The Attitude and Determination Control Subsystem (ADCS) for the Tethering and Ranging Mission of the Georgia Institute of Technology (TARGIT)

AE 8900 MS Special Problems Report

Space Systems Design Lab (SSDL)

Daniel Guggenheim School of Aerospace Engineering

Georgia Institute of Technology, Atlanta, GA

Author:

Abhijit Harathi

Advisor:

Dr. Brian C. Gunter

May 14, 2021

The Attitude and Determination Control Subsystem (ADCS) for the Tethering and Ranging Mission of the Georgia Institute of Technology (TARGIT)

by Abhijit Harathi

TARGIT Overview

The Tethering And Ranging mission of the Georgia Institute of Technology (TARGIT) is a CubeSat mission that aims to demonstrate target detection and tracking via LiDAR and other sensors. A 3U CubeSat will be deployed from the ISS. After going through system checkouts and other procedures, it will deploy and inflate a tetrahedron-shaped target. The CubeSat will image and track the target with sensors in order to keep it within its primary imager's field of view. Once it is a certain distance away, the CubeSat will use laser ranging technology to detect the target. Throughout all this, the CubeSat will be going through other mission modes to charge up, occasionally reduce its angular rates and stabilize itself, and keep a certain pointing configuration.

In order to achieve and transition between the different mission modes, TARGIT has an attitude determination and control subsystem (ADCS) that maneuvers the spacecraft to and maintains different pointing configurations based on the mission mode required. It uses sensors to determine its orientation and it uses actuators to change or maintain its orientation as needed.

TARGIT ADCS Overview and Configuration

TARGIT's ADCS can be broken down into hardware and software. As mentioned earlier, the hardware that the ADCS uses consists of sensors and actuators. The flight unit's sensors and actuators are listed in Table 1, along with the manufacturers, names, and quantities. Note that the Epson M-G364 Inertial Measurement Unit (IMU) has both a gyroscope and accelerometer. Also, the coarse sun sensor (CSS) and magnetic torque rods were developed in-house, so there are no associated manufacturers and names for them.

Table 1. Hardware components for ADCS.

Hardware Type	Unit Type	Manufacturer and Name	Quantity
Sensors	Magnetometer	Honeywell HMC1053	1
	Inertial Measurement Unit (IMU)	Epson M-G364	1
	Horizon Sensor	FLIR Lepton 2.5	2
	Pixycam	Pixycam Pixy2	1
	Fine Sun Sensor (FSS)	SolarMEMS nanoSSOC-D60	2
	Coarse Sun Sensor (CSS)	N/A	3
	GPS	NovAtel OEM719	
Actuators	Magnetic Torque Rods	N/A	3
	Reaction Wheels	Maryland Aerospace MAI-400	3

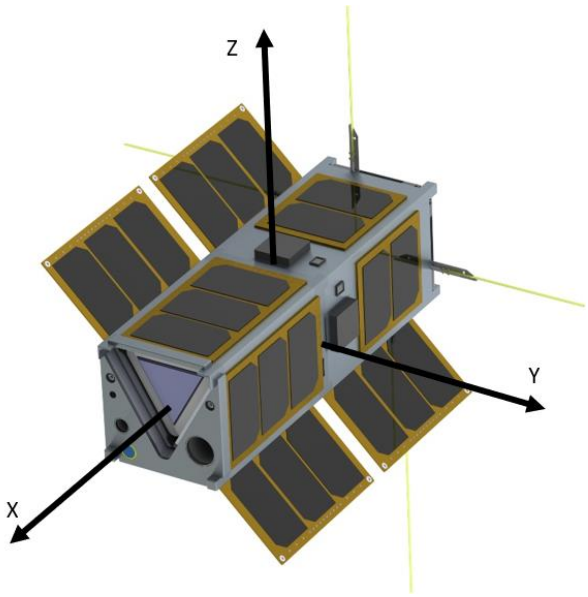


Figure 1. TARGIT spacecraft with body axes.

Figure 1 shows the TARGIT spacecraft with body axes X, Y, and Z. The 3U CubeSat has a set of fold-out deployable solar panels. The X-axis is aligned with the laser ranging payload that will be used for science operations, while the Y and Z axes are normal to the faces on which the solar panels are attached. The spacecraft will be in a "flying-wing" configuration in nominal operations, with the x-axis aligned with the direction of the velocity vector and the solar panels pointing zenith. This results in the laser ranging payload facing in the direction of the anti-velocity vector, which is also the direction in which the inflatable target will be deployed.

The ADCS components that are mounted internally are the magnetometer, the IMU, the magnetic torque rods, and the reaction wheels. The sensing done by the magnetometer, GPS, and IMU does not require any external vision, while the actuators create internal torques, negating the need for external placement. Meanwhile, the horizon sensors, Pixycam, the fine sun sensors, and coarse sun sensors are mounted externally. The horizon sensors are mounted normal to the two zenith facing body panels such that they can detect the Earth's curvature in the flying-wing configuration. The Pixycam is mounted facing the anti-velocity direction because, like the laser, it also needs to be able to see the target. The fine sun sensors are mounted on the zenith-facing sides of the body to provide the best sun-pointing control to charge the solar panels. Two of the course sun sensors are on the nadir-facing sides of the body, while one is mounted facing the direction of the velocity vector. If the sun is not seen by the fine sun sensors but is picked up by the coarse sun sensors during sun-pointing, the spacecraft's orientation will be adjusted accordingly until the fine sun sensors can pick up the direction of the sun in order to maximize charging.

The ADCS software uses the information gathered from the sensors above for calibration to be done before launch and for algorithms for different mission modes on-orbit. There is calibration done beforehand for the magnetometer and coarse sun sensor. ADCS control algorithms will be used for detumble, sun pointing, local vertical local horizontal (LVLH) pointing, target tracking, and momentum management. A Kalman filter algorithm will also be used

to assist with the control algorithms and for general attitude estimation. In addition, a least squares algorithm will be used for detecting the direction of the sun based on the different sun sensor measurements. All these algorithms are explained in detail in the Algorithms section.

TARGIT ADCS Hardware

The following sections describe the ADCS hardware's functionality. Note, any application program interface (API) referenced can be found on the *targit-flight* repository, in *hardware_apis/src/hardware_apis*.

Magnetometer

The ADCS uses the three-axis Honeywell HMC1053 magnetometer to detect the Earth's magnetic field on orbit. It will be used in the detumble and Kalman filter algorithms. It is an analog sensor that outputs to an analog-to-digital converter (ADC), which interfaces with the on-board computer (OBC) through I^2C communication protocol. There is also a GPIO connection to a set/reset circuit on the magnetometer, which is used to send a pulse to the circuit before each reading. This allows internal magnetic components to be reset such that a previous reading does not affect the next reading. The ADC outputs voltage values corresponding to the x, y, and z axis in the magnetometer's reference frame in units of volts, These values range from 0 to the reference voltage of the ADC, which is set to 3.3V. These measurements are converted to the spacecraft's body frame in the flight code.

Before being put on the flight unit that is sent up to orbit, this magnetometer must be calibrated beforehand. The Engineering Sciences and Mechanics (ESM) building on campus has a Helmholtz cage through which an artificial magnetic field can be created. Using another magnetometer, the magnetic field inside the cage can be read, and a field will be created such that the magnetometer reads zero across all three axes. The other magnetometer will then be replaced with the Honeywell magnetometer and the output from the Honeywell magnetometer is taken. Because the field should be zero, if the magnetometer does not read values of zero, the output it reads is used as a calibration offset that is subtracted from future measurements.

