WPI

Last modification: October 29, 2024

RBE 2002 (B-24)
Lab 2: Events, Messages and the IMU
Lab Activities

Be sure to follow along with the Worksheet at the end of this document while you do the activities.

In this lab, you will program your robot to drive out an arbitrary number of intersections, turn around, and return to the start. While the behaviour is hardly extraordinary, the lab will give us the opportunity to introduce the IMU, sensor imperfections, and calibration. We will also explore the concept of *messages as events* and *guard conditions*, conditionals that dictate actions depending on auxiliary variables.

There are many ways to execute turns with the Romi. And while dead reckoning with a timer is simple and reasonably effective, for this lab, you will use the gyroscope built into the IMU. You will start by exploring the functionality and limitations of the gyroscope, after which you'll add code to calibrate the gyroscope. You will then program the basic behaviour of executing a turn. The lab will culminate in commanding the robot to drive N intersections, turn around, and return to the start. You are encouraged to increase the challenge by navigating to an arbitrary intersection.

Preparation

Unfortunately, the starter code already contains a proper checker for detecting intersections. Though you will not need to write that function, be sure to review it in light of the concepts presented in lecture.

In addition, the navigation code in the starter code is unnecessarily complex. While it would be useful for an arbitrary arrangement of streets and intersections, we will limit ourselves to a so-called Manhattan grid, which will simplify navigation tremendously.

You will probably want to remove some of the code related to modes for the Romi. Specifically, you should no longer need the `SETUP` mode, so you can comment out that part of the code and save a little program space. You may want to use the `TELEOP` mode when we get to lifting, but if your robot defaults to `AUTO` mode, you can save some of the clunkiness of the IR remote.

The datasheet for the IMU is linked at the top of `LSM6.h`.

Lab Activities

The template code contains several placeholders for this week's activities (it also contains the intersection checker, unfortunately, and you should at least review how that works).

1 Getting Started

Procedure

1. Create a branch of your repository using your choice of tools. Name the branch `week-02-nav` and make sure that you have switched to it. You will need to commit any changes you've made before you switch (or you will have to stash those changes).

2 The Gyroscope

We'll start by exploring the gyroscope. Uncomment the line `__IMU_DEBUG__` flag in the `platformio.ini` file and run the code from last week. You should see a bunch of zeros scroll by. By default, in `Robot::HandleOrientationUpdate()`, the code is outputting the heading (yaw angle) of the Romi as calculated from the gyroscope readings, except that we did remember to remove the code that does those calculations, so it doesn't update – it will be up to you to code up the calculations.

To start, we'll ignore the gyroscope *bias* and "naïvely" add code to update the heading of the Romi.

Procedure

1. Review the code in `Robot::RobotLoop()`. At or near the end of that function, the Romi checks to see if there is new data available from the IMU. When there is new data, the code calls `Robot::HandleOrientationUpdate()`, which you will need to complete. In this case, each new reading is handled in the same way that events are handled:

   ```
   if new reading -> process that reading
   ```

   Then if there is a new reading, we can execute any number of tasks that rely on the updated heading.

   To see how the Romi checks if there is a new reading, dig into the `LSM6::checkForNewData()` function. Note that the line that reads,

   ```
   if(getStatus() & 0x01)
   ```

   needs to be updated. What *should* it read? See the `STATUS_REG` register (`0x1E`) for hints.

2. To help with deciphering the workings of the IMU, add code to print `millis()` and the *raw* gyro values whenever a new reading is available. Note that class `Robot` is declared a `friend` class of `LSM6`, so you don't have to mess with getters and setters. You will want to put the print statements inside the `#ifdef/#endif` pair in `Robot::HandleOrientationUpdate()`. Doing so will make it easy to turn the printing on and off, as desired. Print all three values – one for each axis – all on one line. You can print the TAB character with `'\t'` to make the numbers more readable.

