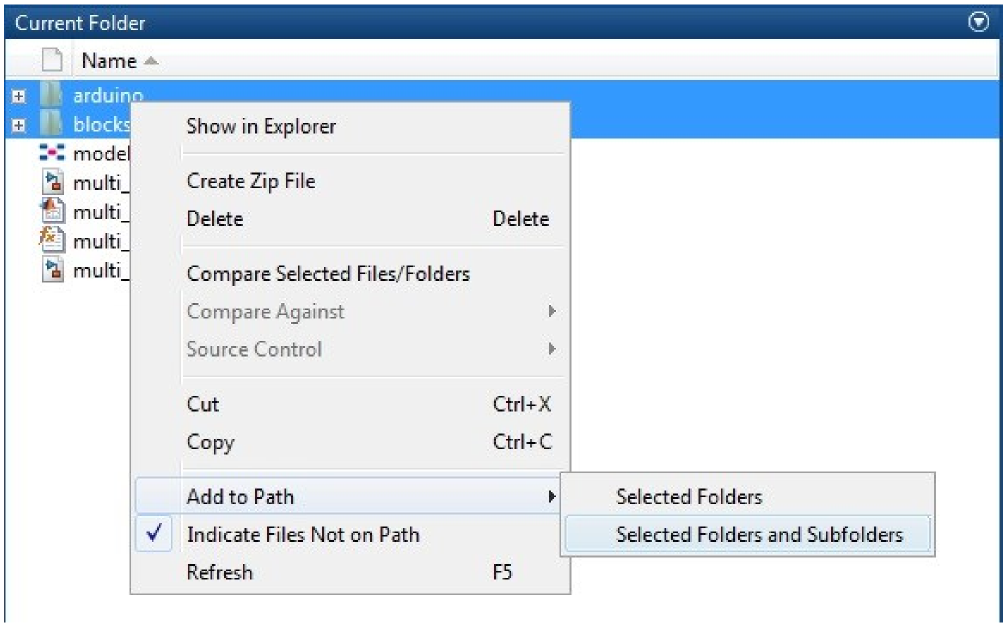


Figure 6. MATLAB project directory

Once installation of the APM2 Simulink Blockset is complete, the active project directory within MATLAB should contain an Arduino and a blocks folder. It is essential that both folders and all subfolders be added to the project directory (Figure 6). With these folders included in the project directory, the Simulink framework will now recognize the APM2 blocks within the model. Alternatively, additional blocks are included with the APM2 Simulink Blockset which allow for GPS integration and other sensors. These blocks can be added through the Simulink library browser.

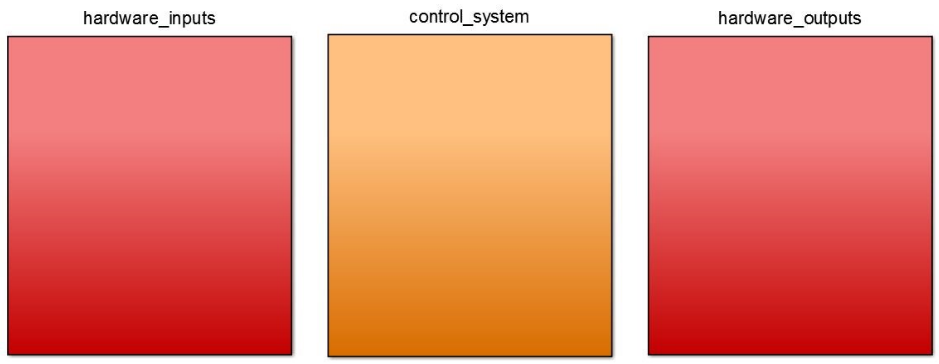


Figure 7. multi_sim.slx ROTH Simulink framework

Opening multi_system.slx will display the framework architecture (Figure 7). The hardware_inputs and hardware_outputs blocks are most essential for compiling the control system for ROTH. Guidance, navigation and control algorithms may be placed into independent blocks provided that the hardware_inputs and hardware_outputs blocks are within the same model and all of the proper global signals are referenced between the blocks.

B. Input Signals

All commands, inertial measurements and inertial navigation estimates are determined within the hardware_inputs block (Figure 8). Within this block is the IMU block, Barometric Sensor block, RC Read block, Sensor Fusion block, Command block and various LED indicator blocks.

