# ADCS subsystem

## ADCS Mission

As the name suggests, the ADCS (attitude detection and control system) is responsible for the detection and control of the attitude of the CubeSat. The attitude of a CubeSat (or any other body in space) is its orientation, angular velocity, and angular acceleration. So the responsibility of the ADCS subsystem is to detect those attributes using sensors and feed them to microcontroller to get the desired attitude and output those values on the actuators to achieve the desired attitude.

A famous application of CubeSats is to take images of earth, horizon, or stars. In these applications, it’s obvious that pointing the CubeSat in the right direction is a must. Otherwise, they will not have control over the camera angle, and you can only pray that the camera aligns to your target at the right time. The odds are very high. Pray hard in that case.

Please note that in our CubeSat, we have no jets. Hence, we don’t have control over the displacement of the CubeSat. We can by no means control the position, speed, or acceleration. The movement track is predetermined and precalculated before launch. Once launched, only God rules for the universe (aka: physics) defines the CubeSat displacement. Therefore, it’s not the responsibility of the ADCS subsystem.

## How it works.

In the previous section, you learnt what ADCS mission is and what it isn’t. Its mission is to first detect the attitude of the CubeSat, and then control that attitude to tune it to the desired state. In this section, we talk about how the ADCS achieves his mission.

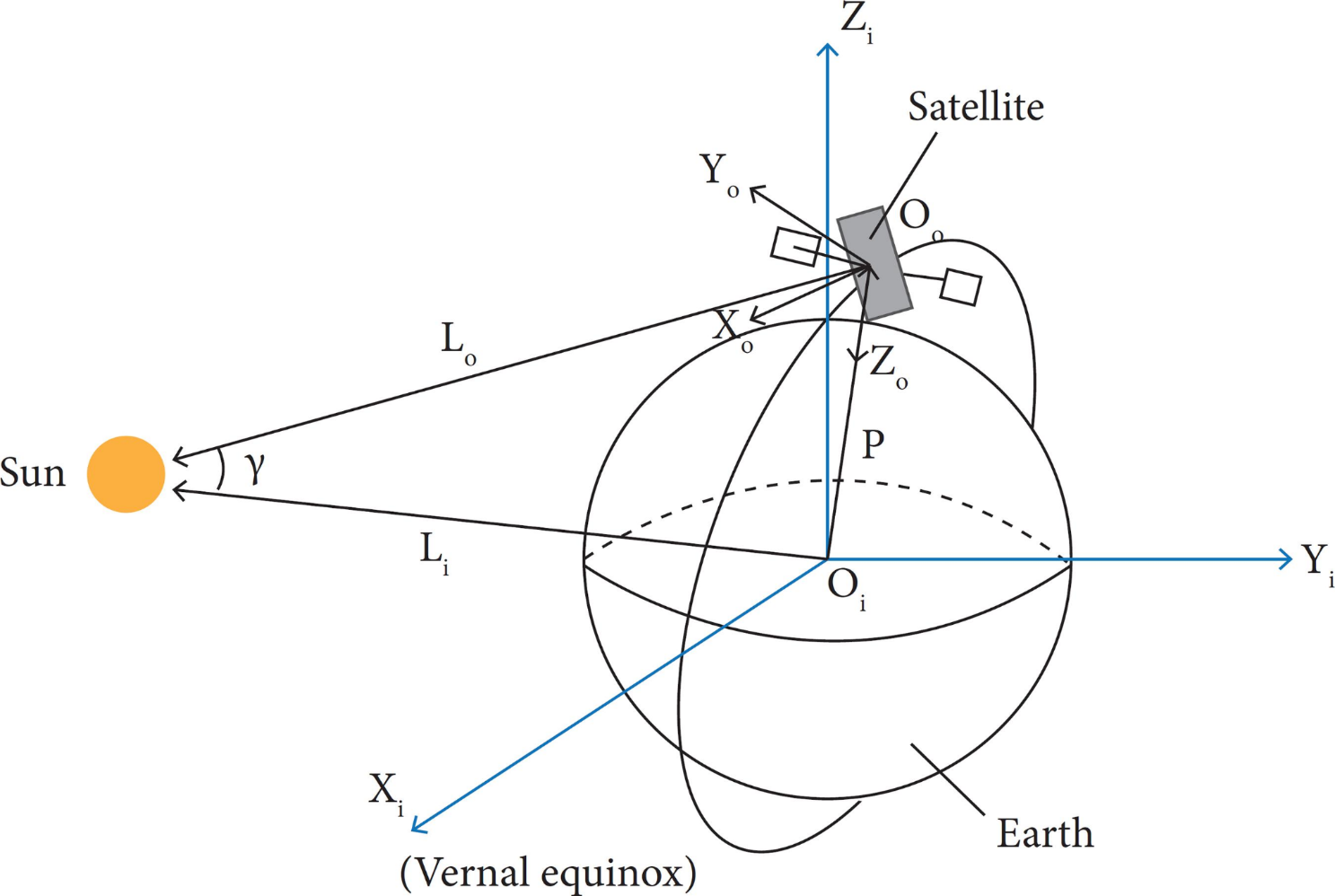
Before diving deep, we assume that the reader has decent knowledge of some topics like vectors and frames of reference. For example, we assume the reader knows what a local frame of reference is and how one frame can be translated to another one.

