

## Lab 1

**Deadline:** As posted to Piazza

### **Contents:**

1. Lab 1 instructions page 1 - 4
2. Submission instructions and Grading Rubric – page 5
3. Appendix: A serial driver example - page 6 -7

### **Hardware/ Sensors**

1. USB-based GNSS puck, issued one per team.

### **Data collection Policy**

Everyone in the team needs to:

1. Write your own device driver for data acquisition.
2. Collect your own datasets individually. You will share the GPS based GNSS puck issued with your teammates.


### **0. More ROS tutorials**

Complete these two ROS tutorials:

- [Recording And Playing Back Bag Data](#)
- [Reading Messages From a Bag File](#)

### **1. Set up the GPS puck**

If you are using a virtual machine like virtual box, you may have to

1. Connect the USB.
2. Go to the Settings of the VM
3. Select the USB settings from the left pane
4. Add the USB device in the list from by clicking on the  icon
5. Boot the VM with the USB device plugged in
6. Make sure the device is selected/captured in USB settings (click on it) before you can see it in `/dev/ttyUSB*`

When you will connect your GPS device, you can see the device name with this terminal command.

```
$ ls -lt /dev/tty* | head
```

You will see a list of devices from which you have to figure out your device file identifier. Let us say it is `/dev/ttyUSB2`

Then you need to set the read write permissions for reading the device properly.

```
$ sudo chmod 666 /dev/ttyUSB2
```

You are all set for reading the device using minicom now, and for more information please check out this URL, which is also posted to Piazza: [A quick guide to use minicom](#)

Get minicom with:

```
$ sudo apt install minicom
```

Configure your device's settings in minicom:

```
$ sudo chmod 666 /etc/minicom
```

```
$ minicom -s
```

Go to serial port setup, change the Bps/Par/Bits to "4800" and Hardware Flow Control to "No".

Run minicom with

```
$ minicom
```

To save data to a text file, you need to use `-c` flag on minicom. In minicom, Ctrl-A brings up your options, including to exit. Save as `gps-data.txt`. When you want to stop writing to the file, press "ctrl-a" and then "x"

To check the contents of the file: `$ more gps_data.txt`

<b>Linux reminder!</b>
If you want to understand any command, you can either go to website man pages of linux or type the following on terminal: <code>\$ man &lt;command_name&gt;</code>

## **2. Write a Device Driver for GNSS puck to parse \$GPGGA**

As we will see in class, the GNSS puck provides several differently formatted messages. We will focus our attention on the messages that are formatted according to the \$GPGGA format.

2.1 Ensure you can read the data from the puck (If you do not have the puck, you can use minicom as a virtual puck & the same driver should work with the real puck)

2.2 Parse the \$GPGGA string for the latitude, longitude, and altitude. *We have provided an example device driver for a depth sensor in the appendix section so that you can use that as a template*

2.3 Convert the latitude & longitude to UTM.

2.4 Define a custom ROS message (called *gps\_msg.msg*) with a Header, Latitude, Longitude, Altitude, UTM\_easting, UTM\_northing, Zone, Letter as fields.

- *Please match the naming & capitalization*
- *Ensure correct data types.*
- *The Header is supposed to be a ROS Header data type. This is very important especially when you start working with tf's and do sensor fusion in ROS*
- *The Latitude & Longitude are supposed to be signed floats.*
- *The ROS Header should contain the GPGGA time stamp & not your system time (as it may be out of sync which could cause problems in a real-world system).*
- *The frame\_ID should be a constant "GPS1\_Frame" since our publisher is giving us data from the solo GPS sensor we gave you.*

2.5 Your ROS node should then publish this custom ROS message over a topic called */gps*

2.6 You now have a working driver, let's make it more modular. Name this GPS driver as *gps\_driver.py* and add a feature to run this file with some argument. This argument will contain the path to the serial port of the GPS puck (*example /dev/ttyUSB2* . *Ofcourse the puck will not always be at the same port so it allows us to connect it anywhere without the script failing*)

2.7 Even though this driver is now sufficiently modular, on a real robot we can have many sensors & launching their drivers individually can be too much work. This is where we shall use the power of ROS.

