

SINGLE MODELING ENVIRONMENT FOR CONSTRUCTING HIGH-FIDELITY PLANT AND CONTROLLER MODELS

Jim Ledin, P.E., Ledin Engineering, Camarillo, California
Mike Dickens, The MathWorks, Inc., Natick, Massachusetts
Jay Sharp, The MathWorks, Inc., Natick, Massachusetts

Abstract—Modern guidance, navigation and control (GN&C) system development relies heavily on modeling, simulation, and real-time testing for both the control system and the plant. Model-based design has proven to be an effective framework for these development activities, especially for designing the control system model and reusing that model in multiple activities. However, plant modeling has historically not had as much success, despite the need for accurate plant models in order to develop high quality controllers. The difficulties in modeling the plant are many and far-reaching, historically leading to the development of specialized six degrees-of-freedom (6DOF) C and FORTRAN simulations that model specific aspects of the physical system. Although these simulations work well for the particular application for which they were developed, it is often difficult to apply the models to new plants, make design changes outside the scope of the original implementation, or reuse the models in new development tasks such as hardware-in-the-loop (HIL) simulation. Recent advances in off-the-shelf plant modeling software now provide a single simulation environment supporting the construction of high-fidelity plant and controller models. These models can be reused by converting them into compact, efficient C code for embedded controller implementation and hardware-in-the-loop testing applications. This paper focuses on recent advances in plant modeling with an emphasis on its use in real-time HIL testing for aerospace applications.

I. INTRODUCTION

OVER the past several decades, the development of GN&C systems has come to rely increasingly on the application of modeling and simulation techniques. Properly applied, modeling and simulation can drastically reduce the amount of hardware prototype development required and drive down the risk of

confronting significant problems during the system integration and system test phases.

System simulation¹ focuses at the level of the entire system as it operates in its intended environment. This type of simulation enables testing of the system in its intended operational modes, as well as under dangerous emergency conditions, without risking loss of life or valuable assets. Environmental conditions that may be difficult or impossible to access for system tests (such as icy roads in the middle of summer, or the conditions of outer space) can be simulated using computers and appropriate software algorithms. Simulation allows intricate test sequences to be performed quickly and repeatedly at relatively low cost.

A simulation containing a control system can be thought of as an executable specification of the control system algorithms. In the past, a standard practice has been to develop and validate a simulation of the control system, then use that simulation as an executable specification in a separate software development task to implement the embedded code. Today, code generation tools are available that produce fast, compact, production-quality C language source code from simulation models. The use of automatic code generation eliminates much of the work involved in the implementation and debugging of the embedded software.

Prototype hardware and software components can be tested at the subsystem level using hardware-in-the-loop (HIL) simulations long before a complete system prototype becomes available for testing. HIL simulations run in real time and perform input/output operations with the system or subsystem under test such that the test item “thinks” it is operating as part of a real system in its operational environment. These subsystems can be tested under nominal conditions as well as at (and beyond) their intended operational boundaries. HIL simulation provides the ability to thoroughly test subsystems using simulation early in the development process. This can greatly reduce the

debugging time and project risk compared to the alternative approach of waiting until a prototype is completed before performing integration and testing.

By combining the benefits of model-based GN&C system design using advanced simulation tools, embedded controller code generation, and HIL simulation, the system development cycle can be significantly accelerated while simultaneously reducing project risk through early testing of critical system elements.

This paper provides an example of these processes applied to the development of a 6DOF guidance and control system for a helicopter. A simulation of the helicopter and its guidance and control systems is first developed in the modern, block diagram-based simulation environment of Simulink[®], a product of The MathWorks, Inc.² Within the Simulink diagrams, supervisory logic is modeled using state-transition diagrams in Stateflow[®], a companion product to Simulink.

The embedded software for the guidance and control systems is then generated directly from the Simulink diagrams as C language source code using the MathWorks products Real-Time Workshop[®] and Stateflow Coder. The C source code is then compiled and deployed in the dedicated real-time execution environment of xPC Target. xPC Target and its companion xPC Target Embedded Option provide a real time execution environment for embedded systems using PC-compatible processors.

The software tools used in this paper are all members of the MATLAB[®] family of software tools from The MathWorks, Inc. These tools provide complete support for system analysis and simulation, model-based GN&C system design, embedded code generation, and deployment in embedded hardware. This paper demonstrates how the MATLAB family of products accelerate the system development process while simultaneously reducing project risk.

Though the helicopter model used in this paper is simplified in some ways, the general techniques described here are applicable across a wide variety of problem domains for plants that range in scale from simple to extremely complex.

