The Extended Mars Lander
Exercise 2: Control

A.M.P. Tombs
arthur.tombs@cantab.net

September 18, 2013

1 Introduction

This document is an extension to the Mars Lander exercise in Part I of the Engineering Tripos, which covered numerical simulation of three-dimensional positional dynamics in both Octave/Matlab and C++, using a Verlet integration scheme. An autopilot was subsequently developed as part of the linear systems course in Part IB, to make the lander hover in a stable way. Familiarity with the Part I exercises is assumed, and is highly recommended, but is not essential for the completion of this extension.

The first document of this extended exercise ('Exercise 1: Dynamics') introduces techniques for the simulation of rotational motion, which are implemented in the Lander program. Working though the Dynamics exercise is not a prerequisite for this exercise, but is encouraged. The increase in complexity brings with it a series of challenges for successful automatic control, which is the focus of this exercise. Here is an overview of the topics covered:

- A brief recap of relevant control material (from Part IB).
- An introduction to state-space system representations (from course 3F2).
- Tasks for the student to complete, involving automatic control of the lander.

2 Linear Systems and 1D Control

At its most general level, Control Theory concerns itself with the behaviour of *dynamical systems*, usually described by differential equations relating inputs to outputs. By restricting analysis to certain classes of systems (e.g. Linear Time-Invariant), we can specify some systematic approaches for analysis and control.

The *Transfer Function* relating two variables can be found by taking the *Laplace Transforms* of their respective time-domain equations. The resulting expression is a representation of how the output $y(t)$ varies in response to an arbitrary input $x(t)$:

$$G(s) = \frac{\mathcal{L}\{y(t)\}}{\mathcal{L}\{x(t)\}} = \frac{Y(s)}{X(s)} \implies y(t) = \mathcal{L}^{-1}\{G(s) \times \mathcal{L}\{x(t)\}\} \tag{1}$$

The Laplace-domain function $G(s)$ contains information about the behaviour of the system, which can be analysed: frequency dependence can be found by evaluating $G(j\omega)$, and asymptotic behaviour (what happens to the output after a long time) is obtained by finding the *poles* (values of s at which $G(s)$ becomes infinite).

- If $\text{Re}(s) > 0$, the output magnitude will grow unbounded $\implies$ unstable.
- If $\text{Re}(s) = 0$, the output magnitude will remain constant $\implies$ marginally stable.
- If $\text{Re}(s) < 0$, the output magnitude will reduce to zero $\implies$ asymptotically stable.
  - As $\text{Re}(s)$ becomes more negative, the rate at which the output reduces will increase.
- As $|\text{Im}(s)|$ increases, the output becomes more oscillatory.

The aim of a control system is that a) an unstable system can be stabilized, and/or b) given a desired frequency-response. This is done by employing *feedback* – setting the input to some function of the system output e.g. $x(t) = -ky(t)$. The resulting system is described as *closed-loop.*

3 State-space representation

State-space representation is one approach for modelling systems with multiple inputs and outputs, thus allowing multidimensional control. The key assumption made with this type of model is that the system can be described at any point in time by a finite number of *state variables*, which completely determine the dynamical behaviour of the system in the future (i.e. the system has no memory of past states, only of current states).

The key equations that define a linear, state-space system model relate a vector of inputs $\underline{u}$, states $\underline{x}$, and outputs $\underline{y}$ by the following matrix equations:

$$\begin{aligned} \dot{\underline{x}} &= A\underline{x} + B\underline{u} \\ \underline{y} &= C\underline{x} + D\underline{u} \end{aligned} \tag{2}$$

The first equation (in $\underline{x}$) describes how the system state evolves over time, relating inputs to the rate of change of the internal system states. The second equation represents how the system states are output to the 'world', and so allows modelling of a system that is being observed experimentally (i.e. without perfect knowledge of the system state).

It should be noted that there may be many different sets of state variables that can be chosen to describe a system, as there are usually more measurable variables (or derived quantities) than there are degrees of freedom. It is often useful to select variables that result in simpler differential equations, or the smallest possible matrix dimensions.