- *Create a launch file called "driver.launch"*
- *This launch file should be able to take in an argument called "port" which we will specify for the puck's port.*
- *Have this launch file run your gps\_driver.py with the argument that was passed when it was launched.*
- *If you have done everything correctly, run the following command & you should get the same results you were getting at 2.5*

```
$ roslaunch driver.launch port:="/dev/ttyUSB0" #Or basically any ttyUSB*
```

### **3. Go outside and collect data**

3.1 Stationary data outdoors: In a new rosbag recording, go outside and collect 10 minutes of data at one spot. Name this rosbag “stationary\_data.bag”

3.2 Walk in a straight line outdoors: In a new rosbag recording, walk in a straight line for a few hundred meters. Name this rosbag “walking\_data.bag”

### **4. Data analysis and report**

4.1 Read the rosbag data into MATLAB (requires ROS toolbox) or python (make sure you install bagpy with `pip3 install bagpy`, etc.):

<https://www.mathworks.com/help/ros/ref/rosbag.html>

<https://www.mathworks.com/help/ros/ref/readmessages.html>

4.2 Analyze stationary data: Examine the data at your leisure in a comfortable spot! by plotting it or analyzing its statistics.

- What does this say about GPS navigation?
- What can you say about the distribution of the error in GPS?
- What is a good error estimate?
- Can we put bounds to these errors?
- What is/are the source(s) of these errors?

4.3 Analyze straight line walk data: Examine the utm data (by plotting it or doing statistics on it)

- What does this say about GPS navigation when moving?
- How does the error estimate change as you move as opposed to stay in a spot? What can you say about the distribution of noise in this case?

4.4 Report: Analyze the data as asked above and write your observations in a **brief 1 - 2 page report with plots/charts** and submit a **PDF copy of it in your GitLab repository**. You need to tell us what you are inferring from a graph/data (*we like to know what you learnt and for that we need both your graph & remarks in your report 😊*). Please ensure all plots are labeled & titled.

## **How to Submit Lab1**

1. In your class repo 'EECE5554', create a folder called LAB1
2. Inside LAB1, create sub-directory structure 'src'.
3. Push your device driver and analysis code to GitLab. The directory structure should be:
  - src (directory)
    - gps-driver (a directory which will also be the name of your custom ROS package)
      - launch (directory with the driver.launch file)
      - python (directory with the gps\_driver.py file)
      - msg (directory with the gps\_msg.msg file)
      - CMakeLists.txt & package.xml (if they are there in your original workspace)
    - report.pdf (file)
    - analysis-scripts (directory with all your analysis scripts & a requirements.txt for dependencies)
    - data (directory with the two bag files)
    - CMakeLists.txt
4. Please do not give us your entire ROS workspace, just the src folder with the Cmake files
5. Upload your lab1/src (local computer) to GitLab. Push your local commits to (remote) GitLab server. You can verify this by visiting gitlab.com and making sure you can see the commit there. Your repo structure should look similar to  
'<Path\_to\_repo>/EECE5554/LAB1/src/'
6. Upload your **pdf report to GitLab (also under the src directory)**
7. Ensure that all file names, parameters & folder structures match this document. To verify your code works, you can clone your repository in a separate folder, do a *catkin\_make* in the LAB1 folder, source the devel/setup.bash in this clone and then run the *roslaunch* command mentioned in 2.7.

## **Grading Rubric (10 Points)**

- 2 points for working device driver
- 3 points for Analysis of Stationary Data
- 3 points for Analysis of Moving Data
- 2 Points for overall Presentation of Report

Late submission policy is mentioned in the syllabus.