II. PLANT MODEL

The helicopter model used in this example is simplified, yet contains enough realism to present a challenging design problem. Figure 1 is a diagram of the helicopter system.

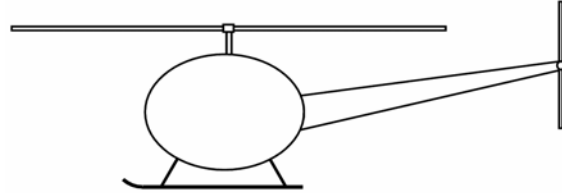


Figure 1 Helicopter configuration

The helicopter configuration is typical of a small manned unit. The main rotor is located above the cabin area and rotates about a vertical main rotor shaft. The tail rotor is mounted on a boom that extends to the rear and rotates in a vertical plane with its axis of rotation aligned in the left-right direction.

We will model the helicopter as three rigid components: a two-blade main rotor, a tail rotor, and the helicopter body. As a simplification, the rotors are assumed to rotate at constant angular velocities. The standard helicopter control actuators will be used, as described below.

- **Main rotor collective.** This actuator adjusts the angle of attack of both main rotor blades simultaneously. It controls the amount of lift generated by the main rotor.
- **Main rotor front-back cyclic.** This actuator adjusts the angle of attack of the blades in opposing directions, but does so primarily during the portion of the blade rotation when it is in the front-back direction. This control has the effect of producing a moment that pitches the helicopter's nose up or down.
- **Main rotor left-right cyclic.** This actuator adjusts the angle of attack of the blades in opposing directions, but does so primarily during the portion of the blade rotation when it is in the left-right orientation. This control has the effect of producing a moment that rolls the helicopter to the left or to the right.
- **Tail rotor collective.** This actuator adjusts the angle of attack of the tail rotor blades simultaneously. It controls the side force generated

by the tail rotor, which allows control of the yaw orientation of the helicopter.

To keep things simple, we will not model the complexity and subtleties of the helicopter's aerodynamics here. However, note that Simulink does provide the necessary capabilities for modeling complex aerodynamic systems in a straightforward manner. Instead, we will assume that the forces and moments generated by each of the helicopter's control actuators is proportional to the control input. This first-order approximation is reasonably valid for control inputs about an equilibrium hovering condition that are not too aggressive. Aerodynamic modeling is limited to representing the drag force, which limits the speed of the helicopter through the atmosphere.

We will assume that the helicopter possesses a navigation system that tracks the vehicle's position, velocity, and angular orientation without error. In reality, such error-free navigation systems do not exist. However, with the availability of small Global Positioning System (GPS) receivers and inertial sensors (gyros and accelerometers), this assumption is not as unrealistic as it would have been a few years ago. We will not model the navigation system's internal behavior. Instead, we will directly use the helicopter's true position, velocity, and angular orientation developed in the simulation as inputs to the control system.

Figure 2 shows the top-level Simulink diagram representing the helicopter plant model. The model inputs are the forces and torques generated by the four control actuators, all described by scalar values. The outputs are the helicopter's position, velocity, angular rotation rate, and Euler angles (roll, pitch, yaw). The outputs are all in the form of three-element vectors.

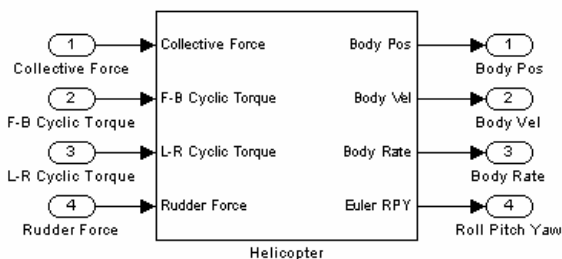


Figure 2 Top-level helicopter plant model

Figure 3 shows the contents of the Helicopter block of Fig. 2. In this figure, the blocks with small squares or circles at the points where lines connect to them model the physical components of the helicopter and the mechanical joints between them. These blocks are from the SimMechanics library. SimMechanics is a companion product to Simulink that provides a capability for modeling systems composed of collections of rigid bodies with mechanical joints of various types connecting them. The equations of motion describing the rotating blades are quite complex. Using SimMechanics to model the dynamic behavior of these bodies significantly reduces the effort required in developing and validating simulations of systems that contain mechanical motion.