3.1 Linearizing equations

In order for a non-linear system to be represented in this standard form, it must first be *linearized* about some *equilibrium state*. For control purposes, it is important that the system behaviour is well approximated by the linear representation – by considering arbitrarily small variations of state variables about constant values, this is generally true (think back to Taylor expansions of functions to explain this). One common class of non-linear system may be described by equations of the following form:

$$\dot{x}_i = f_i(\underline{x}, \underline{u}) \tag{3}$$

Elements of the matrices A and B are then found by taking partial derivatives of this function, evaluated about some equilibrium $\underline{x}_e, \underline{u}_e$ (which satisfies $A\underline{x}_e + B\underline{u}_e = \underline{0}$):

$$\begin{aligned} A_{ij} &= \left.\frac{\partial f_i}{\partial x_j}\right|_{\underline{x}=\underline{x}_e,\ \underline{u}=\underline{u}_e} \\ B_{ij} &= \left.\frac{\partial f_i}{\partial u_j}\right|_{\underline{x}=\underline{x}_e,\ \underline{u}=\underline{u}_e} \end{aligned} \tag{4}$$

3.2 Open-loop behaviour

For an open-loop system (i.e. $\underline{u} = \underline{0}$ for all time), the asymptotic behaviour can be analysed by finding *eigenvalues* and *eigenvectors* of the matrix A. These eigenvalues are related to the transfer-function poles used in 1D control, and determine the asymptotic behaviour of the open-loop system in the same way. You should recall that eigenvalues are found by solving the equation $\det(A - \lambda I) = 0$.

3.3 State feedback

State feedback is implemented by setting the system inputs $\underline{u}$ to a linear sum of the state vector $\underline{x}$ (implicitly assuming that all state variables are available to the controller) and some external reference signal $\underline{r}$, thus making a closed-loop system:

$$\begin{aligned} \text{let } \underline{u} &= -K\underline{x} + M\underline{r} \\ \implies \dot{\underline{x}} &= (A - BK)\underline{x} + BM\underline{r} \end{aligned} \tag{5}$$

If the state variables are chosen so that they should be controlled to zero, the reference signal may be removed.

As in the 1D case, the asymptotic behaviour of the closed-loop system can be analysed in terms of the eigenvalues of $A - BK$. The elements of the gain matrix K can be designed to give desired characteristics – there are standard methods for doing this (including the *root-locus* method), both analytically and numerically.

3.4 Controllability

A system is described as *controllable* if it is possible, by some combination of inputs, to move the system between any two arbitrary states, within a finite time period. Although not proved here, this property can be tested for by constructing a *controllability matrix*:

$$P = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix} \tag{6}$$

where n is the number of state variables. If P has full rank (i.e. all rows are independent), the system is controllable. Otherwise, there exist some combinations of state variables that cannot be reached by the system, no matter what inputs are given. Given that all obtainable states lie within the column space of P, the subspace of states that are unobtainable can be found as the left null space of P (solutions of $P^T\underline{x} = \underline{0}$).

Similar to controllability is the concept of *observability*, which applies when only the system outputs $\underline{y}$ are available for feedback (i.e. the values of $\underline{x}$ cannot be used directly). As we are considering full state-feedback, this will not be explored further.

4 The Lander

As with all dynamical systems, in order to develop a control system for the lander module, we must begin with the equations of motion. The first of these equations describes changes in position $\underline{r}$ of the lander in response to forces (gravitational attraction $\underline{F}_G$, thrust $\underline{F}_T$, and aerodynamical drag $\underline{F}_D$). These values are expressed relative to some *global* coordinate system.

$$\begin{aligned} m\ddot{\underline{r}} &= \sum \underline{F} \\ &= \frac{-GMm}{|\underline{r}|^2}\hat{\underline{r}} + \underline{F}_T + \underline{F}_D \end{aligned} \tag{7}$$

The second equation describes the rotational behaviour of the lander in response to torques $\underline{Q}$. This is a form of *Euler's equations* for rigid-body motion, where the values are expressed in the lander's *local* reference frame.

$$\dot{\underline{\omega}} = I^{-1}(I\underline{\omega} \times \underline{\omega} + \underline{Q}_T + \underline{Q}_D) \tag{8}$$

The vector $\underline{\omega}$ is an *angular velocity*, representing an axis about which the lander is instantaneously rotating. I is the inertia matrix for the lander, calculated about its centre of mass.

