ME 133b - Lab 2

Inertial Navigation & Visual Odometry

--------------------------------------------------------------------------------------------------------------------Due Wed. Feb. 20, 2019

This lab investigates two of the three most common forms of *odometry*: inertial navigation (with an IMU) and visual odometry (using LiDAR scan matching). You have already experimented with the third form of odometry: wheel odometry.

You will use a turtlebot instrumented with an Inertial Measurement Unit (IMU) , and a 2-dimensional scanning LiDAR. You are to program the turtlebot to make known motion that ends at the same point where the robot starts; e.g., a circle or a square.. You will track the motions using the CAST OptiTrack system to get the ground truth motions of the vehicle. You will then estimate the robot’s motion using 2 different techniques: inertial navigation, and visual odometry. The goal is to get practical experience with the usefulness and shortcomings of these methods.

# **IMU Data processing**

The following steps can be completed in any data processing software such as Excel or MATLAB. Make sure that the data from the accelerometer and gyroscope have been converted into *m/s*2 and deg/s respectively.

1. First we’ll start out with a simple integration of the altitude of the system, given by the Z axis acceleration measured by the IMU. Subtract the static offset observed at the start of the test when the car was stationary, which should be around 9.81*m/s*2 given this axis is biased by gravity.
2. Double integrate the Z axis value using the IMU sampling rate as the integration time step. If you want a more accurate estimate you could use time stamp data. For the homework submission, plot the vertical position of the system over the duration of the test, and comment on the final altitude compared to the initial. How different are they despite the fact that we were driving along a flat surface?
3. Next we’ll be implementing the equations for inertial navigation taught in class to try and track the vehicle’s position in the XY plane. You’ll need to use the Z axis gyroscope reading to track the orientation of the vehicle on the plane, and the X and Y accelerations to track its position, as represented in the figure below. Remember to calibrate the sensor values by subtracting the static offset from all of the values and set initial velocity and position to zero.
4. In the homework submission, plot the vehicle’s position in the X Y plane over the course of the test run. Comment on the shape of the trajectory compared to the OptiTrack measurement.
5. Lastly, we’ll see how accurate our orientation estimates can be when tracking external stimuli such as the Earth’s magnetic field with exteroceptors like a magnetometer. Using the relation *atan*2(*Xmag, Ymag* ) to give you the angle the vehicle is facing relative to magnetic north, plot the orientation of the vehicle over the duration of test. Does it end up facing the right direction?

Include the equations you use for your labsubmission.

**Extra credit (5 points):** Integrate the complete 3 DOF orientation of the vehicle using all three gyroscope readings via the methods taught in class, and plot the resulting Euler-Angles over the course of the test run.



# **LiDAR Data processing and Visual Odometry**

# You can use one of two approaches to implement your visual odometry estimate. The most basic approach is to use an existing ROS package named ’laser\_scan\_matcher’ (that incidentally was written by a Caltech PhD student) to process all of the distance measurements from the LIDAR sensor, convert them into a point cloud, then attempt to match each successive scan with previous scans and track motion of the sensor, which is equivalent to tracking the motion of the vehicle, modulo a body-fixed displacement between the vehicle reference frame and the sensor’s reference frame. Alternatively, as described below, you can receive significant extra credit for developing your own scan matching code.

**Playing back sensor data**

1. Load the .bag file you created during your lab-time into your machine’s ’home’ folder, which can be navigated to in the terminal with the command: $ cd ~
2. You will also need to download the configuration file for rviz which sets up the GUI to play back the sensor data from our test which be found [here](http://robotics.caltech.edu/~jbowkett/turtlebot_rgbd_lidar.rviz), and loaded into your virtual machine or ROS system.
3. We can now check the .bag file to make sure it has all the data we need to run the test. To do this, open a terminal with *CTRL + ATL + T* and navigate to wherever you placed the .bag file, then run the command (replacing bag\_name with whatever yours is called) which should produce an output similar to the figure below:

$ rosbag info bag\_name.bag



1. If any of the topics have 0 messages or are missing then there may have been problems with your lab set up. In that case you may instead choose one of the back up bags.
2. Once all of the files in place we can start ROS by using the same terminal to enter the command:

 $ roscore

1. Next we can open the rviz GUI in another terminal by navigating to the location of the turtle- bot\_rgbd\_lidar.rvz file and then using the command:

 $ rviz rviz –d turtlebot\_rgbd\_lidar.rviz

1. Finally, we can play back the .bag file to see what the turtlebot saw as it was driving around during the demonstration, by navigating to the location of the file in a new terminal and running the command:

$ rosbag play --clock bag\_name.bag



1. In order to check the position of the odometry estimate against our ground truth measurement we’ll need to print out the transform between ’base\_link’ and ’odom’ by running the following command in a new terminal WHILE the .bag file is being played back:

$ rosrun tf tf\_echo /odom /base\_link

How does the final location compare to the position we measured with OptiTrack? Compare the two values in your lab report.

**Interpreting the data**

As you play back the bag file you should see two points clouds in rviz much like the figure above, the thin line comes from the 180° sweep of the lidar, and the rectangle from the RGBD camera. The three sets of axes represent the coordinate frames of the robot’s footprint, the RGBD camera, and the lidar. These are all relative to the global fixed frame named odom which the turtlebot calculates by fusing data from its encoders and IMU. We can see how this drifts by unchecking the box next to DepthCloud (seen in red above), and changing the Decay Time variable for the LaserScan from 0 to 60 seconds.