The magnetometer reads Earth's magnetic field in Gauss, so a conversion must be done from the ADC voltage values to the magnetometer values in Gauss. The following equation is used for the conversion for the value for the x, y, and z-axis outputs from the magnetometer and is handled in the honeywell_mag.py API:

$$b_{Gauss} = \frac{b_{volts} - V_{ref}}{SV_{cc}G}$$

b_{Gauss} is the magnetic field sensed by the magnetometer in units of Gauss and b_{volts} is the magnetic field sensed by the magnetometer in units of volts. The magnetometer circuit contains 3 MAX4208 inverted amplifiers through which the magnetometer outputs are sent to amplify their voltage. V_{ref} is this reference voltage, while G refers to the gain by which the magnetometer outputs are amplified. S refers to the sensitivity of the magnetometer measurements, provided in the datasheet, while V_{cc} is the power supplied to the magnetometer. Note that V_{ref} is exactly half of

V_{cc}. This allows us to discern between positive and negative outputs from the magnetometer. The values for V_{ref}, S, V_{cc}, and G are provided in the table below.

Table 2. Variable Values for magnetometer output conversion.

Variable	Value
V_{ref} (V)	1.65
S (1/Gauss)	1 x 10-3
V_{cc} (V)	3.3
G	413

The honeywell_mag.py API can be referenced for more information.

IMU

The gyroscope and accelerometer used by the ADCS are part of the Epson M-G364 inertial measurement unit (IMU). This six-axis IMU interfaces with the OBC through SPI communication protocol for the flight unit, but it can also use UART if needed. The gyroscope measures the angular velocity of the spacecraft, with a range of -100 to 100 degrees per second, while the accelerometer measures linear acceleration of the spacecraft, with a range of -3 G to 3 G. 1 G here is as reference value, determined by the manufacturer, of 9.80665 m/s^2. The IMU outputs values for its gyro and accelerometer in terms of degrees per second and mG (1e-3 G) in 16-bit words. The conversion from bits to the proper values is handled in the epson.py API, and this API can be referenced for more information.

Horizon Sensor

The horizon sensor suite used by the ADCS contains two infrared radiation (IR)-imaging sensors. Earth emits radiation and the sensors will be placed such that the sensors see Earth's horizon in their field-of-view (FOV). Calculations are done to curve-fit the horizon seen by the sensors to determine how much the spacecraft needs to rotate about its pitch and roll axes such that the flying-wing configuration can be maintained. From this, a nadir-pointing vector can be calculated. Consequently, the horizon sensor will be vital to the LVLH pointing mode and will be used in the Kalman filter algorithm, discussed more in the Algorithms section.

To get the nadir-pointing vector, first the image taken by the horizon sensor must be processed for edge detection. A mask is applied to the image to make pixels greater than a defined threshold white, while the others below the threshold are black. From this, the Earth will be very clearly defined compared to the rest of the image. The Prewitt edge detection method is then used to isolate Earth's horizon edge.

Using the points corresponding to the horizon, a best fit radius and center of Earth in the image coordinate system is calculated using a least squares optimization method. The following link can be referred to for more information on the Python module and methods used: https://scipy-cookbook.readthedocs.io/items/Least_Squares_Circle.html.

Pitch is calculated between the image boresight and the horizon via the following equation:

$$p = 0.6375(\sqrt{x_c^2 + y_c^2} - R_e)$$

R_e refers to the calculated radius of the Earth from the least squares method mentioned earlier, while x_c and y_c are the center coordinates of the circle. 0.6375 comes from the vertical field of view in degrees divided by the vertical dimensions of the edge in pixels.

Roll is calculated from the horizon sensor's image as the following:

$$r = \arctan\left(\frac{y_c}{x_c}\right)$$

A nadir vector can be produced using the previously calculated pitch and roll angles from the image plane. Since the horizon sensors are mounted at fixed angles with respect to the spacecraft body, the altitude of the spacecraft needs to be added to the equations to correct the nadir vector. The equations below show how to get the nadir-pointing unit vector in the image frame, later translated to the spacecraft body frame:

$$\theta = \arccos\left(\frac{R}{R + 400}\right)$$

$$p_{corr} = p + \theta - \arccos\left(\frac{R}{R + h}\right)$$

$$\eta = 90 - \theta + p_{corr}$$

$$u_{nadir} = R_z(r)R_x(\eta) * [0\ 0\ 1]'$$

u_{nadir} is the unit vector pointing in the nadir direction, while R_z and R_x are rotation matrices used to rotate the $[0\ 0\ 1]'$ vector first by η about the x-axis and then by r degrees about the z-axis in the image frame. More information on this can be found in the *ImageTestScript.py* API and *HSP.py* API.

Pixycam

The Pixycam used by the ADCS, the Pixy 2, is an imaging system that can be trained to detect different colors. It interfaces with the BeagleBone Black flight computer through I^2C communication protocol. It will be used to detect and track the target during science operations, out to a certain distance. the target is one color, but because of the different shades that it might appear as at different points in the orbit, due to lighting effects, it must be trained to detect these different shades as well. This will be done prior to launch by training the Pixycam to detect the target in conditions from dimly lit to completely lit. When the Pixycam finds the target in its field of view, it generates a bounding box around the target and gives an x and y-coordinate of the center of that bounding box with respect to the x-y origin of the field of view, which is defined at the top left frame of the field of view. More information, as well as a diagram of what the field of view would look like, is shown in the "Target Tracking" section later on when ADCS algorithms are discussed. For more information on how the Pixycam interacts with the flight computer, check the *pixy_i2c.py* API.

FSS

The FSS uses 4 photodiodes and an internal processor that calculates the azimuth and elevation incidence angles of an incoming sun ray. The following diagram shows how the angles are defined with respect to the sun sensor's body frame. The incoming sun ray is used to form a rectangular prism, with lines tracing out to the opposite edges of the two shorter faces from the point of origin, along the x and y axes. From there, angles α and β are defined from those lines with respect to the z-axis, respectively, in the xz and yz planes.