The IMU block can be selected to set the anti-aliasing low pass filter frequency, gyro scale, accelerometer scale and sample time. The anti-aliasing low pass filter frequency must be less than half of the system's operating frequency. For standard operation, a filter frequency of 42Hz, gyro scale of 250 degrees per second and accelerometer scale of 2g are adequate.

The operator's radio commands are output from the RC Read block to the control system. Each channel outputs a number corresponding to the pulse width of the radio's transmitted PWM signal. This signal is centered about a pulse width of 1500us with a range of 1000us to 2000us.

Overrun detection uses the red LED of the flight controller to indicate the software is incapable of maintaining the specified cycle time. Should the red LED show during operation, reducing the sample time will help to prevent overrunning. It is recommended to keep the sample time above a minimum of 0.02 seconds or 50Hz.

All signals and command data are output from the Flight Controller Data Block as a set of global goto tags. These tags are listed along the right side of the block. The signals of these tags can be referenced in any part of the model, such as the control system. Global tags exist for all of the IMU signals - ax, ay, az, p, q, r - as well as for the pressure and temperature signals.

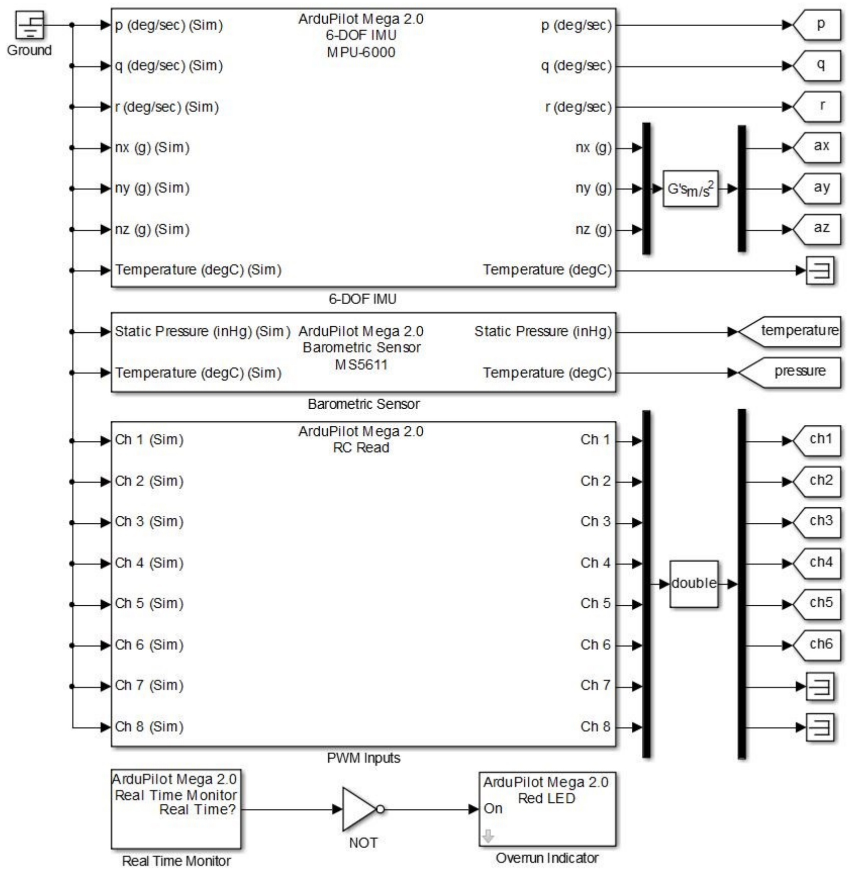


Figure 8. hardware_input block

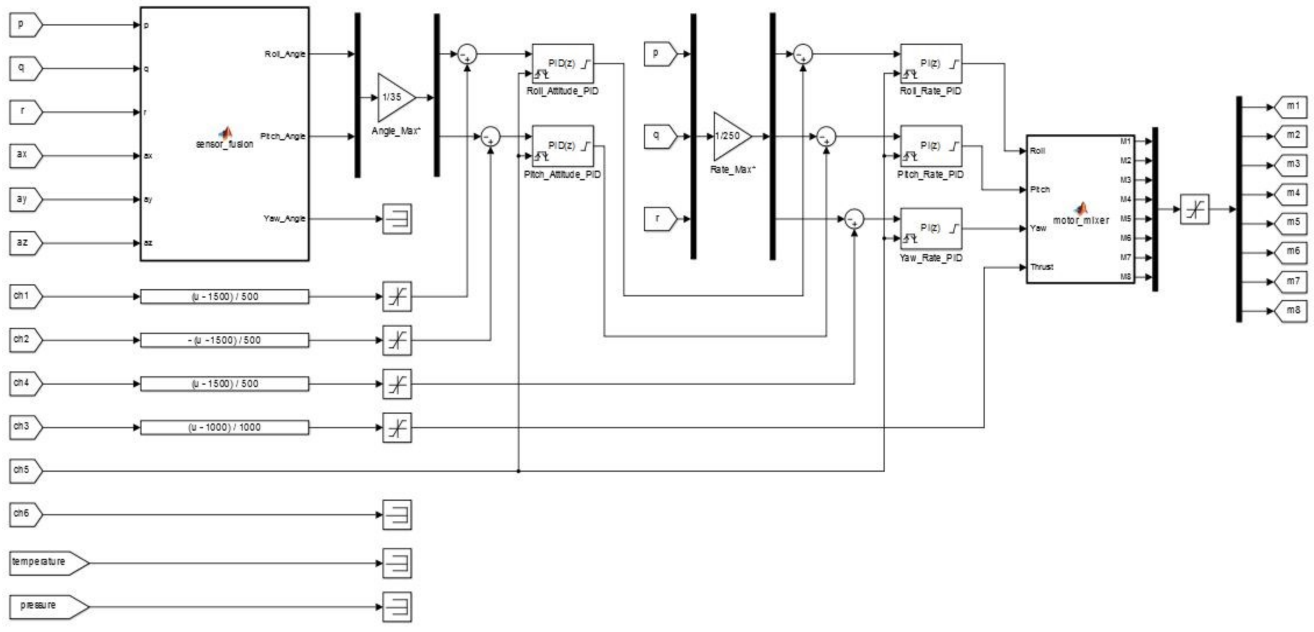


Figure 9. control_system block

C. Control System