Back to our track. The work of ADCS is better divided into two parts: detection and control. So, for now, let’s discuss the detection part. Detection is based on a key idea that is simple and straightforward. Let's assume there’s two vectors. If we can measure (or model) those vectors in two different frames of reference, we can then relate one frame to the other. And this is exactly what we need. We need to relate the frame of the CubeSat to the one frame of earth. If we can do that, we will know how the CubeSat is oriented with respect to earth. In our implementation, the two vectors used are the sun vector, and the geomagnetic vector.

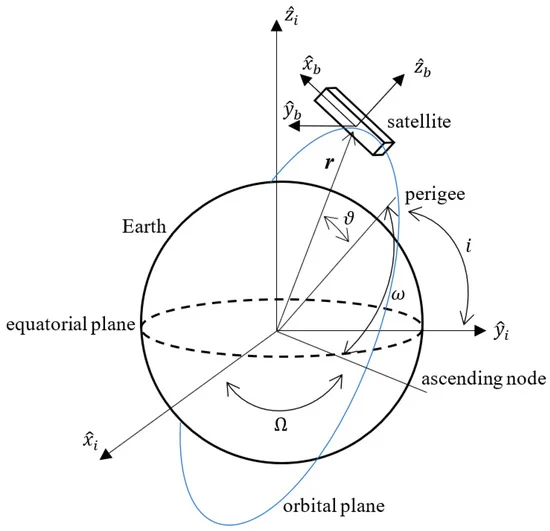
Diagram

Description automatically generated

The sun vector is obviously the vector pointing to the sun. In the frame of earth, it is pointing from the center of the earth to the sun. and in the frame of the CubeSat, it’s pointing from the cube center to the sun. Now we need to get that vector in the two frames. In the CubeSat frame we simply used the solar panels to roughly estimate where the sun is. In the earth frame, we used a model that once you told it the time, it tells you where the sun should be based on some mathematical equations.