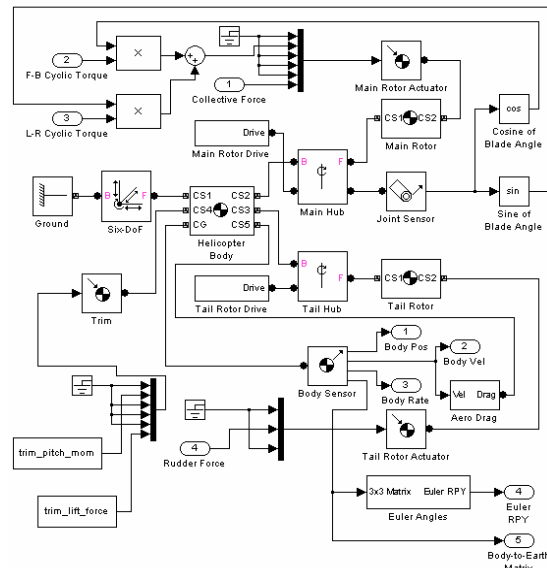


Figure 3 Detailed helicopter plant model

The primary blocks in Fig. 3 are described below.

- The Ground and Six-DoF blocks represent the fixed Earth coordinate system and its relation to the freely moving helicopter body respectively.
- The Helicopter Body, Main Rotor, and Tail Rotor blocks represent the three rigid bodies constituting the helicopter physical model.
- The Main Hub and Tail Hub blocks model the joints connecting the two rotors to the helicopter body. Each joint has one axis of rotational freedom.

- The Main Rotor Drive and Tail Rotor Drive blocks contain subsystems (lower level diagrams) that drive both the rotors at constant angular rates.
- The Trim, Main Rotor Actuator, and Tail Rotor Actuator blocks allow forces and moments to be applied to the helicopter body, main rotor, and tail rotor respectively.
- The Aero Drag subsystem applies a drag force proportional to the square of the helicopter's speed in the negative velocity direction.
- The Joint Sensor and Body Sensor blocks enable measurement of the main rotor's angular orientation about the vertical axis and the body's position, velocity, angular rotation rate, and Euler angles.
- The Collective Force and Rudder Force inputs represent the lift force generated by the main rotor and the side force produced by the tail rotor respectively.
- The Sine of Blade Angle and Cosine of Blade Angle blocks compute the sine and cosine of the main rotor's angular position about its rotational axis. The resulting values multiply the L-R Cyclic Torque (left-right) and F-B Cyclic Torque (front-back) cyclic inputs respectively. These computations model the moment produced by the variation of the blade's angle of attack resulting from cyclic control inputs.
- The `trim_pitch_mom` and `trim_lift_force` blocks represent constant values of pitching moment and lift force applied to the body to attain a trimmed configuration for the hovering helicopter. When trimmed, the main rotor axis is oriented vertically and there is no net translational or angular acceleration on the helicopter.

The mass properties selected for the helicopter body, main rotor, and tail rotor are representative of a model helicopter. To simplify the determination of the mass properties of the three components, each is approximated as a cylinder in terms of mass, length, and radius. A shorter, thick cylinder represents the helicopter body/tail boom assembly, while the blades are modeled as relatively long, thin cylinders.

The rotational velocities of the main and tail rotors are representative of the rotor speeds of a model helicopter. The simulated drive mechanism forces the blades to rotate at a constant angular velocity. This is another simplification; a more realistic blade speed model would account for the torques produced by blade aerodynamic loads and model the response of the engine and drive train to those loads and to variations in the throttle setting.

In operation, this model must execute with a step time short enough that several samples are taken for each rotation of the main rotor blade. This is because (assuming a fixed, nonzero cyclic control input) the moment produced by the main rotor varies sinusoidally during each rotation of the main rotor. A step time of 1.0 millisecond was found to be sufficient for modeling the moment variations due to the collective control.

III. CONTROLLER DESIGN

Once the helicopter model described in the previous section has been developed, control system design can begin. This is a fairly complex system, so we will design the controller in steps.

Although it would be straightforward to develop a high-order linear plant model from the nonlinear Simulink model, it is often not feasible to use such a model directly in designing a controller. The resulting controller would be a complex linear system that is likely to be subject to numerical instability and sudden failure in situations where the actual system's behavior deviates from that of the linear model.

Instead, as a first step, we will assume that this MIMO system can be modeled as a collection of SISO systems. Although some degree of cross coupling is likely, the assumption of SISO systems enables straightforward application of various design approaches such as root locus, pole placement, or LQR design³. Assuming the control system design resulting from this approach is sufficiently robust, it will then be possible to experiment with it to determine which controller elements would most benefit from a later application of more complex MIMO design techniques.

Although the ultimate goal for this control system is to control the motion of the helicopter in three-dimensional space, it is first necessary to control its angular orientation. In fact, if we have good control of

the helicopter's Euler angles, we can use that to control the helicopter's horizontal motion by altering the pitch and roll Euler angles. Pitch and roll motions tilt the main rotor disk relative to the vertical, which results in a net force from the main rotor in the horizontal plane. This force moves the helicopter in the desired direction.

