LAB_1 - EECE 5554 Robotics Sensing and Navigation

Term: Spring 2021

Deadline: 10 PM on Monday, Feb 7, 2022

Contents:

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 ?**

1. 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

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>.

2. **<u>Write Device Driver for GNSS puck to parse $GPGGA</u>**

   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 with header, latitude, Longitude, Altitude, utm_easting, utm_northing, Zone, letter as fields.

   Your ros node should then publish all these values in your custom defined ros msg.

3. **<u>Go outside and collect data</u>**

   2.1 Stationary data outdoors

   Data-collection

   Go outside and collect 10 minutes of data at one spot in a **rosbag**.

   2.2 Walk in a straight line outdoors

   In a new ros-bag recording, walk in a straight line for a few hundred meters.

4. **Data analysis and report:**
   Read the rosbag data into matlab/python.
   4.1 Analyse stationary data
   Examine the data at your leisure (in a warm 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 a bounds to these errors? What is the source of these errors?

4.2 Analyse 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)**
     - **Report.pdf (file)**
     - **Analysis scripts**
     - **data**
4. 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/'
5. 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...")
```