3. Place your Romi on the table and run the code. You can leave the power off so that the motors don't start running unexpectedly. Observe (eyeball) the raw values from the gyroscope. Record the raw values and, consulting the datasheet for the IMU, record the equivalent drift rate. You can find the output data rate and full-scale settings in `LSM6::enableDefault()`. Table 3 in the datasheet can be used to decipher the settings. Does the timestep make sense?

4. Do some experiments to determine the direction of each of the gyroscope axes. Record your answers in the Worksheet.

5. In `Robot::HandleOrientationUpdate()`, add code to update and print the heading in any state other than `IDLE` (we'll update the bias in `IDLE` in a moment). You need only worry about rotations about the z-axis. You will need to account for both the sensitivity and the data rate, both of which are calculated for you and stored in variables in the `LSM6` class. What do you notice about the heading? Does the drift agree with your calculations above?

6. Slowly turn the Romi for one complete rotation (we'll program it to turn itself later – just do it by hand here). Did it change by 360 degrees?

2.1 Reducing the bias

As discussed in lecture, the gyroscope is plagued by *bias* – a non-zero offset that manifests as an apparent slow rotation of the Romi. Later in the term, you'll explore a means to remove the bias for the pitch angle by observing gravity, but since there is no external reference to correct the yaw angle, the best we can do is try to minimize the effects of bias.

To do that, you'll update a running average of the gyroscope output whenever the Romi is motionless (e.g., in the `IDLE` state). The running average will take a short amount of time to stabilize, but it will work well.

Procedure

1. Add code to update the bias when in `IDLE` mode. You will need to implement a running average in `IMU::updateGyroBias()`. You will need to experiment with the weight for your running average.

2. Update your heading calculation to remove the bias from the gyroscope reading. Repeat the rotation from before. Did it work better? If you print out the adjusted gyroscope value (instead of just the raw value), does it approach 0?

2.2 Gyroscope Settings

Here you will explore the output data rate (ODR) and full-scale (FS) settings.

Procedure

1. Rapidly, but carefully, turn the Romi by 90 degrees. Did it register the correct rotation?

2. Experiment with the `FS` setting. Can you get the gyroscope to be more reliable at high speeds?
3. Add code to turn a pin HIGH[1] immediately before the code calls `LSM6::CheckForNewData()` and LOW immediately afterwards. Slap an oscilloscope on the pin and observe the output. You should see several short pulses and one long one every once in a while (Why?). Given the length of an IMU transaction, how fast could you read the IMU, in principle?
4. Experiment with different ODR values until you find one that is a good compromise between rapid updates and not tying up the Romi with too many updates from the IMU.
5. Record the final values you used for the two.

3 Navigating the Arena

Here, you will enable your Romi to execute turns using the gyroscope to track the motion. Although you have calibrated the gyro, it will not be able to keep the absolute heading for the duration of a typical *run*. Instead, we'll use the *relative* heading, which won't drift much over the duration of a typical *turn*.

As discussed in class, you'll need to update some of the navigation code, but first, let's focus on turning.

3.1 Turning

Procedure

1. Add two `int8_t` variables, `currDirection` and `targetDirection`. We make them integers corresponding to `0 = EAST`, `1 = NORTH`, and so forth. By making them integers, it is easy to do "turning math," and we can easily constrain the values by using the `%` modulus operator.
2. Add code to the `Robot::EnterTurning()` function to set a target direction from the current direction, based on the desired amount of turns (which should also be an integer – you'll need to change the function definition). Command the Romi to rotate in the correct direction at a reasonable speed.
3. Add code to check for the completed turn. While you will have to deal with turns in both directions at some point, you can concentrate on positive turns first. You only need to check for completion when there is a heading update, so you can move the checker inside the conditional triggered by `CheckForHeadingUpdate()`. Also, while proper checker structure is generally a good idea, here it becomes cumbersome. Better is to use a state variable to reduce the complexity of the checker (for example, if the current direction is the same as the target direction, the checker can skip the logic of checking the heading).
4. Add code to `Robot::HandleTurnComplete()` to handle the completed turn. You will need to update the direction. For testing, it is best to just go to the `IDLE` state.
5. Add code to respond to one of the arrow keys on your remote so you can test your turning algorithm.