4.1 Altitude control

In the Part IB course, the lander was analysed in terms of its altitude as a 1D control problem. Considering this same case of purely vertical motion (and assuming just one, vertical thruster), we can re-state the control problem in terms of a state-space representation. First we define a variable z to be the distance of the lander from the center of the planet:

$$\text{let } z = |\underline{r}| \tag{9}$$

In this vertical state, the thruster force $\underline{F}_T$ is given by the maximum vertical thrust T_v scaled by a throttle t. These expressions can be substituted into Equation (7) to give the governing differential equation of the system:

$$\ddot{z} = \frac{-GM}{z^2} + \frac{T_v t}{m} + \frac{\underline{F}_D}{m} \tag{10}$$

As this equation involves the second time-derivative of z, we must choose $\dot{z}$ as one of our state variables in order to represent it in the state-space form – this is a standard approach to dealing with higher-order derivatives.

$$\underline{x} = \begin{bmatrix} z \\ \dot{z} \end{bmatrix} \implies \dot{\underline{x}} = \begin{bmatrix} \dot{z} \\ \ddot{z} \end{bmatrix} \tag{11}$$

The input to this system is simply the throttle scalar t:

$$\underline{u} = \begin{bmatrix} t \end{bmatrix} \tag{12}$$

Now, we select an equilibrium state $\underline{x} = \underline{x}_e$ about which to linearize the equations of motion.

$$\text{equilibrium} \implies \dot{\underline{x}} = \underline{0} \implies \dot{z} = \ddot{z} = 0 \tag{13}$$

This allows us to calculate the throttle value required to maintain equilibrium (noting that there is no drag when the lander is stationary):

$$\ddot{z}|_{z=z_e} = \frac{-GM}{z_e^2} + \frac{T_v t_e}{m} + \cancelto{0}{\frac{\underline{F}_D}{m}} = 0 \implies t_e = \frac{GMm}{T_v z_e^2} \tag{14}$$

From here on, our equations will be in terms of the error between the desired and actual states $\delta\underline{x} = \underline{x}_e - \underline{x}$. Calculating partial derivatives of Equation (10) to obtain elements of the A and B matrices:

$$\begin{aligned} \left.\frac{\partial \dot{z}}{\partial z}\right|_{\underline{x}_e,\underline{u}_e} &= 0 & \left.\frac{\partial \dot{z}}{\partial \dot{z}}\right|_{\underline{x}_e,\underline{u}_e} &= 1 \\ \left.\frac{\partial \ddot{z}}{\partial z}\right|_{\underline{x}_e,\underline{u}_e} &= \frac{2GM}{z_e^3} + \frac{1}{m}\frac{\partial \underline{F}_D}{\partial z} & \left.\frac{\partial \ddot{z}}{\partial \dot{z}}\right|_{\underline{x}_e,\underline{u}_e} &= \frac{1}{m}\frac{\partial \underline{F}_D}{\partial \dot{z}} \\ \left.\frac{\partial \dot{z}}{\partial t}\right|_{\underline{x}_e,\underline{u}_e} &= 0 & \left.\frac{\partial \ddot{z}}{\partial t}\right|_{\underline{x}_e,\underline{u}_e} &= \frac{T_v}{m} \end{aligned} \tag{15}$$

Note that when calculating partial derivatives for one variable, all other variables are held constant. As $\dot{z}$ only varies indirectly with z (through Equation (10)), the first element of A is zero. To a good approximation, both the gravitational acceleration and aerodynamical drag can be assumed to be constant with respect to changes in altitude. Assembling these values into matrices gives the following state-space realization:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ \frac{T_v}{m} \end{bmatrix} \tag{16}$$

The matrices C and D are, for this example, chosen to output the system state without modification – this is used to indicate that we are able to measure both the lander's altitude and climb-rate.

$$C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad D = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{17}$$

We can now apply state feedback (by setting the throttle input $\underline{u}$ to a linear combination of the states $\underline{x}$), and design a controller for holding a constant altitude:

$$\begin{aligned} \underline{u} &= -K\delta\underline{x} \\ &= -\begin{bmatrix} k_1 & k_2 \end{bmatrix} \begin{bmatrix} \delta x_1 \\ \delta x_2 \end{bmatrix} \end{aligned} \tag{18}$$

The system now behaves as an open-loop system with a modified A matrix:

$$\begin{aligned} \dot{\delta\underline{x}} &= (A - BK)\delta\underline{x} \\ &= \begin{bmatrix} 0 & 1 \\ \frac{-T_v k_1}{m} & \frac{-T_v k_2}{m} \end{bmatrix} \delta\underline{x} \end{aligned} \tag{19}$$

