TableSat Project

Sven Giorno, Enoch Lee

December 15, 2017

Contents

Introduction

TableSat is a 1-DOF satellite simulation platform driven by two computer fans that apply + and − torque, respectively. It also carries different sensors: a gyrometer, three magnetometers and four light sensors. The main purpose of this project is to implement programs in order to control in real time the rotational rate and the heading of the system.

In the first section, we will address the different controllers that we tried in order to get a stabilized rotational rate. In the next sections, we will interest ourselves in the calibration of the different sensors available. Once the calibration is achieved, we will address in the last section our implementation of a controller allowing to control the heading of the satellite.

Note that all of the plots shown in this report can be reproduced by running the MATLAB codes using the data text files included with the report.

1 Rate Control

1.1 Tested control laws

In order to stabilize the angular velocity, we tried different controllers.

1.1.1 Bang-bang controller

This is the simplest controller we can use. The idea is to use a constant thrust acceleration, with a positive input if the error is negative, and a negative input if the error is positive. Once the error is lower than a given tolerance, we maintain the thrust for 30s. If during this lapse of time, the error grows higher than the tolerance, we re-enter the control loop. Otherwise, we enter the goal state. This process is illustrated in the hybrid state automaton represented in Figure 1. However, this controller did not work on this system. It relies on the assumption that a given

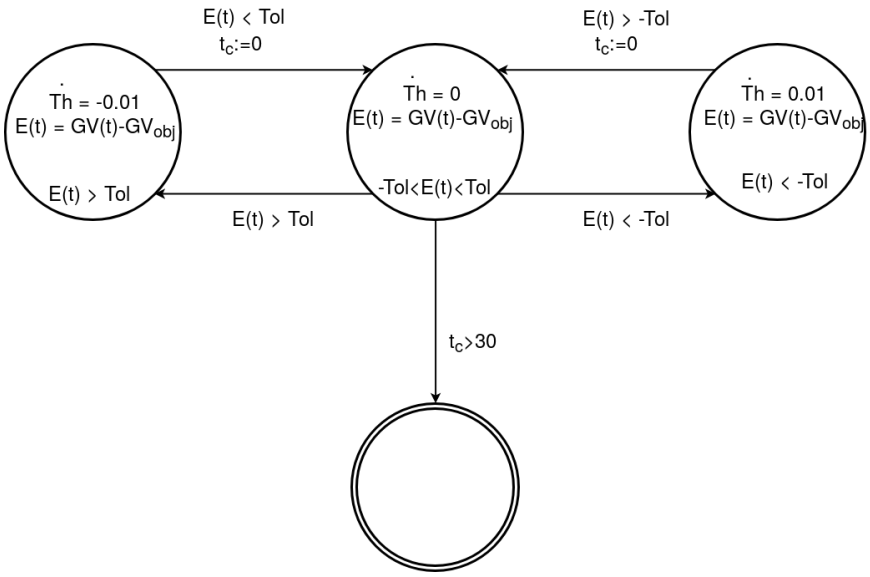


Figure 1: Graphical representation of the bang-bang controller as an automaton. Th is the thrust, $E(t)$ is the error and Tol is a user specified tolerance.

thrust is equivalent to only one angular speed. This assumption has been proven to be false: even though there is a correlation between the absolute value of the thrust and the rotational rate of TableSat, a constant thrust is far from producing an approximately constant angular velocity and hence the controller failure.

1.1.2 Proportional controller

We then needed to make the controller more robust, and so, more complex. We thus tried a proportional controller chose to constantly update the thrust, even during the 30s stabilized phase.

The proportional controller has one main advantage over the previous one: the thrust acceleration is not a constant anymore. This property allows for the calibration of the response to the error: the higher the error, the greater the increase in the thrust. As such, the control is able to to reach the desired value faster.

This advantage comes at a cost, though: the response is based on a proportionality constant, K_p, which needs to be finely tuned. If this coefficient is too high, we will reach the desired value quickly, but stabilize quite slowly because of strong oscillations. On the other hand, if the coefficient is too low, it will take a lot of time to reach the desired value. These cases are illustrated in Figure 2.

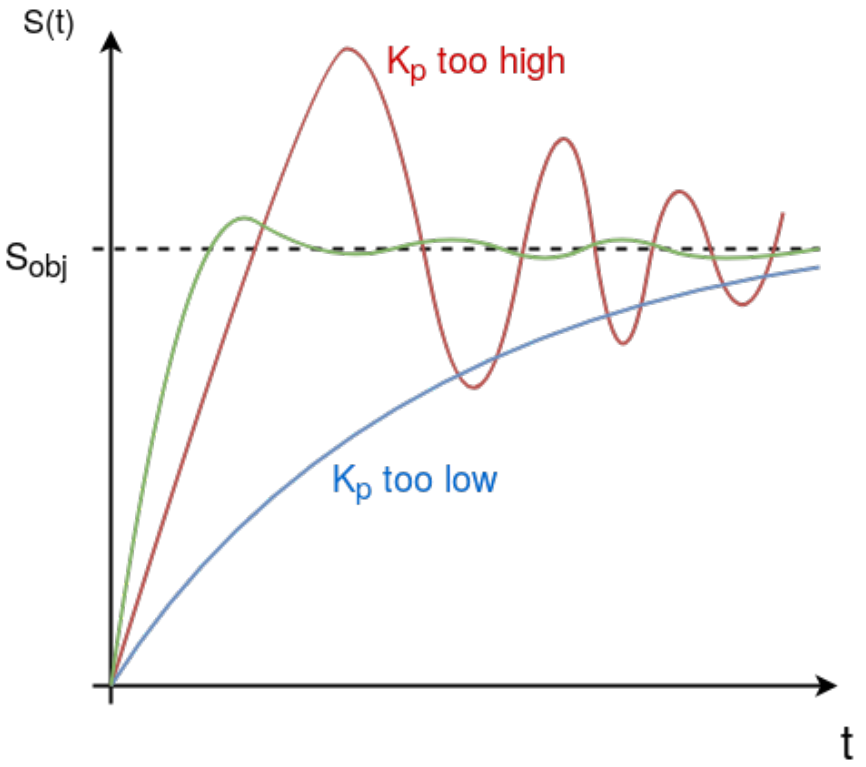


Figure 2: Measured signal using a proportional controller for different proportionality constants.

While this controller was working on TableSat, the stabilization took too much time. We thus decided to try a slightly more complex controller.

