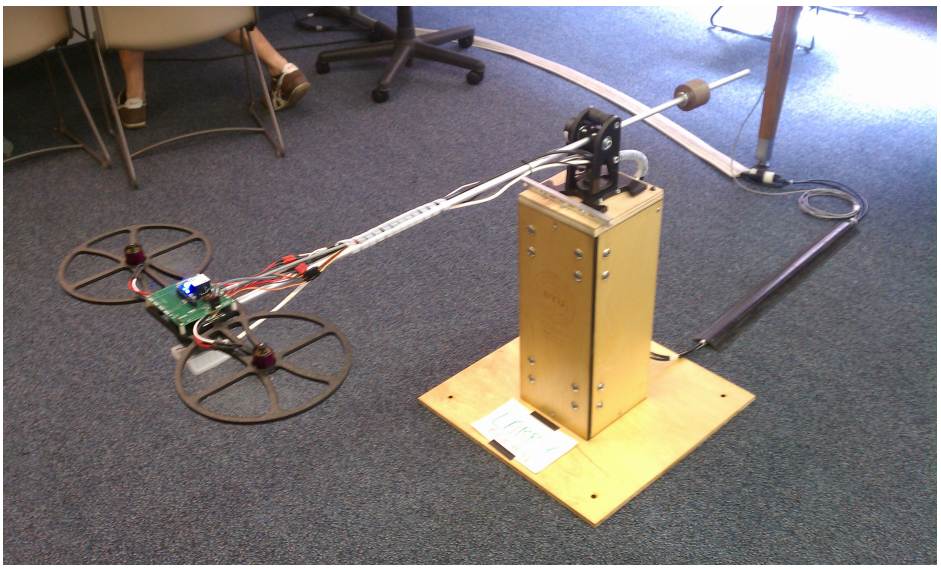
A Guide to the BYU Whirlybird
Version 4.0.1



Randal W. Beard, Tim McLain
Brigham Young University

Updated: October 2015

Contents

Introduction

This objective of this document is to provide a detailed introduction to the BYU whirlybird including laboratory assignments that are intended to complement an introductory course in feedback control. The whirlybird is a three degree-of-freedom helicopter with complicated nonlinear dynamics, but the dynamics are amenable to linear analysis and design. The whirlybird has been designed with encoders on each joint that allow full-state feedback, but it is also equipped with an IMU (rate gyros and accelerometers) and a downward looking camera that resemble the types of sensors that will be on a real aerial robot. The sensor configuration allows estimated state information obtained from the IMU and camera to be compared to truth data obtained from the encoders, thereby decoupling the state feedback and state estimation problems, and allowing each to be tuned and analyzed independently. We believe that this decoupling is essential to help students understand the subtleties in feedback and estimator design.

The design process for control system is shown in Figure 1. For the whirlybird, the system to be controlled is a physical system with actuators (motors) and sensors (encoders, IMU, camera). The first step in the design process is to model the physical system using nonlinear differential equations. While approximations and simplifications will be necessary at this step, the hope is to capture in mathematics, all of the important characteristics of the physical system. The mathematical model of the system is usually obtained using Euler-Lagrange method, Newton's method, or other methods from physics and chemistry. The resulting model is usually high order and too complex for design. However, this model will be used to test later designs in simulation and is therefore called the Simulation Model, as shown in Figure 1. To facilitate design, the simulation model is simplified and usually linearized to create lower-order (and usually linear) design models. For any physical system, there may be multiple design models that capture certain

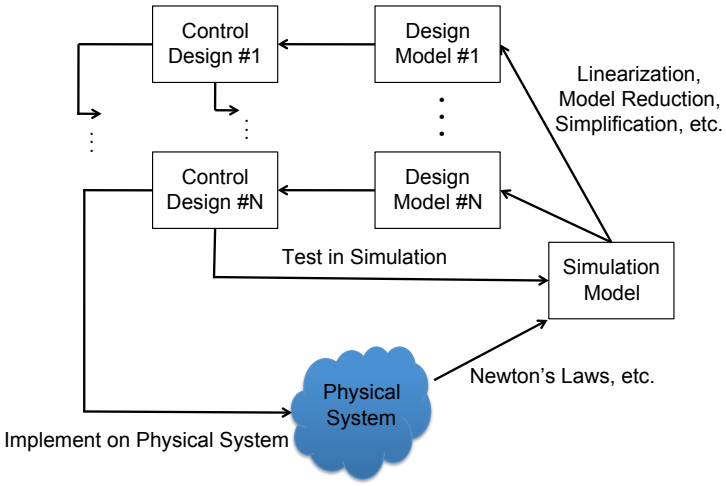


Figure 1: The design process. Using physics models the physical system is modeled with nonlinear differential equations resulting in the simulation model. The simulation model is simplified to create design models which are used for the control design. The control design is then tested and debugged in simulation and finally implemented on the physical system.

aspects of the design process. For the whirlybird, we will decompose the motion into longitudinal (pitching) motion and lateral (rolling and yawing) motion and we will have different design models for each type of motion. As shown in Figure 1, the design models are used to develop control designs. The control designs are then tested against the simulation, which sometimes requires that the design models be modified or enhanced if they have not captured the essential features of the system. After the control designs have been thoroughly tested in simulation, they are implemented on the physical system and are again tested and debugged, sometimes requiring that the simulation model be modified to more closely match the physical system.

The purpose of these notes, and the associated laboratory assignments, is to walk you through the design cycle for the whirlybird. We will consider three different design methodologies: successive loop closure, loopshaping, and observer-based control. The simulation model will be implemented in Matlab and Simulink. We believe that visualizing the physical system is an important part of the simulation model. Therefore the first lab assignment described in Chapter **??** will describe how to draw animations in Simulink. Implementing complicated mathematical models in Simulink requires knowl-

edge of s-functions, which are typically unfamiliar to students. Therefore the second lab assignment described in Chapter **??** shows how to create s-functions. A simulation model for the whirlybird is derived in Chapter **??** using the Euler-Lagrange method, and the associated lab is to implement this model in a Simulink s-function. The fourth lab, described in Chapter **??**, is to develop the design models that will be used in subsequent labs. Developing physically realizable design specifications is an important part of the design process, and so the fifth lab assignment, described in Chapter **??** is to develop appropriate design specifications. The final three labs are to design implement controllers for the whirlybird using three different design techniques. These labs can be done in any order. In Chapter **??**, successive loop closure using PID is used to control the whirlybird. In Chapter **??**, the loopshaping method is used, and Chapter **??** walks the student through an observer-based design, where an extended Kalman filter is used to track a target in the camera plane.

Lab 1

Animations in Simulink

We believe that visualizing the control design in the simulation stage helps the designer to develop physical intuition for the system and is an important debugging tool. Therefore, the first step in developing a simulation model for the whirlybird will be to create an animation in Matlab/Simulink. Students unfamiliar with Matlab and/or Simulink should read the `Getting Started` sections of the Matlab and Simulink documentation, which can be found by typing `helpdesk` at the Matlab prompt.

1.1 Handle Graphics in Matlab

When a graphics function like `plot` is called in Matlab, the function returns a *handle* to the plot. A graphics handle is similar to a pointer in C/C++ in the sense that all of the properties of the plot can be accessed through the handle. For example, the Matlab command

```
>> plot_handle=plot(t,sin(t))
```

returns a pointer, or handle, to the plot of `sin(t)`. Properties of the plot can be changed by using the handle, rather than reissuing the `plot` command. For example, the Matlab command

```
>> set(plot_handle,'YData',cos(t))
```

changes the plot to `cos(t)`, without redrawing the axes, title, label, and other objects that may be associated with the plot. If the plot contains

drawings of several objects, then a handle can be associated with each object. For example,

```
>> plot_handle1 = plot(t,sin(t))
>> hold on
>> plot_handle2 = plot(t,cos(t))
```

draws both `sin(t)` and `cos(t)` on the same plot, with a handle associated with each object. The objects can be manipulated separately without redrawing the other object. For example, to change `cos(t)` to `cos(2t)`, issue the command

```
>> set(plot_handle2,'YData',cos(2*t))
```

We can exploit this property to animate simulations in Simulink by only redrawing the parts of the animation that change in time, and thereby significantly reducing the simulation time. To show how handle graphics can be used to produce animations in Simulink, we will provide three detailed examples. In Section 1.2 we illustrate a 2D animation of an inverted pendulum using the fill command. In Section 1.3 we illustrate a 3D animation of a spacecraft using lines to produce a stick figure. In Section 1.4 we modify the spacecraft animation to use the vertices-faces data construction in Matlab.

1.2 Animation Example: Inverted Pendulum

Consider the image of the inverted pendulum shown in Figure 1.1, where the configuration is completely specified by the position of the cart y, and the angle of the rod from vertical θ. The physical parameters of the system are the rod length L, the base width w, the base height h, and the gap between the base and the track g. The first step in developing the animation is to determine the position of points that define the animation. For example, for the inverted pendulum in Figure 1.1, the four corners of the base are

$$(y + w/2, g), (y + w/2, g + h), (y - w/2, g + h), (y - w/2, g),$$

and the two ends of the rod are given by

$$(y, g + h), (y + L\sin\theta, g + h + L\cos\theta).$$

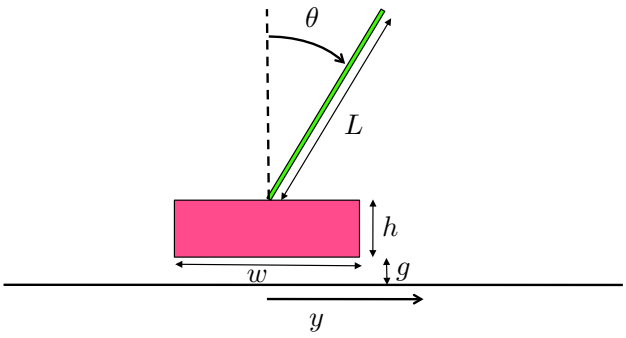


Figure 1.1: Drawing for inverted pendulum. The first step in developing an animation is to draw a figure of the object to be animated and identify all of the physical parameters.

Since the base and the rod can move independently, each will need its own figure handle. The `drawBase` command can be implemented with the following Matlab code:

```
1     function handle = drawBase(y,width,height,gap,handle)
2       X = [y-width/2, y+width/2, y+width/2, y-width/2];
3       Y = [gap, gap, gap+height, gap+height];
4       if isempty(handle),
5         handle = fill(X,Y,'m');
6       else
7         set(handle,'XData',X,'YData',Y);
8       end
```

Lines 2 and 3 define the X and Y locations of the corners of the base. Note that in Line 1, `handle` is both an input and an output. If an empty array is passed into the function, then the `fill` command is used to plot the base in Line 5. On the other hand, if a valid handle is passed into the function, then the base is redrawn using the `set` command in Line 7.

The Matlab code for drawing the rod is similar and is listed below.

```
1     function handle = drawRod(y,theta,L,gap,height,handle)
2       X = [y, y+L*sin(theta)];
3       Y = [gap+height, gap + height + L*cos(theta)];
4       if isempty(handle),
5         handle = plot(X,Y,'g');
6       else
```

```
7         set(handle,'XData',X,'YData',Y);
8       end
```

The main routine for the pendulum animation is listed below.

```
1     function drawPendulum(u)
2         % process inputs to function
3         y        = u(1);
4         theta    = u(2);
5         t        = u(3);
6
7         % drawing parameters
8         L = 1;
9         gap = 0.01;
10        width = 1.0;
11        height = 0.1;
12
13        % define persistent variables
14        persistent base_handle
15        persistent rod_handle
16
17        % first time function is called, initialize plot and persistent vars
18        if t==0,
19            figure(1), clf
20            track_width=3;
21            plot([-track_width,track_width],[0,0],'k'); % plot track
22            hold on
23            base_handle = drawBase(y,width,height,gap,[],'normal');
24            rod_handle  = drawRod(y,theta,L,gap,height,[],'normal');
25            axis([-track_width,track_width,-L,2*track_width-L]);
26
27        % at every other time step, redraw base and rod
28        else
29            drawBase(y,width,height,gap,base_handle);
30            drawRod(y,theta,L,gap,height,rod_handle);
31        end
```

The routine `drawPendulum` is called from the Simulink file shown in Figure 1.2, where there are three inputs: the position y, the angle θ, and the time t. Lines 3-5 rename the inputs to y, θ, and t. Lines 8-11 define the drawing parameters. We require that the handle graphics persist between function calls to `drawPendulum`. Since a handle is needed for both the base and the rod, we define two persistent variables in Lines 14 and 15. The `if` statement in Lines 18-31 is used to produce the animation. Lines 19–25 are called once

at the beginning of the simulation, and draw the initial animation. Line 19 brings the figure 1 window to the front, and clears it. Lines 20 and 21 draw the ground along which the pendulum will move. Line 23 calls the `drawBase` routine with an empty handle as input, and returns the handle `base_handle` to the base. Line 24 calls the `drawRod` routine, and Line 25 sets the axes of the figure. After the initial time step, all that needs to be changed are the locations of the base and rod. Therefore, in Lines 29 and 30, the `drawBase` and `drawRod` routines are called with the figure handles as inputs.

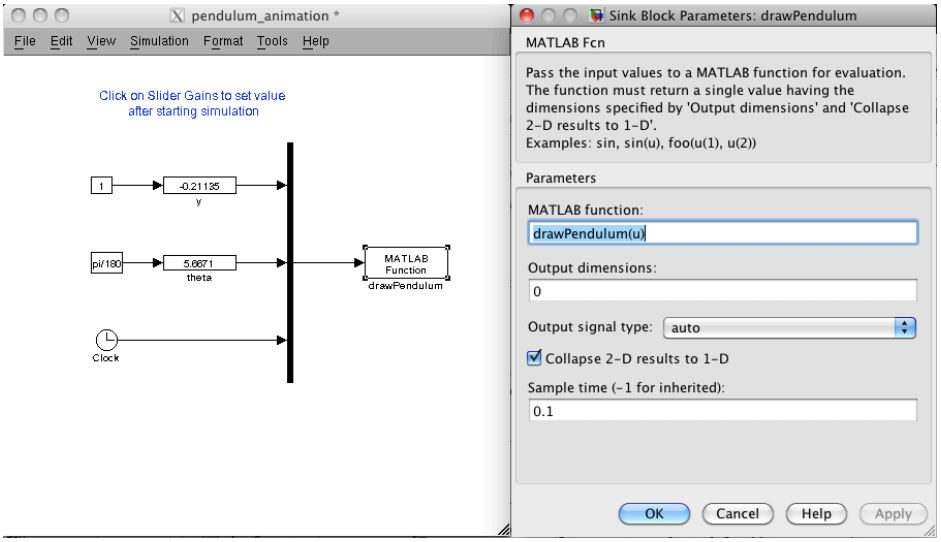


Figure 1.2: Simulink file for debugging the pendulum simulation. There are three inputs to the matlab m-file `drawPendulum`: the position y, the angle θ, and the time t. Slider gains for y and θ are used to verify the animation.