We can now investigate stability of the closed-loop system by finding eigenvalues of this matrix in terms of the two gain constants k_1 and k_2:

$$\lambda = \frac{T_v}{2m}\left(-k_2 \pm \sqrt{k_2^2 - \frac{4mk_1}{T_v}}\right) \tag{20}$$

Examining the case where $k_2 = 0$ (i.e. purely proportional control) it is simple to show that there is no value of k_1 that will produce an asymptotically stable system:

$$\lambda = \begin{cases} \pm\sqrt{\frac{-T_v k_1}{m}} & k_1 < 0 \implies \text{unstable} \\ \pm j\sqrt{\frac{T_v k_1}{m}} & k_1 \geq 0 \implies \text{marginally stable} \end{cases} \tag{21}$$

Augmenting the state vector with $\dot{z}$ has had the added benefit of allowing derivative action in the controller, which we have now shown is necessary for asymptotic stability.

In order to design for particular closed-loop behaviour (characterised by the values of λ), we could attempt to solve for K by hand, but in general it is easier to use software to design the controller for us. In Octave/Matlab[1], the command `K = place(A, B, P)` will calculate a gain matrix `K` that achieves the desired closed-loop poles given in `P`.

1. In The Lander program, run scenario 7 to observe the case of proportional-only control (i.e. $k_2 = 0$). The graphing view (selected from the main menu) is useful to see how the altitude changes over time. Is the variation perfectly sinusoidal? Why/why not? Plotting the climb rate may help to understand what is going on here.

2. The file `Exercise_02_altitude.m` gives a starting point for designing feedback gains. Complete the indicated lines to set up the system model, and place the two closed-loop poles at values close to $-1 + 0i$. If using Matlab, the poles must not be equal to each other[2]; values of -0.9 and -1.1 are suggested.

   Run the script in Octave to calculate values for k_1 and k_2, and try them in the Lander – the `Autopilot settings` menu allows you to type values in directly. Once you have obtained working gain values, use them to overwrite the default values in `lander.cpp` – the lines setting these are near the top of the file.

   While the scenario is running, try setting the demand altitude to different values to observe the step response of the lander.

[1]You will need the Control toolbox; instructions on how to install it are given on the Camtools site.

[2]Due to the algorithm that Matlab uses for this function, the number of repeated poles must not be greater than the rank of B (in this case 1). Octave uses the SLICOT library, which does not suffer from this limitation.

3. Try placing the eigenvalues at different locations in the complex plane e.g. you may want to make the altitude converge more quickly. What limits do you run into (that are not part of the linear model)?
4. In `lander.cpp`, modify scenario 7 to run without the attitude stabilizer (the line is near the end of the file, and marked with the word 'STUDENTS'). Recompile the program, and run scenario 7 again. What happens differently? Why?

4.2 Attitude control

In this section we will address the main aim of the document: creating a feedback controller for the lander's *attitude* (its orientation in 3D space). Firstly, we choose to define the lander's orientation in terms of three parameters ϕ, θ and ψ, known as *Euler angles*. Figure 1 shows how these angles are used to transform a coordinate frame through three successive rotations – it should be clear that any possible 3D rotation can be achieved by this method.

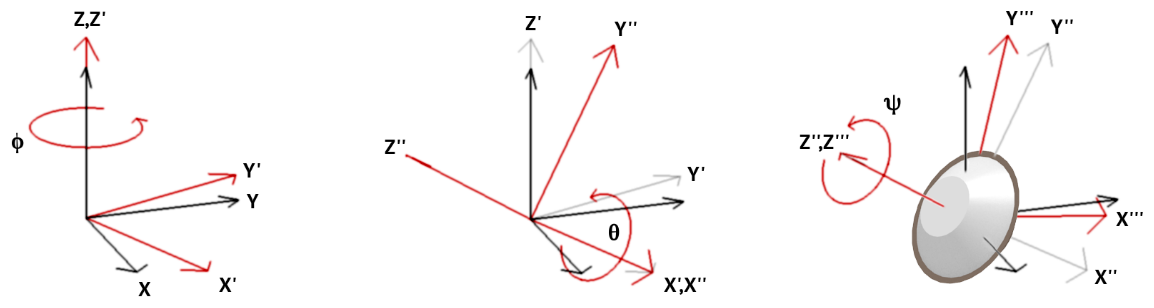

Figure 1: The three ZXZ Euler angle rotations acting on a coordinate frame. The first rotation ϕ is around the original Z-axis Z, the second rotation θ is around the rotated X-axis X', and the third rotation ψ is around the twice-rotated Z-axis Z''.

