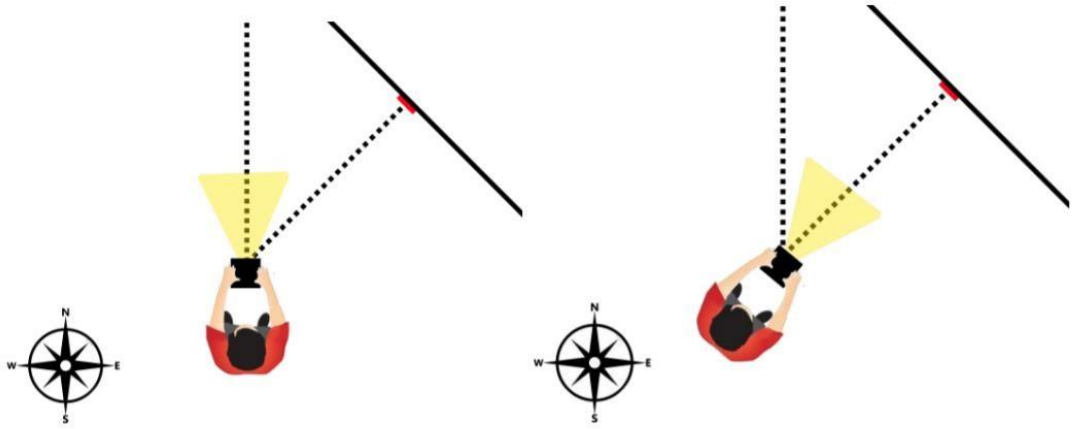
ADCS Lab

In this activity, we will be looking at how well your ADCS sensors work without sensor fusion. We will be getting roll, pitch, and yaw data from the combination of your accelerometer and magnetometer, as well as from your gyroscope. We will do this by coding your camera to automatically take a picture at a given RPY angle.



Set Up

You will need the following items for this lab:

- An item to image, such as a piece of colored plastic (look for objects that you have already in your environment)
- A protractor (if you don't have a physical protractor, try downloading an app on your phone or printing out a paper protractor)
- Something to affix your target to at the same height you will be holding your camera. You can lean it against something, use the Velcro in your kit, or whatever solution you come up with.

Note: for this activity, if you choose to stand while rotating as shown in the graphic above, you will be holding your Flat Sat with the camera facing the wall. Be sure to look at your IMU data with this in mind – which is roll, pitch, and yaw in this orientation?

Download Skeleton Code

1. Download the three Python files on Edly: sensor_calc_V2.py, auto_camera_V2.py, and plot_V2.py.
2. In sensor_calc, you will be defining your roll/pitch/yaw and calibration functions. These functions will be imported to auto_camera and plot, so make sure you test them thoroughly.
    - **Note:** sensor_calc is not fully set up, and you will need to input some equations/code before you can run plot.py. If you run plot.py before setting up sensor_calc, plot.py will not work.
3. plot.py is already plots your roll/pitch/yaw values to help you troubleshoot. When you have sensor_calc set up, you can run: `plot_v2.py am` to plot the values calculated by the accelerometer and magnetometer, and run: `plot_v2.py gyro` to plot the values calculated by the gyroscope. Feel free to change this code around to plot whatever you want!
4. Auto_camera will be the code you will write to take the picture for this activity. Currently, it is set up to have two arguments passed to the command line: dir, which is the RPY angle you are slewing about, and target_angle, which is the amount you want to turn. Feel free to change this to suit your solution, but, as always, do not modify the skeleton code.

#1- Accelerometer and Magnetometer Code

1. Let's write the code to determine RPY from the accelerometer and magnetometer inside the sensor_calc.py file. Make use of NumPy's arctan2 functions to find your orientation.
2. Some equations you'll need (be careful of radians vs degrees):
    a. $roll = arctan(\frac{accelY}{\sqrt[2]{accelX^2+accelZ^2}})$
    b. $pitch = arctan(\frac{accelX}{\sqrt[2]{accelY^2+accelZ^2}})$
3. Then, to find yaw, we need to use the magnetometer.
    a. $mag_x = magXcos(pitch) + magYsin(roll)sin(pitch) + magZcos(roll)sin(pitch)$
    b. $mag_y = magYcos(roll) - magZsin(roll)$
    c. $yaw = arctan\left(\frac{-mag_y}{mag_x}\right)$
4. Some things to consider in your RPY functions:
    a. How do you want to define "zero angle"?
    b. What should happen when your angle moves into a different quadrant?

5. Once you have the RPY code written, run `plot_v2.py` or `plot_v2.py am` . It will automatically run the helper function to set the initial position. If you've done everything correctly, you should see a live plot. Don't worry too much about accuracy yet, we will be getting there soon.

#2- Gyroscope

1. Let's repeat what we just did, but by using the gyroscope. There are functions set up in sensor_calc.py.
2. You will need to integrate the angular rate to get the orientation. The equation is as follows:
    a. $\boldsymbol{\theta = \theta_{-1} + \omega_{gyro}\Delta t}$
3. Make sure you use the right gyroscope output for each RPY. (hint: gyroX may not correspond to roll. Look on the IMU itself for a guide)
4. For your first value, you won't be able to use this formula because you won't have a previous angle or a time step! Be sure you have a way to make an initial estimate. (hint: use the accel/mag!)
5. Once you have the RPY code written, run `plot_v2.py gyro` . If you have coded it correctly, your data should be similar to your accelerometer/gyro data.

#3- Calibrating your Sensors

You may have noticed that your RPY data isn't completely accurate, that is because each sensor on an IMU becomes inaccurate under certain conditions. We can use this to our advantage by using each sensor to calibrate the others, increasing our overall accuracy.

Calibration is simple:

1. Take some data from the sensor
    a. For the magnetometer, shake it around while calculating this data. This is because you're trying to find how much it is offset from the Earth's magnetic field
    b. For the gyroscope, keep it still. This is because you're trying to find out what it is measuring when it is supposed to be measuring zero rotation.
2. Find the min and max values
3. Average the min and max values and subtract this value from every reading you get.

Note: this is only for hard iron calibration for the magnetometer. Follow the instructions here for soft iron.

#4- Imaging Code

Set your Image Target

Pick the position you're starting from and the surface you're imaging from and measure the angle between them. Do this with a protractor if you have it, a ruler, and some trig if you don't. Write this angle down. You can also measure angles using a phone compass. Affix your target to the imaging surface. (See image on page 1 for angle reference/example)

Write your Image Taking Code

1. Now, write a script to take a picture when you have turned your CubeSat to the angle you specified earlier in `auto_camera_v2.py`, remember to include the var and target_angle variables if you want to change the defaults.
    a. Ex: `auto_camera_v2.py pitch 45`
2. Run it! How centered is your final image? What might this say about the accuracy of your sensor readings?