WPI

RBE 2002: Unified Robotics II
Lab 4 Activities: IMU

Last modification: November 12, 2022

Individual Lab Activity: IMU Fundamentals

1 IMU Fundamentals

Pololu provides a nominal library for interfacing with the IMU and we have updated it to add some additional functionality. The IMU runs on 3.3V – but no worries! – they've built a level shifter into the Romi.

1.1 Setup

Procedure

1. Clone the `Week-04` repository for your team as you have done in previous weeks. This week, we have added the worksheet questions to two markdown files, one for individuals and one for the group. Though there are "thought experiements" and questions peppered in these instructions, you are formally responsible for the questions in the `.md` files.

2. Create a branch for yourself, as you have done in the past. Checkout your branch and rename `users/individual.md` to your name. Answer the introductory questions therein.

1.2 Basic functionality

The first thing to do is to establish basic communication with the IMU and determine/verify the coordinate system for each sensor.

Procedure

1. Open the `imu` example, found in the `sample-code` folder in your team's github repo. You will need to set up your `platformio.ini` file as before and use `lib_extra_dirs` to point to the IMU library in the `lib` folder.

2. Upload the program to be sure it runs. If there are any glitches, find an instructor.

3. Look at the source code to see what the buttons do in this example program.

4. Set your Romi on the table and observe the data from the accelerometer and the gyroscope in turn. Are the gyroscope values zero? How about the accelerometer? What is the value of the acceleration in the z-axis? How many g does this correspond to?

5. Do some experiments to determine/verify which direction the axes point. A good experiment is to accelerate the Romi in various directions and then twist it. You can use the Arduino Serial Plotter to help visualize the readings. Determine/confirm the orientation of axes for both the accelerometer and the gyroscope. Are the coordinate systems right-handed? Do they coincide with each other? Record your findings in the table in your `.md` file.

6. With the Romi sitting in its normal position, record the acceleration for each axis in the first row of the table in your `.md` file (where it says, "Top pointing up"). Write your answers in integer values of g (e.g., "1g", "-42g").

7. Without moving the Romi yet, make *predictions* for the readings on each axis if you were to point the front straight up. You are welcome to work with your teammates here, but don't discuss your predictions until you've each made one. Record your predictions in the second row of the table in your `.md` file ("Front pointing up").

8. After you have made your predictions, set the Romi on its end so the front is pointing straight up. **Boldify** your correct answers. *Do not go back and change your answers!* That's not the point of this exercise, and you will *not* lose points if your predictions were incorrect.

9. Repeat the process for each of the positions in the table. Each time, make a prediction and then check your results and embolden your correct answers.

1.3 Calibration

Procedure

1. In `LSM6::enableDefaults()`, find the lines of code that set the the output data rate and the full-scale range for the accelerometer and the gyroscope.

2. Given the setting for the full-scale, consult Table 3 in Section 4.1 of the datasheet to determine the *sensitivity* of each sensor. Record the values in the table in your `.md` file.

3. Design and perform an *experiment* to measure the *sensitivity* and *offset* of each accelerometer axis. Describe your experiment in the Lab Worksheet and record your findings.

4. Set your Romi on a solid surface with the z-axis pointing upward. Run the program again and calculate the noise (as the std. dev. *in gravities*) for each axis of the accelerometer and record the results. You'll want to grab 200 - 300 readings and use a spreadsheet or `MATLAB` (or just calculate the noise in the program itself).

5. Gently tap your Romi and note the accelerometer readings. You don't need to calculate anything here, but remember this when we get to the angle calculations.

Show your results to an instructor for an electronic sign-off. Have the data from your observations ready so they can verify that you have done the calculations properly.

1.4 Timing

Here you will explore the timing of the readings.

Procedure

1. Attach an oscilloscope to the clock (`SCL`) and data (`SDA`) of the I2C bus. With the program running in accelerometer-only mode, capture a transaction. You don't have to interpret it, but estimate how long it takes for your Romi to read the IMU. Remember that a read transaction is actually a pair of write/read transactions – record the total time for both.