The governing dynamical equations for orientation, expressed in terms of the Euler angles, are given here:

$$\begin{aligned}\dot{\phi} &= \omega_x \frac{\sin\psi}{\sin\theta} + \omega_y \frac{\cos\psi}{\sin\theta} \\ \dot{\theta} &= \omega_x \cos\psi - \omega_y \sin\psi \\ \dot{\psi} &= \omega_x \frac{-\sin\psi}{\tan\theta} + \omega_y \frac{-\cos\psi}{\tan\theta} + \omega_z\end{aligned} \tag{22}$$

Collecting the angles into a vector $\underline{E}$ and taking partial derivatives, the linearized equations can be written in the following form:

$$\begin{aligned}\delta\dot{\underline{E}} = \begin{bmatrix}\delta\dot{\phi} \\ \delta\dot{\theta} \\ \delta\dot{\psi}\end{bmatrix} &= \begin{bmatrix} 0 & \dfrac{-(\omega_x\sin\psi + \omega_y\cos\psi)}{\tan\theta\sin\theta} & \dfrac{\omega_x\cos\psi - \omega_y\sin\psi}{\sin\theta} \\ 0 & 0 & -(\omega_x\sin\psi+\omega_y\cos\psi) \\ 0 & \dfrac{\omega_x\sin\psi+\omega_y\cos\psi}{\sin^2\theta} & \dfrac{-\omega_x\cos\psi+\omega_y\sin\psi}{\tan\theta}\end{bmatrix}\delta\underline{E} \\ &+ \begin{bmatrix}\dfrac{\sin\psi}{\sin\theta} & \dfrac{\cos\psi}{\sin\theta} & 0 \\ \cos\psi & -\sin\psi & 0 \\ \dfrac{-\sin\psi}{\tan\theta} & \dfrac{-\cos\psi}{\tan\theta} & 1\end{bmatrix}\delta\underline{\omega}\end{aligned} \tag{23}$$

Note that all terms in the first matrix contain an ω_i factor, and so will be zero if the lander is to be controlled to rest (assuming $\sin\theta \neq 0$).

In order to add a derivative component to the attitude controller, it may seem logical to obtain similar expressions for the second time-derivatives of the Euler angles. A better choice, however, is to use the three components of the angular velocity vector – this is an example of when two sets of variables encode the same information. Equation (8) can be expanded to give the evolution of the three components in response to a net torque $\underline{Q}$:

$$\begin{aligned}\dot{\underline{\omega}} &= I^{-1}(I\underline{\omega} \times \underline{\omega} + \underline{Q}) \\ &= \begin{bmatrix}((I_{yy} - I_{zz})\omega_2\omega_3 + Q_1)/I_{xx} \\ ((I_{zz} - I_{xx})\omega_1\omega_3 + Q_2)/I_{yy} \\ ((I_{xx} - I_{yy})\omega_1\omega_2 + Q_3)/I_{zz}\end{bmatrix}\end{aligned} \tag{24}$$

Here the inertia matrix $I = \begin{bmatrix}I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz}\end{bmatrix}$. Defining $\alpha = \frac{I_{yy} - I_{zz}}{I_{xx}}$, $\beta = \frac{I_{zz} - I_{xx}}{I_{yy}}$ and $\gamma = \frac{I_{xx} - I_{yy}}{I_{zz}}$, the Equation (24) may be rewritten as a matrix equation:

$$\delta\dot{\underline{\omega}} = \begin{bmatrix}0 & \alpha\omega_z & \alpha\omega_y \\ \beta\omega_z & 0 & \beta\omega_x \\ \gamma\omega_y & \gamma\omega_x & 0\end{bmatrix}\delta\underline{\omega} + \begin{bmatrix}\frac{1}{I_{xx}} & 0 & 0 \\ 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & \frac{1}{I_{zz}}\end{bmatrix}\delta\underline{Q} \tag{25}$$