1.1.3 PID controller

The PID controller is an improvement to the proportional controller. It allows for a finer control over the command by taking into account the integral and the differential error. The purpose of the integral term is to decrease the amount of time needed to reach the range of the desired value, by taking into account all of the previous errors. The differential term, on the other hand, helps to damp the oscillations around the desired value to get a quicker stabilization by counterbalancing the change in the command due to the proportional and integral controllers.

Once again, the downside of this controller is that each coefficient has to be tuned to get an efficient stabilization. The procedure to calibrate the parameters is as follows:

1. Select $K_i = K_d = 0$ and $K_p > 0$. Run the controller and adapt K_p based on its output: if the rotational rate increases too slowly, increase K_p ; on the contrary if the rotational rate increases too quickly and largely exceeds the desired value, decrease K_p.

2. Once K_p has been properly tuned, keep $K_d = 0$ and select a non-zero value for K_i. This coefficient is supposed to shorten the time it takes to get from 0 to the desired output, before the stabilization phase. Since K_i is controlling the integral of the error, it should stay quite low compared to K_d, otherwise it will introduce an inertia in the controller, which will prevent it to stabilize.

3. Finally, select a non-zero value for K_d. It must be high enough to damp the oscillations during the stabilization phase, without slowing down the ramp-up phase. Since this coefficient is controlling the derivative of the error, it should be much higher than K_p – otherwise it won't have any appreciable effect.

1.1.4 Final controller

The controller we decided to use is a slightly modified version of the PID controller. Based on our experiment with the system, we observed that there was a quite high latency between a thrust input and a change in the rotational rate. Hence, instead of coding a "continuous" version of the controller, we implemented a "semi-continuous" version, updating the thrust every 0.5s, to take this latency into account.

Furthermore, because of the large dead-band for the thrust input, we are starting at a non-zero thrust, corresponding to a small rotational rate. We run this constant initial non-zero thrust for 30s – in order to reach a more or less stable rotational rate – before entering the control loop. This allows the ramp-up phase to be shorter.

The controller we used is illustrated in Figure 3 below.

In our code, we used the following parameters: $K_p = 0.013$, $K_i = 0.00002$ and $K_d = 0.22$. The next subsection is presenting the results we had in one run using this controller.

1.2 Results

Figures 4a-4c are showing the results provided by the file `rate_control.cpp` for three different commands.

As we can see on these plots, the controller has been properly tuned: the ramp-up phase takes about 1.5min, but gets exactly to the desired output without any high oscillations during the stabilization phase.

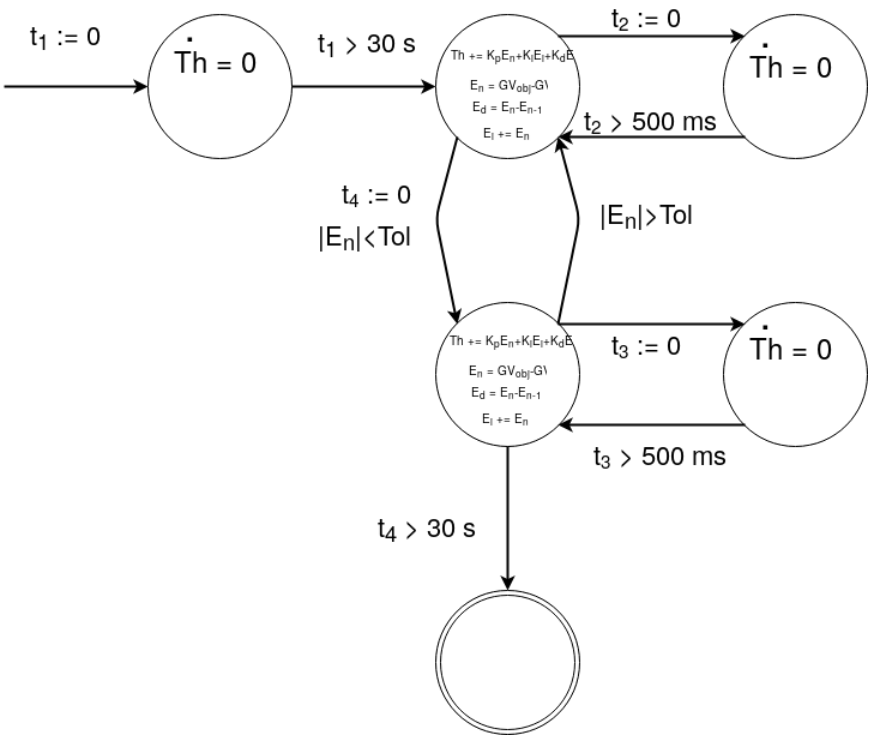


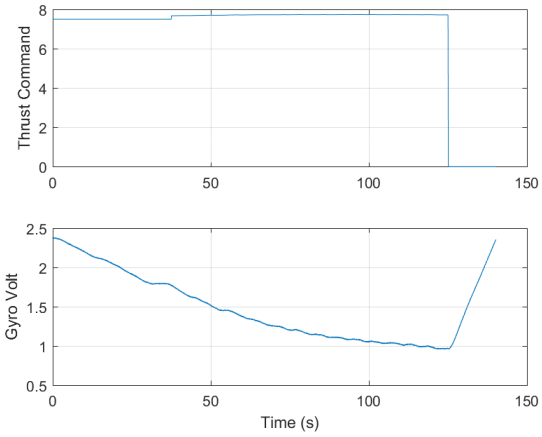
Figure 3: Final rate controller.

2 Data Collection

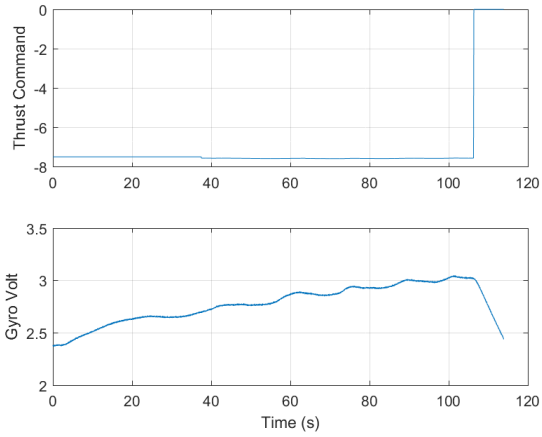
The data collection is done using the `daq.cpp` function which calls the `rate_control.cpp` file within `rate_control.cpp` that controls the TableSat and commands it to reach a desired speed, measured in gyro-volts. A gyro-volt of 3.5 is used to generate all the plots used in this section and to illustrate the effects of moving-average filters. The function captures time, rate gyro, magnetometer, and coarse sun sensor data in the data file `tsat_data.txt` at 200Hz. Using MATLAB, the data is then extracted and plotted.