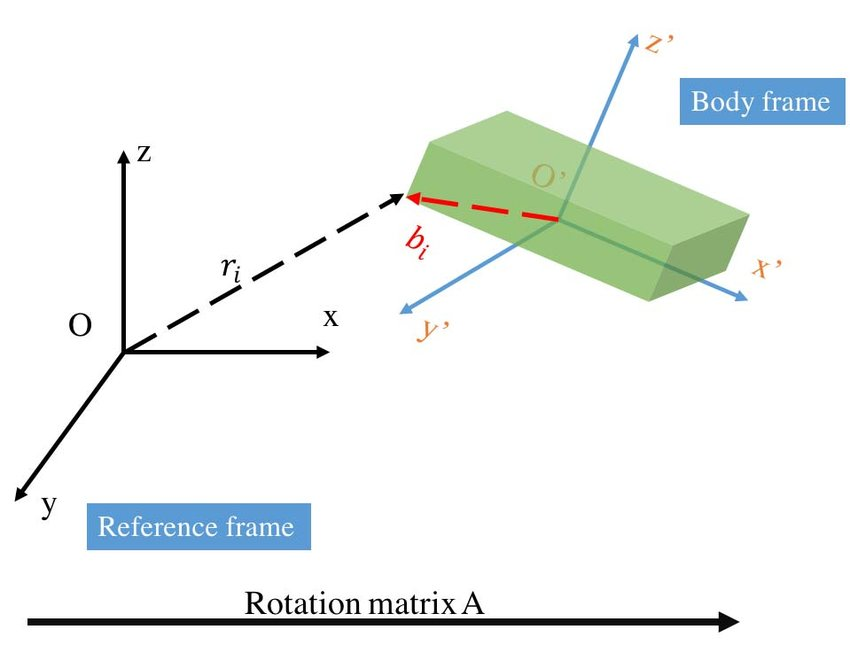
The geomagnetic vector, on the other hand, is the vector of the magnetic field of the earth. Like the sun vector, it is directly sensed by the CubeSat using magnetic compass sensors to get the local vector. It’s then calculated in the frame of earth using a model that takes position and time and gives the modeled geomagnetic vector.



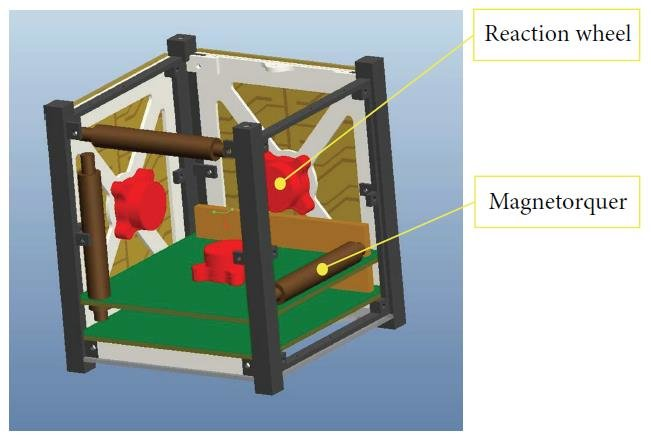
Having the two pairs of vectors; one pair in the CubeSat local frame of reference, and the other pair in the earth frame of reference, we can now try to relate those two vectors to each other. Basically, if you want to relate some set vectors to another set, you try to calculate the translation matrix. That matrix is defined so that when it’s multiplied by one specific set of vectors, the output is the other set of vectors. If you want to translate in the opposite direction, use the inverse of the original matrix.

In our case, the translation matrix is called rotation matrix instead, and it’s defined as:

Finding that rotation matrix is the output of the detection process.



The second process is attitude control and it’s much easier that detection. We have two kinds of actuators to influence the CubeSat attitude: reaction wheel and magnetic torquer. Reaction wheel is famous. It’s some wheel or disk attached to a motor in such a way that allows the motors turn the wheel. Reaction wheel affects the attitude such that when it’s turned in one way, the CubeSat body turns in the opposite way proportionally. Magnetic torquer simply consists of two perpendicular coils. When electric current flows in those coils, an induced magnetic field is generated. The induced magnetic field reacts to the geomagnetic field and tries to set the attitude of the CubeSat to orientation of least conflict.

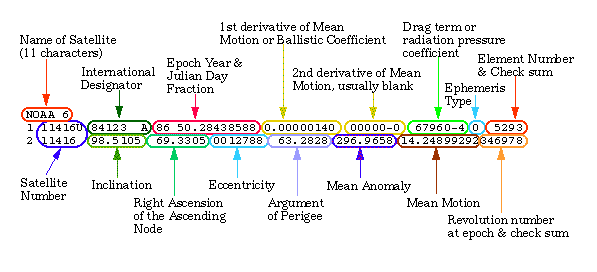


We have mentioned before that we use mathematical models to calculate the reference vectors in the frame of the earth if we know time and position. But we didn’t discuss how to obtain and keep track of time and position in the first place. However, we will now.

A hardware peripheral called RTC, or Real-Time Clock is used to keep track of time. It’s initially set with the help of the OBC clock and/or the GPS clock. The RTC is periodically checked against the same two previously mention sources to encounter any drift or bad initialization errors.