Both the control system from the dynamic model and ROTH framework may be interchanged for testing. The control system can be dragged and dropped from the dynamic model into the ROTH framework when simulation is complete. The following section details the functionality of the attitude control system included with the APM Multicopter Development Kit for Simulink (Figure 9).

i. Attitude Estimation

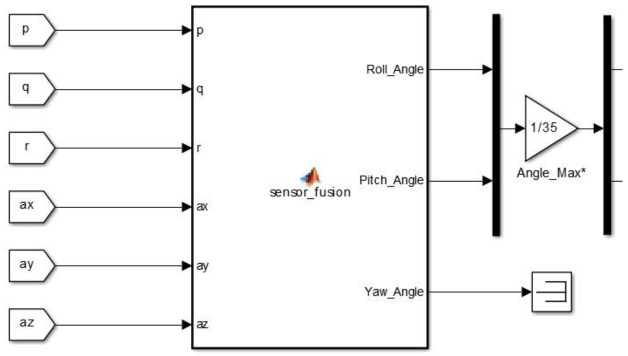


Figure 10. sensor_fusion block

To obtain the best estimate of the aircraft's current state, a series of Kalman filters are used within the sensor_fusion block of the control system (Figure 10). Due to limitations of the flight controller's processing power, the Kalman filter has been simplified into several two state Kalman filters.

Attitude estimation for pitch and roll is performed by estimating the Euler angles of the aircraft and the gyro bias due to noise and gyro drift. Body rates are first converted to Euler rates. Assuming the IMU is properly mounted, the acceleration due to gravity can be used to determine the aircraft's approximate attitude; however, the signals obtained from the accelerometer are much more unreliable than the gyroscope data. Accelerometer computed attitude is compared with the integrated Euler rates in order to determine the gyro bias. Following the first iteration the Euler rates will continue to update and gyro bias will be computed and removed from the original signal using the statistical estimates of the Kalman Filter.