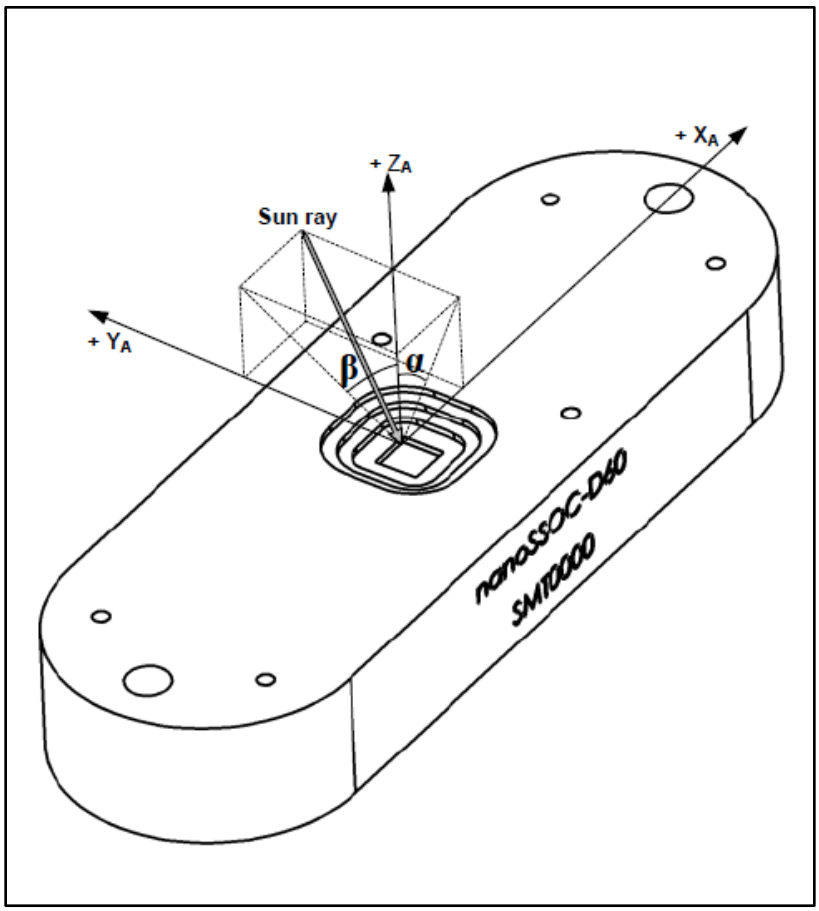


Figure 2. Definitions of the sun ray incidence angles in the FSS body axes.

The equations below use those two angles to produce a three-axis unit vector, also in the sun sensor's body frame, in the direction of the sun. This sun vector will be used for the sun pointing, target tracking, and Kalman filter algorithms.

$$\boldsymbol{s} = [\frac{\tan(\alpha)}{\sqrt{\tan(\alpha)^2 + \tan(\beta)^2 + 1}} \quad \frac{\tan(\beta)}{\sqrt{\tan(\alpha)^2 + \tan(\beta)^2 + 1}} \quad \frac{1}{\sqrt{\tan(\alpha)^2 + \tan(\beta)^2 + 1}}]'$$

α and β refer to the incidence angles shown in Figure 3. These equations are handled in the fss.py API, and this API should be referenced for more information on how communication with the FSS is handled.

CSS

The CSS consists of a photodiode on a circuit board developed in-house that communicates through I^2C protocol. This diode uses a machine learning algorithm to provide a set of lookup tables that are used to determine the location of the Sun. Based on this table, the CSS can give a rough angle of the direction of the sun. It is not as accurate at determining a sun vector as the FSS is, which is why the CSS boards are not placed on the sides on which the solar panels are also facing. Both the FSS and CSS boards will be necessary for sun pointing, but a weighted least squares algorithm, discussed later, will be used to determine the sun vector from all the information gathered from the different sensors. More information on the CSS communication can be found in the css.py API.

GPS

The GPS is used for orbit determination purposes as it gets externally determined position updates. These will be useful for the Sun, Earth, and magnetic field models that will be used by the Kalman filter. These models need a reference position for which the sun vector, Earth nadir vector, and magnetic field vectors can be used as reference in the Kalman filter algorithm described later on. For more information, the *oem719.py* API can be referenced.

Magnetic Torque Rods

The magnetic torque rods built in-house for the flight unit are three separate metal rods that have insulated copper wire wrapped around them, with one torque rod per axis. Electricity is sent through the copper wires using pulse-width modulation (PWM), and as the electricity goes around through the coils, a magnetic dipole is created that interacts with the magnetic field, creating a torque on the spacecraft system. Voltage sent through the torque rods via PWM channels goes through high frequency changes between an on and off state, essentially allowing for control of the average voltage depending on the duration of the on state compared to the off state. This allows for control of the torque acting on the spacecraft from the dipole moment. The torque rods are used in the detumble and momentum management algorithms, but they can also be used as backups for the other algorithms. The specific methods for sending commands can be seen in the pwm.py API.

Reaction Wheels

The reaction wheels used by the ADCS are the Maryland Aerospace MAI-400 reaction wheels. There are three reaction wheels on the flight unit, one to provide a torque about each spacecraft body axis. Each one is controlled independently with its own printed circuit board attached to the unit. Interacting with the reaction wheels via the OBC can be achieved through I^2C and UART communication protocol. To control the reaction wheels, each wheel can be individually set to be commanded to a certain speed or torque. Each reaction wheel unit has a tachometer as well to determine the wheel speed in RPM. Check the *maryland.py* API for more information. The reaction wheels are used in all the control algorithms, with the exception being detumble since that relies on the magnetic torque rods. If needed, however, they can be used as a backup for detumble.

TARGIT ADCS Software

The software used by TARGIT ADCS can be found on the *adac_kit* repository on GitHub. In addition, software is tested in simulation or on the engineering unit. Results for some of these tests will be presented in the individual algorithm sections below. Future testing will be conducted for the more complicated algorithm tests and should be updated in this document post-testing.

Algorithms

The algorithms utilized by the ADCS consist of estimation and control algorithms. Although we have a method for calculating the sun sensor vector, because we have multiple sensors employed, a weighted least squares method is employed to determine the best-fit vector. The Kalman filter is used to estimate attitude, and it requires as many available sensors as possible to determine the best estimate.

The first control algorithm discussed is detumble, which is used for angular rate damping during initial deployment, and other events in which the satellite is spinning at unexpected rates and needs to be de-spun before transitioning into other modes. The second algorithm, sun pointing, is used to point the solar panels normal to the sun to maximize charging. This will be vital especially when the battery level reaches a threshold below which charging is necessary in order to be able to operate the satellite.

For nominal operations, LVLH pointing will be used to keep the spacecraft in its flying-wing configuration to keep the solar panels pointed zenith and the payload section facing the anti-ram direction. This is vital to keep the target behind the spacecraft such that as it inflates, as this minimizes the risk of the tether (that is attached to the target) getting caught on some part of the spacecraft. This LVLH pointing will have heavy usage of the ADCS sensor and actuator suite since it will need to use Kalman filtering and potentially momentum management.

During the science operations mode, the spacecraft will be pointing its Pixycam imaging system and laser ranging system at the target. The target tracking algorithm will be utilized, and it is a special case of the LVLH pointing mode. Again, a Kalman filter will be used to estimate attitude while keeping the Pixycam centered on the target as best as possible. Once the Pixycam is centered, an optimal roll angle will be calculated to attempt to keep the solar panels pointed at the Sun while still tracking the target. The last main algorithm that will be discussed is momentum management, which consists of using the magnetic torque rods to desaturate the momentum in the reaction wheels.

Further information on these algorithms can be found in the sections below, while the code can be found on the *adac-kit* and *adcs-proto* repositories on GitHub, in *adac-kit/src/adac_kit* or *adcs-proto/apps*.

Sun Sensor Vector Calculation

Because there are multiple sun sensors used to calculate the sun vector, a recursive weighted least squares algorithm is used to determine the Sun's position. An apriori estimate, x_o,

and a covariance estimate, P_0, are taken from the previous time step. Pre-defined estimates are used at the start of the estimation period as there is no previous time steps.