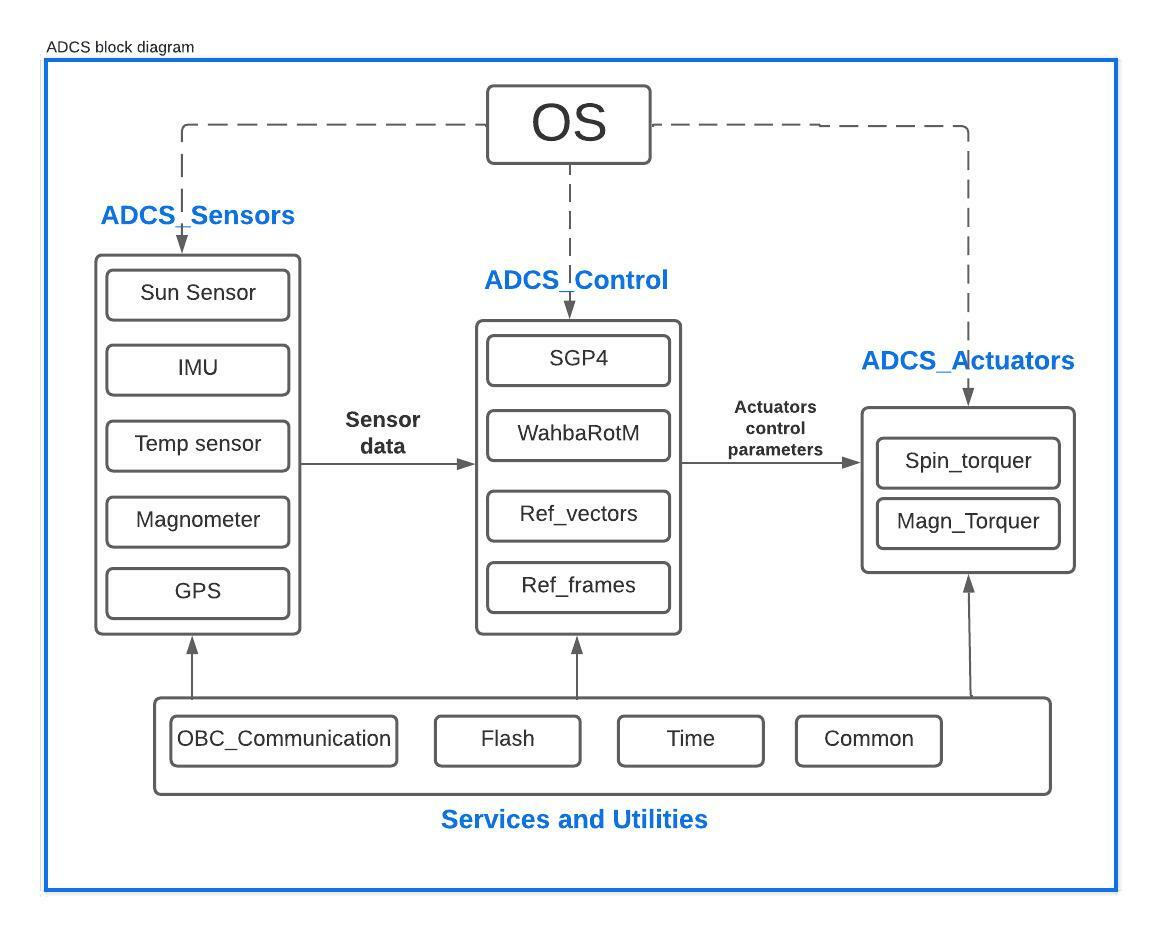
Position on the other hand is as easy. To accurately calculate the current position of the Cubesat, we use some piece of software called “Space Global Propagator” or SGP for short. And it’s very simple in principle. If you know the position, velocity and some other parameters at some point in the past, you can propagate (pridect) those parameters at the present. This is possible because the cubesat in assumed to be in free space and therefore obeying well-defined planetary physics rules.