1.3 Animation Example: Spacecraft Using Lines

The previous section described a simple 2D animation. In this section we discuss a 3D animation of a spacecraft with six degrees of freedom. Figure 1.3 shows a simple line drawing of a spacecraft, where the bottom is meant to denote a solar panel that should be oriented toward the sun.

The first step in the animation process is to label the points on the spacecraft and to determine their coordinates in a body fixed coordinate system.

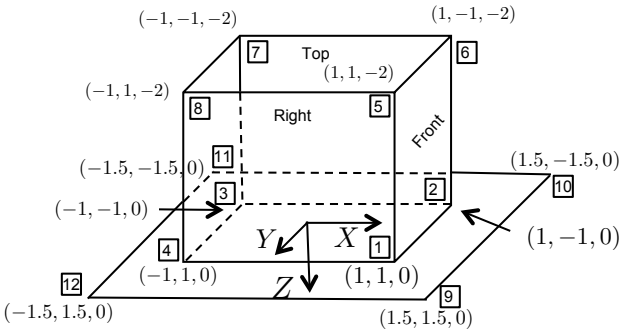


Figure 1.3: Drawing used to create spacecraft animation. Standard aeronautics body axes are used, where the x-axis points out the front of the spacecraft, the y-axis points to the right, and the z-axis point out the bottom of the body.

We will use standard aeronautics axes with X pointing out the front of the spacecraft, Y pointing to the right, and Z pointing out the bottom. The points 1 through 12 are labeled in Figure 1.3 and specific coordinates are assigned to each label. To create a line drawing we need to connect the points in a way that draws each of the desired line segments. To do this as one continuous line, some of the segments will need to be repeated. To draw the spacecraft shown in Figure 1.3 we will transition through the following nodes $1-2-3-4-1-5-6-2-6-7-3-7-8-4-8-5-1-9-10-2-10-11-3-11-12-4-12-9$ Matlab code that defines the local coordinates of the spacecraft is given below.

```
function XYZ=spacecraftPoints;
  % define points on the spacecraft in local NED coordinates
  XYZ = [...
     1    1   0;... % point 1
     1   -1   0;... % point 2
    -1   -1   0;... % point 3
    -1    1   0;... % point 4
     1    1   0;... % point 1
     1    1  -2;... % point 5
     1   -1  -2;... % point 6
     1   -1   0;... % point 2
     1   -1  -2;... % point 6
    -1   -1  -2;... % point 7
    -1   -1   0;... % point 3
```

```
        -1   -1  -2;... % point 7
        -1    1  -2;... % point 8
        -1    1   0;... % point 4
        -1    1  -2;... % point 8
         1    1  -2;... % point 5
         1    1   0;... % point 1
         1.5  1.5 0;... % point 9
         1.5 -1.5 0;... % point 10
         1   -1   0;... % point 2
         1.5 -1.5 0;... % point 10
        -1.5 -1.5 0;... % point 11
        -1   -1   0;... % point 3
        -1.5 -1.5 0;... % point 11
        -1.5  1.5 0;... % point 12
        -1    1   0;... % point 4
        -1.5  1.5 0;... % point 12
         1.5  1.5 0;... % point 9
         ]';
```

The configuration of the spacecraft is given by the Euler angles ϕ, θ, and ψ, which represent the roll, pitch, and yaw angles respectively, and p_n, p_e, p_d, which represent the North, East, and Down positions respectively. The points on the spacecraft can be rotated and translated using the Matlab code listed below.

```
function XYZ=rotateBody(XYZ,phi,theta,psi);
  % define rotation matrix
  R_roll = [...
          1, 0, 0;...
          0, cos(phi), -sin(phi);...
          0, sin(phi), cos(phi)];
  R_pitch = [...
          cos(theta), 0, sin(theta);...
          0, 1, 0;...
          -sin(theta), 0, cos(theta)];
  R_yaw = [...
          cos(psi), -sin(psi), 0;...
          sin(psi), cos(psi), 0;...
          0, 0, 1];
  R = R_yaw*R_pitch*R_roll;
  % rotate vertices
  XYZ = R*XYZ;
```

```
1     function XYZ = translateBody(XYZ,pn,pe,pd)
2       XYZ = XYZ + repmat([pn;pe;pd],1,size(XYZ,2));
```

Notice the order of the rotations performed. First the spacecraft is rolled about the North axis by the angle ϕ. Next it is pitched about the East axis by the angle θ. Finally, it is yawed about the Down axis by the angle ψ. Drawing the spacecraft at the desired location is accomplished using the following Matlab code.

```
1     function handle = drawSpacecraftBody(pn,pe,pd,phi,theta,psi,handle)
2       % define points on spacecraft in local NED coordinates
3       NED = spacecraftPoints;
4       % rotate spacecraft by phi, theta, psi
5       NED = rotateBody(NED,phi,theta,psi);
6       % translate spacecraft to [pn; pe; pd]
7       NED = translateBody(NED,pn,pe,pd);
8        % transform vertices from NED to XYZ (for matlab rendering)
9        R = [...
10           0, 1, 0;...
11           1, 0, 0;...
12           0, 0, -1;...
13           ];
14       XYZ = R*NED;
15      % plot spacecraft
16      if isempty(handle),
17        handle = plot3(XYZ(1,:),XYZ(2,:),XYZ(3,:));
18      else
19         set(handle,'XData',XYZ(1,:),'YData',XYZ(2,:),'ZData',XYZ(3,:));
20        drawnow
21      end
```

Lines 9–14 are used to transform the coordinates from the North-East-Down (NED) coordinate frame, to the drawing frame used by Matlab which has the x-axis to the viewers right, the y-axis into the screen, and the z-axis up. The `plot3` command is used in Line 17 to render the original drawing, and the `set` command is used to change the `XData`, `YData`, and `ZData` in Line 19. A rendering of the spacecraft is shown in Figure 1.4.

The disadvantage of implementing the animation using the function `spacecraftPoints` to define the spacecraft points, is that this function is called each time the animation is updated. Since the points are static, they only need to be defined once. The Simulink mask function can be used to define the points at the beginning of the simulation. Masking the `drawSpacecraft` m-file in

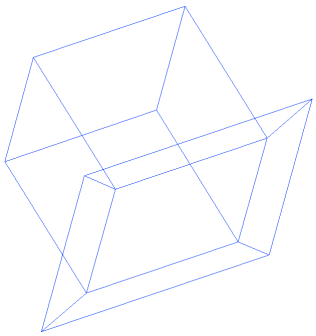


Figure 1.4: Rendering of the spacecraft using lines and the `plot3` command.

Simulink, and then clicking on `Edit Mask` brings up a window like the one shown in Figure 1.5. The spacecraft points can be defined in the initialization window as shown in Figure 1.5 and passed to the `drawSpacecraft` m-file as a parameter.

1.4 Animation Example: Spacecraft using vertices and faces

The stick-figure drawing shown in Figure 1.4 can be improved visually by using the vertex-face structure in Matlab. Instead of using the `plot3` command to draw a continuous line, we will use the `patch` command to draw faces defined by vertices and colors. The vertices, faces, and colors for the spacecraft are defined in the Matlab code listed below.

```
1     function [V, F, patchcolors]=spacecraftVFC
2     % Define the vertices (physical location of vertices
3       V = [...
4           1    1    0;... % point 1
5           1   -1    0;... % point 2
6          -1   -1    0;... % point 3
7          -1    1    0;... % point 4
8           1    1   -2;... % point 5
9           1   -1   -2;... % point 6
10         -1   -1   -2;... % point 7
11         -1    1   -2;... % point 8
12          1.5  1.5  0;... % point 9
```

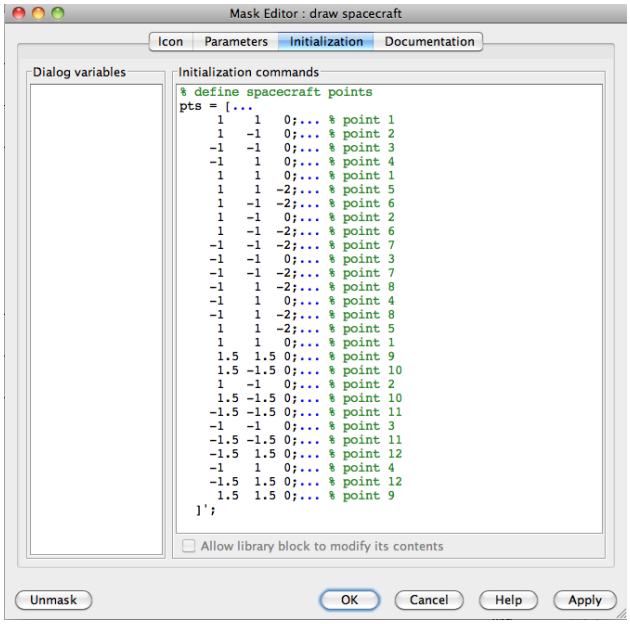


Figure 1.5: The mask function in Simulink allows the spacecraft points to be initialized at the beginning of the simulation.

```
13          1.5 -1.5  0;... % point 10
14        -1.5 -1.5  0;... % point 11
15        -1.5  1.5  0;... % point 12
16      ];
17    % define faces as a list of vertices numbered above
18      F = [...
19            1, 2,  6,  5;...  % front
20            4, 3,  7,  8;...  % back
21            1, 5,  8,  4;...  % right
22            2, 6,  7,  3;...  % left
23            5, 6,  7,  8;...  % top
24            9, 10, 11, 12;... % bottom
25            ];
26    % define colors for each face
27      myred    = [1, 0, 0];
28      mygreen  = [0, 1, 0];
29      myblue   = [0, 0, 1];
30      myyellow = [1, 1, 0];
31      mycyan   = [0, 1, 1];
32      patchcolors = [...
33        myred;...    % front
```

```
34          mygreen;...  % back
35          myblue;...   % right
36          myyellow;... % left
37          mycyan;...   % top
38          mycyan;...   % bottom
39          ];
```

The vertices are shown in Figure 1.3 and are defined in Lines 3–16. The faces are defined by listing the indices of the points that define the face. For example, the front face, defined in Line 19 consists of points $1-2-6-5$. Faces can be defined by N-points, where the matrix that defines the faces has N columns, and the number of rows is the number of faces. The color for each face is defined in Lines 32–39. Matlab code that draws the spacecraft body is listed below.

```
1     function handle = drawSpacecraftBody(pn,pe,pd,phi,theta,psi, handle)
2       [V, F, patchcolors] = spacecraftVFC;  % define points on spacecraft
3       V = rotate(V', phi, theta, psi)'; % rotate spacecraft
4       V = translate(V', pn, pe, pd)'; % translate spacecraft
5       R = [...
6           0, 1, 0;...
7           1, 0, 0;...
8           0, 0, -1;...
9           ];
10      V = V*R;  % transform vertices from NED to XYZ (for matlab rendering)
11      if isempty(handle),
12      handle = patch('Vertices', V, 'Faces', F,...
13                     'FaceVertexCData',patchcolors,...
14                     'FaceColor','flat');
15      else
16        set(handle,'Vertices',V,'Faces',F);
17      end
```

The transposes in Lines 3 and 4 are used because the physical positions in the vertices matrix V are along the rows instead of the columns. A rendering of the spacecraft using vertices and faces is given in Figure 1.6.

Lab 1: Assignment

The objective of this assignment is to create a 3D graphic of the whirlybird that is correctly rotated and translated to the desired configuration.

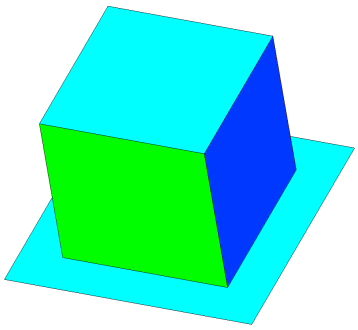


Figure 1.6: Rendering of the spacecraft using vertices and faces.

1. Create an animation drawing of the whirlybird shown in Figure 1.7. Use the parameters given Appendix A.

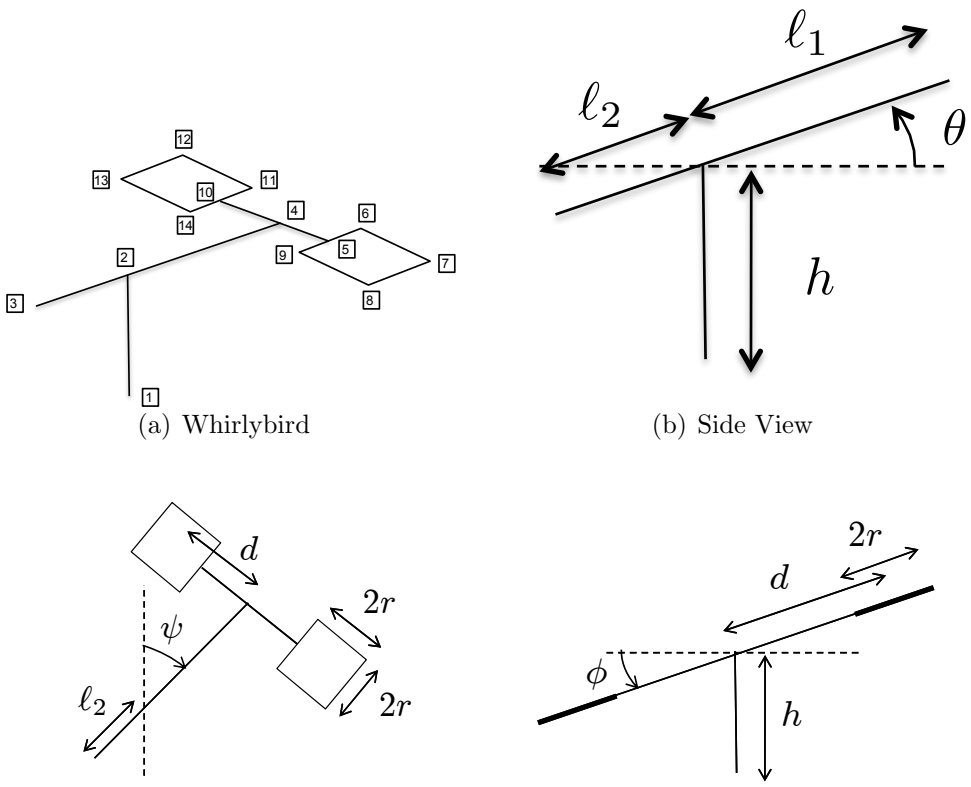
2. Create a Simulink model where the input to the animation file is the roll angle ϕ, the pitch angle θ, and the yaw angle ψ. Verify that the whirlybird is correctly rotated and translated in the animation. Hint: Point 2 in Figure 1.7(a) is the intersection of the three axes of rotation, ψ, θ, and ϕ. Making the origin of your NED reference frame coincide with point 2 simplifies the drawing the the whirlybird. Perform the rotations of the moving portions of the whirlybird in this order: (1) roll about North axis by ϕ, (2) pitch about East axis by θ, (3) yaw about Down axis by ψ .