Parameters for each of the filters can be modified at the top of the code in the sensor_fusion block (Figure 11). These parameters must be properly tuned to match the noise of the sensors. Q_gyroBias and Q_angle are functions of the noise characteristics of the gyroscope and accelerometer respectively. R_angle essentially determines which source of sensor data should be trusted as a true measurement. Increasing the value of R_angle will improve the performance of the filter, but the accelerometer leveling data will have less of an effect on the attitude estimation. Decreasing the value of R_angle may reduce the performance of the filter, but it will prevent gyro drift from building over time. Accelerometer data is taken directly from the IMU signals, so any acceleration of the aircraft, including centrifugal acceleration, will remain part of the data interpreted by the filter.

```
%============================Constants=============================

delta_t = 0.01;
degrees = 180 / pi;
radians = pi / 180;

%Attitude Estimator
Q_gyroBias = 0.000000087;
Q_angle = 0.000039;
R_angle = 20;
```

Figure 11. sensor_fusion parameters

ii. Attitude Control (Outer Loop)

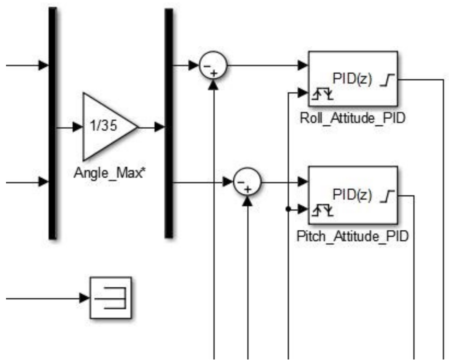


Figure 12. Attitude controller

The outer loop of the control system is known as the attitude controller. The attitude controller uses PID controllers to maintain a specified roll and pitch angle (Figure 12). Observing the right hand rule in the NED coordinate frame, positive pitch raises the nose of the aircraft and positive roll raises the left side of the aircraft. Attitude commands are interpreted from the input pwm radio signal and scaled to range in magnitude from -1 to 1. These signals are compared with the Euler angle data from the sensor_fusion block. In order to maintain similar unit conventions, the gain "Angle_Max*" is set to the inverse of the maximum desired angular deflection. The integral term of the PID controller is configured to be externally reset by the rising or falling edge of channel 5. Channel 5 may be configured to operate as a mode switch to change between control systems during flight, such as the attitude and rate control system. Saturation limits are also applied to the PID controller such that the output of the attitude controller is bounded from -1 to 1. The attitude controller acts as an outer loop to the rate controller by passing the output signals to the rate controller as command data.

iii. Rate Control (Inner Loop)

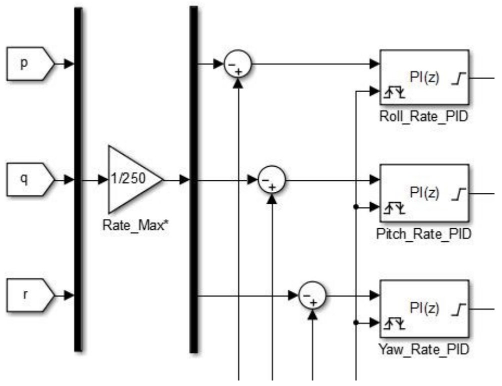


Figure 13. Rate controller

The Rate Controller block is responsible for maintaining a set angular velocity. It is the inner loop of the control system through which all commands must be processed. The rate controller uses a PI controller to achieve the desired angular velocity (Figure 13). Rate commands ranging from -1 to 1 are compared with the angular velocity data from the IMU. In order to maintain similar unit conventions, the gain Rate_Max* is set to the inverse of the maximum desired angular velocity. The integral term of the PI controller can be configured to be externally reset by the rising or falling edge of a signal. Saturation limits are also applied to the PI controller such that the output of the rate controller is bounded from -1 to 1. The outputs from the rate controller are subsequently passed to the motor_mixer block so that the appropriate pwm values are sent to the electronic speed controllers.

iv. Motor Mixer

Interpretation of the control system commands is handled by the motor_mixer block within the control_system block. This block mixes the signals of roll, pitch, yaw and thrust such that the appropriate motors deliver an increase or decrease in thrust. By default, the motor mixer is setup to handle a four motor configuration with the forward heading between the front two motors. This is termed a "X" configuration. The motor mixer can be customized to accommodate up to eight motors in various configurations (Figure 14). The output of the motor mixer is the pulse width of the pwm signal for each output channel; therefore, it is important that saturation limits be set to keep this pulse width between values of 1000us and 2000us. The pwm signal is sent to the electronic speed controllers through the RC Write block at an update frequency of 50Hz, regardless of the system's operating frequency (Figure 16). Ideally, the pwm signal would be refreshed at up to 490Hz, but this requires modification of the underlying libraries of the APM2 Simulink Blockset and may not result in noticeably increased performance; therefore, it will remain a limitation of the current system.