In summary, the design approach for this system will consist of the following steps:

1. Given the nonlinear helicopter simulation, develop a set of SISO linear models assuming each Euler angle is primarily controlled by its associated actuator.
2. Design SISO controllers for each of the three axes.
3. Implement and test the Euler angle controllers.
4. Design a SISO altitude controller.
5. Design two SISO controllers for moving the helicopter in the horizontal plane. One of these is for the front-back motion and the other is for the left-right motion.
6. Implement and test the position controllers.

To begin the development of the Euler angle controllers, it is first necessary to select an equilibrium operating condition about which to linearize the nonlinear model of Fig. 3. We will use a hovering condition for this purpose. In hover, the net lift from the main rotor equals the sum of the weights of the helicopter components. The net moment produced by the main rotor must be equal and opposite to the moment produced by the weights and moment arms of the helicopter components.

Since this model assumes the helicopter body and rotors are rigid bodies at fixed relative locations, it is a straightforward matter to compute the net lift force and pitching moment the main rotor must produce to maintain equilibrium. These constants are placed in the `trim_pitch_mom` and `trim_lift_force` blocks in preparation for the linearization procedure. The helicopter must also be initialized with zero translational and angular velocity and with the main rotor axis aligned vertically relative to the Earth.

In addition to the steps given above, we will take the additional step of stopping the rotor motion during

linearization. While not realistic for modeling purposes, this avoids introducing the dynamics of the rotor motion into the linear system model. This simplifies the process of generating the linear model and reduces the order of the resulting model. It also allows the blade's placement to enable control about the desired axis. Placing the blade at 0° (aligned front-back relative to the body) allows a moment produced by the front-back cyclic control actuator to control the helicopter's pitch motion. Placing the blade at 90° (aligned sideways) allows a moment produced by the left-right cyclic to control the helicopter's roll motion.

As the blade rotates, the angle of attack due to a constant cyclic control input varies sinusoidally. The length of the blade's moment arm in the controlled direction also varies sinusoidally. For example, with a roll command applied to the left-right cyclic and a zero front-back cyclic input, the main rotor exerts a maximum moment when it is aligned in the sideways direction. When it is in the front-back orientation, it generates no moment because at that point there is no angle of attack due to the cyclic input and there is no moment arm allowing the blade to produce a rolling torque.

The result of these effects is that the moment produced by the blade in response to a fixed left-right cyclic control input varies in proportion to $\sin^2 \psi$, where ψ is the main rotor rotation angle about the vertical axis, with $\psi = 0$ when the rotor is aligned in the front-back direction.

The linear model derived using a stationary main rotor blade assumes the blade remains stationary. Since the moment resulting from a control input actually varies with $\sin^2 \psi$, it is necessary to account for this variation. The moment produced over each half-cycle is identical, so we can limit the analysis to a half cycle. Integrating $\sin^2 \psi$ over a half cycle (from 0 to π)

produces the result $\frac{\pi}{2}$. A fixed blade orientation ψ would have a multiplier of 1 instead of the $\sin^2 \psi$ variation. Integrating 1 from 0 to π gives the result π . Since $\frac{\pi}{2}$ is half of π , we see that the control

effectiveness of the rotating blade is exactly half that of the stationary blade used in the linearization.

To compensate for this effect in the resulting linear models, we simply multiply the linear model by 0.5. This accounts for the actual diminished effectiveness of the cyclic control inputs relative to the artificial fixed blade positions used in the linearization.

With the trim values and initial state (including main rotor orientation) specified, the following commands produce a linear state-space model of the helicopter model in the MATLAB command window.

```
[a,b,c,d] = linmod2('Helicopter');
sys = ss(a,b,c,d);
```

The `linmod2` command creates a set of state-space matrices with the inputs and outputs shown in Fig. 2. The resulting linear model has 4 inputs, 12 outputs (consisting of four 3-element vectors), and 16 states.

The next step is to extract SISO models from the 16-state system model representing the transfer function from each of the angular control inputs (the two cyclic controls and the tail rotor collective) to the corresponding Euler angles. For example, pitch motion is controlled by the front-back cyclic. The command to extract the linear model from pitch cyclic input (input number 2) to pitch Euler angle (output 11) is as follows:

```
pitch_sys = minreal(sys(11, 2));
```

In the previous statement, the `minreal` function eliminates states from the model that do not affect the dynamics between the SISO system's input and output. The result is the second order system

$$\frac{\theta}{M_{FB}} = \frac{11.3746}{s^2}, \text{ where } \theta \text{ is the pitch Euler angle}$$

and M_{FB} is the moment applied to the main rotor blade in the front-back plane. It is not surprising that the 16-state full system model reduces to such a simple form when examined as a SISO system: we are simply applying a moment to a collection of rigid bodies at fixed positions relative to each other. Although this is not a precise description of the helicopter in operation, it is suitable for use in controller design.