3. Extra Credit: Improve the whirlybird animation so that it looks more realistic.

Lab 1: Hints

1. When drawing the whirlybird, it is important to correctly identify the orientation of the whirlybird axis. Pay close attention to Figure 1.3 and its explanation in the text to identify the whirlybird axis.

2. When using *handle graphics*, a separate handle must be defined for each object that moves independently. For example, the whirlybird base does not roll, pitch or yaw and therefore should have its own handle.



(a) Whirlybird

(b) Side View

(c) Top View

(d) Front View

Figure 1.7: Specifications for animation of whirlybird.

3. When using the *patch* function, the number of vertices that define a face must be 3 or greater and every face must have the same number of vertices. Faces with fewer vertices can be drawn by repeating the final point through the end of the row.

4. When using the *patch* function, the number faces must be equal to the number of face colors. If the number of faces differs from the number of face colors then the object will not be drawn.

Lab 2

S-functions in Simulink

This chapter assumes basic familiarity with the Matlab/Simulink environment. For additional information, please consult the Matlab/Simulink documentation. Simulink is essentially a sophisticated tool for solving interconnected hybrid ordinary differential equations and difference equations. Each block in Simulink has the structure

$$\dot{x}_c = f(t, x_c, x_d, u); \qquad x_c(0) = x_{c0} \tag{2.1}$$

$$x_d[k+1] = g(t, x_c, x_d, u); \qquad x_d[0] = x_{d0} \tag{2.2}$$

$$y = h(t, x_c, x_d, u) \tag{2.3}$$

where $x_c \in \mathbb{R}^{n_c}$ is a continuous state with initial condition x_{c0}, $x_d \in \mathbb{R}^{n_d}$ is a discrete state with initial condition x_{d0}, $u \in \mathbb{R}^m$ is the input to the block, $y \in \mathbb{R}^p$ is the output of the block, and t is the elapsed simulation time. An *s-function* is a Simulink tool for explicitly defining the functions f, g, and h and the initial conditions x_{c0} and x_{d0}. As explained in the Matlab/Simulink documentation, there are a number of methods for specifying an s-function. In this Chapter, we will overview the level-1 m-file s-function. We will show how to implement the system specified by the standard second order transfer function

$$Y(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} U(s). \tag{2.4}$$

The first step is either case, is to represent (2.4) in state space form. Using control canonical form we have

$$\begin{pmatrix}\dot{x}_1\\ \dot{x}_2\end{pmatrix} = \begin{pmatrix}-2\zeta\omega_n & -\omega_n^2\\ 1 & 0\end{pmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix} + \begin{pmatrix}1\\ 0\end{pmatrix} u \tag{2.5}$$

$$y = \begin{pmatrix}0 & \omega_n^2\end{pmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix}. \tag{2.6}$$

2.1 Level-1 M-file S-function

The code listing for an m-file s-function that implements system (2.5) and (2.6) is shown below. Line 1 defines the main m-file function. The inputs to this function are always the elapsed time t, the state x, which is a concatenation of the continuous state and discrete state, the input u, a `flag`, followed by user defined input parameters, which in this case are ζ and ω_n. The Simulink engine calls the s-function and passes the parameters t, x, u, and `flag`. When `flag==0`, the Simulink engine expects the s-function to return the structure `sys` which defines the block, initial conditions `x0`, an empty string `str`, and an array `ts` that defines the sample times of the block. When `flag==1` the Simulink engine expect the s-function to return the function $f(t, x, u)$, when `flag==2` the Simulink engine expects the s-function to return $g(t, x, t)$, and when `flag==3` it expects the s-function to return $h(t, x, u)$. The switch statement that calls the proper functions based on the value of `flag` is shown in Lines 2–11. The block setup and the definition of the initial conditions is shown in Lines 13–27. The number of continuous states, discrete states, outputs, and inputs are defined in Lines 16-19, respectively. The direct feedthrough term on Line 20 is set to one if the output depends explicitly on the input u: for example, if $D \neq 0$ in the linear state space output equation $y = Cx + Du$. The initial conditions are defined on Line 24. The sample times are defined on Line 27. The format for this line is `ts = [period offset]`, where `period` defines the sample period, and is 0 for continuous time, or -1 for inherited, and where `offset` is the sample time offset, which is typically 0. The function $f(t, x, u)$ is defined in Lines 30–32, and the output function $h(t, x, u)$ is defined in Lines 35–36.

```
1   function [sys,x0,str,ts] = second_order_m(t,x,u,flag,zeta,wn)
2       switch flag,
3       case 0,
```

```
4             [sys,x0,str,ts]=mdlInitializeSizes;  % initialize block
5         case 1,
6             sys=mdlDerivatives(t,x,u,zeta,wn);  % define xdot = f(t,x,u)
7         case 3,
8             sys=mdlOutputs(t,x,u,wn);           % define xup = g(t,x,u)
9         otherwise,
10            sys = [];
11        end
12
13     %=============================================================================
14    function [sys,x0,str,ts]=mdlInitializeSizes
15        sizes = simsizes;
16        sizes.NumContStates  = 2;
17        sizes.NumDiscStates  = 0;
18        sizes.NumOutputs     = 1;
19        sizes.NumInputs      = 1;
20        sizes.DirFeedthrough = 0;
21        sizes.NumSampleTimes = 1;     % at least one sample time is needed
22        sys = simsizes(sizes);
23
24        x0  = [0; 0];  %  define initial conditions
25        str = [];  % str is always an empty matrix
26        % initialize the array of sample times
27        ts  = [0 0];      % continuous sample time
28
29     %=============================================================================
30    function xdot=mdlDerivatives(t,x,u,zeta,wn)
31        xdot(1) = -2*zeta*wn*x(1) - wn^2*x(2) + u;
32        xdot(2) = x(1);
33
34     %=============================================================================
35    function y=mdlOutputs(t,x,u,wn)
36        y = wn^2*x(2);
```

Lab 2: Assignment

In the next chapter we will see that the equations of motion of the whirlybird, if the pitch angle is set to $\theta = 0$ are given by the equations

$$\ddot{\phi} = \frac{\dot{\psi}^2 \sin\phi\cos\phi(J_y - J_z) + d\,(f_l - f_r)}{J_x} \tag{2.7}$$

$$\ddot{\psi} = \frac{-2\dot{\psi}\dot{\phi}\sin\phi\cos\phi(J_y - J_z) + \ell_1\,(f_l + f_r)\sin\phi}{\left(m_1\ell_1^2 + m_2\ell_2^2 + J_y\sin^2\phi + J_z\cos^2\phi\right)}. \tag{2.8}$$

While we will not use these equations for the simulation model, in this lab we will implement these equations and connect the equations of motion to the animation that you created in Lab 1, to give you practice in writing an s-function.

1. Rename your Simulation animation file `whirlysim.slx`, and create a new a matlab file called `param.m`.

2. Enter all of the parameters from Appendix A into the parameter file. A trick that can be used to enable all future parameters to be passed into the simulink model using only one argument, is to put all parameters in a structure. For example, I use the structure "P" to denote parameters. So `P.m1` is the first mass, etc. The feedback gain defined above can be labeled `P.K`.

3. Using Equation (2.7) and (2.8), and the equation

   $$\ddot{\theta} = 0,$$

   create a model of the whirlybird system using f_l and f_r as the inputs and $x = (\phi, \dot{\phi}, \theta, \dot{\theta}, \psi, \dot{\psi})^T$ as the state and the output. Set the initial conditions to $\phi = 20$ deg, $\dot{\phi} = 0$ deg/s, $\theta = 0$ deg, $\dot{\theta} = 0$ deg/s, $\psi = 30$ deg, and $\dot{\psi} = 0$ deg/s.

4. Connect the s-function to the animation block developed in last week's lab.

5. The simulation is difficult to maneuver using sliders for f_l and f_r. We can improve things slightly by converting to input sliders for force F and torque τ applied to the head of the whirlybird. Note that the

applied torque about the roll angle is $\tau = d(f_l - f_r)$ and the applied force is $F \triangleq f_l + f_r$. In matrix form you can write

$$\begin{pmatrix} F \\ \tau \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ d & -d \end{pmatrix} \begin{pmatrix} f_l \\ f_r \end{pmatrix}.$$

Inverting the matrix we get

$$\begin{pmatrix} f_l \\ f_r \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2d} \\ \frac{1}{2} & -\frac{1}{2d} \end{pmatrix} \begin{pmatrix} F \\ \tau \end{pmatrix}.$$

Therefore, given a desired F and τ, the required right and left forces are

$$f_l = \frac{1}{2}F + \frac{1}{2d}\tau$$
$$f_r = \frac{1}{2}F - \frac{1}{2d}\tau$$

Use the simulink matrix gain block to convert input sliders in F and τ into the system inputs f_l and f_r. The equilibrium force for the system will be $F = 1.73\ N$.

6. Clean up the Simulink diagram so that it looks nice.

Lab 2: Hints

1. Your file called `param.m` should be a Matlab script. Matlab scripts do not have input or output arguments. Run the script before starting your simulink simulation. Type `help script` at the matlab prompt or view the Matlab online documentation for more information about scripts. If the script changes, then you will need to re-run `param.m`. Another option is to click on the small wheel in the simulink diagram and select "Model Properties" as shown in Figure 2.1, and then select "Callbacks" and "InitFcn" as shown in Figure 2.2. Typing `param.m` in the window will cause the script to be executed each time the simulation is initialized, i.e., at the beginning of each simulation.

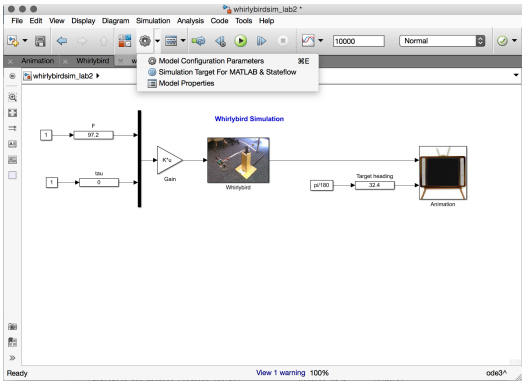


Figure 2.1: Select model properties, to auto run `param.m` at the beginning of each simulation.

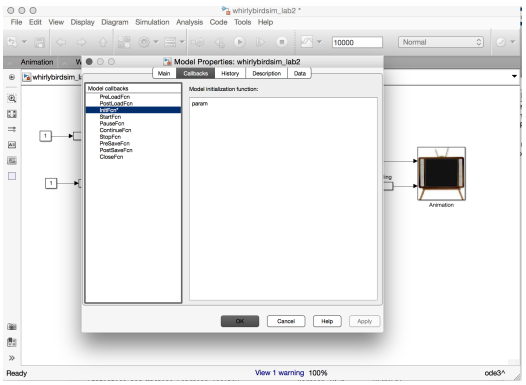


Figure 2.2: In the `InitFcn` callback, type `param`. This will cause simulink to execute the script `param.m` at the beginning of each simulation run.

2. Remember to check the inputs of your function that plots the whirlybird. The input vector u was 3×1 for lab 1 and now it will be 6×1: ϕ, θ, ψ, $\dot{\phi}$, $\dot{\theta}$, $\dot{\psi}$.

3. Inside your Simulink S-Function code you should be making changes to `mdlInitializeSizes`, `mdlDerivatives`, and `mdlOutputs`.

Lab 3

Whirlybird Simulation Model

The objective of this Chapter is to develop the simulation model for the whirlybird using the Euler-Lagrange equation

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) - \frac{\partial L}{\partial q} = Q. \tag{3.1}$$

The variable L is called the Lagrangian, the variable q is a vector of generalized coordinates, and the variable Q is a vector of generalized forces.

The physical parameters of the whirlybird are given in Appendix A, where m_1 is the mass of the whirlybird head (including the rotors, IMU, and camera), m_2 is the mass of the counterweight, ℓ_1 is the length of the rod from the pivot point to the center of mass of the whirlybird head, ℓ_2 is the length of the rod from the pivot point to the counter weight, d is the distance from the center of mass of the whirlybird head to the motor, and h is the height of the pivot point above ground. We will assume that the counter weight is a point mass and that the rods are massless.

The unit vectors $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ denote a right handed coordinate system centered at the pivot point and oriented so that $\mathbf{i}$ and $\mathbf{j}$ form a plane that is parallel to the ground, with $\mathbf{i}$ pointing along the rod and $\mathbf{j}$ pointing to the right when the pitch and yaw angles are both zero, and $\mathbf{k}$ pointing into the ground.

The yaw angle ψ is the positive rotation about $\mathbf{k}$, the pitch angle θ is the positive rotation about $\mathbf{j}$, and the roll angle ϕ is the positive rotation about $\mathbf{i}$. The forces exerted by the left and right motors are denoted as f_r and f_ℓ. A cartoon of the whirlybird is shown in Figure 3.1.

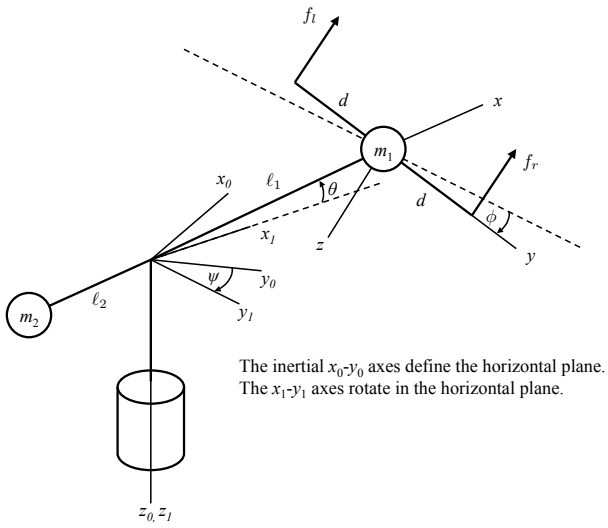


Figure 3.1: Whirlybird coordinate frames and nomenclature.