A measurement covariance matrix at each time step, R_k is used to provide weights based on the amount of trust that the algorithm has in each measurement. For example, if an FSS and CSS can both see the sun, the FSS's measurement would be given a higher weightage because it is supposed to have better resolution. If one sensor cannot see the sun, it would be removed from the calculation and essentially given 0 weightage. Similarly, if it is clear that a sun sensor is providing very poor data, that measurement should not be affecting data, and it would be weighted down accordingly. To estimate the sun vector measurement, the following equation is used to get the updated "best fit" sun sensor measurement, $\widehat{\boldsymbol{x_k}}$, similar to getting a line of best fit:

$$\widehat{\boldsymbol{x_k}} = \overrightarrow{\boldsymbol{x_0}} + \boldsymbol{K_k}(\overrightarrow{\boldsymbol{y_k}} - \boldsymbol{H}\overrightarrow{\boldsymbol{x_0}})$$

In this case, $\mathbf{K_k}$ is equal to:

$$\boldsymbol{K_k} = \boldsymbol{P_0}\boldsymbol{H^T}(\boldsymbol{H} * \boldsymbol{P_0}\boldsymbol{H^T} + \boldsymbol{R_k})^{-1}$$

H is a state to measurement matrix, while $\overrightarrow{\boldsymbol{y_k}}$ is the sun sensor measurement at each time epoch. To get the a posteriori estimation error covariance, $\mathbf{P_k}$, the equation below is used:

$$\boldsymbol{P_k} = (\boldsymbol{I_{3x3}} - \boldsymbol{K_k}\boldsymbol{H})\boldsymbol{P_0}(\boldsymbol{I_{3x3}} - \boldsymbol{K_k}\boldsymbol{H})' + \boldsymbol{K_k}\boldsymbol{H}(\boldsymbol{I_{3x3}} - \boldsymbol{K_k}\boldsymbol{H})'$$

The updated $\mathbf{x_k}$ and $\mathbf{P_k}$ can be used as the a priori update in the next sun vector estimate. The *estimation.py* file should be referenced for more code-specific information.

Kalman Filter

Attitude determination or estimation for LVLH pointing and for other 3-axis control modes, such as target tracking, is achieved through using an extended Kalman filter. It estimates the spacecraft body quaternion in the inertial frame and the gyroscope bias using two or more reference vectors. This will consist of the magnetometer and sun vector in sunlight, and the magnetometer and horizon sensor nadir vector in eclipse. The description of the Kalman filter will be broken into multiple steps for clarity.

<u>Dynamic and Kinematic Models</u>

The dynamic equation for the satellite can be expressed using Euler's equation:

$$\dot{\boldsymbol{\omega}}_{ib}^{b} = \boldsymbol{J^{-1}}[\boldsymbol{T}^{b} - [\boldsymbol{\omega}_{ib}^{b} \times]\,\boldsymbol{J}\,\boldsymbol{\omega}_{ib}^{b}]$$

J is the inertia matrix, $\boldsymbol{\omega}_{ib}^{b}$ represents the angular velocity of the body frame with respect to the inertial frame, and $\boldsymbol{T}^{b}$ is the sum of torques acting on the spacecraft in the body frame. $[\boldsymbol{\omega}_{ib}^{b} \times]$ represents a skew symmetric matrix that performs the cross product shown below.

$$[\boldsymbol{\omega}_{ib}^{b} \times] = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}$$

The quaternion attitude kinematics are represented by:

$$\dot{q} = \frac{1}{2}\Omega(\omega)q$$

q is the orientation of the spacecraft body frame with respect to the inertial frame, **ω** is the angular velocity of the body frame with respect to the inertial frame, and $\boldsymbol{\Omega}(\boldsymbol{\omega})$ is defined by:

$$\boldsymbol{\Omega}(\boldsymbol{\omega}) = \begin{bmatrix} -[\boldsymbol{\omega}\times] & \boldsymbol{\omega} \\ -\boldsymbol{\omega}^T & 0 \end{bmatrix}$$

Measurement Model

Vector measurements from sensors are used to determine attitude and gyroscope bias. These will come from the magnetometer, sun sensors, and horizon sensors, which will output magnetic field vectors, sun vectors, and earth nadir vectors. Given an estimate of a reference direction in the inertial frame, the estimated body frame measurement is:

$$\hat{\boldsymbol{r}}^b = A\left(\mathbf{q}_i^b\right)\hat{\boldsymbol{r}}^i$$

$\hat{\boldsymbol{r}}^{\boldsymbol{i}}$ is the reference vector in the inertial frame, and $\hat{\boldsymbol{r}}^{\boldsymbol{b}}$ is the corresponding reference direction in the body frame. $\boldsymbol{A}\left(\boldsymbol{q}_{\boldsymbol{i}}^{\boldsymbol{b}}\right)$ represents the transformation matrix from inertial to body frame given the quaternion. $\hat{\boldsymbol{r}}^{\boldsymbol{i}}$ is found based on the measurement used. For example, if a magnetic field measurement was being used, then $\hat{\boldsymbol{r}}^{\boldsymbol{i}}$ would be the magnetic field direction at the spacecraft's location in the inertial frame, which can be found using the IGRF model for Earth's magnetic field. This is why having a GPS is crucial for Kalman filtering.

The gyroscope measurement model assumes that the measured angular velocity has a bias and zero mean Gaussian white noise. The bias also contains zero mean white noise. The bias dynamic model is therefore:

$$\dot{\boldsymbol{\beta}} = 0$$

Subtracting the bias from the measured angular velocity gives us:

$$\hat{\boldsymbol{\omega}} = \boldsymbol{\omega}_m - \hat{\boldsymbol{\beta}}$$

In this case, $\boldsymbol{\omega}_m$ is the measured angular rate and $\hat{\boldsymbol{\beta}}$ is the estimated gyroscope bias, and $\hat{\boldsymbol{\omega}}$ would be the estimated angular velocity.

State Propagation

The filter must propagate its current state to the next time step. This state propagation occurs between a set of set of sensors reading, and it uses discrete time propagation. Note that the subscript $k+1$ in the equations below is used to denote the propagated quantity at the next timestep, superscript – denotes a value before a sensor update and superscript + denotes a value after a sensor update.

$$\hat{\mathbf{q}}_{\boldsymbol{k+1}}^{-} = \boldsymbol{\Theta}(\hat{\boldsymbol{\omega}}_k^{+})\hat{\mathbf{q}}_k^{+}$$

$$\boldsymbol{\beta}_{k+1}^{-} = \boldsymbol{\beta}_k^{+}$$

The discrete time quaternion propagation matrix $\boldsymbol{\Theta}$ is given by:

$$\boldsymbol{\Theta}(\hat{\boldsymbol{\omega}}_k^+) = \begin{bmatrix} \cos\left(\frac{1}{2}\|\hat{\boldsymbol{\omega}}_k^+\|\Delta t\right)\mathbf{I_{3x3}} - [\hat{\psi}_k^+ \times] & \hat{\psi}_k^+ \\ -\hat{\psi}_k^{+T} & \cos\left(\frac{1}{2}\|\hat{\boldsymbol{\omega}}_k^+\|\Delta t\right) \end{bmatrix}$$

$$\hat{\psi}_k^+ = \frac{\sin\left(\frac{1}{2}\|\hat{\boldsymbol{\omega}}_k^+\|\Delta t\right)\|\hat{\boldsymbol{\omega}}_k^+\|}{\|\hat{\boldsymbol{\omega}}_k^+\|}$$

$$\hat{\boldsymbol{\omega}}_k^+ = \boldsymbol{\omega}_m - \boldsymbol{\beta}_k^+$$

Δt represents the discrete time step in seconds. The propagated state covariance matrix, $\mathbf{P}_{k+1}^-$, is propagated accordingly:

$$\mathbf{P}_{k+1}^- = \boldsymbol{\Phi}_k \mathbf{P}_k^+ \boldsymbol{\Phi}_k^T + \mathbf{G}_k \mathbf{Q}_k \mathbf{G}_k^T$$