The motor mixer can be configured for use with a combination of eight pwm controlled actuators. For example, these pwm signals can be used to drive both electronic speed controllers and servos for many aircraft, ground vehicles or robotic mechanisms. This scenario assumes a four rotor configuration where the forward heading is aligned with the first motor. This is termed a "+" configuration. Since only four rotors will be used, outputs M5-M8 are disabled by sending a PWM signal of 1000us. It is important to note that the signals being sent to the motor mixer are values ranging from -1 to 1, with the exception of the thrust signal which ranges from 0 to 1. After combining these signals the resulting value must be multiplied by 1000 and added to the idle_PWM value. The value of idle_PWM should be set to 1000us unless it is desirable that the motors idle at a low RPM. This is achieved by increasing the idle_PWM in small increments until the motors begin to rotate, or reducing the idle_PWM value in small increments until the motors no longer rotate. The most important part of the motor mixer is the equation used to combine roll, pitch, yaw and thrust for each actuator. Assuming that motor one is forward of the aircraft and rotating clockwise, applying an increase in thrust to motor one will result in zero roll, positive pitch and positive yaw; therefore, the equation for motor one has a roll coefficient of zero, pitch coefficient of positive one and yaw coefficient of positive one. This combined weighting of values must then be scaled by thrust, divided by two and added back to the thrust value (Equation 1). The MATLAB code for the Quad +, Quad X and Hex X mixer configurations are included in the motor_mixer block (Figure 15). The mix selection within the motor_mixer block can be switched during flight to accommodate motor failures and experimental airframe configurations.

$$M1 = \left(\frac{(Pitch+Yaw)*Thrust}{2} + Thrust\right) * 1000 + idle\_PWM \tag{1}$$

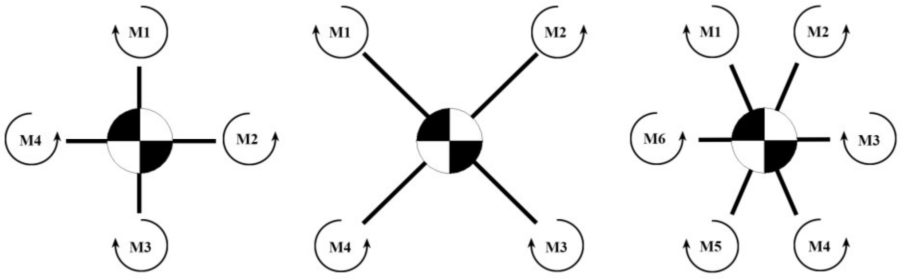


Figure 14. Motor layout

```
function [M1, M2, M3, M4, M5, M6, M7, M8] = motor_mixer(Roll, Pitch, Yaw,
Thrust)

%===========================Settings & Constants===========================

idle_PWM = 1000;
aircraft_mixer = 0;
                    % Quad [X] = 0
                    % Quad [+] = 1
                    % Hex  [X] = 2

%===============================Quad X Mixer===============================

if aircraft_mixer == 0;         % Quad X
    M1 = ((Roll + Pitch - Yaw) * Thrust / 2 + Thrust) * 1000 + idle_PWM;
    M2 = ((-Roll + Pitch + Yaw) * Thrust / 2 + Thrust) * 1000 + idle_PWM;
    M3 = ((-Roll - Pitch - Yaw) * Thrust / 2 + Thrust) * 1000 + idle_PWM;
    M4 = ((Roll - Pitch + Yaw) * Thrust / 2 + Thrust) * 1000 + idle_PWM;
    M5 = 1000;
    M6 = 1000;
    M7 = 1000;
    M8 = 1000;
elseif aircraft_mixer == 1;     % Quad +
    M1 = ((Pitch - Yaw) * Thrust / 2 + Thrust) * 1000 + idle_PWM;
    M2 = ((-Roll + Yaw) * Thrust / 2 + Thrust) * 1000 + idle_PWM;
    M3 = ((-Pitch - Yaw) * Thrust / 2 + Thrust) * 1000 + idle_PWM;
    M4 = ((Roll + Yaw) * Thrust / 2 + Thrust) * 1000 + idle_PWM;
    M5 = 1000;
    M6 = 1000;
    M7 = 1000;
    M8 = 1000;
elseif aircraft_mixer == 2;     % Hex X
    M1 = ((Pitch + Roll / 2 - Yaw) * Thrust / 2 + Thrust) * 1000 + idle_PWM;
    M2 = ((Pitch - Roll / 2 + Yaw) * Thrust / 2 + Thrust) * 1000 + idle_PWM;
    M3 = ((-Roll - Yaw) * Thrust / 2 + Thrust) * 1000 + idle_PWM;
    M4 = ((-Pitch - Roll / 2 + Yaw) * Thrust / 2 + Thrust) * 1000 + idle_PWM;
    M5 = ((-Pitch + Roll / 2 - Yaw) * Thrust / 2 + Thrust) * 1000 + idle_PWM;
    M6 = ((Roll + Yaw) * Thrust / 2 + Thrust) * 1000 + idle_PWM;
    M7 = 1000;
    M8 = 1000;
else
    M1 = 1000;
    M2 = 1000;
    M3 = 1000;
    M4 = 1000;
    M5 = 1000;
    M6 = 1000;
    M7 = 1000;
    M8 = 1000;
end
%==========================================================================
```