Both data sets from magnetometers and CSS exhibit periodic behavior, with that of the magnetometers showing a sinusoidal behavior and that of the CSS showing step function behaviors. The data from Mag3, the magnetometer pointing towards the z-axis, is producing notably nosier data than its x- and y- axis (Mag1 and Mag2) counterparts. On the other hand, data from the sun sensors are relatively consistent across the board. In our experience, sun sensors produce consistent readings across all tests while that of magnetometers are relatively inconsistent. Typically, inconsistencies occur when the battery is changed recently or any physical changes and adjustments are made to the TableSat. In all tests, however, Mag3's data are consistently noisier than those of Mag1 and Mag2.

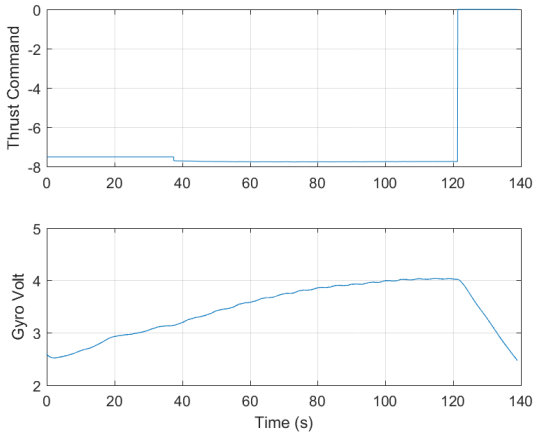
A moving average filter with various window sizes is applied to gauge the effectiveness of the filter. The results for filters with window sizes $n = 5$ and $n = 10$ are plotted below. The moving average filter calculates the value at the current timestep by averaging it with n immediate neighbors



(a) 1 gyro-Volt.



(b) 3 gyro-Volts.



(c) 4 gyro-Volts.

Figure 4: Thrust command (in V) and rotational rate (in gV) against time for different gyro-Volts commands.

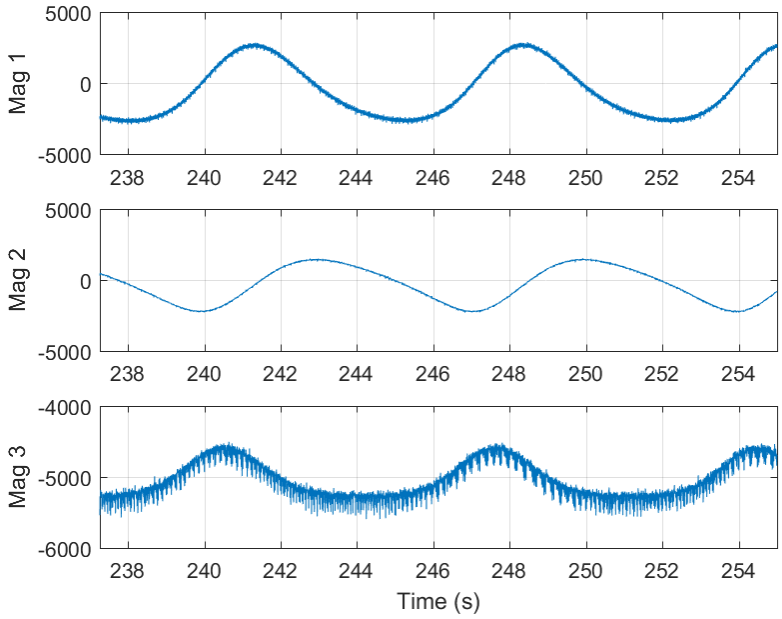


Figure 5: Magnetometer Readings during the stabilized phase (3.5 Gyro-volt).

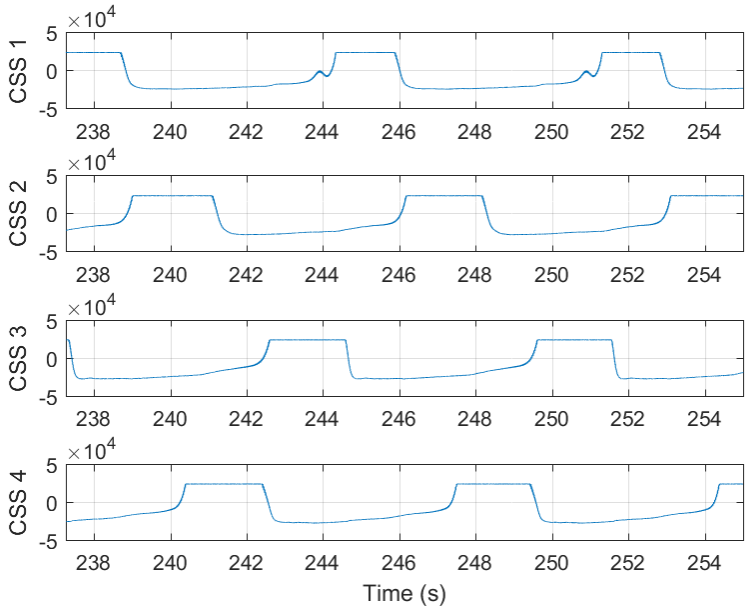
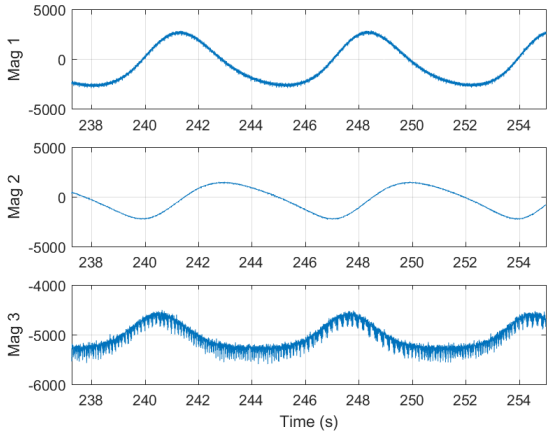
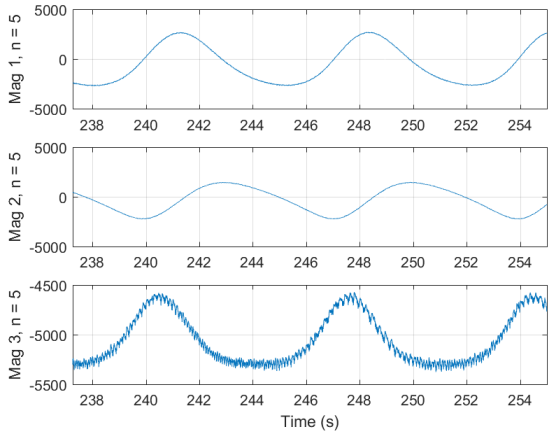