Here, the covariance transition matrix $\boldsymbol{\Phi}$ is given by:

$$\boldsymbol{\Phi}_k = \begin{bmatrix} \boldsymbol{\Phi}_{11} & \boldsymbol{\Phi}_{12} \\ \mathbf{0}_{3x3} & \mathbf{I}_{3x3} \end{bmatrix}$$

$$\boldsymbol{\Phi}_{11} = \mathbf{I_{3x3}} - [\hat{\boldsymbol{\omega}}_k^+ \times]\frac{\sin\left(\frac{1}{2}\|\hat{\boldsymbol{\omega}}_k^+\|\Delta t\right)}{\|\hat{\boldsymbol{\omega}}_k^+\|} + [\hat{\boldsymbol{\omega}}_k^+ \times]^2 \frac{(1 - \cos(\|\hat{\boldsymbol{\omega}}_k^+\|\Delta t))}{\|\hat{\boldsymbol{\omega}}_k^+\|^2}$$

$$\boldsymbol{\Phi}_{12} = [\hat{\boldsymbol{\omega}}_k^+ \times]\frac{(1 - \cos(\|\hat{\boldsymbol{\omega}}_k^+\|\Delta t))}{\|\hat{\boldsymbol{\omega}}_k^+\|^w} - \mathbf{I_{3x3}}\Delta t - [\hat{\boldsymbol{\omega}}_k^+ \times]^2 \frac{(\|\hat{\boldsymbol{\omega}}_k^+\|\Delta t - \sin(\|\hat{\boldsymbol{\omega}}_k^+\|\Delta t))}{\|\hat{\boldsymbol{\omega}}_k^+\|^3}$$

The process noise matrices, $\mathbf{Q}_k$ and $\mathbf{G}_k$, are given by:

$$\mathbf{Q}_k = \begin{bmatrix} \left(\sigma_\omega^2 \Delta t + \frac{1}{3}\sigma_\beta^2 \Delta t^3\right)\mathbf{I_{3x3}} & \left(-\frac{1}{2}\sigma_\beta^2 \Delta t^2\right)\mathbf{I_{3x3}} \\ \left(-\frac{1}{2}\sigma_\beta^2 \Delta t^2\right)\mathbf{I_{3x3}} & \left(\sigma_\beta^2 \Delta t\right)\mathbf{I_{3x3}} \end{bmatrix}$$

$$\mathbf{G}_k = \begin{bmatrix} -\mathbf{I_{3x3}} & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \mathbf{I_{3x3}} \end{bmatrix}$$

σ_ω^2 is the variance of the gyroscope angular rate measurement and σ_β^2 is the variance of the gyro bias. These values can be taken from the Epson-MG364 datasheet or determined experimentally.

Measurement Update

After state propagation, measurement updates for the filter's estimate of the state can be determined. Equations to find the Kalman gain matrix, $\mathbf{K_k}$, estimated error in the 3x1 vector part of the error quaternion, δq, and error in gyro bias, $\delta\boldsymbol{\beta}$, are shown and used to update the new estimate of the attitude quaternion and bias, as well as the new covariance matrix estimate.

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}_k)^{-1}$$

$$\begin{bmatrix} \delta q \\ \delta\boldsymbol{\beta} \end{bmatrix} = \mathbf{K}_k(\mathbf{z}_k - \mathbf{h}(\hat{\mathbf{q}}_k^-))$$

$$\hat{\mathbf{q}}_k^+ = \begin{bmatrix} \delta q & \sqrt{1-\delta q^T \delta q} \end{bmatrix} \otimes \hat{\mathbf{q}}_k^-$$

$$\boldsymbol{\beta}_k^+ = \boldsymbol{\beta}_k^- + \delta\boldsymbol{\beta}$$

$$\mathbf{P}_k^+ = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\mathbf{P}_k^-$$

Note that the scalar component of the error quaternion is recovered using $\sqrt{1-\delta q^T \delta q}$. This new error quaternion is added multiplicatively to the current estimate by quaternion multiplication, denoted by ⊗. The bias is updated by adding the bias error to the previous estimate.

The matrix $\mathbf{H}_k$ is the measurement sensitivity matrix, found from using each of the N reference measurement vectors $\mathbf{r}_i$ in the inertial frame:

$$\mathbf{H}_k = \begin{bmatrix} 2[A(\hat{\mathbf{q}}_k^-)\mathbf{r}_1 \times] & \mathbf{0}_{3x3} \\ \vdots & \vdots \\ 2[A(\hat{\mathbf{q}}_k^-)\mathbf{r}_N \times] & \mathbf{0}_{3x3} \end{bmatrix}$$

Again, the inertial reference directions come from the different models that rely on GPS position outputs.

$\mathbf{R}_k$ is the sensor noise matrix, which is shown below:

$$\mathbf{R}_k = \begin{bmatrix} \sigma_1^2\ \mathbf{I_{3x3}} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_N^2\ \mathbf{I_{3x3}} \end{bmatrix}$$

σ_i^2 is the measurement variance for the ith sensor used. This concept is also used in the weighting matrix for the sun sensor estimate: a lower σ_i^2 indicates more trust. As with the sun sensor estimation technique, if a sensor estimate is missing, the variance for the unavailable sensor may be set to an arbitrarily large number, such as 10^{10}. This will cause the filter to ignore the invalid measurement and just "trust" the other measurements as specified.

The Kalman gain matrix is multiplied by the difference between the real sensor measurements and the estimated sensor measurements $\mathbf{h}(\hat{\mathbf{q}}_k^-)$ to find the estimated state error for the attitude quaternion and gyro bias. With N measurement vectors $\mathbf{b}_i$ and their corresponding reference vectors in the inertial frame $\mathbf{r}_i$, $\mathbf{z_k}$, and $\mathbf{h}(\hat{\mathbf{q}}_k^-)$ are defined:

$$\mathbf{z}_k = \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_N \end{bmatrix}$$

$$\mathbf{h}(\hat{\mathbf{q}}_k^-). = \begin{bmatrix} A(\hat{\mathbf{q}}_k^-)\mathbf{r}_1 \\ \vdots \\ A(\hat{\mathbf{q}}_k^-)\mathbf{r}_N \end{bmatrix}$$

Detumble

Because of the docking and unloading mechanisms for CubeSats, there will be some initial tumbling when the CubeSat is deployed for its mission. A detumble algorithm must be utilized to gradually eliminate this spin with the use of the magnetic torque rods and magnetometer. There may be other times where the spacecraft is tumbling due to science operations, and if necessary,

the detumble algorithm can be used in those circumstances as well. For this mode, the only torques that the ADCS supplies to the spacecraft are through the magnetic torque rods. Two methods of control laws can be used here, but only the first one will be used on-orbit.

The first control law that can be used relies on only magnetometer estimates and uses a common B-dot algorithm. Angular velocity is determined by the time rate of change in the magnetic field vector in the body frame. The control law and the equation for calculating the time rate of change of the magnetic field vector are presented below:

$$\mathbf{m} = -k\dot{\mathbf{b}}$$

$$\dot{\mathbf{b}} = \frac{\boldsymbol{b}_{k+1} - \boldsymbol{b}_k}{\Delta t}$$

m is the commanded dipole moment, *k* is the control gain, and $\dot{\mathbf{b}}$ is the time rate of change of the magnetic field vector. This rate is calculated by taking the difference of two consecutive magnetometer outputs divided by the time interval between the samples. Using a moving average filter or some other type of low-pass filter will help reduce noise in the magnetic field vector outputs.

