A blue letter p on a black background

AI-generated content may be incorrect.

***Logbook***

Aiden Hains

[20962562@student.curtin.edu.au](mailto:20962562@student.curtin.edu.au)

|  |  |
| --- | --- |
| **Date** | **Work Accomplished** |
| 16/4 | Researched what a b-dot controller is, found a few articles on a standard b-dot and a non-standard one.  A screenshot of a computer  AI-generated content may be incorrect. |
| 16/4 | I watched Monte Carlos videos on the mathematics behind b-dot controllers, along with some more information about attitude determination, some orbital mechanics and magnetorquers.  <https://www.youtube.com/watch?v=uNZeDMmDdbo&list=PL_D7_GvGz-v3mDQ9iR-cfjXsQf4DeR1_H&index=7> |
| 17/4 | Researched what a magnetorquer was.  My thoughts:  It is essentially the armature section of a dc motor, it uses the current input, and the earth’s magnetic field that passes through it to produce a torque about axis that it is aligned with.  The basic principle of using it to detumble is that the torque applied by the magnetorquer is supposed to oppose the change in the earth’s magnetic field that passes through this axis.  Since the end result should be that CubeSat flies through the inertial magnetic field, converts it to the magnetic field in the body frame and thus this body frame magnetic field should not change over time, the angular velocity in each axis of the CubeSat should not change, therefore being de-tumbled.  Thus, the moment applied by a magnetorquer is as follows:  moment = n\*A\*I,  where n = number of turns in the wire,  A = cross sectional area of the wire assembly.  I = current.  The torque applied by a magnetorquer is then the cross product between the magnetic moment and the earth’s magnetic field in the coils frame of reference. |
| 17/4 | Researched the basic principle of the b-dot controller.  The idea is that the rate of change of the magnetic field passing through the magnetorquer should eventually drop to zero. Which means that the CubeSat is not spinning on any of its axis.  This means that if you have some way of reducing that rate of change of magnetic field, then you have a way of de-tumbling the CubeSat.  The simplest method is to produce a torque that opposes this rate of change, with some gain added. This is the basic b-dot algorithm, take the derivative of the magnetic field through each of the body axis of the CubeSat, and multiply it by some gain value ‘k’, and use that in the equation that defined the angular acceleration, which by integration, affects the angular velocity.  The equation for the b-dot is then:  Where  (and B is in the body frame)  There are other methods of doing this as well, that I currently do not know how to explain.  For next time:  Research another b-dot algorithm and start coding a CubeSat orbiter. |
| 18/4-21/4 | Coding a simple orbiter in MATLAB using monte carlos’ code as a base.  Plot of orbiter:  A graph of a sphere with colored lines  AI-generated content may be incorrect.  This took much longer than expected as I forgot to add in all the small details.  The method of making the above plot was first to get an initial state that the CubeSat was in, so I gave it a standard 56-degree 600km altitude LEO.  The starting xyz position is the altitude on only the x direction, so no y or z components in the inertial frame as seen by the red axes above, which is correct because a zero-angle right ascension of ascending node coincides with the vernal equinox direction, where y = z = 0.  Then the velocity in each inertial direction must be found, at is assumed that the CubeSat is not moving on or out of the x-direction at the initial state, but moving perpendicular to it, which means that there is velocity on both the y and z direction, defined by the speed at which a 600 km circular orbit shall move at multiplied by the cosine and sine respectively of the orbital inclination.  Then the acceleration in each axis is taken as just gravity acting alone.  The initial quaternion pose frame of the CubeSat is the exact same pose as the earths xyz frame for simplicity.  Then the angular velocity is defined by me to be some random number positive or negative to show the angular velocity behavior. |
| 21/4 | Added magnetic field components to the simulation with the IGRF model, basically simulating a perfect sensor (hopefully will add sensor noise at some point).  The IGRF model is very strange, it returns magnetic field components in the North East Down frame, which I don’t fully understand. But with the help of monte carlos’ videos, I was able to get these strange components into the inertial x y and z components frame, using the quaternion orientation components which makes a lot more sense.  The magnetic field of the inertial frame over the orbit : |
| 22/4 | Wrote my own satellite function, it takes in the xyz position, translation velocity in xyz, the angular velocity of the CubeSat in the body frame, and the quaternion pose of the CubeSat.  It then calculates the derivative of all these components.  The derivative of the position is simply the translational velocity of xyz.  The derivative of the translational velocity is taken as the acceleration due to gravity. It is possible to take other forces into account, but at this scale its hard to tell what the solar radiation pressure is, also its much less significant than gravity at this low earth orbit.  The derivative of the quaternions is taken as a kind of complicated relationship using the angular velocity as pre-defined matrix, the relationship is show below:    Pqr is the angular velocity in the xyz directions of the body frame respectively.  The derivative of the angular velocity is the really important section of this satellite module, as this is the section that the b-dot algorithm can take effect. Since the formula is as such:    The ‘I’ matrix is the 3x3 inertia matrix, which is to say its just a 1x3 matrix with each x y and z inertia, taken as a column vector, multiplied by the identity matrix so that it can be multiplied and inverted and cross-produced with.  The omega above is the angular velocity, pqr. And the ‘Tmt’ vector is just the torque applied by the magnetorquer.  This relationship is the most important as this is where the de-tumbling happens.  There are a few more additions to this satellite module that use quaternions to change reference frames, but the previously stated relationships are the main equations that define the CubeSat behaviour.  