TLE or Two-Line Elements, is the data format fed to the propagator. It holds information about time and position as well as the track parameters. So, if we have one TLE frame, and we know what time it is, the propagator will tell us where we should be. TLEs are supposed to be created based on some ground RADARs monitoring the state of our CubeSat in space and time. The ground station is responsible for acquiring most recent TLE frames and sending them to the CubeSat through the communication link. On the CubeSat, TLE frames are received by the communication subsystem, sent to the OBC, and finally to the ADCS to be used with the propagator.



## ADCS software static design

We believe that the reader is now ready with a complete description of the subsystem. So, It’s now time to discuss the overall static design. A static design is meant to define the modules of the system and how they relate to each other. A static design also defines the interfaces of each module and the other modules using those interfaces. Before discussing the static design, it may be better to have a look at the block diagram first as it’s simpler and clearer.



Have a good look at the block diagram. It is pretty much self-explaining. You may notice the four main groups of the ADCS:

* Sensors
* Controller
* Actuators
* Services and utilities.

We will now discuss those four groups alongside with the OS component.

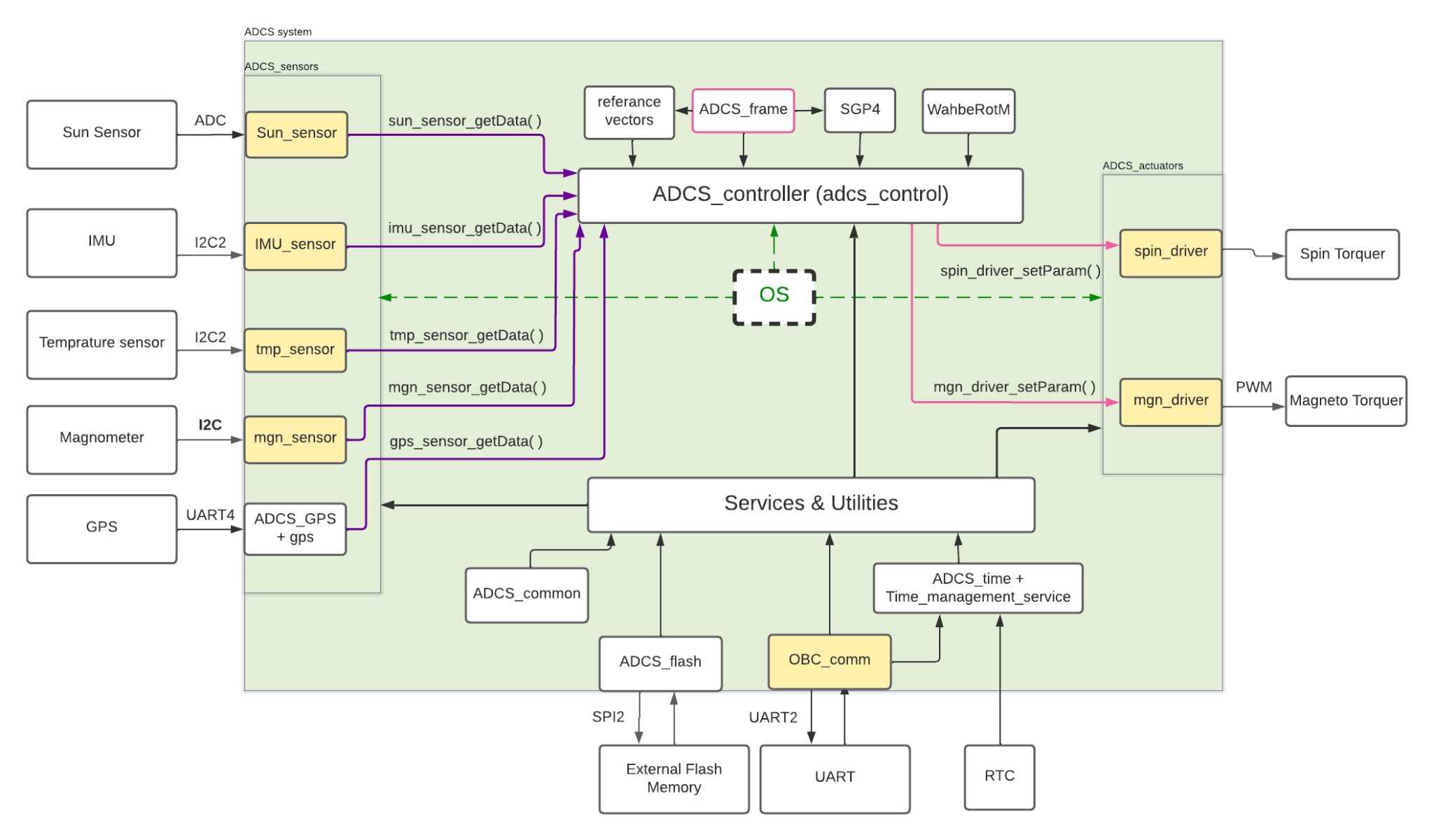
Starting with the sensors group. It contains five sensors, three of which we mentioned before; sun sensor to get the sun vector in local frame, magnetometer, or magnetic compass to get the geomagnetic vector in the local frame, and the GPS to provide time and position information when possible. Two more sensors are temperature sensor and IMU sensor. The temperature is necessary to be sensed to account for errors due to drift in temperature. The IMU sensor itself contains three sub sensors; an accelerometer to measure acceleration, a gyroscope to sense rotation, and a magnetometer that may be used with the other magnetometer for sanity check or as backup.