[1] An LED pin is a good choice.

3.2 Navigation

To simplify the project, we'll limit the arena to a grid of intersections, which we can represent by i, j pairs. Then, since the robot's direction is represented by a cardinal direction, it becomes straightforward to update i and j based on the direction the robot is driving.

Procedure

1. In the `Robot` class, add variables to keep track of the Romi's location. Specifically, you'll need to track `iGrid` and `jGrid` and you'll also want a target destination, `iDest` and `jDest`.
2. Add code to `HandleIntersection()` to update i or j, depending on the direction the robot is driving.
3. Finally, write code to command your Romi to drive out `N` intersections (using the IR remote), turn around, and return to the start. You will need to code in a guard condition: the action of the Romi will change depending on if the Romi has reached the target intersection or not.

4 Verification

Test your Romi by commanding it to go different counts out and back. Record your results in the table in the Worksheet. How well does it perform?

5 Extra Challenges

There are lots of extensions that you can do to challenge yourself a little more. Some ideas include:

- Implement a PID routine for turning. You will want to change the completion condition from "passes the target" to "close enough" to make it work well.
- On that note, a very challenging exercise would be to create a turning motion that ramps up and then back down to execute turns.
- Add the ability to drive to any i, j intersection. You can assume no obstacles, so your algorithm might look something like: First, drive until the correct j coordinate, then turn and drive to the correct i coordinate.

Worksheet

Turn in one copy of this worksheet for the pair.

Question

What is the default output data rate (ODR) for the gyroscope? How does the rate compare to the time step as indicated by `millis()`?

Question

The line of code below sort of works, but it's not correct.

```
if(getStatus() & 0x01)
```

What should it read? Why? Walk us through the steps to check if there is new information.

Question

Which direction does the positive direction of each gyroscope axis point? Write one of: FWD, BKWD, LEFT, RIGHT, UP, DOWN, STRANGE, or CHARM for each axis. What is the approximate average of the *raw* gyroscope values on each axis? What angular velocity does that correspond to? Don't forget your units!

Axis	+ Direction	raw value	ω
x			
y			
z			

Question

When you slowly turn the Romi 360 degrees, how far does it think it has turned? Why is there a discrepancy? Explain by referencing the data in the previous question.

Question

After calibration, is the turn more accurate? Explain.

Question

When you rapidly turn the Romi by 90 degrees, how accurate is it? Explain.

Question

From the oscillosocpe readings, how long does a typical IMU transaction take? What is the theoretical upper limit on how fast you can read the IMU?

Question

What did you settle on for the ODR and FS? Why did you choose those values? What are the trade-offs for different FS values? What are the trade-offs of different data rates?

Question

Although we're specifying that you will use the IMU for managing turns, list out at least three methods you could use for executing turns. List the advantages and disadvantages of each.

Question

Before you get started coding, draw out a state machine for the navigation of the arena, keeping in mind that turning (being non-blocking) is a *state*. Be sure to label the transitions with events and actions.

Question

Command your Romi to drive 1, 2, and 3 intersections before turning around. Record how well it does for at least 5 iterations each.

Intersection count	Attempts	Successes
1		
2		
3		

Comment on the Romi's performance. Were there any failures? Did it ever lose the line? Qualitatively, how well does it complete turns? Note: failures are not the end of the world. Be honest and don't go overboard trying to solve the occasional miscue.

Question

Take a video of your Romi navigating the arena and paste a clickable link here.

Question

Create a *Release* of your code on GitHub (either student's code) and paste a clickable link to it here.

6 Wrapping up

Question

Team Member Contributions

1. Independently, describe how you contributed to the design tasks for this week's activities.

   Student A _____

   Student B _____

2. Assign a weight (as a percentage) to each team member's contribution.

Student	Percentage

3. If either of you went beyond expectations for this lab, describe any extra work that you did. There is no requirement that both students work on the same activities, so be sure to identify who did what.