The whole function is:  % This function computes the derivative of the position, velocity,  % quaternions and angular velocity for numerical integration  function dstatedt = Satellite(t, state)  % format: position xyz (1:3), velocity in xyz directions (4:6),  % quarternion orientation (7:10), angular velocity (11:13)  % init\_state = [x0; y0; z0; xdot0; ydot0; zdot0; quart0; p0; q0; r0];  quart = state(7:10); % [q1, q2, q3, q4]  angvel = state(11:13); % [p, q, r]  p = angvel(1);  q = angvel(2);  r = angvel(3);  %%% get inertial parameters %%%  InertialParams  %%% Gravity Model %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  % Get earth params  Planet  dist\_vect = state(1:3); %[x, y, z]  dist = norm(dist\_vect);  distvect\_hat = dist\_vect/dist;  F\_gravity = ((-G\*M\*mass)/(dist^2))\*distvect\_hat;  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  %%% Translation Kinematics %%%%%%%  vel = state(4:6); % [vx, vy, vz]  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  %%% Translational Dynamics %%%%%%%  force = F\_gravity;  accel = force/mass;  p = angvel(1);  q = angvel(2);  r = angvel(3);  rpqmat = [ 0, -r, q;  r, 0, -p;  -q, p, 0];  %accel = (1/mass)\*state(1:3) - (rpqmat\*vel);  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  %%% Rotational Kinematics %%%%%%%%%%%%%%%%%%%  angvel\_mat = [ 0 -p -q -r ;  p 0 r -q ;  q -r 0 p ;  r q -p 0 ];  quartder = (1/2).\*(angvel\_mat\*quart);  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  %%% Rotational Dynamics %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  invInertia = inv(Inertia);  H = Inertia\*angvel;  x = state(1);  y = state(2);  z = state(3);  magfieldinertial = MagneticField(x, y, z)';  magfieldbody = TBIquat(quart)\*magfieldinertial;  MagnetorquerParams  mag\_torque = Controller(magfieldbody, angvel, n, A);  angvelder = invInertia\*(-cross(angvel, H) + mag\_torque);  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  dstatedt = [vel; accel; quartder; angvelder]; |
| 23/4 | Changed numerical integrator to a RK4 style using monte carlos’ one as a base. Since ode45 is extremely slow.  The new integrator is:    It takes small iterative additions to the current state then adds them all together. |
| 24/4 | Programmed my first b-dot controller using the iterative method of the magnetic field derivative, which is the standard b-dot.  The photo below shows the b-dot (look for magfield\_deriv):    The angular velocity from this de-tumbling (after a whole load of time figuring out what value of k wouldn’t send the CubeSat into infinite spin or do nothing)    After hours and days of debilitating I realised that this is the best I could ever get from the iterative method, I am not sure where I went wrong, but it was a very interesting way of doing b-dot.  This made me think I could do better, so I researched some more and realised that monte carlos’ aerospace paper, would have the most in-depth dive into orbital mechanics. Which it did and I found another way of writing a b-dot without using any iterative method  For next time:  Re-write algorithm to be non-iterative |
| 7/5 | The basic idea for the new b-dot is that the derivative of the magnetic field is equal to that of the cross product between the angular velocity and the magnetic field in the body frame.  Then the magnetic moment applied by the magnetorquer is a gain value ‘k’ multiplied by this cross product.  After this the torque applied is then moment cross producted with the magnetic field through the body:  This method is much more analytical and relies on currently known values of the angular velocity and magnetic field to reliably make decisions on how much torque to apply, also the de-tumbling works much better than the iterative method:  A graph of a function  AI-generated content may be incorrect.  The value for ‘k’ that resulted in this is 40,000. And step size of 1 second.  The CubeSat de-tumbles much quicker and actually tends to zero, making it the superior model.  For next time:   * Change step size |
| 14/5 | Using the same detumbling algorithm, I started decreasing the step size by factors of 10, first with a step size of 0.1 seconds, which resulted in this detumbling:  A graph of a function  AI-generated content may be incorrect.  Which detumbles in a single orbit, a bit slower than the 1 second step size, however, it does still show stability even with more unstable integration.  Also, the k value of 40,000 stayed the same.  After this I tried out what an even smaller step size would do, so 0.01 seconds, and I did have to tweak the gain value a bit to get it to fully detumble, but it did seem to not go to absolute zero, but plateau around 1-2 degrees per second, so more testing of the gain may be needed, or the step size might be too small for this application:  A graph of a graph showing a number of objects  AI-generated content may be incorrect.  The gain for this is 1,000,000. |
| 15/5 | Started and finished the block diagram:  A diagram of a diagram  AI-generated content may be incorrect. |
| 15/5 | Started writing the logical flowchart for all functions.  Finished main function flowchart, still need Satellite, and controller.  For next time:  Finished satellite and controller function flowcharts |
| 17/5 | A diagram of a flowchart  AI-generated content may be incorrect.Finished all three flowcharts: |
| 18/5 | My reflections:   * B-dot algorithm is a very powerful way of de-tumbling a satellite, and seems to have quite a few methods of actually implementing it, its not stricly rigid, and many ideas of b-dot are all possible with varying success rates. * I believe the reason my iterative b-dot didn’t work as expected probably came down to the fact I didn’t write all my code with the iterative method in mind first. What I mean by this is that each iteration I get the derivative of the kinematics and dynamics, but the amount of torque that is being applied in the angular acceleration equation is working on either outdated information and not using the correct derivative value. Im not entirely sure but it is an interesting thought. * Regardless, the cross product between the angular velocity and magnetic field seems to be quite robust, accounting for less reliable integration. * I would have liked to add in sensor noise to really show robustness but I just didn’t have time to implement it. With this in mind I would have liked to learn about kalmann filters and sensor noise reduction but that a little out of scope for this project. * All in all this was really fun, a little head-banging-against-a-wall at times but I did enjoy seeing the graph when the angular velocity went to zero. |