The second control law presented uses a magnetometer estimate and angular velocity estimate, $\boldsymbol{\omega}$, to determine the commanded dipole moment. The law is presented below:

$$\mathbf{m} = \frac{k}{\mathbf{b}^T\mathbf{b}}(\boldsymbol{\omega} \times \mathbf{b})$$

This controller is more efficient in theory as the moment is applied perpendicular to the angular velocity of the spacecraft, so this maximizes the rate damping. However, this method is more complicated as it requires more sensors. In addition to the gyro on the IMU, getting an angular velocity estimate requires a Kalman filter, which requires additional sensor output (from the sun sensor or horizon sensor), and they might not be available at the time. Using a Kalman filter would also be more complex computationally and since there could be convergence issues, it would be a lot simpler to use the first control law. The tradeoff here would be that it takes more time for detumble to reduce and eliminate the angular rates, but if the chance of success is higher, the first control law should be used. The *control.py* file in *adac-kit* and the *detumble.py* file in *adcs-proto* can be referenced for more information.

The detumble algorithm with the first control law was tested on the TARGIT engineering unit. For this case, the spacecraft was placed on the air-bearing in the Helmholtz cage and was given some initial spin about its z-axis. Data was recorded from various sensors, and a graph showing the angular rates of the CubeSat over time is shown below:

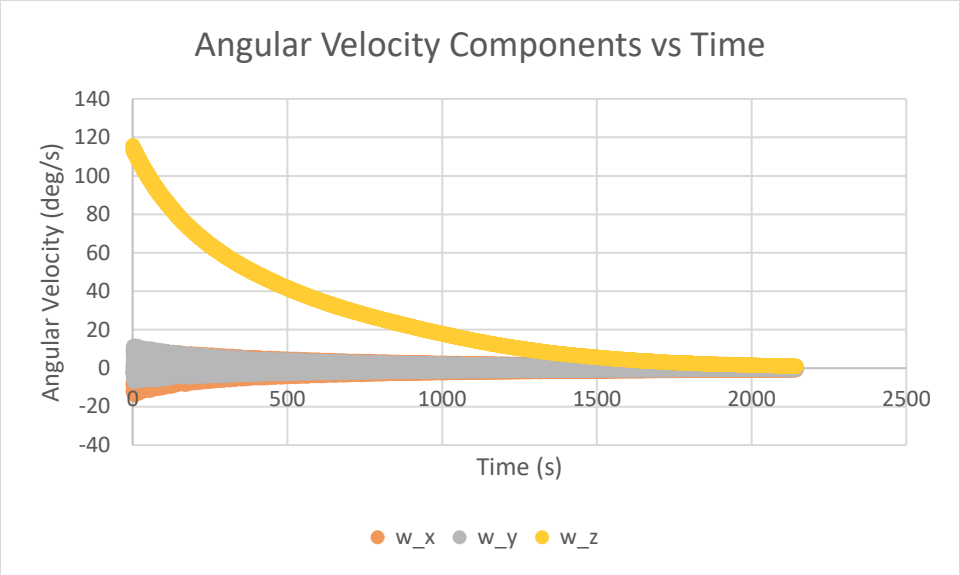


Figure 3. Angular velocity plotted over time during detumble algorithm implementation.

Because of the nature of the setup, it is hard to isolate the initial spin to just the z-axis as it was done by hand. Consequently, it is seen that there is some initial wobble, resulting in a spin, but a much smaller one, about the other axes. Regardless, the detumble algorithm reduces spin across all axes to essentially zero over time. It takes roughly 2200 seconds, or about 37 minutes, which is expected considering this controller is not as efficient as a controller that also takes angular velocity estimates.

Sun Pointing

In order to maximize the charging efficiency of TARGIT's solar panels, a sun-pointing mode is used to orient the spacecraft such that the incidence angle of the solar panels to the sun is normal (90º) and maintain this pointing configuration. The ADCS sun pointing mode uses feedback from the fine and coarse sun sensors to actuate the reaction wheels to minimize any error in pointing. The control method uses a proportional derivative (PD) feedback controller to regulate the angle between a reference vector and the observed sun vector. The reference vector is defined in the body frame to align the solar panels to the sun.

As with the detumble algorithm, two control laws can be used: one that does not use angular velocity estimates, and one that does. First, some initial definitions must be made. Given the reference vector in the body frame, **e**, the measured sun vector, **s**, the error signal, **θ**, can be calculated as:

$$\boldsymbol{\theta} = \cos^{-1}(\mathbf{e}^T\mathbf{s})\frac{\boldsymbol{\varepsilon}}{\|\boldsymbol{\varepsilon}\|}$$

$$\boldsymbol{\varepsilon} = [\mathbf{e}\times]\mathbf{s}$$

ε represents the axis of rotation from **e** to **s**. Using this error signal, a PD controller can be used to calculate the torque that must be exerted by the spacecraft:

$$\boldsymbol{\tau} = k_p\boldsymbol{\theta} + k_d\dot{\boldsymbol{\theta}}$$

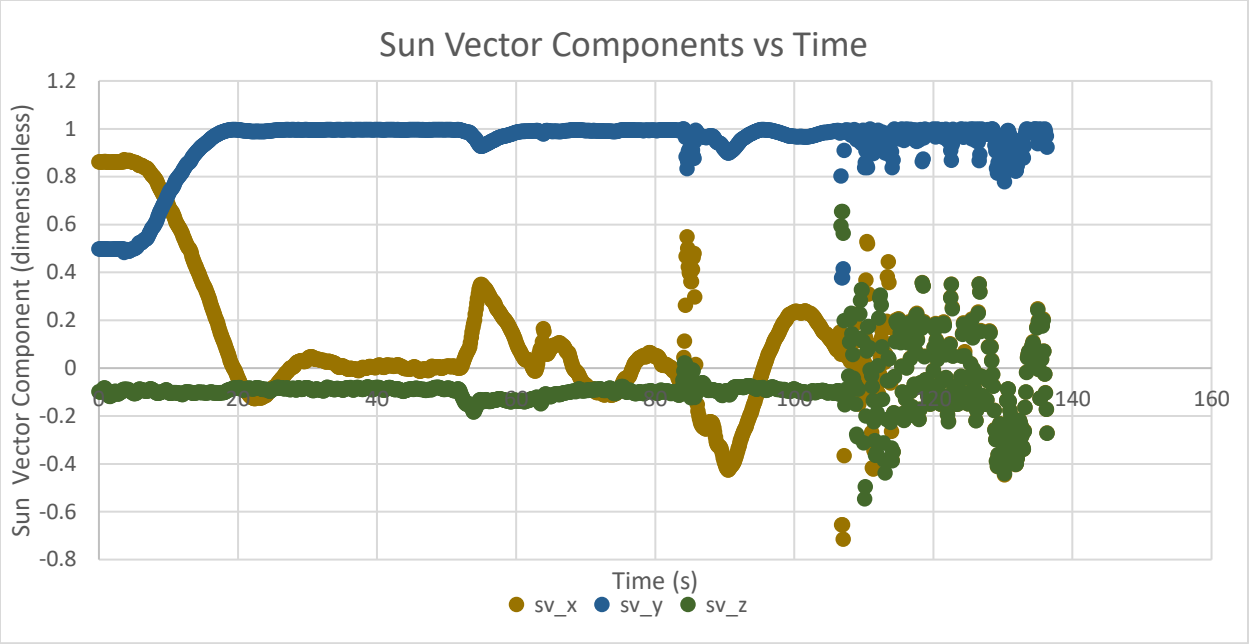
k_p and k_d represent the proportional and derivative gain of the controller. Because an angular velocity estimate is not used, a slight spin may be induced about the sun vector in a steady state condition, (less than 1 deg/s). If angular velocity can be estimated, a more robust control law can be used. Again, to estimate angular velocity, a Kalman filter must be utilized and can be more computationally and power hungry because of all the software and hardware that must be employed. The control law is shown below:

$$\boldsymbol{\tau} = k_p\boldsymbol{n}\sin(\boldsymbol{\theta}) + k_d\boldsymbol{\omega} + [\boldsymbol{\omega}\times](\mathbf{J}\boldsymbol{\omega} + \mathbf{h})$$

$$\boldsymbol{n} = \frac{[\mathbf{e}\times]\mathbf{s}}{\sin(\boldsymbol{\theta})}$$