We will use the pole placement design method here because of its directness and simplicity. To produce a satisfactorily responsive system, we select the

performance specifications to be a 1.0 second settling time and a damping ratio of 0.8. Although the pole placement method does not take into account the control effort, we can examine the performance of the resulting controller and adjust the settling time to a larger value or use the LQR design method in a subsequent design iteration if we find the control effort is excessive.

The observer pole locations are selected to be the closed loop poles multiplied by 3. This results in observer poles that converge rapidly in comparison to the closed loop response while minimizing the sampling rate required for a discrete controller implementation.

The following commands design the observer-controller for the pitch motion. Note that we have included a factor of 0.5 to account for the diminished effectiveness of the cyclic control due to the rotation of the main rotor.

```
t_settle = 1;
damp_ratio = 0.8;
obs_pole_mult = 3;
[N_p, ssobsctrl_p, sscl_p] = ...
    ss_design(0.5*pitch_sys, ...
        t_settle, damp_ratio, ...
        obs_pole_mult);
```

The last statement above calls a custom MATLAB function named `ss_design`, which performs pole placement observer-controller design using the features of the MATLAB Control System Toolbox.

The feedforward gain (N_p) and the state-space observer controller (`ssobsctrl_p`) are integrated into the Simulink diagram as shown in Fig. 4. The controller's r input receives the commanded pitch angle and the y input is the measured pitch angle. The output u is the moment applied to the main rotor by the front-back cyclic.

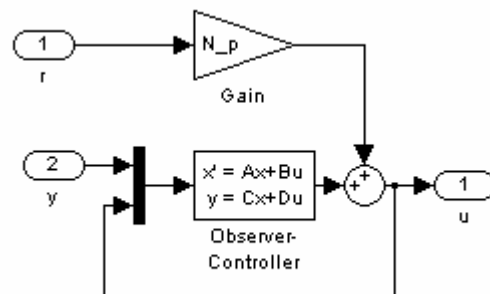


Figure 4 *Pitch observer-controller in Simulink*

Controllers for the left-right cyclic to roll Euler angle and tail rotor collective to yaw Euler angle are designed with similar procedures, except that the yaw controller has a settling time specification of 4 seconds to limit the yaw rate magnitude. After integrating the controllers with the helicopter plant model, the resulting system is shown in Fig. 5. The inputs are now the main rotor collective force and the three commanded angular orientations for the helicopter.

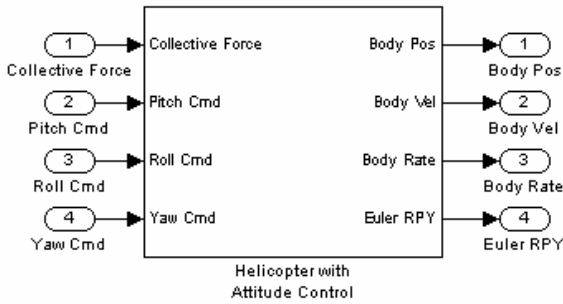


Figure 5 *Helicopter with attitude controllers*

It is now possible to perform some testing of the Euler angle control system. Recall that the model has been set up at a trim condition for hovering. The four inputs in Fig. 5 represent variations from the hovering condition. With all four inputs set to zero, the system should hover in place. Since the position of the helicopter is not yet being controlled, we can expect it to drift from its initial position.

The stability and responsiveness of the angular orientation controllers and the cross coupling between them can be tested by putting a small step command (say, 1°) into each of the pitch, yaw, and roll angle commands and observing the response. Note that changes to the pitch and roll of the rotor disk will introduce a force in the horizontal direction, which will cause the helicopter to begin moving. However, for the small step angle we are using, the horizontal force will be quite small.

Figure 6 displays the responses to 1° step commands applied separately to each of the pitch, yaw, and roll axes. This figure shows that the angular orientation control is stable in response to each of these inputs and the responses converge to the commanded values.