3.1 Kinetic and Potential Energy

The first step in the derivation of the equations of motion of the whirlybird is to find the total kinetic and potential energy of the system. The total kinetic energy of the system is given by

$$K = \sum_i \left(\frac{1}{2} m_i \mathbf{v}_i^T \mathbf{v}_i + \frac{1}{2} \boldsymbol{\omega}_i^T \mathbf{J}_i \boldsymbol{\omega}_i \right),$$

where the sum is over all objects in the system, m_i is the mass of object i, $\mathbf{v}_i$ is the velocity of the center of mass of object i, $\boldsymbol{\omega}_i$ is the angular velocity of object i, and $\mathbf{J}_i$ is the inertia matrix of the i^{th} object. Since the velocity and angular velocity of the base are zero, and since we are assuming massless rods, they do not contribute to the kinetic energy of the system. Therefore, only the whirlybird head and the counterweight contribute to the potential energy. Since the counterweight is assumed to have a point mass, its inertia matrix is zero. Therefore, the kinetic energy of the whirlybird is given by

$$K = \frac{1}{2} m_1 \mathbf{v}_1^T \mathbf{v}_1 + \frac{1}{2} \boldsymbol{\omega}_1^T \mathbf{J}_1 \boldsymbol{\omega}_1 + \frac{1}{2} m_2 \mathbf{v}_2^T \mathbf{v}_2, \tag{3.2}$$

where $\mathbf{v}_1$, $\boldsymbol{\omega}_1$, and $\mathbf{J}_1$ are the velocity of the center of mass, angular velocity, and inertia of the whirlybird head respectively, and $\mathbf{v}_2$ is the velocity of the counter weight.

In the inertial frame, the position of the center of mass of the whirlybird head is given by

$$p_1 = R_z(\psi)R_y(\theta)\begin{pmatrix}\ell_1\\0\\0\end{pmatrix} = \begin{pmatrix}c_\psi & -s_\psi & 0\\ s_\psi & c_\psi & 0\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}c_\theta & 0 & s_\theta\\ 0 & 1 & 0\\ -s_\theta & 0 & c_\theta\end{pmatrix}\begin{pmatrix}\ell_1\\0\\0\end{pmatrix} = \begin{pmatrix}\ell_1 c_\theta c_\psi\\ \ell_1 c_\theta s_\psi\\ -\ell_1 s_\theta\end{pmatrix},$$

where to make the expressions easier to read, we have used the notation $c_\phi \triangleq \cos\phi$ and $s_\phi \triangleq \sin\phi$. Similarly, the position of the counterweight is given by

$$p_2 = R_z(\psi)R_y(\theta)\begin{pmatrix}-\ell_2\\0\\0\end{pmatrix} = \begin{pmatrix}-\ell_2 c_\theta c_\psi\\ -\ell_2 c_\theta s_\psi\\ \ell_2 s_\theta\end{pmatrix}.$$

Differentiating with respect to time we get

$$\mathbf{v}_1 = \dot{\mathbf{p}}_1 = \begin{pmatrix}-\ell_1\dot\theta s_\theta c_\psi - \ell_1\dot\psi c_\theta s_\psi\\ -\ell_1\dot\theta s_\theta s_\psi + \ell_1\dot\psi c_\theta c_\psi\\ -\ell_1\dot\theta c_\theta\end{pmatrix}$$

$$\mathbf{v}_2 = \dot{\mathbf{p}}_2 = \begin{pmatrix}\ell_2\dot\theta s_\theta c_\psi + \ell_2\dot\psi c_\theta s_\psi\\ \ell_2\dot\theta s_\theta s_\psi - \ell_2\dot\psi c_\theta c_\psi\\ \ell_2\dot\theta c_\theta\end{pmatrix}.$$

The angular velocity of the whirlybird, expressed in the body frame, is

$$\boldsymbol{\omega} = \begin{pmatrix}p\\q\\r\end{pmatrix} = \begin{pmatrix}1 & 0 & -s_\theta\\ 0 & c_\phi & s_\phi c_\theta\\ 0 & -s_\phi & c_\phi c_\theta\end{pmatrix}\begin{pmatrix}\dot\phi\\\dot\theta\\\dot\psi\end{pmatrix} = \begin{pmatrix}\dot\phi - \dot\psi s_\theta\\ \dot\theta c_\phi + \dot\psi s_\phi c_\theta\\ -\dot\theta s_\phi + \dot\psi c_\phi c_\theta\end{pmatrix}.$$

The inertia matrix of the whirlybird head is given by

$$\mathbf{J} = \begin{pmatrix}J_x & 0 & 0\\ 0 & J_y & 0\\ 0 & 0 & J_z\end{pmatrix}.$$

Let $q = (\phi, \theta, \psi)^T$ and working out the expression for Equation (3.2) gives

$$K = \frac{1}{2}\dot q^T M(q)\dot q$$

where

$$M(q) \stackrel{\triangle}{=} \begin{pmatrix} J_x & 0 & -J_x s_\theta \\ 0 & m_1\ell_1^2 + m_2\ell_2^2 + J_y c_\phi^2 + J_z s_\phi^2 & (J_y - J_z) s_\phi c_\phi c_\theta \\ -J_x s_\theta & (J_y - J_z) s_\phi c_\phi c_\theta & (m_1\ell_1^2 + m_2\ell_2^2 + J_y s_\phi^2 + J_z c_\phi^2) c_\theta^2 + J_x s_\theta^2 \end{pmatrix}.$$

Since there are no springs in the system, the only source of potential energy is due to gravity. Therefore the potential energy is given by

$$P = m_1 g h_1 + m_2 g h_2 + P_0$$

where h_1 is the height of the center of gravity of the whirlybird head relative to zero pitch, h_2 is the height of the counterweight relative to zero pitch, and P_0 is the potential energy of the system at zero pitch. From geometry we have

$$P(q) = m_1 g \ell_1 s_\theta - m_2 g \ell_2 s_\theta + P_0.$$

3.2 Euler-Lagrange Equations

The Lagrangian is defined as

$$\begin{aligned} L &\stackrel{\triangle}{=} K(q, \dot{q}) - P(q) \\ &= \frac{1}{2} \dot{q}^T M(q) \dot{q} - P(q). \end{aligned}$$

The generalized forces are

$$Q = \begin{pmatrix} d(f_l - f_r) \\ \ell_1 (f_l + f_r) c_\phi \\ \ell_1 (f_l + f_r) c_\theta s_\phi + d(f_r - f_l) s_\theta \end{pmatrix}.$$

Taking the partial of L with respect to q gives

$$\frac{\partial L}{\partial q} = \frac{1}{2} \begin{pmatrix} \dot{q}^T \frac{\partial M}{\partial q_1} \\ \dot{q}^T \frac{\partial M}{\partial q_2} \\ \dot{q}^T \frac{\partial M}{\partial q_3} \end{pmatrix} \dot{q} - \frac{\partial P}{\partial q}.$$

Also

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}} = \frac{d}{dt} M(q) \dot{q} = M(q) \ddot{q} + \dot{M} \dot{q}.$$

Therefore, the Euler-Lagrange equation

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = Q$$

gives

$$M(q)\ddot{q} + \left[\dot{M} - \frac{1}{2}\begin{pmatrix} \dot{q}^T\frac{\partial M}{\partial q_1} \\ \dot{q}^T\frac{\partial M}{\partial q_2} \\ \dot{q}^T\frac{\partial M}{\partial q_3}\end{pmatrix}\right]\dot{q} + \frac{\partial P}{\partial q} = Q.$$

If we define

$$c(q,\dot{q}) \triangleq \dot{M}\dot{q} - \frac{1}{2}\begin{pmatrix} \dot{q}^T\frac{\partial M}{\partial q_1} \\ \dot{q}^T\frac{\partial M}{\partial q_2} \\ \dot{q}^T\frac{\partial M}{\partial q_3}\end{pmatrix}\dot{q},$$

then the equations of motion are given by

$$M(q)\ddot{q} + c(q,\dot{q}) + \frac{\partial P}{\partial q} = Q.$$

Working through the math, we get

$$c(q,\dot{q}) = \begin{pmatrix} -\dot{\theta}^2(J_z - J_y)s_\phi c_\phi + \dot{\psi}^2(J_z - J_y)s_\phi c_\phi c_\theta^2 \\ -\dot{\theta}\dot{\psi}c_\theta\left[J_x - (J_z - J_y)(c_\phi^2 - s_\phi^2)\right] \\ - - - \\ \dot{\psi}^2 s_\theta c_\theta\left[-J_x + m_1\ell_1^2 + m_2\ell_2^2 + J_y s_\phi^2 + J_z c_\phi^2\right] \\ -2\dot{\phi}\dot{\theta}(J_z - J_y)s_\phi c_\phi - \dot{\phi}\dot{\psi}c_\theta\left[-J_x + (J_z - J_y)(c_\phi^2 - s_\phi^2)\right] \\ - - - \\ \dot{\theta}^2(J_z - J_y)s_\phi c_\phi s_\theta - \dot{\phi}\dot{\theta}c_\theta\left[J_x + (J_z - J_y)(c_\phi^2 - s_\phi^2)\right] \\ -2\dot{\phi}\dot{\psi}(J_z - J_y)c_\theta^2 s_\phi c_\phi + 2\dot{\theta}\dot{\psi}s_\theta c_\theta\left[J_x - m_1\ell_1^2 - m_2\ell_2^2 - J_y s_\phi^2 - J_z c_\phi^2\right] \end{pmatrix},$$

and

$$\frac{\partial P}{\partial q} = \begin{pmatrix} 0 \\ (m_1\ell_1 - m_2\ell_2)gc_\theta \\ 0 \end{pmatrix}.$$

3.2.1 Summary

The equations of motion are

$$M(q)\ddot{q} + c(q,\dot{q}) + \frac{\partial P}{\partial q} = Q, \tag{3.3}$$

where

$$M(q) = \begin{pmatrix} J_x & 0 & -J_x s_\theta \\ 0 & m_1\ell_1^2 + m_2\ell_2^2 + J_y c_\phi^2 + J_z s_\phi^2 & (J_y - J_z) s_\phi c_\phi c_\theta \\ -J_x s_\theta & (J_y - J_z) s_\phi c_\phi c_\theta & (m_1\ell_1^2 + m_2\ell_2^2 + J_y s_\phi^2 + J_z c_\phi^2) c_\theta^2 + J_x s_\theta^2 \end{pmatrix},$$

$$c(q,\dot{q}) = \begin{pmatrix} -\dot{\theta}^2 (J_z - J_y) s_\phi c_\phi + \dot{\psi}^2 (J_z - J_y) s_\phi c_\phi c_\theta^2 \\ -\dot{\theta}\dot{\psi} c_\theta \left[J_x - (J_z - J_y)(c_\phi^2 - s_\phi^2) \right] \\ --- \\ \dot{\psi}^2 s_\theta c_\theta \left[-J_x + m_1\ell_1^2 + m_2\ell_2^2 + J_y s_\phi^2 + J_z c_\phi^2 \right] \\ -2\dot{\phi}\dot{\theta}(J_z - J_y) s_\phi c_\phi - \dot{\phi}\dot{\psi} c_\theta \left[-J_x + (J_z - J_y)(c_\phi^2 - s_\phi^2) \right] \\ --- \\ \dot{\theta}^2 (J_z - J_y) s_\phi c_\phi s_\theta - \dot{\phi}\dot{\theta} c_\theta \left[J_x + (J_z - J_y)(c_\phi^2 - s_\phi^2) \right] \\ -2\dot{\phi}\dot{\psi}(J_z - J_y) c_\theta^2 s_\phi c_\phi + 2\dot{\theta}\dot{\psi} s_\theta c_\theta \left[J_x - m_1\ell_1^2 - m_2\ell_2^2 - J_y s_\phi^2 - J_z c_\phi^2 \right] \end{pmatrix},$$

$$\frac{\partial P}{\partial q} = \begin{pmatrix} 0 \\ (m_1\ell_1 - m_2\ell_2) g c_\theta \\ 0 \end{pmatrix},$$

and

$$Q = \begin{pmatrix} d(f_l - f_r) \\ \ell_1 (f_l + f_r) c_\phi \\ \ell_1 (f_l + f_r) c_\theta s_\phi + d(f_r - f_l) s_\theta \end{pmatrix}.$$

3.3 Model of the Motor-Propeller

In the previous sections, we modeled the motors as producing a force perpendicular to the whirlybird x-y plane. The input to the motors is actually a pulse-width-modulation (PWM) command that regulates the duty cycle of the current supplied to the motor. The pulse width modulation command is constrained to be in the range $[0, 100]$, where 0 represents zero duty cycle, or zero current supplied to the motor, and 100 represents full current supplied to the motor.

While there are dynamics in the internal workings of the motor, we will neglect these dynamics and model the relationship between PWM command u and force as

$$F = k_m u.$$

To experimentally find the value of k_m we apply an equal PWM signal to each motor and increase this signal until the force balances the whirlybird

at constant pitch angle. In the equilibrium position the forces cancel and we have

$$\ell_1 F = m_1 \ell_1 g - m_2 \ell_2 g.$$

Setting $F = f_l + f_r = k_m(u_l + u_r)$, and solving for k_m we get

$$k_m = \frac{m_1 \ell_1 g - m_2 \ell_2 g}{\ell_1 (u_l + u_r)}.$$

While PWM is the input to the system, it is easier to think in terms of torque and force applied to the whirlybird. The relationship between PWM commands and force and torque are given by

$$\begin{pmatrix} \tau \\ F \end{pmatrix} = k_m \begin{pmatrix} d & -d \\ 1 & 1 \end{pmatrix} \begin{pmatrix} u_l \\ u_r \end{pmatrix}. \tag{3.4}$$

Inverting the matrix and solving for PWM command gives

$$u_l = \frac{1}{2k_m}\left(F + \frac{\tau}{d}\right) \tag{3.5}$$

$$u_r = \frac{1}{2k_m}\left(F - \frac{\tau}{d}\right). \tag{3.6}$$

Lab 3: Assignment

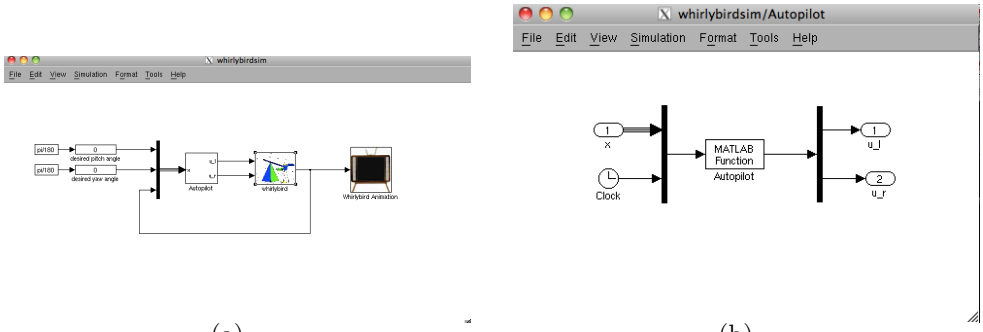
1. (Optional) Carefully work the derivation of the equations of motion for the whirlybird by hand to verify that they are correct. Please send corrections to the instructor.