Here, **n** is the unit vector along the axis of rotation from **e** to **s, ω** is the angular velocity estimate determined by the Kalman filter, and **h** is the reaction wheels' internally stored angular momentum. Because **h** affects the gyroscope's reading, it is included as a correction term. To improve the accuracy of these controllers and reduce jitter, simple signal processing methods such as a moving average filter or low-pass filter can be implemented to filter the measured sun angle after the weighted least squares method is used.

Shown below are the results of using the sun-pointing algorithm with the engineering unit and a strong lamp to act as the sun. The sun vector components are plotted over time as a unit vector. The spacecraft spins about the sun sensor's y-axis axis, which is why the y-component is essentially fixed over time near 0. It is not exactly 0 since there was some height difference between the CubeSat and the lamp. To align the sun vector with the "sun", the unit vector must be almost all in the z-axis, which occurs here over time, as the x-component drops to 0. This means that the sun sensor was facing normal to the "sun". At the end, there is some very interesting information plotted. At this point, the light was turned off to see what the data would record, and it is shown to be very spread out and unusable. Care must be taken to not include sun vector data when it cannot see the sun or else it will cause a lot of bad data to be passed into the algorithms used.



More information can be found in *sun_point.py* and *sun_point_alt.py* in adcs-proto.

LVLH Pointing

The LVLH pointing algorithm, as described earlier, is used to maintain the spacecraft in a flying wing configuration for nominal operations. This consists of the deployable solar panels pointed zenith and the laser payload pointed anti-ram. The full suite of sensors and actuators will be used, although the Kalman filter will switch between using sun sensor data and horizon sensor data, depending on whether the spacecraft is in sunlight or in eclipse. The flow diagram below shows the relationships between the sensor outputs and software inputs, and the relationships between the software outputs and actuator inputs.

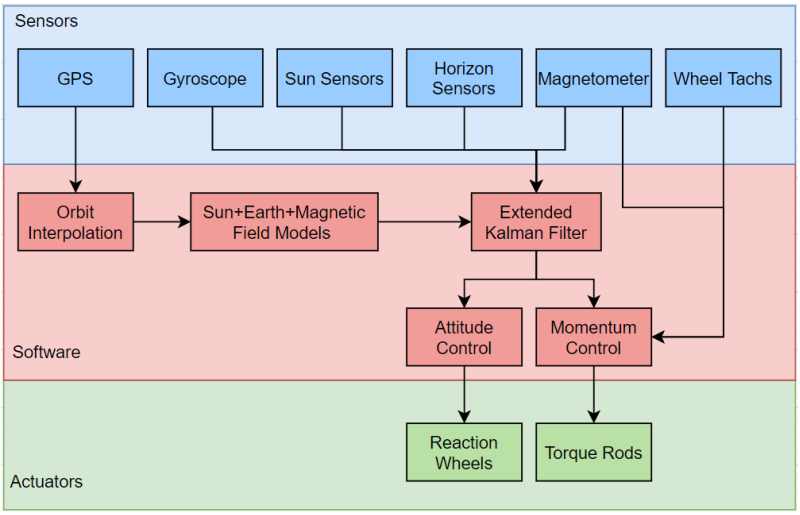


Figure 4. Information flow for the LVLH Pointing Algorithm.

As mentioned earlier, the Kalman filter will be used to provide an estimate of the vehicle's current orientation in the inertial frame as well as gyro bias. The attitude control algorithm for LVLH pointing uses the attitude estimate and gyro bias to get torques to maintain LVLH, which are then applied to the reaction wheels. Momentum management software, described more later, is then used to actuate the magnetic torque rods.

The control law for LVLH uses a quaternion PD feedback controller. A reference quaternion in the inertial frame, $\mathbf{q}_{ref}$, and reference angular rate, $\boldsymbol{\omega}_{ref}$, is specified for the controller beforehand. The error quaternion is calculated by $\boldsymbol{\delta q} = \mathbf{q} \otimes \mathbf{q}_{ref}^{-1}$, where $\mathbf{q}$ is the current quaternion in the inertial frame. The angular rate error is $\boldsymbol{\omega}_{e,} = \boldsymbol{\omega} - \boldsymbol{\omega}_{ref}$ where $\boldsymbol{\omega}$ is the spacecraft angular velocity. The desired torque in the body frame is:

$$\boldsymbol{\tau} = -\boldsymbol{k}_p \text{sign}(\delta q_4) \boldsymbol{\delta q}_{1:3} - \boldsymbol{k}_d \boldsymbol{\omega}_e - [\boldsymbol{\omega} \times](\mathbf{J}\boldsymbol{\omega} + \mathbf{h})$$

$\boldsymbol{k}_p$ is the proportional gain vector, $\boldsymbol{k}_d$ is the derivative gain vector, and $\mathbf{h}$ is the internal momentum vector from the reaction wheels. $\mathbf{J}$ is the inertia tensor for the spacecraft. The sign function for δq_4 helps to ensure that the shortest path to the desired orientation is taken, rather than rotating about the opposite direction to achieve the same orientation if it is longer. The gains $\boldsymbol{k}_p$ and $\boldsymbol{k}_d$ can be designed by selecting a controller bandwidth ω_n and damping ratio ζ:

$$\boldsymbol{k}_p = \omega_n^2 \mathbf{J}$$

$$\boldsymbol{k}_d = 2\zeta\omega_n \mathbf{J}$$

ω_n should be at least 10 times smaller than the controller update frequency, while ζ can be tuned to achieve the desired response.

$\mathbf{q}_{ref}$ for LVLH pointing can be calculated by using a transformation from the inertial frame to the local frame at the spacecraft's current position. The estimate of the spacecraft's inertial position and velocity, $\mathbf{r_I}$ and $\mathbf{v_I}$, respectively, are used to the LVLH frame to the inertial frame through the following calculations:

$$A_L^I = [\boldsymbol{o}_1 \quad \boldsymbol{o}_2 \quad \boldsymbol{o}_3]$$

$$\boldsymbol{o}_1 = \boldsymbol{o}_2 \times \boldsymbol{o}_3 \quad \boldsymbol{o}_2 = \frac{-(\boldsymbol{r}_I \times \boldsymbol{v}_I)}{\|\boldsymbol{r}_I \times \boldsymbol{v}_I\|} \quad \boldsymbol{o}_3 = \frac{-\boldsymbol{r_I}}{\|\boldsymbol{r_I}\|}$$

The orientation relative to the local frame in the inertial frame can be defined by chaining together rotations with A_I^L, which also is equal to ${A_L^I}^T$.