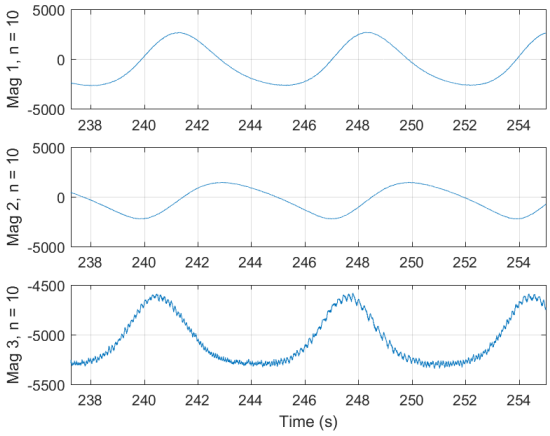
Figure 6: CSS Readings during the stabilized phase (3.5 Gyro-volt).



(a) $n = 0$ (no filter).



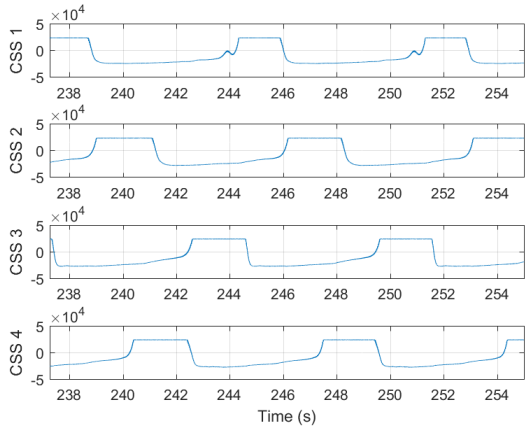
(b) $n = 5$.



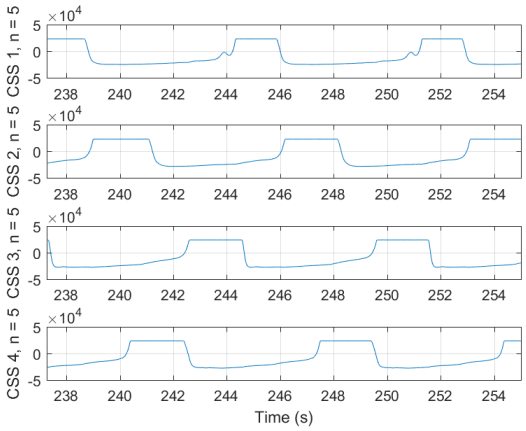
(c) $n = 10$

Figure 7: Magnetometer Readings during the stabilized phase (3.5 Gyro-volt) with a moving average filter using n consecutive data points.

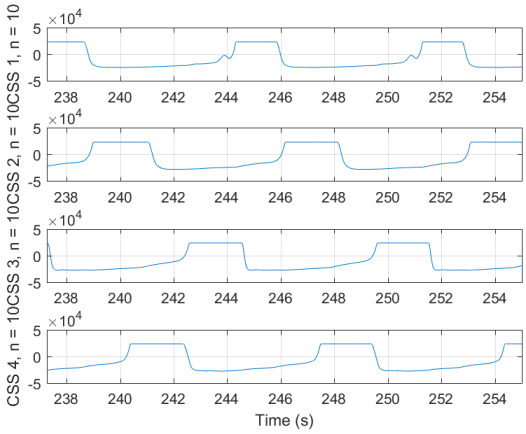
in order to reduce the impact of noise. In the `movingAvg.m` MATLAB script, the moving average filter takes in the previous $n-1$ values in order to estimate the current value. Increasing window size of the filter smooths out the jagged noises in the data set yet shifting it temporally and depressing the amplitude. A window of $n = 5$ clears out noises in Mag3 significantly while the smoothing effect of a filter with a window of $n = 10$ is even more pronounced. However, the mechanism of the calibration method used in this project makes a moving filter redundant for calibration. This will be further discussed in the later sections.



(a) $n = 0$ (no filter).



(b) $n = 5$.



(c) $n = 10$

Figure 8: Sun sensors Readings during the stabilized phase (3.5 Gyro-volt) with a moving average filter using n consecutive data points.

3 Gyro MKS Unit Calibration

Using the the available sensors, we can define a reference point for TableSat. This reference point will allow us to measure the period of rotation of the satellite, T. Assuming the angular velocity is constant over the whole period, this will then give us a point linking gyro-Volts and degrees per second ($\frac{360}{T}$). Repeating this process for different gyro-Volts commands will give us several points on which we will be able to fit a linear function. Figure 9 shows the results for the linear calibration of the gyrosensor. Note that these results have been made for only one rotational direction. The calibration for the opposite direction is the symmetric of this plot with respect to the axis $y = 2.40$.

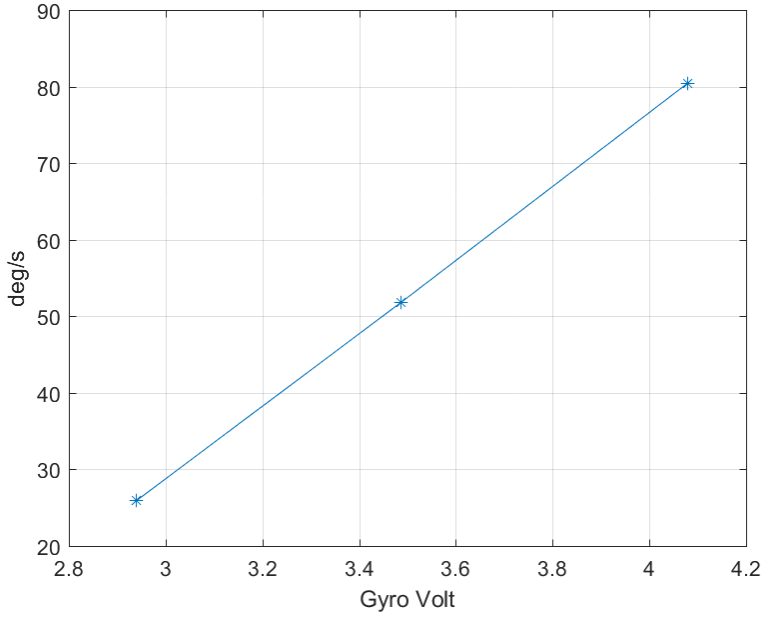


Figure 9: Linear calibration for the gyrosensor.

