Extended Kalman Filter Implementation using GPS and IMU

Jingxi Chen, Francisco Chavez

Halicioglu Data Science Institute

University of California, San Diego

DSC 190 - Dr. Jack Silberman

jic046@ucsd.edu, fchavezg@ucsd.edu

What we have promised

- To implement EKF to do pose estimation with GPS and IMU
- We need to compare the GNSS fused with an external IMU and without IMU fusion to enable us to navigate/race at Purdue next year
- Plug this into EKF and compare the data from both.

Must have

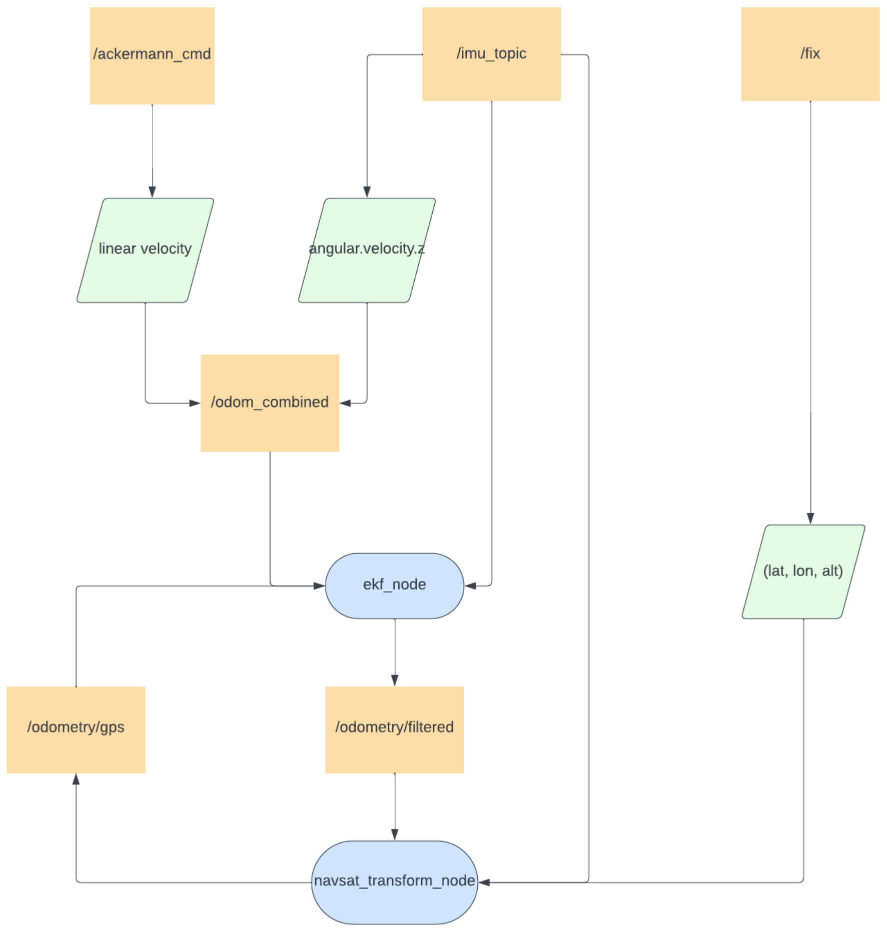
- GPS data (lat, lon, alt)
- IMU data (angular.velocity.z)
- Encoder data (linear velocity)

Nice to have

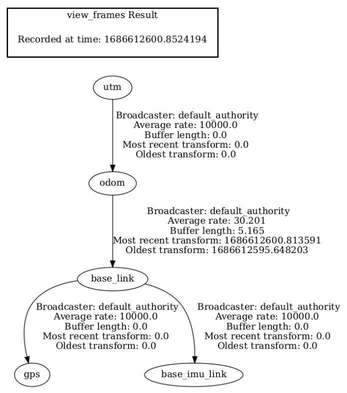
- Reliable Encoder odometry data
- Reliable Magnetometer to help with heading
- Working map frame by SLAM
    - Map_frame was required in EKF, but we did not include because we did not have one