2. Add a statement to print the value of `millis()` immediately before the program prints the sensor output in the `loop()`.

3. Run the program and verify that it's reading at the rate specified in the `enableDefaults()` method.

4. Increase the data rate to the next speed (see the library for choices). Does it still print on the correct schedule? What happens if you increase the data rate again? Keep increasing the data rate until you can't read data at the expected rate. How does the length of the I2C transaction affect the limit at which you can read data? Answer the questions in the `.md` file.

1.5 Gyroscope

Procedure

1. Determine the full-scale output of the gyroscope and then check Table 3 in Section 4.1 of the datasheet to determine the sensitivity.

2. It will be difficult to calculate the sensitivity experimentally (unless you can carefully spin your Romi in a controller manner), but you *can* estimate the bias and noise with a simple experiment. Describe your experiment and record your findings in your `.md` file.

3. Spin the Romi back and forth quickly (but carefully!). Do the readings max out? Try increasing the full-scale output of the gyroscope to 500 dps. Does that help?

Show your results to an instructor for an electronic sign-off. Be prepared to discuss the data rates and sensitivity experiments.

2 Angle calculations

You will build your complementary filter as a group exercise, but here you will individually explore some of the calculations that go into the filter.

2.1 The `STATUS` register

The IMU has a `STATUS` register that can be used to determine if there are new readings available. In fact, you've been using it all along, though you may not have realized it. Using the `STATUS` register, you can use the following procedure to get data from the IMU:

1. Set the accelerometer ODR to the rate at which you want to perform calculations.
2. Set the gyroscope ODR to something faster.
3. Read the status of the accelerometer to determine if new data is available. When new data is available,
   (a) Read the acceleration data,
   (b) Read the gyroscope.

   Since the gyroscope is running at a faster rate, you'll know that both the accelerometer and gyroscope data are "fresh."

   Note that the chip itself actually has a pin that can be set to raise an interrupt signal when new data is available, but alas, Pololu didn't break it out and connect it to the microcontroller. Instead, we'll have to poll the accelerometer for new data. Rats.

2.2 Update `Chassis`

Your first step to building a complementary filter will be to incorporate the IMU into your `Chassis` class. Instead of bringing your robot code here, **you will continue to add functionality to your Week 2 code**. Think of it as your Final Project code from here on out.

1. Open your code from the previous lab. Add an `imu` member object to your `Chassis` class. Add code to `Chassis::init()` to start the IMU. We'll leave it to you to choose your data rate and sensitivity (you'll test and adjust below).
2. Add a method, `updatePitch()` to the `Chassis` class. Later, you'll add your complementary filter there, but for now, `updatePitch()` should check for a new accelerometer reading and print out `millis()` when new data are available. Return the observed angle (later, you'll return the fused angle). Test your code to be sure you have the correct functionality. You should call `updatePitch()` in the main `loop()`.

2.3 Basic angle calculations

Now let's do an angle calculation.

Procedure

1. Returning to `updatePitch()`, add code to calculate the *observed* pitch angle of your Romi from the accelerometer readings. As with kinematics, `atan2()` is still your friend.

2. Add code to print out the observed angle.
3. Run your code. Rotate the Romi on its pitch axis. How do the observations look? Is there noticeable noise? What happens if you gently tap it? Do the angle readings jump?

Show your results to an instructor for an electronic sign-off.

Create a Release of your code. Tag the release with your name and `week04`, e.g. `lewin-greg-week04`. Include a link to your release in the `.md` file.

Group Activity: Sensor fusion

3 The Complementary Filter

Here, you will build the functionality to perform sensor fusion, specifically, you'll implement a complementary filter.

Procedure

1. As in previous activities, discuss among yourselves the pros and cons of each student's solution. Decide on one to merge into your main branch and do so with a Merge Request. Be sure to add comments as part of the merge.

3.1 Filtering

Now let's create the actual complementary filter.