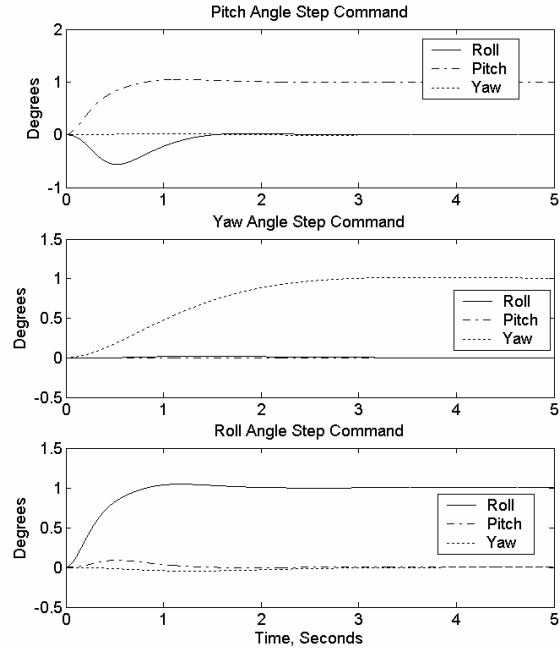


Figure 6 *Step responses of angular orientation controllers*

However, note that in the top plot of Fig. 6 there is significant cross coupling from the pitch angle command into the helicopter's roll angle. This degree of cross coupling may be unacceptable for a final design, but for now we will simply note its presence and continue with the design. The cross coupling resulting from the yaw and roll step inputs is at a much lower level compared to the pitch response.

The next step in the design is to develop a vertical position (altitude) controller. We will again assume a SISO controller is appropriate. The input to the controller is the commanded altitude and the plant output is the measured altitude. The actuator is the main rotor collective force. The altitude controller is designed using the following commands:

```
t_settle = 4;
damp_ratio = 0.8;
obs_pole_mult = 3;
[N_v, ssobsctrl_v, sscl_v] =
ss_design(vert_sys, t_settle, ...
          damp_ratio, obs_pole_mult);
N_v = -N_v; % Sign change
```

The sign change for the feedforward gain (N_v) is necessary because the vertical dimension of the position vector in the model is positive downward, but altitude commands are positive in the upward direction. The resulting observer-controller has the same form as the pitch observer-controller shown in Fig. 4.

A plot of the step response to a 1 meter change in altitude along with the responses of the three Euler angle controllers to this step appears in Fig. 7. Note that cross coupling into the Euler angles is present, but the magnitudes of the responses are just a fraction of a degree about each axis.

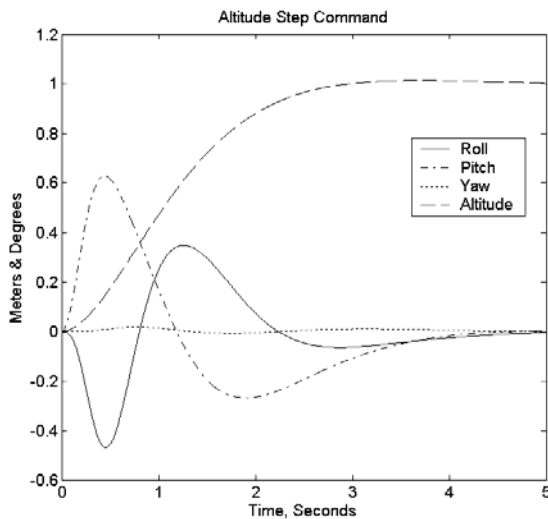


Figure 7 Step response of altitude controller

It is necessary to use limiting in the altitude control loop because the altitude command could become quite large. For instance, suppose the helicopter is commanded to an altitude of 1000 meters instead of 1 meter. A truly linear response would amplify each of the traces in Fig. 7 by a factor of 1000, which is clearly unacceptable. A limiter applied to the output of the altitude controller restricts the amount of force applied by the main rotor collective. This also restricts the magnitude of the cross coupling into the Euler angles from the altitude control system.

The remaining dimensions of control involve the helicopter's motion in the horizontal plane. We can break this motion into two components relative to the helicopter body: forward-backward motion and side-to-side motion. The forward-backward motion is affected

primarily by the tilt of the main rotor about the helicopter pitch axis. For example, tilting the rotor so it is lower when forward of the body results in a horizontal force from the main rotor in the forward direction. Similarly, side-to-side motion is affected by the tilt of the main rotor about the helicopter roll axis.

Viewed in this way, the actuator input for horizontal motion is the commanded main rotor tilt in the desired direction and the output is the helicopter horizontal position in that direction. In principle, we can use any SISO design method to develop controllers for motion in the front-back and side-to-side directions.