## Appendix: Python based ROS node for Depth sensor on an AUV

```
1. #!/usr/bin/env python
2. # -*- coding: utf-8 -*-
3.
4. import rospy
5. import serial
6. from math import sin, pi
7. from std_msgs.msg import Float64
8. from nav_msgs.msg import Odometry
9.
10. def paro_to_depth(pressure, latitude):
11.     '''
12.     Given pressure (in m fresh) and latitude (in radians) returns ocean depth (in m.). Uses the
13.     formula discovered and presented by Leroy and Parthiot in: Claude C. Leroy and Francois Parthiot,
14.     'Depth-pressure relationships in the oceans and seas', J. Acoustic Society of America, March 1998,
15.     p1346-.
16.     '''
17.     # Convert the input m/fw into MPa, as the equation expects MPa.
18.     pressure = pressure * 0.0098066493
19.     # Gravity at Latitude.
20.     g = 9.780318 * (1 + 5.2788e-3*sin(latitude)**2 -
21.                    2.36e-5*sin(latitude)**4)
22.     # Now calculate the 'standard ocean' depth.
23.     Zs_num = (9.72659e2*pressure - 2.512e-1*pressure**2 +
24.              2.279e-4*pressure**3 - 1.82e-7*pressure**4)
25.     Zs_den = g + 1.092e-4*pressure
26.     return Zs_num / Zs_den
27.
28. if __name__ == '__main__':
29.     SENSOR_NAME = "paro"
30.     rospy.init_node('depth_paro')
31.     serial_port = rospy.get_param('~port', '/dev/ttyS1')
32.     serial_baud = rospy.get_param('~baudrate', 9600)
33.     sampling_rate = rospy.get_param('~sampling_rate', 5.0)
34.     offset = rospy.get_param('~atm_offset', 12.121) # in meter ??
35.     latitude_deg = rospy.get_param('~latitude', 41.526) # deg 41.526 N is Woods Hole
36.
37.     port = serial.Serial(serial_port, serial_baud, timeout=3.)
38.     rospy.logdebug("Using depth sensor on port "+serial_port+" at "+str(serial_baud))
39.     rospy.logdebug("Using latitude = "+str(latitude_deg)+" & atmosphere offset = "+str(offset))
40.     rospy.logdebug("Initializing sensor with *0100P4\\r\\n ...")
```

```

39.     sampling_count = int(round(1/(sampling_rate*0.007913)))
40.     port.write('*0100EW*0100PR='+str(sampling_count)+'\r\n') # cmd from 01 to 00 to set sampling
period
41.     rospy.sleep(0.2)
42.     line = port.readline()
43.     port.write('*0100P4\r\n') # cmd from 01 to 00 to sample continuously
44.
45.     latitude = latitude_deg * pi / 180.
46.     depth_pub = rospy.Publisher(SENSOR_NAME+'/depth', Float64, queue_size=5)
47.     pressure_pub = rospy.Publisher(SENSOR_NAME+'/pressure', Float64, queue_size=5)
48.     odom_pub = rospy.Publisher(SENSOR_NAME+'/odom', Odometry, queue_size=5)
49.
50.     rospy.logdebug("Initialization complete")
51.
52.     rospy.loginfo("Publishing pressure and depth.")
53.
54.     odom_msg = Odometry()
55.     odom_msg.header.frame_id = "odom"
56.     odom_msg.child_frame_id = SENSOR_NAME
57.     odom_msg.header.seq=0
58.
59.     sleep_time = 1/sampling_rate - 0.025
60.
61.     try:
62.         while not rospy.is_shutdown():
63.             line = port.readline()
64.             #print line
65.             if line == '':
66.                 rospy.logwarn("DEPTH: No data")
67.             else:
68.                 if line.startswith('*0001'):
69.                     odom_msg.header.stamp=rospy.Time.now()
70.                     try: pressure = float(line[5:].strip())
71.                     except:
72.                         rospy.logwarn("Data exception: "+line)
73.                         continue
74.                     pressure_pub.publish(pressure)
75.                     depth_mes = paro_to_depth(pressure - offset, latitude_deg)
76.                     depth_pub.publish(depth_mes)
77.                     odom_msg.pose.pose.position.z = -depth_mes
78.                     odom_msg.header.seq+=0
79.                     odom_pub.publish(odom_msg)
80.                     rospy.sleep(sleep_time)
81.
82.     except rospy.ROSInterruptException:
83.         port.close()
84.
85.     except serial.serialutil.SerialException:
86.         rospy.loginfo("Shutting down paro_depth node...")
87.
88.

```