2. Modify the s-function from Lab 2 to implement the full equations of motion for the whirlybird. Let the state be given by $x = (\phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi})^T$, and let the input be given by $u = (u_l, u_r)^T$.

3. Add your animation from Lab 1.

4. In hardware, the inputs to the motors will be pulse width modulation (PWM) comments, which we will denote by u_l, and u_r. The possible PWM signals are constrained to be between 0 and 100. Add if statements to the s-function that ensures that u_l and u_r are constrained between 0 and 100.

5. Add a matlab m-file called `autopilot.m` that will be used as the control block for the system. The output of the autopilot will be u_l and u_r. The inputs to the autopilot should be pitch command, yaw command, and the full state of the system. See Figure 3.2. This autopilot block provides the framework to apply closed-loop feedback to our system. We are not quite ready for that yet. For the present, we will apply open-loop force and torque commands from within the autopilot block using Equations (3.5)–(3.6). Apply a torque of zero, and the right amount of force to cause the system to remain stationary when the initial conditions are $\phi(0) = 0$, $\theta(0) = 0$, and $\psi(0) = 0$.



(a) (b)

Figure 3.2: (a) Simulink model for the whirlybird system. (b) The control system is implemented in `autopilot.m` where the input is the system state x, and the current time t.

Lab 4

Whirlybird Design Models

The equations of motion developed in the previous chapter are too complex to be useful for control design. Therefore the objective of this chapter is to develop several different design models for the whirlybird that capture certain behaviors of the system. The essential idea is to notice that since the total force on the head of the whirlybird is $F = f_l + f_r$, the pitching motion can be controlled by manipulating F. Also, since the torque on the head of the whirlybird is $\tau = (f_l - f_r) * d$, the rolling motion can be controlled by manipulating τ. We also note that the yawing motion is driven primarily by the roll angle ϕ.

Accordingly, we define the *longitudinal* motion of the whirlybird, to be the motion involving the state variables $(\theta, \dot{\theta})$, and the *lateral* motion of the whirlybird to be the motion involving the state variables $(\phi, \psi, \dot{\phi}, \dot{\psi})$. While these motions are coupled, the coupling is weak and will be neglected in the design models. Design models for the longitudinal motion, assuming that $\phi = \dot{\phi} = \dot{\psi} = 0$ are derived in Section 4.1. Design models for the lateral motion, assuming that $\theta = \dot{\theta} = 0$ are derived in Section 4.2.

4.1 Design Models for the Longitudinal Dynamics

4.1.1 Nonlinear Model for Longitudinal Dynamics

In this section we will derive a nonlinear design model for the pitch, or longitudinal dynamics. The governing equation for θ is the second line of

Equation (3.3) which is

$$\begin{aligned}(m_1\ell_1^2 + m_2\ell_2^2 + J_y c_\phi^2 + J_z s_\phi^2)\ddot{\theta} + (J_y - J_z)s_\phi c_\phi c_\theta \ddot{\psi} \\ + \dot{\psi}^2 s_\theta c_\theta \left[-J_x + m_1\ell_1^2 + m_2\ell_2^2 + J_y s_\phi^2 + J_z c_\phi^2\right] \\ - 2\dot{\phi}\dot{\theta}(J_z - J_y)s_\phi c_\phi - \dot{\phi}\dot{\psi}c_\theta \left[-J_x + (J_z - J_y)(c_\phi^2 - s_\phi^2)\right] \\ + (m_1\ell_1 - m_2\ell_2)g c_\theta = \ell_1(f_l + f_r)c_\phi. \quad (4.1)\end{aligned}$$

Setting $\phi = \dot{\phi} = \ddot{\psi} = \dot{\psi} = 0$ gives

$$(m_1\ell_1^2 + m_2\ell_2^2 + J_y)\ddot{\theta} + (m_1\ell_1 - m_2\ell_2)g c_\theta = \ell_1(f_l + f_r).$$

Defining $F \triangleq f_l + f_r$ and solving for $\ddot{\theta}$ gives

$$\ddot{\theta} = \left(\frac{(m_2\ell_2 - m_1\ell_1)g}{m_1\ell_1^2 + m_2\ell_2^2 + J_y}\right)\cos\theta + \left(\frac{\ell_1}{m_1\ell_1^2 + m_2\ell_2^2 + J_y}\right)F, \quad (4.2)$$

which is the nonlinear design model for the longitudinal dynamics. In state-variable form we have

$$\dot{x}_{lon} = f_{lon}(x_{lon}, u_{lon}) \triangleq \begin{pmatrix} x_{lon}(2) \\ \left(\frac{(m_2\ell_2 - m_1\ell_1)g}{m_1\ell_1^2 + m_2\ell_2^2 + J_y}\right)\cos(x_{lon}(1)) + \left(\frac{\ell_1}{m_1\ell_1^2 + m_2\ell_2^2 + J_y}\right)u_{lon} \end{pmatrix}, \quad (4.3)$$

where $x_{lon} = (\theta, \dot{\theta})^T$ and $u_{lon} = F$.

4.1.2 Linear State Space Model for Longitudinal Dynamics

The objective is to linearize the equations of motion around $\theta = \theta_e$, and the equilibrium force F_e Convince yourself that $\theta_e = 0$, $F_e = 0$ is not an equilibrium of the system. The first lab assignment is to find F_e so that θ_e is an equilibrium of the system. Defining the deviated state from equilibrium as

$$\tilde{x}_{lon} \triangleq (\theta, \dot{\theta})^T - (\theta_e, 0)^T$$

and the deviated control input from equilibrium as

$$\tilde{u}_{lon} = F - F_e,$$

then the linearization of Equation eq:(4.3) about the equilibrium is given by

$$\dot{\tilde{x}}_{lon} = A_{lon}\tilde{x}_{lon} + B_{lon}\tilde{u}_{lon},$$

where

$$A_{lon} = \frac{\partial f_{lon}}{\partial x_{lon}}(x_{lon,e}, F_e) = \begin{pmatrix} 0 & 1 \\ \frac{(m_1\ell_1 - m_2\ell_2)g\sin(\theta_e)}{m_1\ell_1^2 + m_2\ell_2^2 + J_y} & 0 \end{pmatrix}, \tag{4.4}$$

$$B_{lon} = \frac{\partial f_{lon}}{\partial u_{lon}}(x_{lon,e}, F_e) = \begin{pmatrix} 0 \\ \frac{\ell_1}{m_1\ell_1^2 + m_2\ell_2^2 + J_y} \end{pmatrix}. \tag{4.5}$$

If the output is $y_{lon} = \theta - \theta_e$, then the output equation is

$$y_{lon} = C_{lon}\tilde{x}_{lon},$$

where $C_{lon} = (1, 0)$.

Note that if we design a controller law $\tilde{u}_{lon}(\tilde{x}_{lon})$, then the commanded force will be

$$F = F_e + \tilde{u}_{lon}.$$

4.1.3 Linear Transfer Function Model for Longitudinal Dynamics

The transfer function from $\tilde{u}_{lon}$ to y_{lon} is given by

$$y_{lon}(s) = C_{lon}(sI - A_{lon})^{-1}B_{lon}\tilde{u}_{lon}(s) = \frac{\left(\frac{\ell_1}{m_1\ell_1^2 + m_2\ell_2^2 + J_y}\right)}{s^2 - \left(\frac{(m_1\ell_1 - m_2\ell_2)g\sin(\theta_e)}{m_1\ell_1^2 + m_2\ell_2^2 + J_y}\right)}\tilde{u}_{lon}(s).$$

If the desired equilibrium angle is $\theta_e = 0$, then the transfer function simplifies to

$$y_{lon}(s) = \frac{\left(\frac{\ell_1}{m_1\ell_1^2 + m_2\ell_2^2 + J_y}\right)}{s^2}\tilde{u}_{lon}(s). \tag{4.6}$$

4.2 Design Models for the Lateral Dynamics

4.2.1 Nonlinear Model for Lateral Dynamics

In this section we will derive a nonlinear design model for the side-to-side, or lateral dynamics. The governing equations for ϕ and ψ are the first and third lines of Equation (3.3) which, after setting $\theta = \dot{\theta} = \ddot{\theta} = 0$ are

$$J_x\ddot{\phi} + \dot{\psi}^2(J_y - J_z)s_\phi c_\phi = d(f_l - f_r)$$
$$(m_1\ell_1^2 + m_2\ell_2^2 + J_y s_\phi^2 + J_z c_\phi^2)\ddot{\psi} - 2\dot{\psi}\dot{\phi}(J_z - J_y)s_\phi c_\phi = \ell_1(f_l + f_r)s_\phi.$$

Solving for $\ddot{\phi}$ and $\ddot{\psi}$ gives

$$\ddot{\phi} = \frac{-\dot{\psi}^2(J_y - J_z)s_\phi c_\phi + d(f_l - f_r)}{J_x}$$
$$\ddot{\psi} = \frac{2\dot{\psi}\dot{\phi}(J_z - J_y)s_\phi c_\phi + \ell_1(f_l + f_r)s_\phi}{m_1\ell_1^2 + m_2\ell_2^2 + J_y s_\phi^2 + J_z c_\phi^2}.$$

To decouple the motion from the longitudinal states, we will assume that $f_l + f_r = F_e$, the equilibrium force required to maintain $\theta = \theta_e$. Defining the torque

$$\tau \triangleq d(f_l - f_r)$$

gives

$$\ddot{\phi} = \frac{-\dot{\psi}^2(J_y - J_z)s_\phi c_\phi + \tau}{J_x} \tag{4.7}$$
$$\ddot{\psi} = \frac{2\dot{\psi}\dot{\phi}(J_z - J_y)s_\phi c_\phi + \ell_1 F_e s_\phi}{m_1\ell_1^2 + m_2\ell_2^2 + J_y s_\phi^2 + J_z c_\phi^2}, \tag{4.8}$$

which are the nonlinear design models for the lateral dynamics. In state-variable form, we have

$$\dot{x}_{lat} = f_{lat}(x_{lat}, u_{lat}) \triangleq \begin{pmatrix} x_{lat}(3) \\ x_{lat}(4) \\ \frac{-x_{lat}(4)^2(J_y - J_z)\sin(x_{lat}(1))\cos(x_{lat}(1)) + u_{lat}}{J_x} \\ \frac{2x_{lat}(4)x_{lat}(3)(J_z - J_y)\sin(x_{lat}(1))\cos(x_{lat}(1)) + \ell_1 F_e \sin(x_{lat}(1))}{m_1\ell_1^2 + m_2\ell_2^2 + J_y\sin(x_{lat}(1))^2 + J_z\cos(x_{lat}(1))^2} \end{pmatrix},$$

where $x_{lat} = (\phi, \psi, \dot{\phi}, \dot{\psi})^T$, and $u_{lat} = \tau$. Note that f_{lat} is independent of the yaw angle ψ.

4.2.2 Linear State Space Model for Lateral Dynamics

Note that for the lateral dynamics, the equilibrium is given by $x_{lat,e} = (0, 0, 0, 0)$ and $u_{lat,e} = 0$. Therefore, to a first approximation, the lateral

dynamics are given by

$$\dot{x}_{lat} = A_{lat}x_{lat} + B_{lat}u_{lat}, \tag{4.9}$$

where

$$A_{lat} \triangleq \frac{\partial f_{lat}}{\partial x_{lat}}(x_{lat,e}, u_{lat,e}) = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ \frac{\ell_1 F_e}{m_1\ell_1^2+m_2\ell_2^2+J_z} & 0 & 0 & 0 \end{pmatrix} \tag{4.10}$$

$$B_{lat} \triangleq \frac{\partial f_{lat}}{\partial u_{lat}}(x_{lat,e}, u_{lat,e}) = \begin{pmatrix} 0 \\ 0 \\ \frac{1}{J_x} \\ 0 \end{pmatrix}. \tag{4.11}$$

Since both ϕ and ψ can be measured by the encoders, the output equation is two dimensional and is given by

$$y_{lat} = C_{lat}x_{lat}, \tag{4.12}$$

where

$$C_{lat} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

4.2.3 Linear Transfer Function Model for Lateral Dynamics

Taking the Laplace transform of the first two rows of Equation (4.9) gives

$$\phi(s) = \frac{(1/J_x)}{s^2}\tau(s). \tag{4.13}$$

Taking the Laplace transform of the second two rows of Equation (4.9) gives

$$\psi(s) = \frac{\left(\frac{\ell_1 F_e}{m_1\ell_1^2+m_2\ell_2^2+J_z}\right)}{s^2}\phi(s). \tag{4.14}$$

The structure of Equations (4.13) and (4.14) indicate that the linear equations for the lateral dynamics form a cascade structure as shown in Figure 4.1. As shown in Figure 4.1 the torque τ can be though of as the input to the roll

dynamics given by Equation (4.13). The roll angle ϕ which is the output of the roll dynamics can be thought of as the input to the yaw dynamics given by Equation (4.14). This cascade structure will be exploited in Chapter **??** and **??** where we will design control strategies for the roll dynamics and yaw dynamics separately and enforce suitable bandwidth separation to ensure a good design.

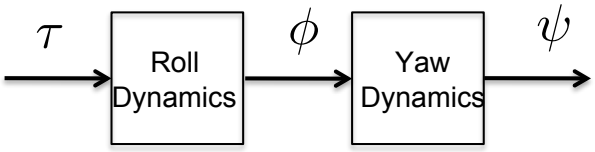


Figure 4.1: The lateral dynamics for the whirlybird can be thought of as a cascade system, where the roll angle drives the yaw dynamics, and the torque drives the roll dynamics.

