Lab # 3: Due Sunday Feb 25th by 11:59pm EST

This homework will further explore the use of LCM for building robots. In addition, we will take an in-depth look at IMUs - in terms of calibration and their role in navigation.

1. Isolate LCM code in its own directory

   You have all built a device driver for the GPS puck. Most of you did this by using the examples directory in the LCM install. For starters I would like you to extract the GPS driver and build it in a local directory. So,

   1. `mkdir ~/my_robot`
   2. In the my_robot directory make a lcmtypes subdirectory to store your type definitions
   3. Move the minimal number of files that you need to make this self-sufficient.
   4. Check to see that you have all shell files you need to compile and run
   5. Run and check your GPS device driver.

   Note to address Ubuntu USB latency issue:

   The Udev rule to make the VN-100 plug & play

   As Sudo create a file under /etc/udev/rules.d called 50-VN-100.rules with the following

   ```
   KERNEL=="ttyUSB[0-9]*", ACTION=="add", ATTRS{idVendor}=="1d6b",
   ATTRS{idProduct}=="0002", MODE="0666", GROUP="dialout"
   ```

   Also, Ubuntu's default latency of 16ms cause issues with high rate sensors, this can be solved by adding the following UDEV rules as 49-USB-LATENCY.rules

   ```
   ACTION=="add", SUBSYSTEM=="usb-serial", DRIVER=="ftdi_sio",
   ATTR{latency_timer}="1"
   ```

   Once the rules have been added, to get udev to recognize the rule, run the following command:

   ```
   sudo udevadm control --reload-rules && sudo service udev restart &&
   sudo udevadm trigger
   ```

   Finally unplug and replug the VN-100 to have it work with the new rules.

2. Add device driver for IMU

   Add the LCM typedef for the IMU in the `~/my_robot/lcmtypes` directory
   Add a device driver for the Vectornav IMU in the `~/my_robot` directory
   Make sure everything compiles correctly
   Run your IMU device driver and test that it works by moving the IMU along different axes
   Now run both the GPS and the IMU simultaneously on your computer
   Note: You should have one run file that spawns off the lcm-logger, the lcm-spy and the two device drivers. Make sure that you can set the serial port for both device drivers in the run file.
   Note: Using LCM-spy verify that the IMU driver is sending messages at the expected frequency that you have set the IMU at.

3. Run individual device drivers on separate machines

   This is also a good time to explore the network options associated with lcm. Check to see that you can run your lcm code without an internet (wifi or hardwired) connection. Look for the note on how to do this.
   https://lcm-proj.github.io/multicast_setup.html
   Also note the udp ttl value. ttl stands for time to live. Figure out what setting you need to make it so that you can run the device drivers for two separate sensors on two separate laptops connected through a network switch. Set your local ip addresses on a 192.168.1.xx network for simplicity. If you do this correctly only one of the machines should be running the logger and the spy.
   Take screen shots and photos showing that this works on your setup.

4. Collect data with IMU and GPS puck

   Begin collecting a time series of the 3 accelerometers, 3 angular rate gyros, 3- axis magnetometers,. Collect at least 1000 points of data with the instrument stationary and **as far away as possible from the computer.** Modify read_lcm.m to make sure that you can import the data into Matlab. **Plot each time series and figure out the noise characteristics of each of the values reported by the IMU**.

   Mount the IMU in your vehicle with electrical tape. Make sure that the x –axis is pointed forward. Fix the GPS puck to the roof – it has a magnetic back and should just stay there. Connect both sensors to your laptop and begin logging. Wait 10 seconds and start up vehicle. Drive course including 360 turns for compass calibration.
   After finishing the mission, shut down logging. Convert the lcm logfile to matlab structs that contain your GPS and IMU data.

5. Integrate IMU data to obtain displacement and compare with GPS.

   Correct magnetometer readings for "hard-iron" and "soft-iron" effects. Calculate the yaw angle from the corrected magnetometer readings.

   Integrate the yaw rate sensor to get yaw angle. One way to do this is to use a 1-pole butterworth lowpass filter. A second way is to use 'trapz' or 'cumtrapz' commands in Matlab and treat your time series as the function that is being integrated. **Compare the two methods.**

   Use a complementary filter to combine the measurements from the magnetometer and yaw rate as described in class to get an improved estimate of yaw angle. You might also find the 'unwrap' command useful.
   Compare your result to the yaw angle computed by the IMU.

   We simplify the description of the motion by assuming that the vehicle is moving in a two-dimensional plane. Denote the position of the center-of-mass (CM) of the vehicle by (X,Y,0), and its rotation rate about the CM by (0,0,ω). We denote the position of the inertial sensor in space by (x,y,0), and its position in the vehicle frame by (x_c,0,0). Then the acceleration measured by the inertial sensor (*i.e.* its acceleration as sensed in the vehicle frame) is

$$\ddot{x}_{obs} = \ddot{X} - \omega\dot{Y} - \omega^2 x_c$$
$$\ddot{y}_{obs} = \ddot{Y} + \omega\dot{X} + \dot{\omega}x_c$$

   where all of the quantities in these equations are evaluated in the vehicle frame.

   1. Assume that $\dot{Y} = 0$ (that is, the vehicle is not skidding sideways) and ignore the offset by setting $x_c = 0$. Then the first equation above reduces to $\ddot{X} = \ddot{x}_{obs}$. Integrate this to obtain $\dot{X}$. **Compute $\omega\dot{X}$ and compare it to $\ddot{y}_{obs}$. How well do they agree? If there is a difference, what is it due to?**

   2. Use the heading from the magnetometer to rotate
$$\ddot{x} = \dot{v} + \omega \times v = \ddot{X} + \dot{\omega} \times r + \omega \times \dot{X} + \omega \times (\omega \times r)$$
      into a fixed (East,North) reference frame. Denote this vector by (v_e,v_n).
      Integrate it to estimate the trajectory of the vehicle (x_e,x_n) and compare with the GPS track.
      Report any scaling factor used for comparing the tracks.

   3. **Estimate x_c.**

NOTE: Denote the position and velocity of the center-of-mass (CM) of the vehicle by **R** and **V**, respectively. The inertial sensor is displaced from the CM by **r** = (x_c,0,0) - note that this vector is constant in the vehicle frame. Let **ω** = (0,0,ω) denote the rotation rate of the vehicle about the CM. The velocity of the inertial sensor is **v** = **V** + **ω** x **r**, and its

corresponding acceleration is:

$$\ddot{x} = \dot{v} + \omega \times v = \ddot{X} + \dot{\omega} \times r + \omega \times \dot{X} + \omega \times (\omega \times r)$$

where we have denoted **v** by $\dot{x}$, and **V** by $\dot{X}$. Taking the x- and y-components of this equation in the vehicle frame gives the two equations quoted above.

Submission Instructions: After you are done please

- Create a folder named "lab3" in the your gitlab repository and copy your my_robot folder, data files, Matlab scripts and any other relevant files into it.
- Add, commit and push your files to the gitlab server
- Go to gitlab.com and verify that the files you have committed, show up there
- Upload your gitlab link and a **pdf** file of the slides to **Blackboard**
- Email the TA and instructor with "EECE-5698 Lab #3 pushed" in the subject line.