Using MATLAB, we can output the linear equation for the calibration. In this case, we obtained:

$$\omega = 47.8604GV - 114.8784 \tag{1}$$

where ω is the rotational rate and GV, the gyro sensor reading.

4 Magnetometer Calibration

The calibration for the magnetometer starts with the data acquired from the `daq.cpp` function where the TableSat is spun for a period of time. Calibration looks only at the period of data where the TableSat is stabilized, that is, gyro-volt reading is within ± 0.1 of the user-inputted desired gyro-volt. Due to the mechanism (to be discussed below) of the calibration, a calibration based on a relatively high gyro-volt is needed, that is, at about 3.5 gyro-volt for TableSat. The data is then imported into MATLAB where further processing is carried out. After the data processing is done, MATLAB outputs a text file that contains all the coefficients needed for the C function which uses these coefficients and translates magnetometer readings into heading angle.

4.1 MATLAB Calibration

The MATLAB portion of the calibration takes in the unfiltered raw data from `daq.cpp` and processes it further by curve-fitting the data and attempts to model the data with an explicit mathematical function. A filter is not needed since the magnetometer data is not noisy enough to warrant a filter. A trend can be easily discerned and the spread of the data is found not to be too far off from the curve-fit. Moreover, it is found that the results of the calibration function are more or less the same, with or without filters of varying sizes. As such, a moving-average filter does not necessarily improve the accuracy nor precision of the curve-fit and the choice of not using any filters was made. The data from Mag3 (pointing at the z axis) can be somewhat inconsistent, dependent on current status of the hardware. It is suspected that this is due to the orientation of this particular magnetometer.

Two different curve fits are considered in the calibration process: namely `fourier8` and `sin8` . The Fourier series model `fourier8` is given by

$$f(t)_{\texttt{fourier8}} = a_0 + \sum_{n=1}^{8}(a_n \cos(npt) + b_n \sin(npt)) \tag{2}$$

where a and b are coefficients generated by the MATLAB fit function and p is a constant generated by the MATLAB fit function and is given by

$$p = \frac{2\pi}{\max(xdata) - \min(xdata)} \tag{3}$$

The Sine model `sin8` is given by

$$f(t)_{\texttt{sine8}} = \sum_{n=1}^{8}(a_n \sin(b_n t + c_n)) \tag{4}$$

where a, b, and c are coefficients generated by the MATLAB fit function to curve-fit the data.

To choose which model to use to curve-fit the given data, the norms of both models are compared. The model with the lowest norm value, and therefore has the least deviation, is chosen

automatically by the MATLAB code and the corresponding coefficient output is exported as a text file ready to be used by the C function to complete the calibration process. The period is obtained by first finding the mean in the inputted data and marking the timestamp whenever the fitting curve passes through this point from below or above. A non-maximum/minimum point is passed through three times in one period and based on this information, MATLAB extracts the period of the fitting model. As the system is not a perfect system and period might fluctuate even though the TableSat has been stabilized, the MATLAB takes in period data from several revolutions and average these values to obtain an average period. Hence, in order to collect meaningful data, the period thus must be relatively constant within this stabilized phase.

The stabilized period is set to be 30s from rate control. This however, cannot be lengthened by too much. Since the stabilized phase is defined as the period of time where gyro-volt reading is within ± 0.1 of the desired value and that actual reading fluctuates within this range even with an adaptive feedback controller, the sampling can easily drop out of the stabilized phase, rendering the data useless. In our experience, a lengthened stabilized phase will lead to multiple gyro-volt out-of-bounds scenarios within the stabilized phase. As such, a short stabilized period for meaningful data collection is needed. In order to pack as many useful periods of data as possible within the short time frame, a relatively high gyro-volt rate is thus required. Yet, the TableSat cannot be spinning too fast as that will result in useless calibration data for the subsequent pointing operations since the controller is designed to operate at a lower speed (to be discussed in the later section).

The text file contains 3 columns, with the first one corresponds to the coefficients for magnetometer 1, the second column corresponds to the coefficients for magnetometer 2, and the third column corresponds to the coefficients for magnetometer 3. In our case, though, we decided at the end not to use the third magnetometer because of the unreliability of it output. The first line of the text file output contains a marker that marks whether the best calibration is done by the Fourier series (1) or the Sine model (2).

For the text file associated with the Fourier series model, the second row contains the p value for respective magnetometer calibration. The third row contains the reference point of which the curve is based on and the period of the data curve-fit at the time frame where the TableSat is stabilized. The fourth row contains the standard deviation of the magnetometer data in order to aid calibrating tolerance needed for pointing. The rest of the rows contain the coefficients in the order of a_0, a_1, b_1, a_2, ..., b_8.

For the text file associated with the Sine model, the second row contains reference point of which the curve is based on and the period of the data curve-fit at the time frame where the TableSat is stabilized. The third row contains standard deviation of the raw data in order to help determine the tolerance needed for the pointing function. The rest of the rows contain the coefficients in the order of a_1, b_1, c_1, a_2, ..., c_8.

A sample calibration result is plotted in Figure 10. It is shown that the calibration method works well for all three magnetometers. The application of moving average filter does not alter the calibration coefficients, and the subsequent curve-fit, in any significant manner and thus is not

applied. The curve-fit of Mg2 can be plotted against that of Mag1 to show a one-to-one relationship in determining the corresponding heading angle. The two-dimensional plot of such a relationship is reproduced in Figure 11. It is shown that their relationship is circular, which is to be expected. From this information, a heading thus can be extracted with relative ease.

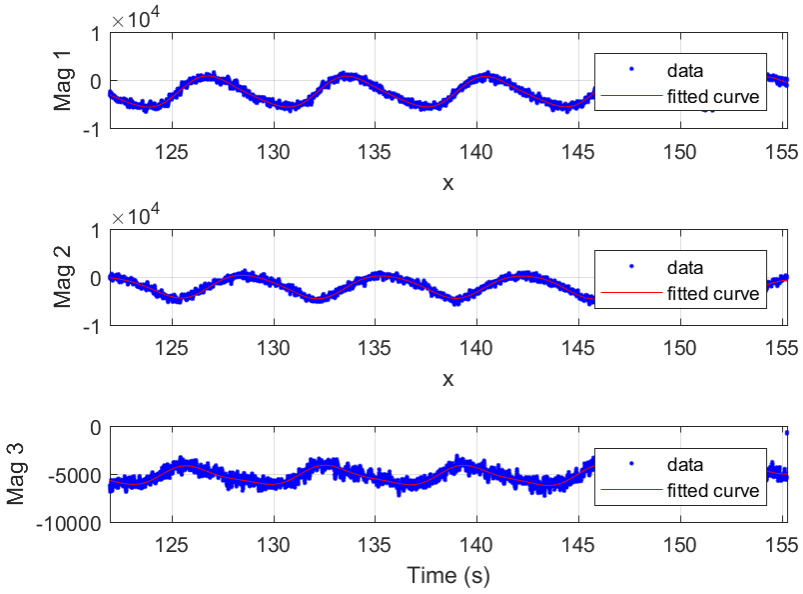


Figure 10: Magnetometer Calibration (`sin8`).