The controller is responsible for calculating the reference vectors; sun and geomagnetic vectors in the earth’s frame of reference, and for calculating the position using the space global propagator, as well as processing the sensor data. Once the controller finishes these three processes, it sends the right control signals to the actuators.

The actuator module does nothing but applying the parameters set by the controller to the actuators. For the services and utilities group, it is a group that serves the whole system. It provides an interface to the OBC communication port, as well as the flash memory, the RTC time peripheral and other common constants used in calculations.

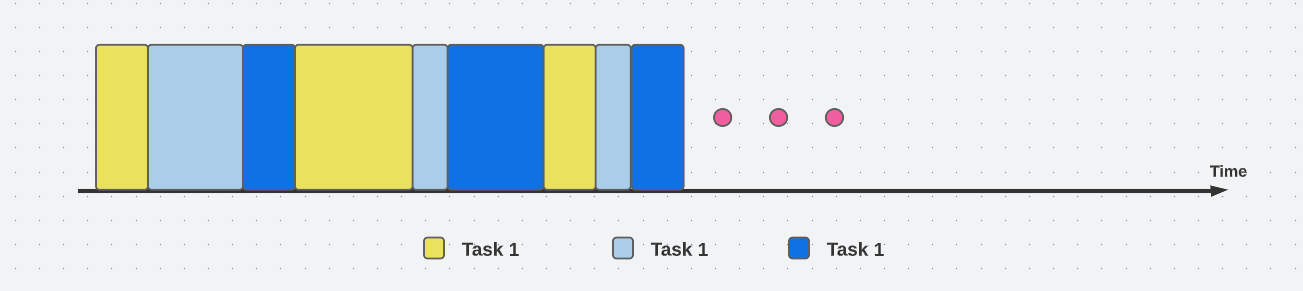
The OS component is any operation technique of your choice. You may use a simple super-loop to run all modules sequentially and repeatedly. You may also use a real-time operating system. You can use anything in between of those two alternatives. For our choice of OS, we chose the TTRD-19A for reliability and the reasons discussed in previous chapters.

Let’s now have a deeper more detailed look over the whole static design diagram. This is a full description to the system’s modules and the interfaces they provide. The inter-module interfaces are mostly setters and getters. This is better for security and reliability.



## ADCS Software Dynamic Design

A dynamic design defines how modules interact dynamically and defines the timings of each module. The final output of a dynamic design process is what I like to call “time line of the processor”, which describes what task runs at what time. The dynamic design is tightly coupled to your choice of OS. For example, if you use the super-loop architecture, there will be no time gaps between running tasks and the processor is always busy. The result will be something like this figure. Of course, This is not the choice for determinism.

