**LAB\_1** - EECE 5554 Robotics Sensing and Navigation

**Term:** Fall 2023

**Deadline**: 10 PM on Friday, Sept 29, 2023

**Contents:**  
1. Lab 1 instructions page 1-2

2. Submission instructions and Grading Rubric – page 3

3. Appendix: A serial driver example - page 4

**Hardware/ Sensors**

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

**Data collection Policy**

Everyone in the team needs to

1. Write their own device driver for data acquisition.
2. Everyone will be collecting their own datasets individually. They will be sharing the GPS device issued to them.
3. **What to Submit for Lab 1 ?**
4. **Setting up the GPS puck**

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 i dentifier. Let us say it is /dev/ttyUSB2

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

$ chmod 666 /dev/ttyUSB2

You are all set for reading the device using minicom now. Configure your device’s settings in minicom. [A quick guide to use minicom](https://wiki.emacinc.com/wiki/Getting_Started_With_Minicom)

For saving data to a text file, you need to use –C flag on minicom. Save as gps-data.txt

Or simply copy paste the terminal output and save it in a text file. Cd

This creates a gps-data.txt file in your current directory. When you want to stop writing to the file, press ctrl c .

$ more gps\_data.txt

For checking the contents of file

**Additional note:**

If you want to understand any command, you can either go to web man pages of linux or type the following on terminal.

$ man <command\_name>.

1. **Write Device Driver for GNSS puck to parse $GPGGA**

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

We need a driver to read in serial data from the puck, parse it 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.*

You need to then convert the latitude and longitude to utm using the python package “utm” as discussed in class.

Define a custom ros message (called gps\_msg.msg) with header, latitude, Longitude, Altitude, utm\_easting, utm\_northing, Zone, letter as fields.

Few things to keep in mind while writing your driver:

* The Header is supposed to be a ROS Header data type. This is very important especially when you start working with tfs and do sensor fusion in ROS.
* 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.

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

You now have a working driver, let’s make it more modular. Name this GPS driver as

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)

**Create a launch file:**

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 before (but with a single command!)

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

1. **Go outside and collect data**

**2.1 Stationary data outdoors**

**Data-collection**

Go outside and collect 10 minutes of data at one spot in a **rosbag**. Name this rosbag “stationary\_data.bag”

**2.2 Walk in a straight line outdoors**

In a new ros-bag recording, walk in a straight line for a few hundred meters. Name this rosbag “walking\_data.bag”

1. **Data analysis and report:**

Read the rosbag data into matlab/python.

**4.1** **Analyse stationary data**  
Examine the data at your leisure 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 on these errors?

What is the source of these errors?

**4.2 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?

**4.3 Report:**

Analyse 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.**

**How to Submit Lab1**

1. In your class repo ‘EECE5554’, create a directory 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 (directory)**

- launch (directory with the driver.launch file)

- python (directory with the driver.py file)

- msg (directory with the gps\_msg.msg file)

- CMakeLists.txt & package.xml (if present in your original workspace)

* + - **Report.pdf (file)**
    - **Analysis scripts**
    - **data**
    - **CmakeLists.txt**

1. 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/’
2. Upload your **pdf report to GitLab (also under the src directory)**

**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**

1. Python based ros node for Depth sensor on an auv

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

import rospy

import serial

from math import sin, pi

from std\_msgs.msg import Float64

from nav\_msgs.msg import Odometry

def paro\_to\_depth(pressure, latitude):

'''

Given pressure (in m fresh) and latitude (in radians) returns ocean depth (in m.). Uses the formula discovered and presented by Leroy and Parthiot in: Claude C. Leroy and Francois Parthiot, 'Depth-pressure relationships in the oceans and seas', J. Acoustic Society of America, March 1998, p1346-.

'''

# Convert the input m/fw into MPa, as the equation expects MPa.

pressure = pressure \* 0.0098066493

# Gravity at Latitude.

g = 9.780318 \* (1 + 5.2788e-3\*sin(latitude)\*\*2 -

2.36e-5\*sin(latitude)\*\*4)

# Now calculate the 'standard ocean' depth.

Zs\_num = (9.72659e2\*pressure - 2.512e-1\*pressure\*\*2 +

2.279e-4\*pressure\*\*3 - 1.82e-7\*pressure\*\*4)

Zs\_den = g + 1.092e-4\*pressure

return Zs\_num / Zs\_den

if \_\_name\_\_ == '\_\_main\_\_':

SENSOR\_NAME = "paro"

rospy.init\_node('depth\_paro')

serial\_port = rospy.get\_param('~port','/dev/ttyS1')

serial\_baud = rospy.get\_param('~baudrate',9600)

sampling\_rate = rospy.get\_param('~sampling\_rate',5.0)

offset = rospy.get\_param('~atm\_offset',12.121) # in meter ??

latitude\_deg = rospy.get\_param('~latitude',41.526) # deg 41.526 N is Woods Hole

port = serial.Serial(serial\_port, serial\_baud, timeout=3.)

rospy.logdebug("Using depth sensor on port "+serial\_port+" at "+str(serial\_baud))

rospy.logdebug("Using latitude = "+str(latitude\_deg)+" & atmosphere offset = "+str(offset))

rospy.logdebug("Initializing sensor with \*0100P4\\r\\n ...")

sampling\_count = int(round(1/(sampling\_rate\*0.007913)))

port.write('\*0100EW\*0100PR='+str(sampling\_count)+'\r\n') # cmd from 01 to 00 to set sampling period

rospy.sleep(0.2)

line = port.readline()

port.write('\*0100P4\r\n') # cmd from 01 to 00 to sample continuously

latitude = latitude\_deg \* pi / 180.

depth\_pub = rospy.Publisher(SENSOR\_NAME+'/depth', Float64, queue\_size=5)

pressure\_pub = rospy.Publisher(SENSOR\_NAME+'/pressure', Float64, queue\_size=5)

odom\_pub = rospy.Publisher(SENSOR\_NAME+'/odom',Odometry, queue\_size=5)

rospy.logdebug("Initialization complete")

rospy.loginfo("Publishing pressure and depth.")

odom\_msg = Odometry()

odom\_msg.header.frame\_id = "odom"

odom\_msg.child\_frame\_id = SENSOR\_NAME

odom\_msg.header.seq=0

sleep\_time = 1/sampling\_rate - 0.025

try:

while not rospy.is\_shutdown():

line = port.readline()

#print line

if line == '':

rospy.logwarn("DEPTH: No data")

else:

if line.startswith('\*0001'):

odom\_msg.header.stamp=rospy.Time.now()

try: pressure = float(line[5:].strip())

except:

rospy.logwarn("Data exception: "+line)

continue

pressure\_pub.publish(pressure)

depth\_mes = paro\_to\_depth(pressure - offset, latitude\_deg)

depth\_pub.publish(depth\_mes)

odom\_msg.pose.pose.position.z = -depth\_mes

odom\_msg.header.seq+=0

odom\_pub.publish(odom\_msg)

rospy.sleep(sleep\_time)

except rospy.ROSInterruptException:

port.close()

except serial.serialutil.SerialException:

rospy.loginfo("Shutting down paro\_depth node...")