However, a linear model of the dynamic behavior from the commanded main rotor tilt angle to the horizontal motion would be quite complex. This model would contain not only the dynamics of the helicopter system; it would also include the observer-controller that controls the rotor tilt angle. The observer-controller resulting from such a model would be of high order and would be susceptible to numerical difficulties. To bypass (at least temporarily) the complexity and difficulty of this approach, we can try using PID control for the horizontal motion instead.

PID control is often used in applications where the plant is of high order and its dynamic behavior is not well understood. For our helicopter model, although we have a complete model of the plant dynamics, the high order controller that would result is not desirable.

We will use PID controllers to drive the helicopter to a specified point in space called a waypoint. A higher-level guidance function can select and sequence through waypoints, allowing the helicopter to be flown along an arbitrary trajectory. However, for control system design purposes, we will assume that there is only one waypoint and the vector (in helicopter body coordinates) from the helicopter's current position to the position commanded by the waypoint horizontal position determines the controller error signal. Since the position error may have an arbitrarily large magnitude, it is necessary to place limiters on the output of the PID controller, which will limit the commanded main rotor tilt angle to a maximum magnitude (selected to be 10°) in the front-back and side-to-side directions.

Because the plant model does not include any effects that would lead to a steady state horizontal position error (such as wind), it is not necessary to use an integral term in the current controller implementation.

This simplifies the PID controller to a PD controller. PD gains that result in satisfactory system performance are determined iteratively.

Figure 8 shows the resulting system's response to a step command to move forward 400 meters. Starting from a trim condition at the Earth coordinate system origin, the main rotor tilts -10° in pitch to produce a force moving the helicopter forward while maintaining constant altitude. The model includes the effect of aerodynamic drag, so the helicopter soon reaches an equilibrium cruising speed. After about 40 seconds the PD controller commands the rotor to pitch in the positive direction, slowing the helicopter as it approaches the waypoint. The helicopter stops at the waypoint and hovers there. Some cross coupling is noticeable from the pitch motion into the roll, but this results in only a peak of 3° of roll motion.

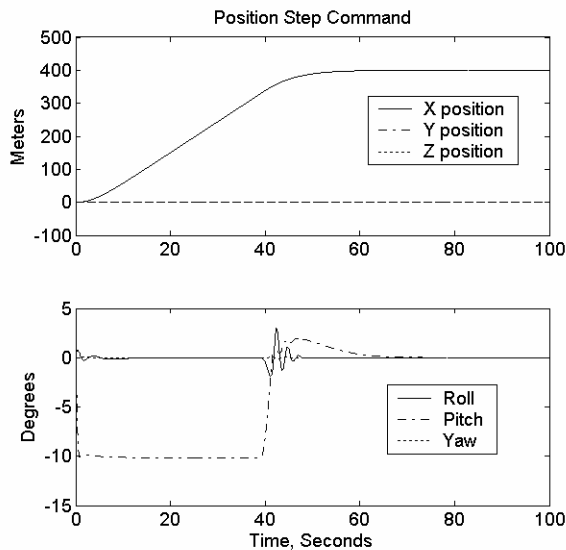


Figure 8 Step response of position controller

This completes the initial iteration of the helicopter control system design. We have observed some areas of the existing design where improvement is possible. In particular, the cross coupling of the pitch angle control into roll has a noticeable effect on performance. A second design iteration could address this issue by using a MIMO design approach to develop a coupled pitch/roll controller that minimizes the cross coupling effects.

Another way to improve the controller design would be to work with a more realistic helicopter model. For example, instead of assuming a perfect inertial measurement system, realistic errors could be added to the simulated position and angular orientation measurements. Further design iteration could attempt to optimize controller performance using LQR controllers and Kalman filter observers.

IV. EMBEDDED CODE GENERATION

After the Simulink model of the helicopter and its GN&C system has been tested thoroughly, the embedded software to implement the GN&C systems can be generated.

The first step in this process is to create a new, empty Simulink diagram and copy only the blocks containing the GN&C software elements from the original model into the new diagram. The copied blocks will have unconnected signals at each point where the software communicates with the plant through an input/output (I/O) device.

Each I/O signal must be connected to an appropriate device driver, represented by a block selected from the xPC Target blockset. Over 150 predefined I/O devices are supported directly and the developer can create new I/O device driver blocks as needed.

If the embedded system employs a PC-compatible CPU, the xPC Target and xPC Target Embedded Option products from The MathWorks provide automated embedded code generation, compilation, and deployment using a provided real time kernel. The xPC Target environment can operate while connected to a host for program downloading and testing purposes, or in a standalone operational system.