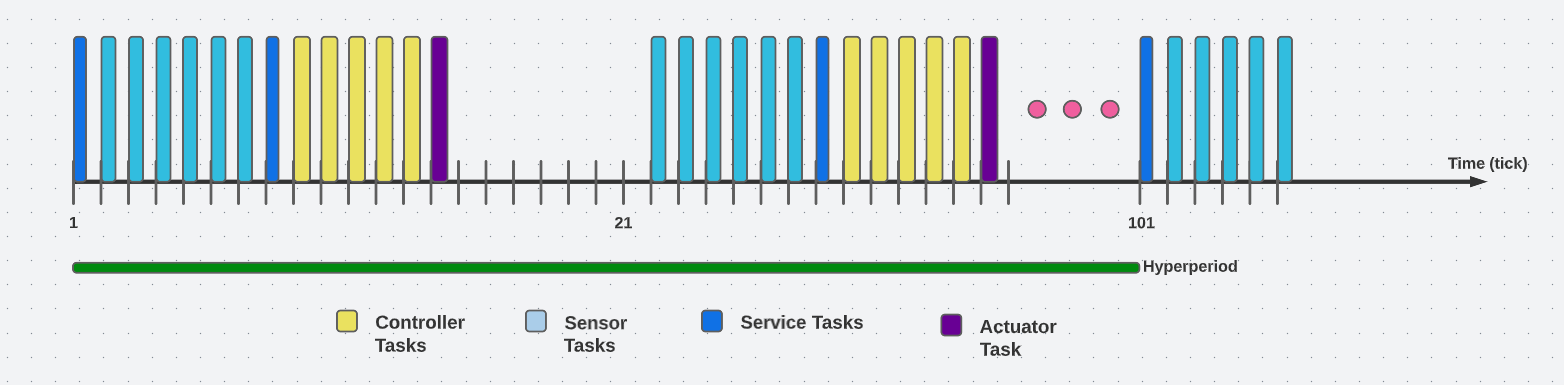


Choosing TTRD-19A as our OS, this will impact our dynamic design. The most key component in TTRD-19A taking effect on our dynamic design is the time-triggered cooperative scheduler. Time-triggered means that the schedular tick source is a timer. Cooperative means that there is no preemption and tasks are allowed to finish on their on. This is as far as the scheduler is concerned. However, there are components in the TTRD-19A OS that prevent tasks from running faster or slower than predefined rate but that’s irrelevant now.

Relating to the modules illustrated in our static design, each module has one initialization function and one update function. We consider the update function of each module to be a task. We then decided on the timing parameters of each task (i.e. task period, task offset, etc..). Based on the timing parameters of individual tasks, we deduced the timing parameters of the whole system (i.e., hyper period, tick period). The output of the dynamic design phase is the figures and tables below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task  Group | Task | Offset (tick) | Period (mS) | Period (tick) |
| TTRD-19A  related tasks | Watchdog Update | 0 | 10 | 1 |
| HEARTBEAT\_SW\_Update | 0 | 1000 | 100 |
| ADC1\_Update | 0 | 500 | 50 |
| PROCESSOR\_TASK\_Update | 0 | 1000 | 100 |
| UART2\_BUF\_O\_Update | 0 | 10 | 1 |
| Service Task | OBC Comm Update | 1 | 1000 | 100 |
| Sensors  related tasks | IMU Sensor Update | 2 | 200 | 20 |
| MGN Sensor Update | 3 | 200 | 20 |
| TMP Sensor Update | 4 | 200 | 20 |
| GPS Sensor Update | 5 | 200 | 20 |
| Sun Sensor Update | 6 | 200 | 20 |
| Health Check Update | 7 | 200 | 20 |
| Service Task | Time Keeping Update | 8 | 200 | 20 |
| Controller  related tasks | Tle Update | 9 | 200 | 20 |
| Sgp4 Update | 10 | 200 | 20 |
| ref\_vectors update | 11 | 200 | 20 |
| attitude Determination | 12 | 200 | 20 |
| attitude Control update | 13 | 200 | 20 |
| Actuators Task | actuators Update | 14 | 200 | 20 |

|  |
| --- |
| TICK Interval |
| 10 mS |
| Hyper period |
| 1000 mS = 100 tick |



Please note that TTRD-19A related task are not present on the timeline chart as they are irrelevant.