Procedure

1. Determine as a group the update rate of your filter. How did you decide on the rate?
2. Add code to your `init()` function to set the gyroscope sampling rate to 4X the accelerometer rate.
3. Add code to `updatePitch()` to read the gyroscope whenever there is a new accelerometer reading. Since the sampling rate is much faster than that for the accelerometer, you don't need to check the status bit.
4. Add a member datum, `estimatedPitchAngle`, to your `Chassis` class. When you call `updatePitch()`, `estimatedPitchAngle` will store the most recent estimate (e.g., θ^{k-1}). After you do the calculations, you'll update `estimatedPitchAngle` so it will be available the next time through the function.
5. Ignoring the bias for the moment, add code to calculate the predicted angle. You'll need the gyroscope sensitivity to get the rotation speed (don't forget to convert it to rad/s!) and you'll need the ODR for the *accelerometer* to get Δt. Both of these values are maintained by the `LSM6` class, so you might want to add a function to it that calculates the predicted motion.
6. Add code to print the prediction angle.
7. Add code to calculate the estimated (filtered) angle. Change the return value so that you're returning the filtered angle.
8. Add code to print the filtered angle. You should be printing four things by my reckoning: `millis()`, the predicted angle, the observed angle, and the filtered angle.

9. Run the code with $\kappa = 1$ and open the Serial Plotter. Are you now using the accelerometer or the gyroscope? How does the "filtered" angle (which isn't really being filtered at all) compare to the observed angle from the accelerometer?
10. Run the code with $\kappa = 0$. Which value are you using now? Set the Romi on your desk and don't move it. What happens to the "filtered" angle?
11. Set $\kappa = 0.5$ and run your program again. Move the Romi through pitching motions; gently tap the Romi. Without dropping it – *be careful of the USB connection!* – start it at some angle and quickly set it down flat. Does the filtered angle track the observed angle? Does it filter out the noise? Does it respond quickly?
12. Experiment with different values of κ. Do you think it should tend towards 0 or 1? What did you find worked well?

3.2 Troubleshooting guide

There are a number of things that can cause poor performance (or complete failure) of your filter. Some of the common ones are listed below, with some trouble shooting hints.

Not filtering out noise well? Try adjusting κ. Which do you want to weight more, the accelerometer or the gyroscope?

Not responsive enough? Try adjusting κ, but also increase the gyroscope sensitivity. Don't forget to change the scale factor in your calculations!

Filtered angle goes the opposite way before it settles down? Classic sign error on your axes.

Gross over- or under-shoot? Check your scale factor. Check your Δt. You remembered to convert to radians, right?

The filtered angle oscillates slowly? The phenomenon shows up when the bias update is poorly scaled.

3.3 Dealing with bias

As discussed above, both the accelerometer and gyroscope are prone to an *offset*, or *bias*. The accelerometer offset can optionally be calibrated out at the beginning of the program by reading the accelerometer while the robot is sitting flat and calculating the average offset.

The gyroscope also has a bias, but it turns out that we can make an estimate of the gyroscope bias as *part of the solution*, so instead of calibrating it out (and having it change anyway), we'll just determine the bias on the fly.

Procedure

1. Add a `gyroBias` member datum to your `Chassis` class. Initialize it to 0.
2. Add code to implement the bias update. Start with $\epsilon = 0.01$.

3. Add code to print the bias, as well as the angles that are already being printed.
4. Run your code. How does the bias change over time? What happens if you make $\epsilon = 1$? Can you explain why?

You will be expected to show a Serial Plotter output while you perform some of the experiments above. The SA will ask you about your parameters (κ, etc.) and how they affect the performance. They may also ask you questions related to the "What about...?" questions in the handout.

4 Challenge

Here you will program your Romi to drive up a slope and stop when it reaches level ground. You will use your complementary filter to detect the slope and flat part. Program your Romi with the following behavior:

- When an IR button is pressed (your choice), the robot starts to drive forward in a straight line towards your ramp,
- When it detects that it is going up the ramp, an LED will light, and
- When it reaches the top of the ramp (which will tip down), it stops and the LED is turned off.

You will need to augment your state machine with a climbing state. What threshold(s) will you use for detecting the slope? Do you think hysteresis could be useful here?

Create a release of your code. Tag it `group#-week04`. Include a link in the `.md` file.