What we have done

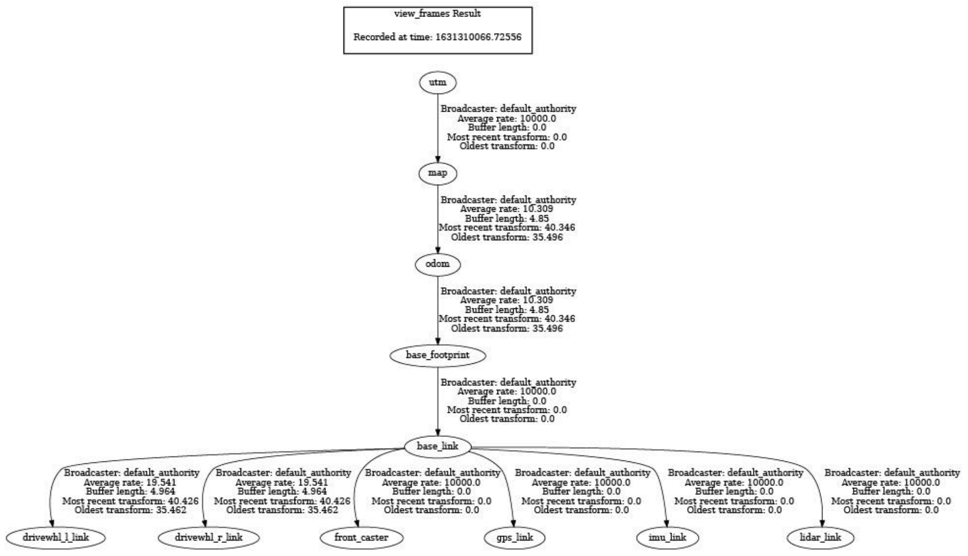
- Often worked on Python using ROS2 Framework trying to read data from the IMU Device
    - Creating functions that can read and handle data
    - Data Wrangling to make sense of the raw data being obtained from IMU
    - Being able to represent IMU data in a meaningful way, such as pose estimate, to positively impact future races
    - Wrote algorithm on python that subscribes to the ROS2 topics receiving data from IMU, and that gives us "yaw" (z) readings to help us orientate with respect to the true North
    - Helped measure the quality of the calibration of the IMU, detecting whether the "yaw" readings we get are, for example, =0 when facing the true magnetic north, =90 when facing true magnetic east, =180 when facing true magnetic south, so on
- Edited ekf_yaml and document the step-by-step change we made:
    - Documentation on ekf_yaml
- Combined linear velocity readings from /ackermann_cmd and angular velocity around the z-axis from /imu_topic → /odom_combined



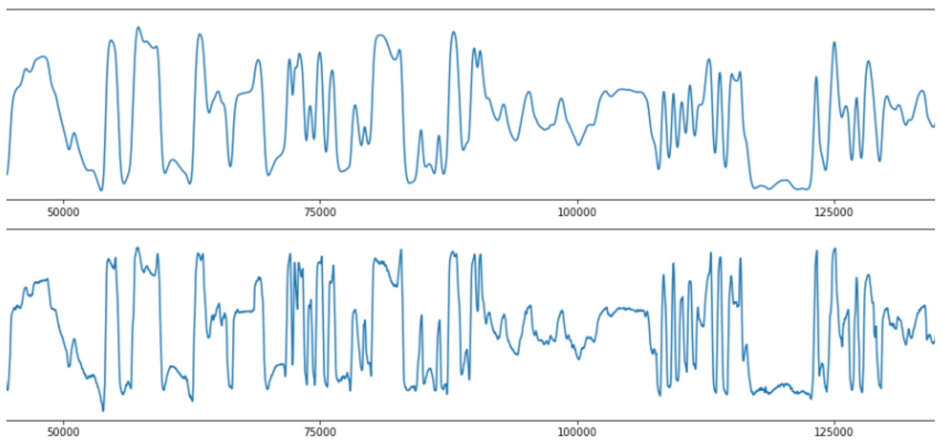
- Incorporated navsat_transform_node to translate the gps coordinates (lat,lon,alt) into pose in the local frame.
- Implemented static transform from base_link to sensors including gps and imu


view_frames Result
Recorded at time: 1686612600.8524194
utm
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
odom
Broadcaster: default_authority
Average rate: 30.201
Buffer length: 5.165
Most recent transform: 1686612600.813591
Oldest transform: 1686612595.648203
base_link
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
gps
base_imu_link

-
- Top is what we were able to get, bottom is what is expected with other sensors like LIDAR and frontcaster



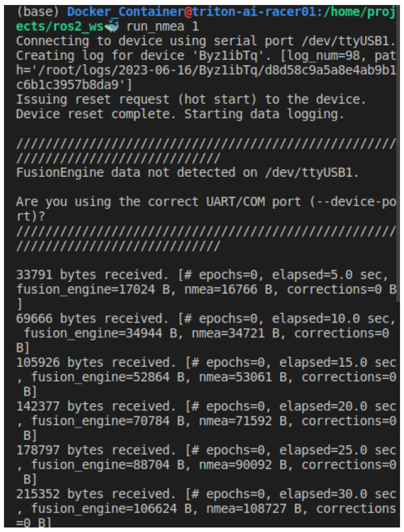
-
    - Tf-tree map of our implementation
- Recorded imu data as mcap and implemented imu_parser.py that turns the recorded mcap into csv
    - Detail step on turning mcap into csv
        - Mcap file to csv
    - With Sepher's help, we got Dead Reckoning with this CSV to plot our path with just the IMU data collected
- Implemented butterworth filter to get more smooth data from the angular velocity yaw output.