Figure 15. motor_mixer block

D. Output Signals and Serial Communication

All output signals from the control system are processed through the hardware_output block. Command signals for m1-m8 are output to the ESCs as pwm signals responsible for setting the rotational velocity of each rotor. The output signals from the control system undergo a unit conversion prior to being sent to the RC Write block.

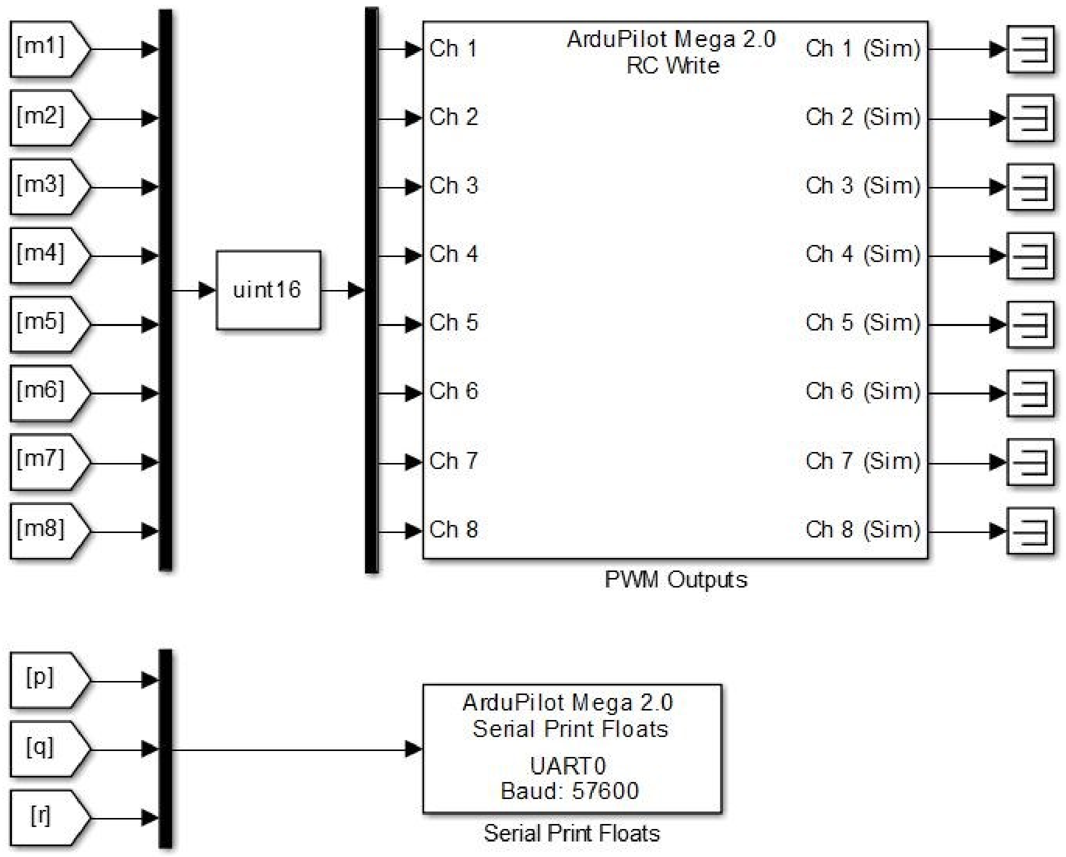


Figure 16. hardware_output block

Debugging and data logging of the control system often requires a real-time communication link between the flight controller and computer. This is accomplished using serial communication with the Arduino command console within the Arduino IDE[2]. To specify which signals will be printed to the command console, the Serial Print Floats block must be place within the model. All signals must be multiplexed into the Serial Print Floats block (Figure 16). The Arduino IDE must be opened and set to connect with the COM port associated with the flight controller. The baud rate of the Serial Print Floats block must match the baud rate of the Arduino command console. The multiplexed signals will be sequentially output to the command console at a frequency matching the sample time of the system. Large arrays of data transmitted at a high frequency will likely result in overrunning of the flight controller, so use of the Serial Print Floats block should be avoided during flight tests.

E. Run on Target Hardware

The final step of the process is to compile and run the Simulink code on the flight controller. Prior to running the software, it is essential that the sample times of all blocks are consistent. There are five locations that require a sample time to be defined. Navigating to Tools > Run on Target Hardware > Options… > Solver allows the sample time of the model to be defined. This sample time must match the sample time of the IMU, Real Time Monitor and sensor_fusion blocks within the hardware_inputs and control_system blocks. The sample time of the sensor_fusion block is defined as delta_t, as shown in Figure 11. Selecting the run button from within the Simulink model will indicate if the code has been successfully compiled. If at this point Simulink produces an error, then there is likely a bug in the control system which must be corrected prior to uploading the control system to the flight controller. If no build errors arise, then the control system is ready to be uploaded to the flight controller via Tools > Run on Target Hardware > Run. If at this point Simulink produces an error then the flight controller may not be properly connected or the incorrect COM port has been selected from Tools > Run on Target Hardware > Options… > Run on Target Hardware. Observe safety when testing flight control algorithms on a powered multicopter with props installed. All systems should be tested with the dynamic model prior to hardware implementation in order to locate potentially hazardous bugs in the software.