Target Tracking

As mentioned earlier, target tracking consists of using the Pixycam to track the target with the Pixycam such that the laser ranging system can also detect the target, and this algorithm is a subcase of the LVLH pointing algorithm. This mode will use the same attitude estimation methods as LVLH, with target pointing being the main control algorithm used. This will be achieved through a boresight control law that uses visual feedback from the Pixycam as it tries to center the target in its field of view. The Pixycam outputs the x and y distances of the bounding box of the

target, which will then be used as errors that need to be minimized through control law. The image below shows the target in the Pixycam's field of view and shows the axes used to calculate error.

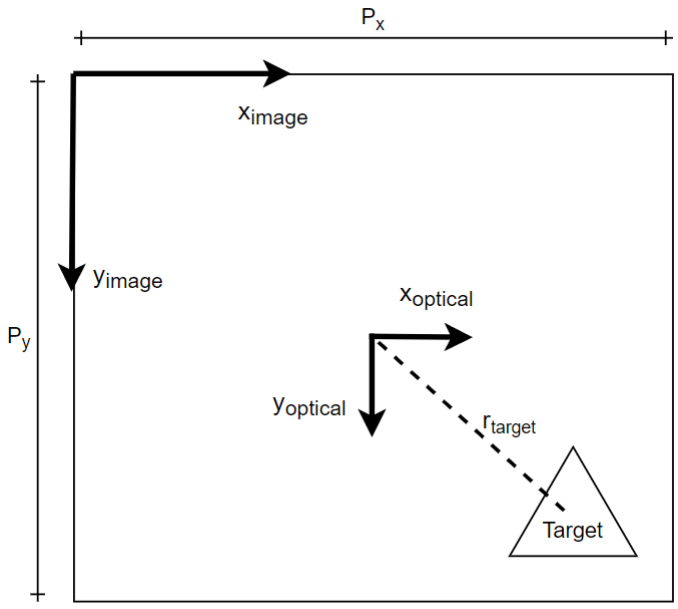


Figure 5. Pixycam's field of view with camera axes, optical axes, and target position.

The position error of the target is found by getting $\boldsymbol{r_{target}} = [x_{target} \quad y_{target}]$, which is the target's position in pixels relative to the optical center of the field of view. The angle errors in the camera frame are:

$$\theta_x = y_{target}\frac{FOV_y}{P_y}$$

$$\theta_y = -x_{target}\frac{FOV_x}{P_x}$$

θ_x and θ_y correspond to the angle errors about the x and y axes, respectively. The FOV is the camera's field of views in radians, and P represents the number of pixels in the x and y directions. The following table can be used as an example, but may differ for the actual Pixycam unit:

Table 3. Example information for the field of view and resolution of the Pixycam.

Field	X	Y
Field of View (deg)	60	40
Resolution (Pixels)	315	207

The error vector in the camera's optical frame, $\boldsymbol{\theta_{optical}} = [\theta_x \quad \theta_y \quad 0]$, is used in a PD feedback controller to drive the error vector to 0. From hardware-in-the-loop (HWIL) testing, it was found that the best way to form the error vector was to use a moving average of the position

of the target. A sample of 12 measurements are taken and the average is taken to get the position of the control sample. The rate of change in position is calculating by taking the difference between the averages of the first six measurements and the averages of the last six measurements. In equation form, the rate of change for x and y would be:

$$\dot{x} = \frac{mean(x_{7:12}) - mean(x_{1:6})}{6\Delta t}$$

$$\dot{y} = \frac{mean(y_{7:12}) - mean(y_{1:6})}{6\Delta t}$$

Using these average values, $\boldsymbol{\theta_{optical}}$ and $\dot{\boldsymbol{\theta}}_{\boldsymbol{optical}}$ are calculated and then rotated to the body frame using:

$$\boldsymbol{\theta_{body}} = \boldsymbol{R}^{\boldsymbol{body}}_{\boldsymbol{optical}}\boldsymbol{\theta_{optical}}$$

$$\dot{\boldsymbol{\theta}}_{\boldsymbol{body}} = \boldsymbol{R}^{\boldsymbol{body}}_{\boldsymbol{optical}}\dot{\boldsymbol{\theta}}_{\boldsymbol{optical}}$$

The body frame error vectors then can be fed into the PD controller to calculated the desired torque on the body frame, to be actuated by the reaction wheels:

$$\boldsymbol{\tau} = k_p\boldsymbol{\theta} + k_d\dot{\boldsymbol{\theta}} + [\boldsymbol{\omega}\times](\mathbf{J}\boldsymbol{\omega} + \mathbf{h})$$

The proportional and derivative gains are k_p and k_d, respectively, the angular velocity estimate is **ω**, the inertia tensor is **J**, and angular momentum stored in the reaction wheels is **h**. Once the controller has gotten the CubeSat to stay on the target, it will then try to roll such that the sun incidence angle to the solar panels is optimized for charging. This event will only be allowed when $\left\|\boldsymbol{\theta_{optical}}\right\| < 0.03\,rad$, to make sure that the target tracking portion of this mode is not affected. The optimal roll angle, θ_{roll}, is calculated by:

$$\theta_{roll} = \text{atan2}\left(\frac{\left\|\mathbf{n}\times\mathbf{s}_{proj}\right\|}{\mathbf{n}^T\mathbf{s}_{proj}}\right)$$

n is the vector normal to the solar panels, while $\mathbf{s}_{proj}$ is the project sun unit vector in the plane perpendicular to the direction of the Pixycam and laser. The time rate change of θ_{roll} can be calculated using a finite difference method similar to the equation for $\dot{\mathbf{b}}$ for detumble. θ_{roll} and $\dot{\theta}_{roll}$ can be added to $\boldsymbol{\theta_{optical}}$ and $\dot{\boldsymbol{\theta}}_{\boldsymbol{optical}}$ in the body frame, respectively.

The *target_track.py* file in *adcs-proto/apps* can be referenced for more information.

Momentum Management

Momentum management is required because the reaction wheels have a maximum value for the amount of momentum that can be "stored" in the wheels. In this algorithm, the magnetic torque rods are used to desaturate the amount of momentum in the reaction wheel. A proportional controller is used to drive down the reaction wheel momentum vector to the desisred quantity. The error in reaction wheel momentum and magnetic field vector is calculated in the body frame and used to command the torque rods by creating the magnetic dipole moment, **m**:

$$\boldsymbol{m} = \frac{k_m}{\mathbf{b}^T\mathbf{b}}[\mathbf{h}_{err} \times]\mathbf{b}$$

k_m is the proportional gain, $\mathbf{h}_{err} = \mathbf{h} - \mathbf{h}_{ref}$ (the reaction wheel angular momentum vector subtracted by the desired reaction wheel angular momentum vector), and $\mathbf{b}$ is the magnetic field vector. The resulting torque is calculated below, where $\mathbf{I_{3x3}}$ is the identity matrix:

$$\boldsymbol{\tau}_m = -k_m(\mathbf{I_{3x3}} - \mathbf{b}\mathbf{b}^T)\mathbf{h}$$

The *control.py* can be referenced for more information.

Acknowledgements

I would like to thank Dr. Gunter for providing the opportunity to work on this project and providing funding and support throughout the years. I would also like to also thank everyone who has worked on the ADCS sub-team over the past several years, especially the current team that is helping the CubeSat get to launch.