Lab 4: Assignment

1. For the longitudinal state-variable equation given by Equation (4.3), find an equation for the equilibrium force F_e so that $x_{lon,e} = (\theta_e, 0)^T$ is an equilibrium of the system.

2. Derive the expressions for A_{lon} and B_{lon} given in Equations (4.4) and (4.5).

3. The lateral state space equations in Equation (4.9) and (4.12) are given in the original state x_{lat} and not a deviated state $\tilde{x}_{lat}$. Explain why this is appropriate for the lateral dynamics and not appropriate for the longitudinal dynamics.

4. Derive the expressions for A_{lat} and B_{lat} given in Equations (4.10) and (4.11).

5. Why does the cascade structure shown in Figure 4.1 makes sense physically.

Lab 5

PD Control of Longitudinal Dynamics

The objective of this lab is to implement PD control for the longitudinal motion of the whirlybird in Simulink. The block diagram for the longitudinal controller is shown in Figure 5.1, where from Equation (4.6)

$$\begin{aligned} b_0 &= \frac{\ell_1}{m_1\ell_1^2 + m_2\ell_2^2 + J_y} \\ &= \frac{0.85 \text{ m}}{(0.891 \text{ kg})(0.85 \text{ m})^2 - (1 \text{ kg})(0.3048 \text{ m})^2 + 0.0014 \text{ kg-m}^2} \\ &= 1.152 \text{ (kg-m}^{-1}\text{)}. \end{aligned}$$

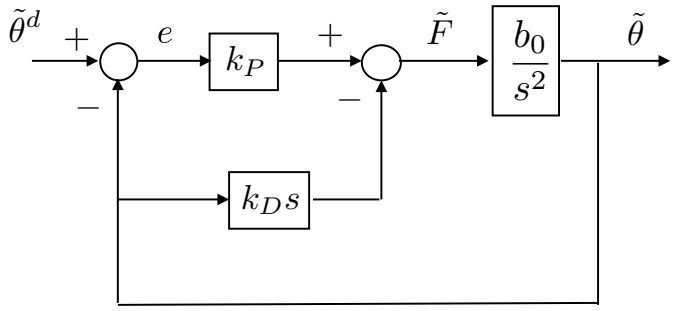


Figure 5.1: PD control design example.

The equilibrium force can be calculated by performing a moment balance

as

$$F_e = (m_1\ell_1 - m_2\ell_2)\cos\theta\frac{g}{\ell_1}.$$

An alternative method for linearizing the dynamics is called *feedback linearization*. From Equation (4.2) the equations of motion for the longitudinal dynamics are given by

$$\begin{aligned}\ddot{\theta} &= \left(\frac{(m_2\ell_2 - m_1\ell_1)g}{m_1\ell_1^2 + m_2\ell_2^2 + J_y}\right)\cos\theta + \left(\frac{\ell_1}{m_1\ell_1^2 + m_2\ell_2^2 + J_y}\right)F, \\ &= \left(\frac{\ell_1}{m_1\ell_1^2 + m_2\ell_2^2 + J_y}\right)\left(F + \frac{(m_2\ell_2 - m_1\ell_1)g}{\ell_1}\cos\theta\right). \end{aligned} \tag{5.1}$$

Therefore, we can cancel the nonlinearity due to $\cos\theta$ by canceling it with a force of

$$F = -\frac{(m_2\ell_2 - m_1\ell_1)g}{\ell_1}\cos\theta + F_c, \tag{5.2}$$

where F_c is a control term to be selected using a PD controller. Plugging Equation (5.2) into (5.1) gives

$$\ddot{\theta}\left(\frac{\ell_1}{m_1\ell_1^2 + m_2\ell_2^2 + J_y}\right)F_c = b_0F_c,$$

where the transfer function is again

$$\Theta(s) = \frac{b_0}{s^2}F_c(s).$$

Lab 5: Assignment

1. Find the transfer function from $\tilde{\Theta}^d$ to $\tilde{\Theta}$ of the closed loop system shown in Figure 5.1.

2. Find the characteristic polynomial $\Delta_{cl}(s)$ of the closed loop system.

3. If the desired closed loop characteristic polynomial is $\Delta_{cl}^d(s) = s^2 + 2\zeta\omega_n s + \omega_n^2$, find formulas for k_P and k_D as a function of ω_n and ζ.

4. Modify your simulink diagram from Lab 3 to implement a PD controller for pitch attitude control. Use the Simulink differentiation blocks for the differentiator. Recall that $\tilde{F} = F - F_e$ and therefore $F = F_e + \tilde{F}$ which implies that the equilibrium force needs to be added to the output of the PD controller.

5. Put the formulas for k_P and k_D in a parameter file and tune the controller based on ω_n and ζ when the input is a square wave with amplitude ± 15 degrees and a frequency of 0.02 Hz.

Lab 5: Hints

- The Simulink diagram should look something like Figure 5.2.

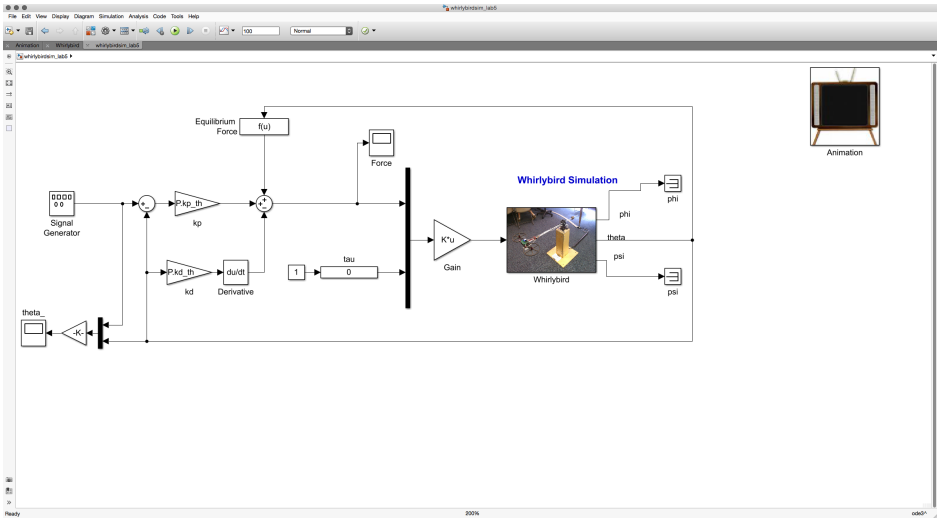


Figure 5.2: PD control for longitudinal dynamics. Feedback linearization is used to cancel the nonlinearity gravity term.

Lab 6

Successive Loop Closure and Limits of Performance

The objective of this lab assignment is to modify the PD controller in the last lab so that the gains are selected to maximize the performance of the longitudinal control loop. We will also use the principle of successive loop closure to design the lateral control loop.

Let u_r and u_ℓ be the PWM commands on the right and left motors. The PWM commands are limited by

$$\begin{aligned} 0 \leq & u_r \leq 100 \\ 0 \leq & u\ell \leq 100. \end{aligned}$$

Recall also from Equation (3.4) that

$$\begin{aligned} F &= k_m u_\ell + k_m u_r \\ \tau &= k_m d u_\ell - k_m d u_r. \end{aligned}$$

If $\tilde{F} = F - F_e$, then

$$\begin{aligned} \left|\tilde{F}\right| &\leq |F| - |F_e| \\ &\leq 2k_m |u_\ell| - |F_e| \\ &\leq 200 k_m - |F_e| \end{aligned}$$

Similarly, a limit on the torque can be derived by noting that the torque is limited by the amount of force each motor must exert to overcome gravity.

Therefore the maximum available force on each motor is $200k_m - F_e$, and the maximum possible torque that can be applied to the whirlybird head during level flight is

$$|\tau| \leq d(200k_m - F_e)$$

Lab 6: Assignment

1. Add constraint blocks to the Simulink diagram that limit u_r and u_ℓ to be between 0 and 100.

2. Using the constraints on force and torque developed in the discussion above, and the physical parameters provided in Appendix A, find proportional and derivative gains for the longitudinal controller to minimize the rise time for a step on θ^d of $A_\theta = 30$ degrees, and so that the damping ratio is $\zeta_\theta = 0.707$. What is the associated rise time? Implement these gains in Simulink and use A_θ and ζ_θ to tune the response.

3. Using the principle of successive loop closure, draw the block diagram for the inner loop of the lateral controller and find the associated closed loop transfer function and closed loop characteristic polynomial. Use the constraint on torque to find the proportional and derivative gains that minimize the rise time for a step on ϕ^d of A_ϕ and that provide a damping ratio of ζ_ϕ.

4. Find the DC-gain of the inner loop and draw the block diagram of the outer loop controlling the yaw angle ψ, where the inner loop has been replaced by its DC-gain. Find the proportional derivative gains for the outer loop that minimize the rise time for a step on ψ^d of $A_\psi = 50$ degrees, and a damping ratio of ζ_ψ.

5. Implement this controller in Simulink and use A_ϕ, ζ_ϕ, and ζ_ψ to tune the response.

Lab 6: Hints:

- Instead of using a constant value for F_e in the control, implement a nonlinear feedforward control input based on Equation (4.1) using the commanded values for θ and ϕ. By setting the $\ddot{\theta}$, $\ddot{\psi}$, $\dot{\psi}$, and $\dot{\phi}$ terms

to zero, the condition for static equilibrium is established. This can be used to calculate the feedforward command.

Lab 7

PID Implementation - Software

The purpose of this lab is to convert the PID design to Matlab m-file implementation that is ready to be converted to hardware implementation. This lab will use the same PID gains used in Lab 6.

7.1 Integrator Gain Selection using Root Locus

Our goal with the longitudinal dynamics is to give a commanded pitch angle and have the whirlybird respond accordingly. This is accomplished by varying the force F exerted by the rotors. The use of an integrator for the longitudinal dynamics is helpful because the equilibrium force is imprecisely known, introducing a steady-state error. The integrator will eliminate this steady state error. The block diagram in Figure 7.1 shows the general form of the closed-loop longitudinal dynamics when PID control is implemented.

The transfer function for this closed-loop system is given by

$$H_{\theta/\theta^c}(s) = \frac{a_{lon}k_{P_\theta}s + a_{lon}k_{I_\theta}}{s^3 + a_{lon}k_{D_\theta}s^2 + a_{lon}k_{P_\theta}s + a_{lon}k_{I_\theta}}. \tag{7.1}$$

where

$$a_{lon} = \frac{l_1}{m_1 l_1^2 + m_2 l_2^2 + J_y}.$$

In Chapter 6, we developed rise-time specifications for the longitudinal dynamics of the whirlybird utilizing the closed loop transfer function of the system without an integrator. In this lab, we will use the values of k_{p_θ} and k_{d_θ}

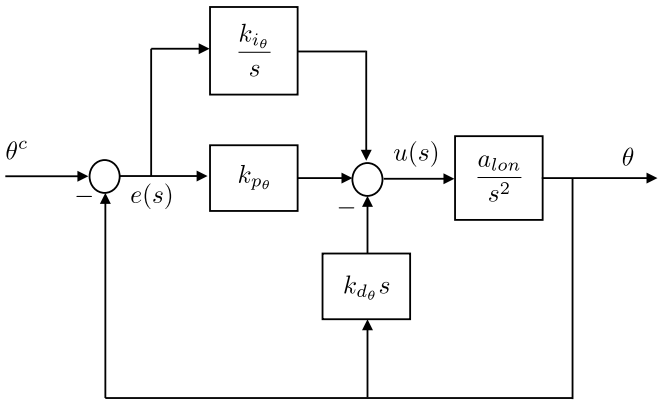


Figure 7.1: PID Diagram for Longitudinal Dynamics

we found to determine an integral gain, k_{i_θ}, that will reject steady-state error without introducing other adverse effects such as large overshoot, increased oscillation, or instability. We will use root locus methods to pick k_{i_θ}.

From Equation (7.1), we see that the closed-loop characteristic equation is given by

$$s^3 + a_{lon}k_{D_\theta}s^2 + a_{lon}k_{P_\theta}s + a_{lon}k_{I_\theta} = 0,$$

which can be placed in Evans form as

$$1 + k_{I_\theta}\left(\frac{a_{lon}}{s\left(s^2 + a_{lon}k_{D_\theta}s + a_{lon}k_{P_\theta}\right)}\right) = 0.$$

After substituting the values for a_{lon}, k_{p_θ}, and k_{d_θ}, the gain k_{i_θ} can be selected by looking at the root locus of this closed-loop system. Figure 7.2 shows an example of the root locus of the characteristic equation plotted as a function of k_{i_θ}.

A similar process can be used to find the integral gain on the outer loop of the lateral controller, where the transfer function from ψ^c to ψ is given by

$$H_{\psi/\psi^c}(s) = \frac{a_{lat}k_{P_\psi}s + a_{lat}k_{I_\psi}}{s^3 + a_{lat}k_{D_\psi}s^2 + a_{lat}k_{P_\psi}s + a_{lat}k_{I_\psi}} \tag{7.2}$$

where

$$a_{lat} = \frac{l_1 F_e}{m_1 l_1^2 + m_2 l_2^2 + J_z}.$$

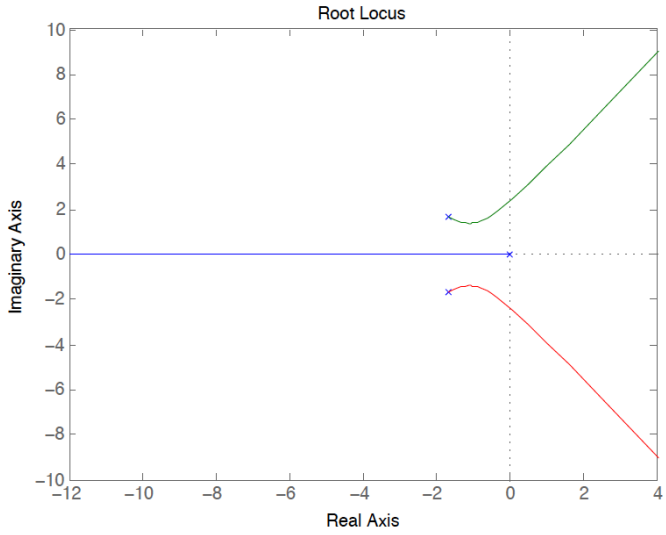


Figure 7.2: Pitch loop root locus as a function of the integral gain k_{i_θ}.

This characteristic equation can be placed in Evans form so that we can generate a root locus plot in MATLAB to determine an appropriate value for k_{i_ψ}.