4.2 C Function

The C function takes in the coefficients and recreates the appropriate equation that is used to model the magnetometers. Since the model is based on the period of time where the TableSat has been stabilized, its starting point is not at $t = 0$. A reference point is thus needed and is outputted by the MATLAB script. Moreover, calibration only works when the period of the rotation has been stabilized; this is again extracted from MATLAB. From these two constants, the calibration C function translates an inputted degree value into the corresponding "time step" in the calibration model.

$$t = Reference + Period(\frac{deg}{360\,\mathrm{deg}}) \tag{5}$$

and inputs this into the Fourier or Sine model given in Eq.(2) and Eq.(4) respectively. This allows the function to find the corresponding sensor reading at each degree input. However, this information might not be too useful in the subsequent pointing operations since an expected sensor reading for a particular degree input does not necessarily imply direction and gives no information for a feedback controller. The inverse is thus needed, that is, given a current sensor reading, a heading angle should be outputted by a function.

Since the curve-fit model is quite complicated and that an explicit analytically solution might

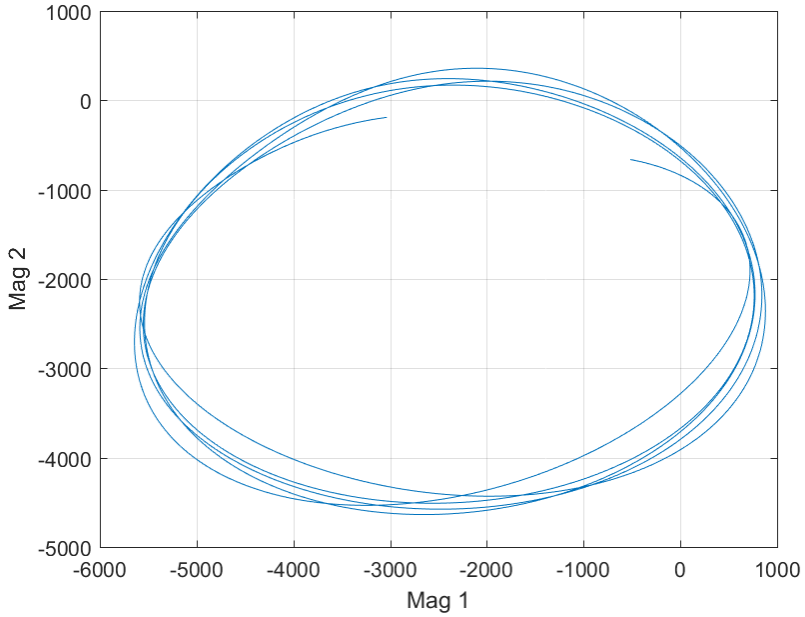


Figure 11: Magnetometer Calibration: relationship between Mag1 and Mag2 readings (`sin8`).

not be attainable, the process of translating a set of sensor readings into heading angle is done numerically. The `sampleDeg()` function samples all degree headings and stores the corresponding model sensor headings into a vector against which the pointing function will check in real time in order to determine current heading and make necessary adjustment based on that information. 36,000 samples at a stepsize of 0.01° are used for this particular process. This thus completes the calibration that maps magnetometer data to a TableSat heading.

5 CSS Calibration

Similar to that of the magnetometer calibration, the calibration for the CSS starts with the data acquired from the `daq.cpp` function where the TableSat is spun for a period of time. Calibration looks only at the period of data where the TableSat is stabilized, that is, gyro-volt reading is within ± 0.1 of the user-inputted desired gyro-volt. Due to the mechanism of the calibration, a calibration based on a relatively high gyro-volt is needed, that is, at about 3.5 gyro-volt. The data is then imported into MATLAB where further processing is carried out. After the data processing is done, MATLAB outputs a text file that contains all the coefficients needed for the C function which uses these coefficients and translates CSS readings into heading angle.

5.1 MATLAB Calibration

The MATLAB portion of the calibration takes in the unfiltered raw data from `daq.cpp` and processes it further by curve-fitting the data and attempts to model the data with an explicit mathematical function. A filter is not needed since the CSS data is not noisy enough to warrant a filter. The same nonlinear curve-fit models as those used in the magnetometer calibration are used in the calibration of the sun sensors. It should be noted that the result are not as good as that of the magnetometers, mostly due to the discontinuous nature of the step function behavior exhibited by the CSS. However, since the sun sensors saturate easily over a revolution and that they provide the same reading over few seconds, they do not provide accurate information regarding the heading of the TableSat. As such, CSS are not chosen for the purpose of the point function, which translates sensor readings into TableSat heading.

Like that in the magnetometer calibration, two different curve-fits models are considered in the calibration process. The Fourier model and the sine model are given in Eq.(2) and Eq.(4) respectively. Similarly, the norms of both models are compared and the one with the lowest norm value is chosen automatically by the MATLAB script. Period is also extracted by the same mechanism as that in magnetometer calibration, that is, marking the mean value at which the curve-fit passes through periodically and finding the period based on this information. Once the calibration is done, we only need to shift the period in order to define the sun as the $0\circ$ heading.

The end product of the MATLAB script is a text file that contains all the coefficients needed for the C function to reconstruct the curve-fit. The text file contains 4 columns, with each storing the corresponding coefficients for the respective 4 CSS found on the TableSat. The first line of the text file output contains a marker that marks whether the best calibration is done by the Fourier series (1) or the Sine model (2).

For the text file associated with the Fourier series model, the second row contains the p value for respective CSS calibration. The third row contains the reference point of which the curve is based on and the period of the data curve-fit at the time frame where the TableSat is stabilized. The rest of the rows contain coefficients in the order of a_0, a_1, b_1, a_2, ..., b_8. Standard deviation of the data is neglected as CSS is not chosen to be used as the main driver for pointing.

For the text file associated with the Sine model, the second row contains reference point of which the curve is based on and the period of the data curve-fit at the time frame where the TableSat is stabilized. The rest of the rows contain coefficients in the order of a_1, b_1, c_1, a_2, ..., c_8. Standard deviation of the data is neglected as CSS is not chosen to be used as the main driver for pointing.

