UNIVERSITY OF TORONTO
INSTITUTE FOR AEROSPACE STUDIES

September 19, 2021

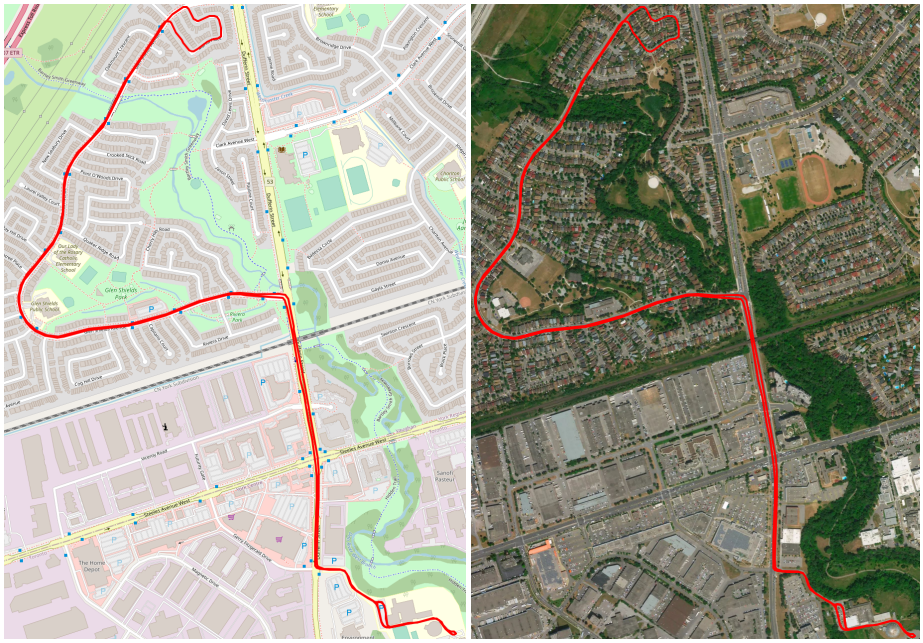
A Guide to the Boreas Dataset

<keenan.burnett@utoronto.ca>

1 Introduction

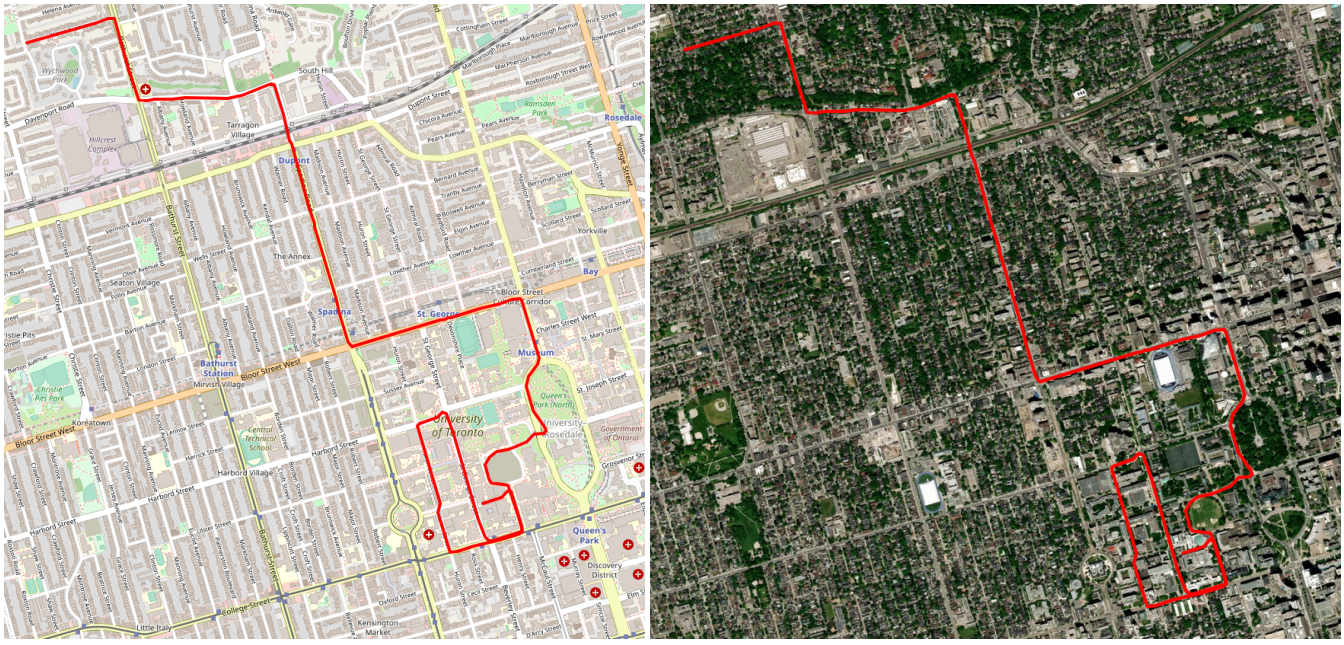
What is contained in this dataset? Our sensor suite includes a 128-beam Velodyne Alpha-Prime 3D lidar, a Navtech CIR204-H 360° radar, a FLIR Blackfly S (5 MP / 10 FPS) monocular camera, and an Applanix POS LV GNSS. All positioning data was obtained by post-processing the Applanix POS LV data. Data was collected during repeated traversals of two routes in Toronto, Canada, across several seasons and weather conditions. The first route, referred to as the Glen Shields route, is shown in Figure 1. The second route, referred to as the St George route, is shown in Figure 2. In the future, we are planning to provide 3D and 2D object labels, semantic segmentation labels, and an HD map for each of the routes.