7.2 Software Implementation in Matlab

As described in Appendix B, the hardware implementation of the whirlybird controller will be accomplished using Labview's Mathscript node. The Mathscript node can implement a small subset of Matlab commands, which do not include function calls or any specialized commands like eig or place. In addition, the Mathscript node cannot implement persistent variables. A Labview diagram of the Mathscript node is shown in Figure 7.3 The inputs to the node are shown on the right of Figure 7.3 and include `Roll`, `Pitch`, `Yaw`, as well as nine gains `Gain1`–`Gain9`, and four variables `Variable1`–`Variable4` which act as persistent variables. The gains will be set from a window in Labview and can be changed during the hardware run.

A Simulink file that emulates the setup in Hardware is contained at whirlybirdLabviewTemplate.zip.

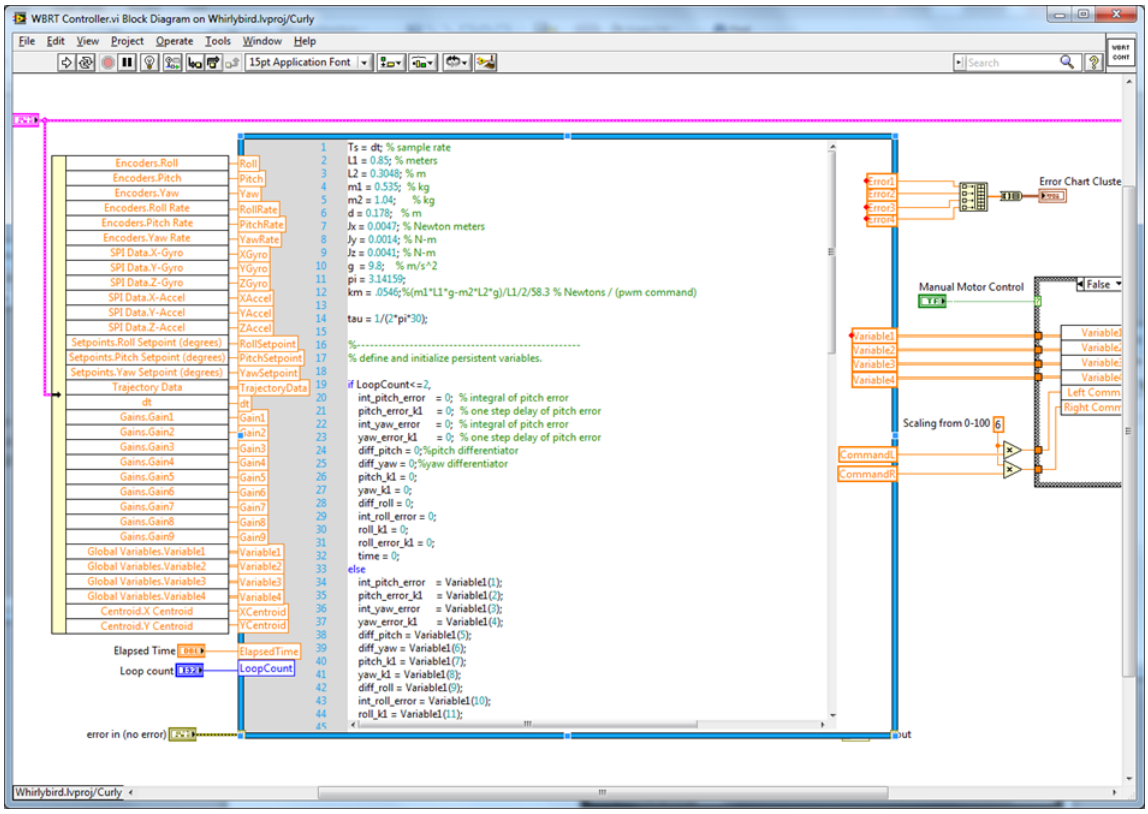


Figure 7.3: WBRT Controller.vi

Lab 7: Assignment

1. Select the gains k_{I_θ} and k_{I_ψ} using root locus techniques as described in the controlbook.

2. Using the Simulink diagram and code provided at:

   whirlybirdLabviewTemplate.zip,

   modify `whirlybird_ctrl.m` to implement the lateral and longitudinal controllers for the whirlybird. Use the gains selected in Lab 6.

Appendix A

Whirlybird Parameters

The whirlybird in the lab has the following parameters:

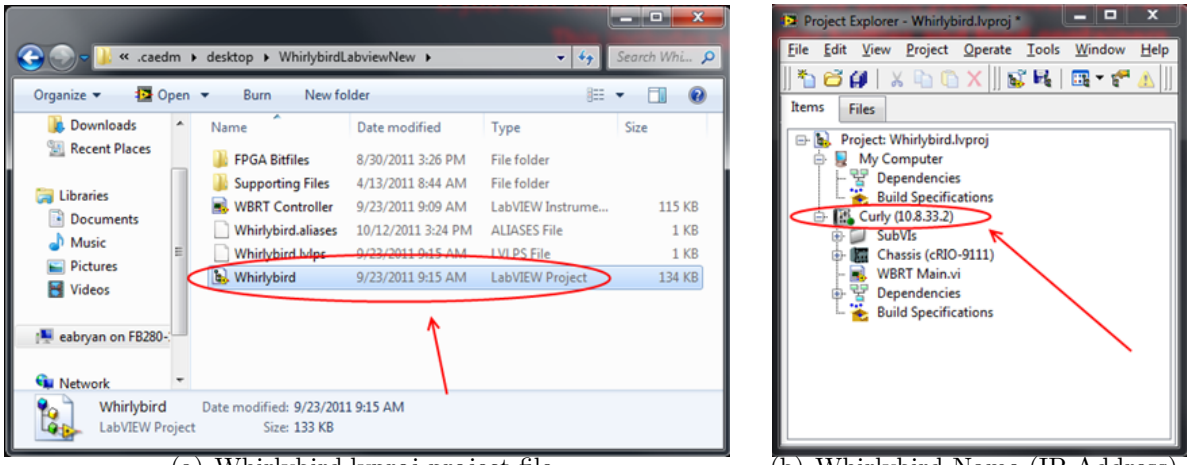
g	9.81	m/s^2
ℓ_1	0.85	m
ℓ_2	0.3048	m
m_1	0.891	kg
m_2	1	kg
d	0.178	m
h	0.65	m
r	0.12	m
J_x	0.0047	kg-m^2
J_y	0.0014	kg-m^2
J_z	0.0041	kg-m^2
k_m	0.0546	N/PWM
σ_{gyro}	8.7266×10^{-5}	rad
σ_{pixel}	0.05	pixel

Appendix B

Using the Hardware

B.1 Getting Started

1. Download the whirlybird project zip file from the class Wiki page. Save on your J: drive. Unzip it.



(a) Whirlybird.lvproj project file (b) Whirlybird Name (IP Address)

Figure B.1: Open LabVIEW project file

2. Locate the LabVIEW project file **Whirlybird.lvproj**. Double-click and it will open in the LabVIEW Project Explorer. Figure B.1(a).

3. To ensure reliable operation of the whirlybird control code within LabVIEW, it is best to recompile the whirlybird virtual instrument. This

takes about 15 minutes. (NOTE: We hope to eliminate the need to recompile soon – check your email for updates!) Here are the steps:

(a) Close WBRT Main.vi Front Panel if it is open.

(b) In Project Explorer under the whirlybird name (Larry, Curly, or Moe), expand Chassis→FPGA Target→Build Specifications. Right click on Whirlybird FPGA Main and select Rebuild.

(c) Choose the local server to recompile the project. Labview will then automatically begin generating 5 intermediate steps. After these steps are complete, a Compilation Status window will appear to finish the rebuild. Be patient – this takes 10 to 15 minutes. If you are asked if you want to save a file, it is recommended that you save them.

(d) After the compilation completes, continue with the steps below.

4. Verify that the name and IP address of the whirlybird you intend to control is listed in the LabVIEW project tree. Figure B.2(b).

5. If the name and IP address are not correct, right-click on the name, select properties, and then change the 'Name' and 'IP Address/DNS Name' entries. See figures B.2(a) and B.2(b)

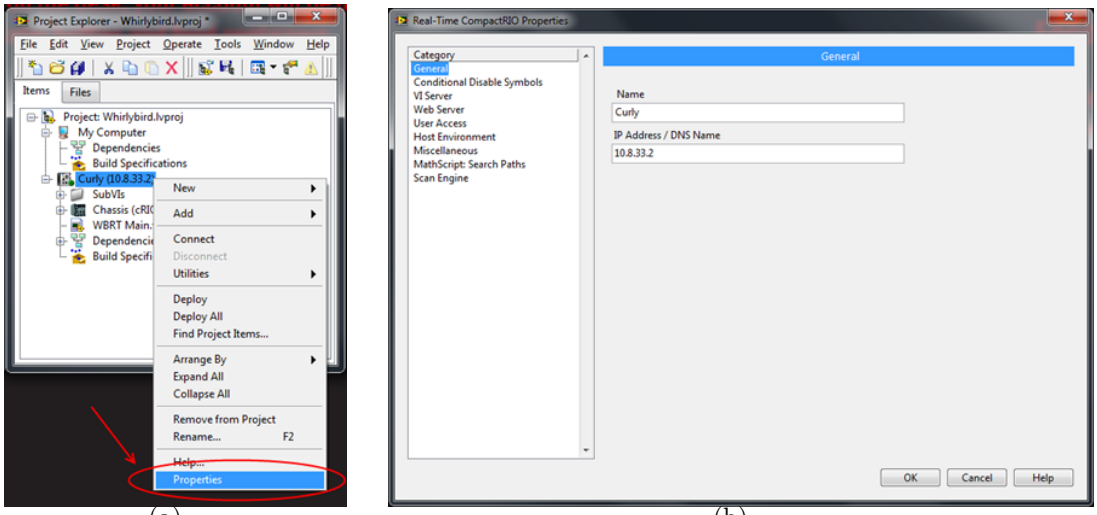
6. Verify that the toggle safety switch cover is up and the switch is closed. Figure B.3.

7. Turn on the whirlybird power supply. Figure B.4.

8. Connect LabVIEW to the whirlybird by right clicking on the name of the whirlybird and selecting "Connect". Figure B.5.

   - If LabVIEW is unable to connect to the whirlybird, check to see that the whirlybird is connected to power and the network cable is plugged in.
   - Be sure the safety switch is closed when the whirlybird power supply is turned on.

9. Once the whirlybird is connected to LabVIEW, return to the project tree and open 'WBRT Main.vi'. Figure B.6



(a) (b)

Figure B.2: Change whirlybird IP address and name

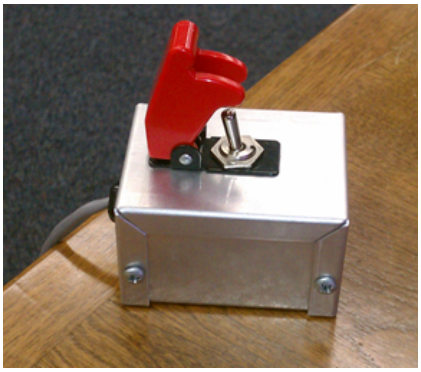


Figure B.3: Safety switch

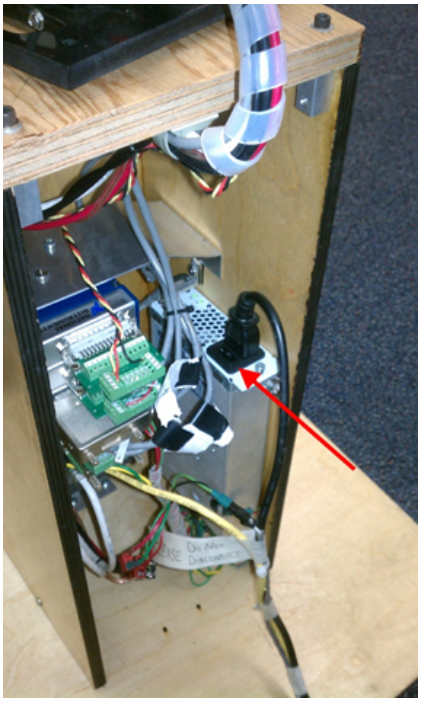


Figure B.4: Power supply switch

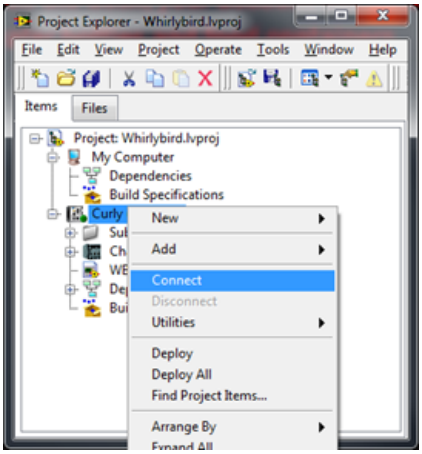


Figure B.5: Connect to whirlybird

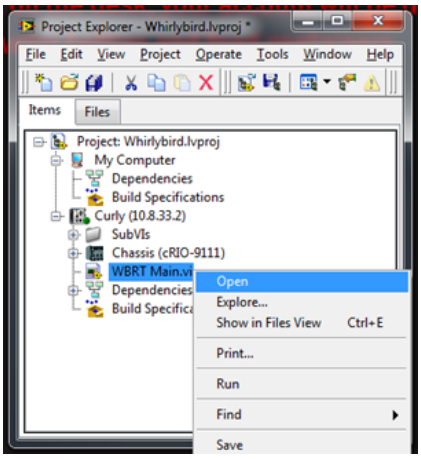


Figure B.6: Open 'WBRT Main.vi'

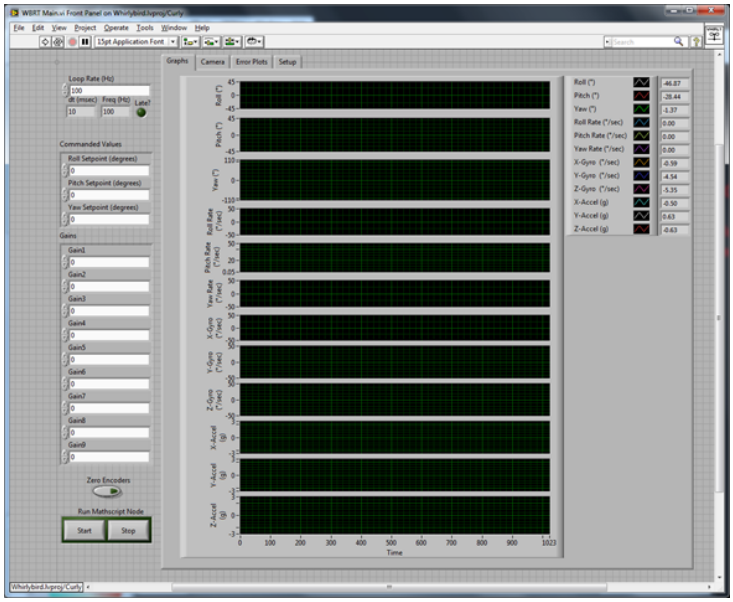


Figure B.7: WBRT Main.vi

10. Wait for the 'Main' window to open. Figure B.7

11. To start the hardware, click on the arrow in the upper left corner of the 'Main' window (Figure B.8). If you are asked to save any files, click yes.