A sample calibration result is plotted in Figure 12. It is shown that the calibration method works relatively well for all four sun sensors. The application of moving average filter does not alter the calibration coefficients, and the subsequent curve-fit, in any significant manner and thus is not applied. Due to the discontinuous nature of the sun sensors and the mechanism of the curve-fit models used, the curve-fits fluctuate near the points of discontinuity. Sun sensors were not chosen to be the main driver for pointing because, in contrast with that of magnetometers, there are headings in the CSS data set where a precise angle is not well defined due to sensor saturation. Moreover, all sun sensors have to be used in order accurately define TableSat heading whereas using two of of the three magnetometers is sufficient if magnetometers were used instead.

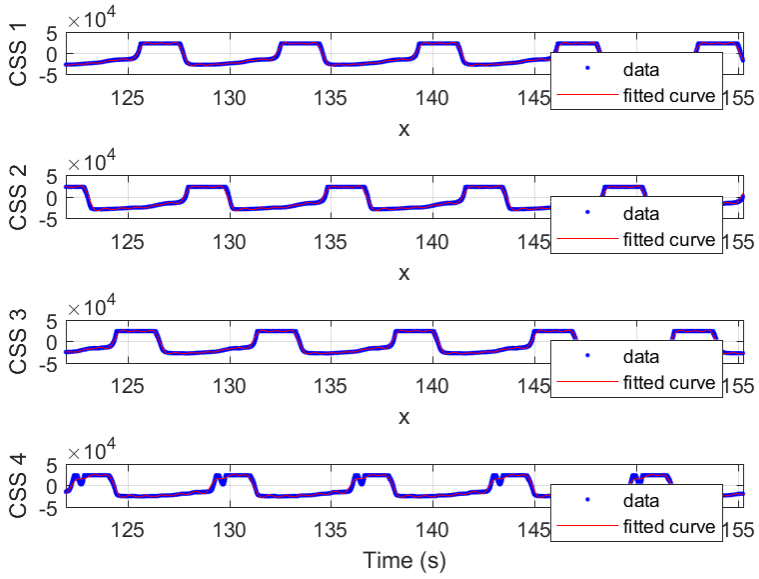


Figure 12: Magnetometer Calibration (`sin8`).

5.2 C Function

Since the mechanism of the calibration is the exact same as that of the magnetometer, the C function is essentially the same. The C function takes in the coefficients and recreates the appropriate equation that is used to model the magnetometers. Since the model is based on the period of time where the TableSat has been stabilized, its starting point is not at $t = 0$. A reference point is thus needed and is outputted by the MATLAB script. Moreover, calibration only works when the period of the rotation has been stabilized; this is again extracted from MATLAB. For these two constants, the calibration C function translates an inputted degree value into the

corresponding "time step" in the calibration model.

$$t = Reference + Period(\frac{deg}{360\,\text{deg}}) \tag{6}$$

and input this into the Fourier or Sine model given in Eq.(2) and Eq.(4) respectively. This allows the function to find the corresponding sensor reading at each degree input. However, this information might not be too useful in the subsequent pointing operations since an expected sensor reading for a particular degree input does not necessarily imply direction and gives no information for a feedback controller. The inverse is thus needed, that is, given a current sensor reading, a heading angle should be outputted by a function.

Since the curve-fit model is quite complicated and that an explicit analytically solution might not be attainable, the process of translating a set of sensor readings into heading angle is done numerically. The `sampleDeg()` function samples all degree headings and stores the corresponding model sensor headings into a vector against which the pointing function will check in real time in order to determine current heading and make necessary adjustment based on that information. 36,000 samples at a stepsize of 0.01° are used for this particular process. This thus completes the calibration that maps CSS data to a TableSat heading.

6 Pointing Control

As explained in the previous sections, we decided to rely on the magnetometer sensors which have proved to be more reliable and accurate than the sun sensors. Since we are not using the sun sensors, the North is chosen arbitrarily based on the calibration.

Because of the high latency between a thrust command and a change in the rotational rate – mostly due to the TableSat's inertia – we decided to implement a Bang-Bang controller in order to point a desired heading. The procedure of the controller is as follows: based on an average of the magnetometer data and on the calibration, we are able to compute the corresponding current heading. We can then compute the associated error with respect to the desired heading. If we are not in the desired range (expected heading plus or minus the tolerance), we give TableSat an impulse in an arbitrary direction, corresponding to a high thrust for a short duration. After stabilization of TableSat's heading, we compute the new error. If the error increases or changes sign, we switch direction, otherwise we keep the same direction. We can then repeat the previous steps until we reach the desired range. Note that as soon as we are within 30$\circ$ from the desired value, we reduce the impulse duration in order to approach it more finely. This procedure is illustrated in Figure 13.

In order to make the controller more robust, we decided to filter the magnetometer's output in real time before using it to get the corresponding heading. This filter consists in a simple average of several magnetometer measures. If the measures are consistent with the calibration, averaging several outputs for a constant heading will yield values very close to the curve fit from the calibration. This way, we can get an accurate heading. Furthermore, the tolerance used to translate sensors output into a heading is based on the standard deviation of the calibration data.