IV. Conclusion

After completion of the dynamic model and control systems, several flight trials were conducted, proving that the ROTH framework of this APM Multicopter Development Kit is a viable option for implementation of experimental control algorithms. It had originally been assumed that the processing power of the Ardupilot Mega may not be enough to support a complete attitude and heading reference system, or AHRS; however, testing indicates that more complex algorithms are capable of running effectively. It should be noted that all hardware has its limitations, and although capable of supporting an AHRS, the flight controller may not be capable of supporting advanced filters, guidance and navigation algorithms. This is in part due to processing limitations, but also due to inaccuracies in the measurements obtained by the MEMS IMU. Kalman filtering proved to be an effective method of estimating the aircraft's state, but it was necessary that these filters be simplified and matrix math be avoided in order to maintain adequate performance. The greatest advantage of the completed framework lies in the user's ability to drag and drop control systems between the dynamic model and ROTH framework. With the ROTH framework, control system may be directly written to the flight controller from within the Simulink environment. Users may now implement a wide variety of systems on an inexpensive, low-cost, airborne platform with the convenience of Simulink's high level programming environment to develop their algorithms.

Acknowledgments

I would like to thank the following individuals:

Don Krupp and Mike Hannan of EV41 at Marshall Space Flight Center for providing project resources and support as well as insight into future applications and development.

Kevin Blankenship for providing an expert understanding of MEMS guidance and navigation algorithms.

Robert Hartley and his team for publication of the APM2 Simulink Blockset, an invaluable tool to my research.

References

[1] Robert Hartley, "APM2 Simulink Blockset," MATLAB Central, 13 November, 2012. URL: http://www.mathworks.com/MATLABcentral/fileexchange/39037-apm2-simulink-blockset

[2]Arduino, "Download the Arduino Software," Arduino, 6 February, 2013 URL: http://arduino.cc/en/Main/Software

[3]Thomas Kølbæk Jespersen, "A practical approach to Kalman filter and how to implement it," TKJ Electronics, 10 September, 2012 URL: http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/

[4]DaeEun Kim, "IMU with Kalman filter," Yonsei University, URL: http://cog.yonsei.ac.kr/quad/quad.htm