-
    - Top is angular velocity after filtering, bottom is before filtering. We found the right parameters to successfully clean up angular velocity data

- Updated p1_runner implementation in docker so we can connect to the correct polaris
-

What did not work as expected? Why?

- GPS:
    - run_nmea did not function as expected
        - Guide to use gnss:
          How to do gnss
        - We followed this guide but it did not work as written in docs because of a possible update to the implementation of p1_runner
        - 

```
(base) Docker_Container@triton-ai-racer01:/home/proj
ects/ros2_ws🐳 run_nmea 1
Connecting to device using serial port /dev/ttyUSB1.
Creating log for device 'Byz1ibTq'. [log_num=98, pat
h='/root/logs/2023-06-16/Byz1ibTq/d8d58c9a5a8e4ab9b1
c6b1c3957b8da9']
Issuing reset request (hot start) to the device.
Device reset complete. Starting data logging.

////////////////////////////////////////////////////
///////////////////////////
FusionEngine data not detected on /dev/ttyUSB1.

Are you using the correct UART/COM port (--device-po
rt)?
////////////////////////////////////////////////////
///////////////////////////

33791 bytes received. [# epochs=0, elapsed=5.0 sec,
fusion_engine=17024 B, nmea=16766 B, corrections=0 B
]
69666 bytes received. [# epochs=0, elapsed=10.0 sec,
 fusion_engine=34944 B, nmea=34721 B, corrections=0
B]
105926 bytes received. [# epochs=0, elapsed=15.0 sec
, fusion_engine=52864 B, nmea=53061 B, corrections=0
 B]
142377 bytes received. [# epochs=0, elapsed=20.0 sec
, fusion_engine=70784 B, nmea=71592 B, corrections=0
 B]
178797 bytes received. [# epochs=0, elapsed=25.0 sec
, fusion_engine=88704 B, nmea=90092 B, corrections=0
 B]
215352 bytes received. [# epochs=0, elapsed=30.0 sec
, fusion_engine=106624 B, nmea=108727 B, corrections
=0 B]
```

            - We couldn't get long/lat/alt readings with the instructions in that google docs above
- IMU:
    - Our original plan was to read the yaw values from the /diagnostics topic that's published by imu_node.py in the sensor2_ws by Sparkfun. This was not usable as it was clearly being affected by magnetometer data, which is heavily affected by the magnetic field around it. We tested this by holding our cellphone close to the

imu and we observed a major change in readings in the yaw values. Even the Compass on our phone was affected by the magnetic field that the ⅕ has.

How would you solve the issues?

- GPS: We replaced the entire p1_runner implementation with an updated p1_runner provided by Sepher (since Point1Nav constantly update their firmware), and run_nmea started working after then, specifically "run_nmea 1" in the ⅕
- IMU: There's no way to solve the issue except for getting another set of imu that has a reliable magnetometer reading.
- Map frame: use SLAM algorithms to map + localize. There are many camera-based SLAM algorithms like OrbSLAM2 or RTABMap.
    - Another option is Cartographer. It allows you to first create a map and then perform pure localization in a 3D environment.

Data collected

- Imu data: linear velocity, linear velocity covariance matrix, angular velocity, angular velocity covariance matrix
        - IMU csv file: https://drive.google.com/file/d/1BUVRRhfCoSK9WPC7nzvw3ZC75rOtkeWJ/view?usp=drive_link
- Sparkfun imu yaw values vs. actual orientation with respect to North.
    - Docs: https://docs.google.com/document/d/15zeuNwC2FeSN1CEgYI2WlHikfqKYozqFND2estZNhsY/edit?usp=sharing
-

Data science techniques used

- We did EDA on countless datasets of topic readings, trying to understand what each topic from IMU and GPS did and how the data was collected in the first place

- Passed our data through python and pandas dataframes to understand it in a tabular way, like we've been doing for the past 4 years
- Handled and cleaned this data through a butterworth filter, and plotted using what we've learned from visualization classes like DSC 106 and other lower-div DSC classes
- Converted data files such as mcap to csv using pandas dataframes to formulate our GPS and IMU readings through a python nested for loop which collect our data in dictionaries to input into a pandas dataframe, to then convert to csv for our map plotting