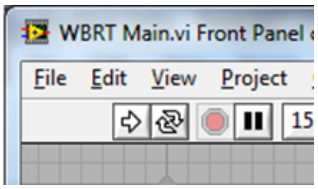


Figure B.8: Click the arrow to begin

12. Once the software is running you will see that the center of the Main.vi window contains state plots that are now plotting in real time. The encoders need to be set to zero. Prop the head up on a chair making sure that the pitch, roll and yaw angles are all at the zero point, as seen in Figure B.9. Zero the encoders by clicking on the button labeled "Zero Encoders" (Figure B.12).

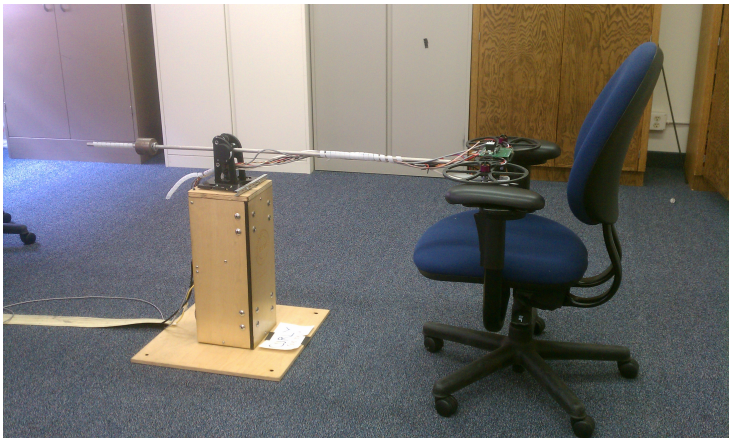


Figure B.9: Setting encoders equal to zero.

13. Now that the encoders are set to zero, remove the chair. It is helpful to set the field labeled "Pitch Setpoint (degrees)" to -20 degrees, and all other Setpoints to zero degrees.

14. Make sure that the roll angle is zero before starting the whirlybird and that the gains are correctly entered.. Click on the Start button to start the whirlybird.

B.2 WBRT Main.vi Front Panel

The WBRT Main.vi Front Panel can be divided into three parts. The left most part contains the run-time variable fields and buttons to start the Mathscript Code and zero the encoders (Figures B.10, B.11, & B.12). Figure B.10 shows the fields where the gains can be entered and adjusted while the whirlybird is flying.

From the Main.vi Front Panel the user may adjust the loop rate. The loop rate can be changed before or during the simulation. The default Loop Rate is 100Hz (Figure B.11). The center of the Main.vi Front Panel is filled with scrolling plots of the whirlybird states, gyro, and accel measurements.

The right side of the Main.vi Front Panel has the state, gyro, and accel instantaneous values (Figure B.13).

B.3 Turn off / Emergency Shut Down

The whirlybird has built-in "power up" and "power down" motor behaviors to reduce the possibility of damaging the whirlybird when turning it on or off. Unless there is risk of bodily injury or damage of equipment, follow the steps in section B.3.1.

B.3.1 Turn Off

To turn off the whirlybird:

1. Click on the "Stop" button (Figure B.12) and wait for the whirlybird motors to stop.

2. Once they have stopped, disconnect from the whirlybird by right clicking on the name of the whirlybird in the project tree and selecting "Disconnect".

3. Turn off the power supply.

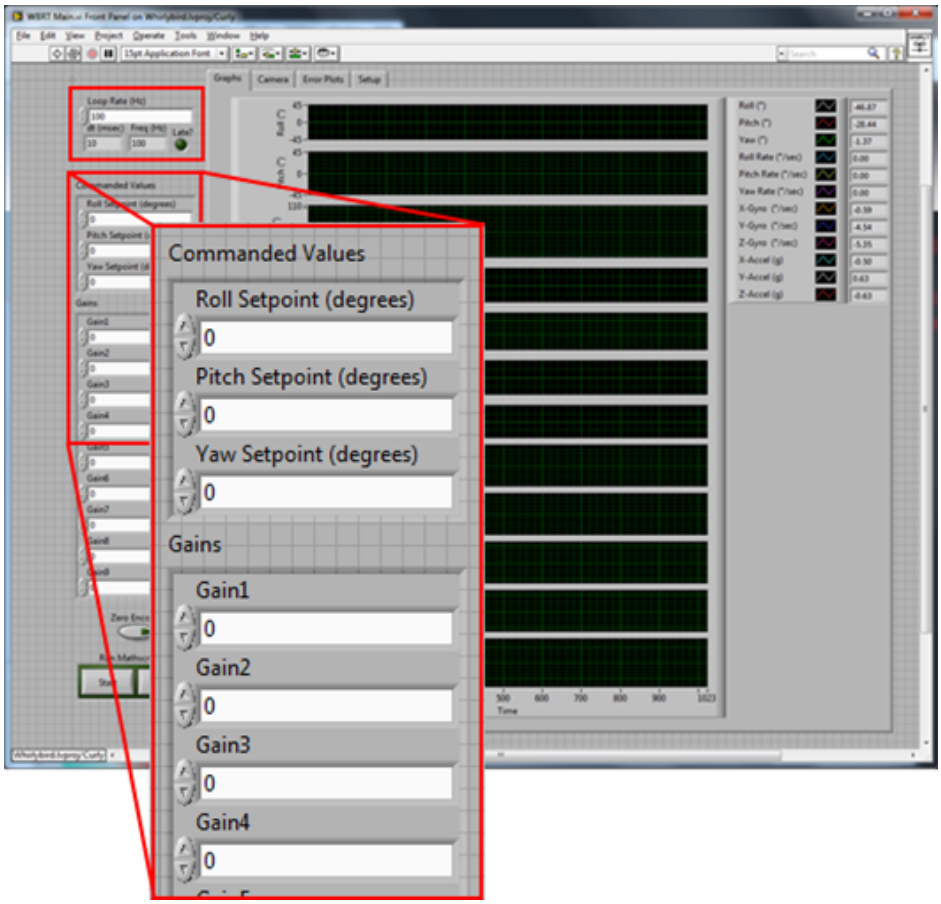

Commanded Values
Roll Setpoint (degrees)
0
Pitch Setpoint (degrees)
0
Yaw Setpoint (degrees)
0
Gains
Gain1
0
Gain2
0
Gain3
0
Gain4
0

Figure B.10:

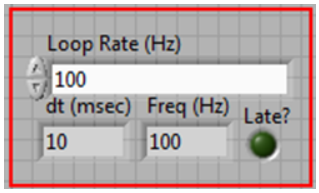


Figure B.11: Loop Rate of Controller.

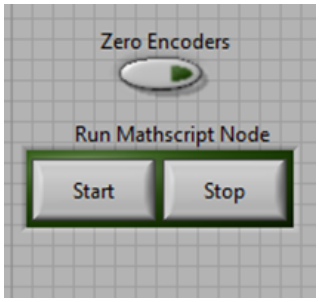


Figure B.12: Zero Encoders & Start/Stop Buttons

B.3.2 Emergency Shut Down

There are various reasons why you may need to turn the whirlybird immediately off, possible bodily injury or damage of equipment to name two. To turn off the whirlybird immediately, close the red cover over the toggle safety switch. The motors will immediately stop spinning and the whirlybird will disconnect.

When the emergency switch is used to shut the whirlybird down, there is a short procedure to follow to reconnect and begin again. To restart the whirlybird after an emergency shutdown,

1. Open the safety switch cover and close the switch.

2. Cycle the power supply by turning it off and waiting for the internal fan to stop spinning, then turning it back on.

3. Reconnect LabVIEW to the whirlybird. Go to the Project Explorer and right-click on the name of the whirlybird and select “Connect”.

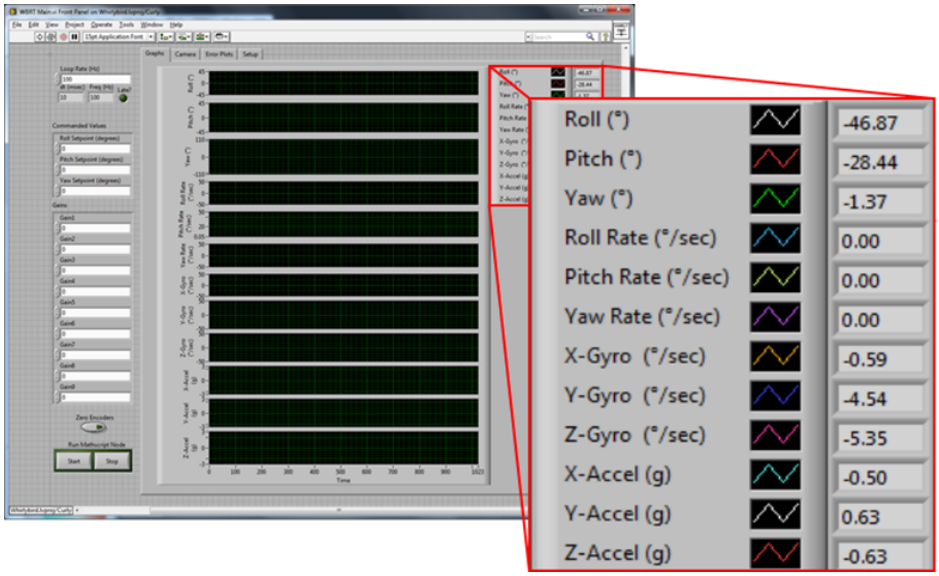


Figure B.13: Main.vi Front Panel: Whirlybird States

B.4 WBRT Controller.vi Block Diagram

B.4.1 Open Controller

1. To open the controller, double click on 'WBRT Controller.vi' in the Whirlybird LabVIEW directory (Figure B.14).

2. The controller front panel will open in LabVIEW (Figure B.15).

3. Press 'Control + E' to open the block diagram. The block diagram should look something like figure 7.3.

B.4.2 Controller Block Diagram explanation

Inside the block diagram (Figure 7.3) there is a blue outlined text box where the control code is entered. The programming language is MathScript and is very similar to MATLAB. Add your control code into the Controller.vi Block Diagram at the location indicated in the script file.

You will notice that the input variables and the output variables are on the left and right side of the text box, respectively.

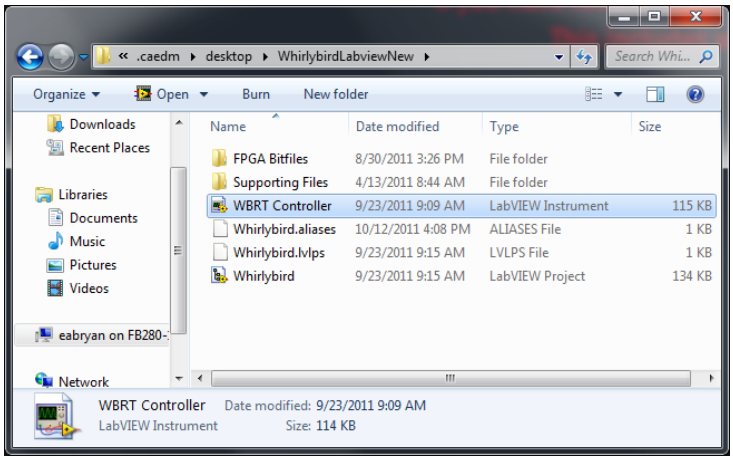


Figure B.14:

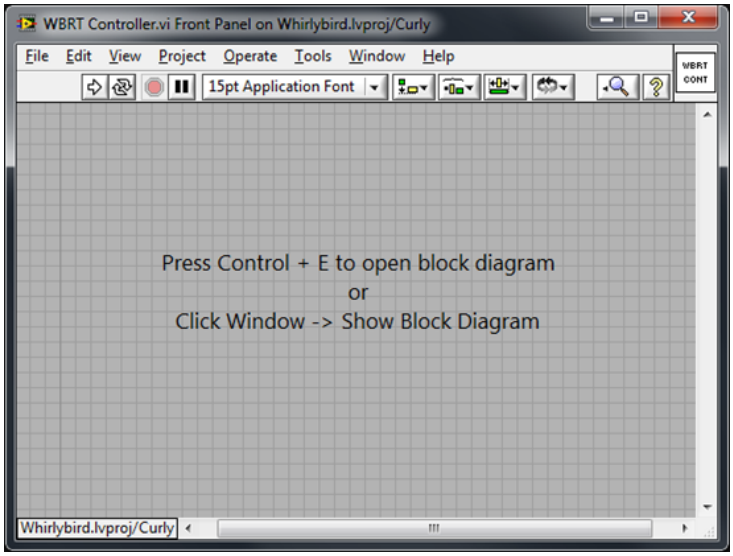


Figure B.15:

- The first 6 variables on the left-hand side of the script window describe the states of the whirlybird in units of radians and radians/sec.
- The next 6 variables describe the whirlybird state information as obtained from the onboard inertial measurement unit (IMU).
- The variables labeled **RollSetpoint**, **PitchSetpoint**, and **YawSetpoint** are to be used as the roll, pitch, and yaw commanded values. You can set these up to be in radians or degrees. These values are read in from the 'Main.vi' window.
- **TrajectoryData** is meant to be a vector of desired attitude values to be read in from a text file. We are not currently using this variable.
- **dt** is set in the Main.vi by changing the 'Loop Rate (Hz)' variable and is in units of milliseconds.
- **ElapsedTime** is the time since your program has been running in units of milliseconds
- **Gain1** through **Gain9** are for your own use. These values, like **dt**, are read in from the 'Main.vi' window and can be changed while the whirlybird is running.
- **Variable1** through **Variable4** are for your own use. These variables behave like persistent variables and will retain their values between calls.
- **CommandL** and **CommandR** are the PWM outputs to the motors, these need to be in the range of 0-100.