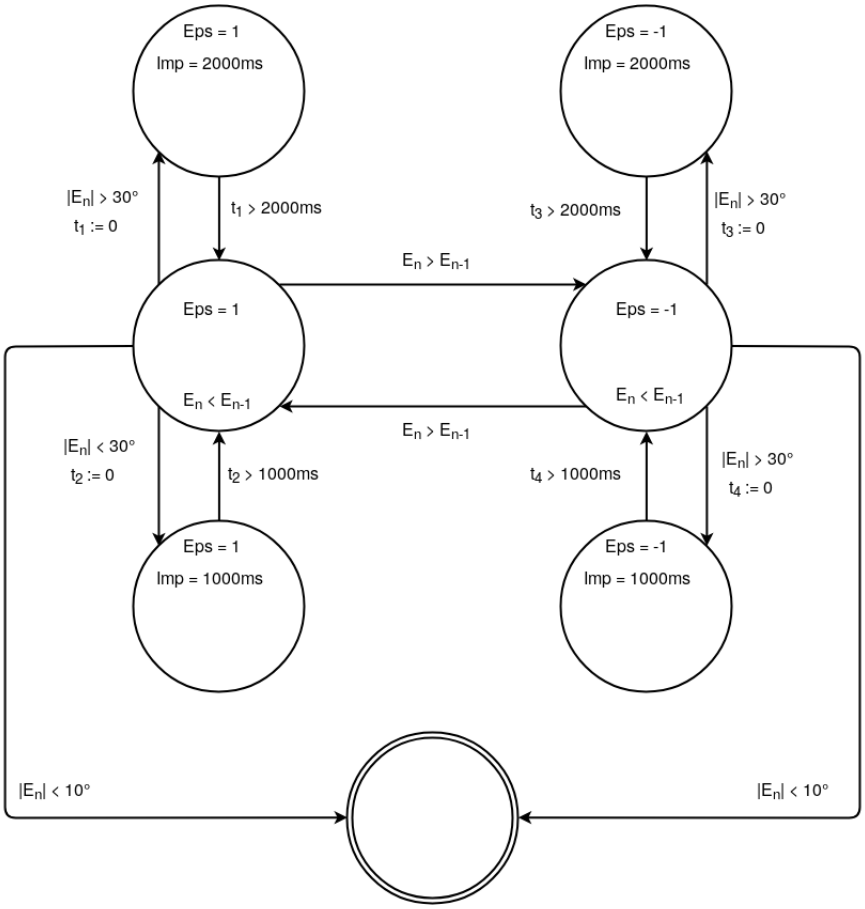


Figure 13: Heading controller. *Eps* stands for the direction of the impulse.

Conclusion

To accurately control a TableSat's heading, a rate control algorithm was first created to ramp up the TableSat in order to record necessary data from the onboard sensors for calibration. It is found that a slightly modified version of a PID controller that is "semi-continuous" can strike a balance between response response latency and precise thrust commands that would ensure a steady revolution speed. A data acquisition program was then implemented in order to collect data. While moving average filters dampen noises levels in the data, it is found that it does not have much of an effect on the calibration due to the method used modelling the data.

A calibration for the Gyro MKS unit was done using multiple samples of varying gyro-volt inputs. A positive linear relationship exists between Gyro-volt readings and revolution speeds. The calibration for the TableSat magnetometers and sun sensors was done using nonlinear curve-fit methods in MATLAB. Two models were used, namely Fourier8 and Sin8. While the sine model, Sin8, approximates the data well if the data resembles a sine wave with limited irregularities, the Fourier series model, Fourier8, is better suited for modelling relatively chaotic or discontinuous yet periodic sensor data. It is found that, in general, the mathematical models for the magnetometers provide a better fit than that of the sun sensors. Their continuous nature provides for an easier and more accurate controller design for pointing and as such, magnetometers were selected to be the main driver for the pointing function. Based on these explicit mathematical models of sensor data, C functions were implemented to recreate these models and numerically calculate the TableSat heading given a sensor input.

A real-time control algorithm for the 1-DOF TableSat was built using calibrated data of magnetometers in order to point towards a desired heading. Two magnetometers were necessary in order to determine the heading of the TableSat. The accuracy of the implemented program can reach within $\pm 5^\circ$ of an inputted heading. This heading is dependent on the current configuration and calibration of the TableSat, that is, a heading of 0 degrees is different each time any physical alterations are made to the TableSat.

In our experience, the program developed here is highly dependent on the current condition of the TableSat. For example, if the TableSat is physically unable to reach a certain rotational speed suitable for sensor calibration, the resultant pointing algorithm will not be able to function well. Also, if the sensor data becomes too noisy and unreadable by the calibration function, the resultant pointing will also be inaccurate. Therefore, future work for the current implementation will be creating a more robust program that would successfully calibrate the TableSat and point it accurately provided that the TableSat can make at least one revolution at any speed.

Annex: How to run the code

The code is divided into two parts: the first one – in C++ – is used to control TableSat and gather the data, which are then used by the second part – in MATLAB – to perform the calibrations. Here is a small documentation summarizing the role of each file – note that this documentation is also written at the beginning of each file:

- `utils.cpp`: this file contains all of the functions that are used by every other files, such as the `commandFan` or `readGyro` functions.
- `rate_control.cpp`: this file implements the PID controller used to stabilize the rotational rate at a desired value for 30s before ramping down. At the end, it outputs a file `rate_data.txt` containing the time, thrust command and gyro rate at each time step during the entire program's run. The time step is 10ms.
- `daq.cpp`: this file relies on the PID controller. It also stabilizes at a desired speed for 30s before ramping down. It outputs a file `tsat_data.txt` which gathers all of the sensor data along with the corresponding thrust command at each time step during the stabilization phase. The time step is 5ms.
- `calibration.cpp`: this file uses the MATLAB `magCalibrationtext` output file to get a numerical inverse of the curve fitting the magnetometer data.
- `point.cpp`: this file relies on `calibration.cpp` and implements a bang-bang controller allowing the user to make the satellite point to a desired heading.

Question 1 – Rotational rate controller

1. Run `rate_control.cpp` with 3GV. It will output a file `rate_data.txt`.
2. Rename the text file as `rate_data_3text`. Note that in order to allow for some flexibility you can control the controller parameters when you run the file as follows: `rate_control Kp Ki Kd` where `Kp`, `Ki` and `Kd` have to be doubles. If no parameters are entered, the standard values described in Section 1.1.4 are used.
3. Repeat the two previous steps with different desired speeds.
4. Copy the renamed text file into the MATLAB directory.
5. Run `plotRate.m`.

Question 2 – Data gathering

1. Run `daq.cpp`. It will output a file `tsat_data.txt`. Note that as for the `rate_control` file, you can enter the controller values.

2. Copy the text file into the MATLAB directory.

3. Run `calibration.m`.

Question 3 – Gyro MKS calibration

1. Assuming the three `rate_data.txt` files are already in the MATLAB directory, run `gyroCalibration.m`.

Question 4 – Magnetometer calibration

1. Assuming the file `tsat_data.txt`is already in the MATLAB directory, run `calibration.m`.

Question 5 – CSS calibration

1. Assuming the file `tsat_data.txt`is already in the MATLAB directory, run `calibration.m`.

Question 6 – Heading controller

1. Assuming the file `tsat_data.txt`is already in the MATLAB directory, run `calibration.m`. It will output a file `magCalibration.txt`.

2. Copy the output file into TableSat.

3. Run the file `point.cpp`. Note that based on the TSat characteristics you are testing the code onto, you might have to change the impulse duration. To do so, you can input the duration of the impulse as a parameter `point impulseDuration`. Note that the duration must be an integer in milliseconds. If no parameter is inputted, a standard value of 1300ms is chosen. Note also that while no specific heading is impossible to target, some headings might be harder to point, based on the quality of the calibration. In our several tests, we observed that headings like to 180◦, 90◦ or 270◦ have consistently proved to be easily achieved, while 0◦ was harder to point to, for some unknown reason.