We can now play the bag file back again by clicking Reset at the bottom left of rviz, then running the ’rosbag play ...’ command again. Pay particular attention to how the apparent location of the walls around the turtlebot drifts, you should see the points diverge from a thin to a thick edged square as the odometry position estimate worsens over time and accumulates.

Take a screenshot of the pattern produced by the accumulated lidar pointcloud using odom as the fixed reference and submit it along with your lab report.

**Using LIDAR to localize**

The goal of this lab is to have you use the lidar data to produce a better estimate of where the robot is relative to the world. First you’ll need to install the ROS lidar SLAM package “laser\_scan\_matcher” which can be loaded with the following terminal command (you may need to run ’sudo apt-get update’ first):

$ sudo apt-get install ros-indigo-laser-scan-matcher

We are also going to have to create a version of our .bag file that does not contain the transformation frames ’odom’ and ’base\_footprint’ due to the fact that ROS does not allow each node in the tf tree to have more than one parent. Previously the wheel odometry topic ’odom’ was providing an estimate of the robot’s position (base\_link) relative to the fixed external frame, but now it will be the ’world’ tf produced by the laser\_scan\_matcher node.

To produce this new .bag file open a new terminal, navigate to your bag file, and run the command (changing bag\_name to the name of your file, can copy the command from [this](http://robotics.caltech.edu/~jbowkett/command.txt) text file):

$ rosbag filter bag\_name.bag no\_odom.bag "topic == ’/camera/depth/camera\_info’ or topic

== ’/camera/depth/image\_raw’ or topic == ’/mobile\_base/sensors/core’ or topic == ’/odom’ or topic == ’/scan’ or topic == ’/tf’ and m.transforms[0].header.frame\_id != ’odom’ and

m.transforms[0].header.frame\_id != ’base\_footprint’"

While this process runs we can prepare rviz to display the localization estimate from the laser scan matching. To do this click Reset at the bottom left corner, and then change Fixed Frame under Global Options from ’odom’ to ’world’ by typing this in manually and hitting enter.

Next, in order to launch the laser scan matcher and play the filtered bag file we’ll need to use a special launch file that can be downloaded from [here](http://robotics.caltech.edu/~jbowkett/lidar_estimate.launch) and placed in the home folder. This assumes your filtered bag file is called ’no\_odom.bag’ so if you changed it you’ll need to modify the launch file to account for this in the text editor (’$ gedit file\_name’ from the terminal).

Finally, we need to let the laser\_scan\_matcher node know that it will be processing data from in the past. First, run the python script [here](http://robotics.caltech.edu/~dpastorm/tf_setup.py) using the command:

$ python tf\_setup.py

In a new terminal, enter the command:

$ rosparam set use\_sim\_time true

In this same terminal we can now run our simulation with the command:

$ roslaunch lidar\_estimate.launch

Watch the accumulated lidar pointclouds in rviz. Do they seem to be more consistent than when they were referenced against the wheel odometry position estimate? Take a screenshot of the accumulated point cloud using ’world’ as the Fixed Frame and submit it along with your lab report. Once again, we can watch the transform between the fixed and robot frames by using the following command in a new terminal WHILE the bag is being played back:

$ rosrun tf tf\_echo /world /base\_link

How does the final position error compare to the error we saw when using the wheel odometry to keep track of the reference frame?

The lidar localized pointcloud was likely better grouped than when referenced with wheel odometry but it still isn’t perfect. In particular it has a habit of drifting when the turtlebot makes rapid changes of direction (as seen at the inflection point of the figure8 example link at the beginning). Much as we discussed in class, how would you go about getting an even better estimate of our current position using all of the sensors we have available?

If you wish to play back the lidar referenced data again you’ll need to click Reset in rviz and *CTRL*

*+ C* in the lidar\_estimate.launch terminal before running the launch file again.

**Extra credit (+30 points)**

The extra credit for this lab involves attempting to implement your own laser scan matching as described by Joel in class, using the wheel odometry calculations we did in the last lab as the initial estimate in the algorithm.

A template for a python ROS node can be found [here](http://robotics.caltech.edu/~jbowkett/me132b_lab3_template.py) which contains the setup for the different topics and tfs you need to read and write from. Implement the algorithm detailed in class so that your node publishes a tf from ’world’ to ’base\_link’ which will then replace the laser\_scan\_matcher node used in this lab. To test your algorithm with the recorded data follow the instructions for the laser\_scan\_matcher node but instead of running the special launch file, start your node with the command (you may need to run ’chmod u+x me133b\_lab3\_template’ first):

$ python me133b\_lab3\_template

You can then play back the filtered bag file with the command:

$ rosbag play --clock no\_odom.bag

Document your work as you will receive credit for even partial completion. A good first step might be to implement laser scan matching for a single pair of scans rather than running it live on a ROS node

# If you are having problems processing your group’s data (particularly in the laser\_scan\_matcher part), try using this bag file.

Figure 2: Inertial navigation with accelerometer and gyroscope