As you will have observed in the first part of the exercise, the lander module is equipped with multiple thrusters, each with a separate throttle control. Neglecting drag forces, the net torque $\underline{Q}$ acting on the lander can be expressed as a summation over the contributions from each of these (each with position $\underline{p}_i$, thrust direction $\hat{\underline{d}}_i$ and throttle t_i):

$$\begin{aligned}\underline{Q} &= \sum_i \underline{p}_i \times -t_i T\hat{\underline{d}}_i \\ &= -T\begin{bmatrix}\uparrow & \uparrow & \\ \underline{s}_1 & \underline{s}_2 & \dots \\ \downarrow & \downarrow & \end{bmatrix}\begin{bmatrix}t_1 \\ t_2 \\ \vdots\end{bmatrix}\end{aligned} \tag{26}$$

where T is the maximum thrust that can be generated by any one thruster. The vectors $\underline{s}_i = \underline{p}_i \times \hat{\underline{d}}_i$ are precalculated for any given thruster arrangement.
Now, to control the system, we collect all our state variables and inputs together into two vectors:

$$\begin{aligned}\underline{x} &= \begin{bmatrix}\phi & \theta & \psi & \omega_x & \omega_y & \omega_z\end{bmatrix}^T \\ \underline{u} &= \begin{bmatrix}t_1 & t_2 & t_3 & t_4\end{bmatrix}^T\end{aligned} \tag{27}$$

The state-space model for attitude is then constructed by blocking together the matrices from equations (23), (25) and (26) (taking care to separate the states variables from the inputs) so that they line up with the appropriate state variables.

$$\dot{\underline{x}} = \begin{bmatrix}A_{11} & A_{12} \\ A_{21} & A_{22}\end{bmatrix}\underline{x} + \begin{bmatrix}B_{11} \\ B_{21}\end{bmatrix}\underline{u} \tag{28}$$

Since this is rather unwieldy to do by hand, the complete attitude system model has been set up for you in the script file `Exercise_02_eul.m`. This script sets up the system model, constructs a controllability matrix, and calculates appropriate gain values to fit desired poles (all are $-1 + 0i$ by default). Run the script in Octave, and confirm that it gives several lines of output (and no errors). According to the controllability analysis, two of the state variables are not controllable – using Figure 1, determine which axis of the lander these variables are associated with. Looking at the thruster positions and directions, is it clear why control is not possible about this axis?

1. In order to successfully control the lander's attitude, you will need to adjust the configuration of the thrusters. Starting with the Octave script, modify the matrix d to direct the four thrusters away from the vertical, according to their x positions – the two with positive x coordinates have thrust vectors of `[0.25, 0.0, -1.0]` before normalization, and the remaining two are in the opposite direction. Running the script again should show that the system is now controllable.

   This new thruster arrangement must now be applied to the Lander C++ code – the relevant lines are at the top of the function `initialize_simulation()` in `lander.cpp`.

   Note that changing the thruster directions or positions will invalidate the gain values calculated in the previous section (although in this case the difference is negligible). One of the outputs of this script is a new `Tv` value that can be used to calculate updated values for k_1 and k_2 in `Exercise_02_altitude.m`.

2. Examine the `autopilot()` function, and make sure you understand what each line is doing. Using the K matrix produced by the script, modify the autopilot to set the throttles based on feedback from the orientation variables (`delta_phi`, `delta_theta` etc.), excluding the altitude terms. You will need to rearrange the code somewhat to allow different throttle values for each of the thrusters.

   NB: The `K` matrix is indexed so that thruster `i` responds to state variable j with coefficient `K[i][j]`. All vectors and variable names in the C++ code correspond to this document.

   To test your attitude controller, run scenario 8 to observe the lander spinning in free space. Turn on the autopilot and check (using the graph view) that the Euler angles and angular velocity components all settle to their demand values.

3. Now add the altitude term back in to the feedback calculations, so that the effects of the two control laws (altitude and attitude) are summed. This combined controller should now allow the lander to stabilize itself and hover in scenario 7.

   Try several different starting conditions and demand altitudes to test how robust this autopilot is. Try making the feedback more/less aggressive, and note the effects.

Extension exercises for you to think about (optional):

- In certain orientations, a phenomenon known as *gimbal lock* will occur in the Euler angles, which can be problematic for this kind of application. Can you develop an autopilot that uses *quaternions* (see the Dynamics teaching document for an introduction to these) in place of Euler angles?
- Try adding in random disturbance to the aerodynamics (modifying the `world_wind` variable in the function `calculate_drag()` will achieve this). How robust is the control? Can it counter a constant disturbance?
- Can you control the ground speed of the lander as well as the orientation and altitude?
- Develop an autopilot capable of de-orbiting the lander, then stabilizing the fall into a hover at a fixed altitude (without using the built-in attitude stabilizer!). You may want to investigate the topics of *gain scheduling* and *model predictive control* to help with this.