For embedded processors of other types, including resource-limited 8- and 16-bit CPUs, the code generation process can be tailored to meet the needs of the situation. The data types of all signals in the Simulink model can be specified in any of a variety of floating-point and fixed-point numeric formats. Through careful selection of signal data types, it is possible to trade off memory usage and execution speed against the precision of numerical results in the GN&C system. The specification of signal data types should be carried out in the Simulink model containing the plant. This allows any signal precision limitations to be tested thoroughly in conjunction with the plant inside the

Simulink environment, prior to working with the embedded processor.

For our helicopter example, the I/O between the embedded processor and the plant takes place entirely across a Controller Area Network (CAN) bus⁴. xPC Target provides support for CAN interfaces, so it is only necessary to connect the appropriate I/O block and configure some settings associated with it.

After the I/O blocks have been added to the Simulink model, the code generation process has been tailored for the execution environment, and the signal data types have been set appropriately, the actual generation of the model code requires only the clicking of a few buttons in a dialog box. Depending on the target processor type (xPC Target or otherwise), the code generation may be followed by automated compilation and linking of the embedded execution image. If the software build is not automated, it is necessary for the developer to perform the appropriate cross-compilation steps to build the image.

The executable image can then be downloaded to the embedded processor for execution. Although some effort may be required in the tailoring of the code generation process for a particular embedded processor, realize that this is a one-time cost. For ongoing iterations of the control system design, changes can be made to the original simulation containing the plant plus the GN&C system. After testing the changes in the Simulink environment, the new GN&C system implementation can be copied to the second Simulink diagram containing just the embedded code and the I/O device interfaces. The embedded code for the new GN&C system version can then be generated and built quickly.

V. HIL SIMULATION

Following the build and installation of the executable image on the embedded processing system, the next stage of development is the thorough testing of the software in a realistic environment. The most obvious way to do this is to operate the embedded system in its intended environment. However, in many cases this is not feasible because a prototype system does not yet exist at the time the embedded code is ready for testing. It also may be impossible to test the system prior to its deployment. For example, the GN&C software for a satellite cannot be tested with the complete embedded system without launching the satellite into space.

Even if a complete system is available, there may be reasons for not executing the initial versions of the embedded software on it. One reason is that it may be unwise to risk damaging expensive prototype hardware with untested software. Another reason is that testing with the prototype hardware may be very costly, as with flight-testing a commercial aircraft, for example.

For the reasons given above, it is highly desirable to employ a system test methodology that allows thorough, repeatable, and inexpensive testing of the embedded software in a laboratory environment.

Hardware-in-the-loop (HIL) simulation¹ provides such a capability. HIL simulation combines the system's embedded software executing on the embedded processing hardware with a real time simulation of the plant and its operational environment. The HIL simulation should employ the same I/O interfaces between the embedded processor and the plant simulation that the actual embedded plant uses to interface with the processor.

The models of the plant and environmental effects used in the HIL simulation must be of sufficient fidelity to enable realistic testing of the embedded software under all significant modes of operation. It should be possible to traverse the system's entire operational envelope in the HIL simulation environment and accurately determine the system's behavior under nominal as well as extreme conditions.

In our helicopter example, the plant model is shown in Fig. 3. This model must be implemented as a real time simulation and interfaced with the embedded processor code containing the GN&C algorithms. Communication between the embedded software and the real time plant simulation takes place across a CAN bus interface.

The xPC Target product enables the simultaneous execution of the real time controller code and the plant and environment simulations on a single PC computer system containing a CAN bus I/O interface. This provides a powerful, inexpensive capability for thoroughly testing the embedded software in any desired scenario.

An additional benefit of the HIL simulation capability is that it is possible to set up and execute test scripts easily, which enables rapid and repeatable regression testing to ensure system performance is not degraded as enhancements are made to the GN&C software design.

VI. CONCLUSION

This paper demonstrated the steps involved in developing a plant simulation, employing a model-based GN&C system design approach, generating embedded code from the GN&C system simulation model, and thoroughly testing the embedded software using HIL simulation. Commercial products available from The MathWorks, Inc. enable and support each of these steps in developing and deploying embedded software.

The net result of the approach described in this paper is thoroughly tested embedded software that is available much earlier in the development cycle compared to traditional development methods that do not integrate the software with the hardware until both software and hardware development are largely completed.

REFERENCES

- [1] Ledin, J., *Simulation Engineering*. Lawrence, KS: CMP Books, 2001.
- [2] <http://www.mathworks.com>
- [3] Franklin, Gene F., J. David Powell, and Abbas Emami-Naeini, *Feedback Control of Dynamic Systems*. Reading, MA: Addison Wesley, 1986.
- [4] <http://www.can-cia.de/can/>