What is this dataset for? This dataset and the associated benchmarks are intended to support odometry (lidar, radar, visual), metric localization (lidar, radar, visual), 3D object detection (lidar) and 2D object detection (radar, visual). By including data taken over several seasons and multiple weather conditions, we expect that this dataset will present a chalenging and exciting benchmark for the aforementioned autonomous driving tasks. We hope that the inclusion of a 360° radar will spur further interest in the use of this sensor towards robust autonomy in all weather conditions.



(a) OpenStreetMap Data (b) Mapbox Satellite Data

Figure 1: The Glen Shields route in Toronto, Ontario, Canada. (Video)



(a) OpenStreetMap Data

(b) Mapbox Satellite Data

Figure 2: The St George route in Toronto, Ontario, Canada. (Video)

2 Sensor Specifications

Boreas' sensor configuration is displayed in Figure 3. The Velodyne Alpha Prime lidar has a 40° vertical field of view, with a range of up to 300m for targets with 10% reflectivity or up to 180m for targets with 5% reflectivity. This lidar has a 0.2° horizontal angular resolution and a 0.1° vertical angular resolution. This sensor typically produces over 2M points per second. The lidar is configured to have a rotational rate of 10Hz, resulting in over 200k points per rotation. We retain only the strongest lidar returns instead of keeping dual returns per laser firing sequence.

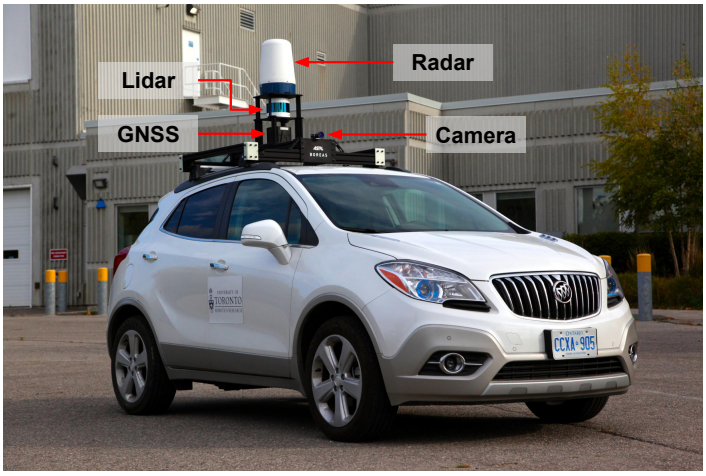
The Navtech CIR204-H radar has a range resolution of 0.0596m per range bin with a total range of 200.256m. The sensor spins at 4Hz and provides 400 measurements per rotation, resulting in a horizontal angular resolution of 0.9°.

The FLIR Blackfly S monocular camera has a resolution of 2448 x 2048. Based on the calibration, the camera has a field of view of approximately $81° \times 71°$. We extract camera images at 10Hz to minimize storage requirements.

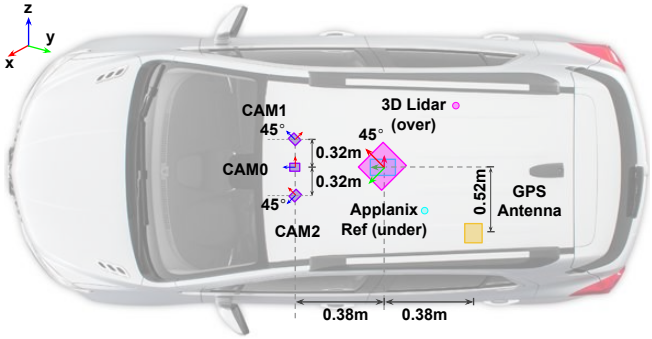
The Applanix POSLV system includes an external wheel encoder, and an extra satellite subscription (RTX) which improves the accuracy. All of the logged Applanix data is post-processed using their proprietary POSPac suite in order to perform a batch optimization over each sequence. The post-processed position data can be expected to have an RMS error between 5 cm and 20 cm, depending on the sequence.

3 Data Organization

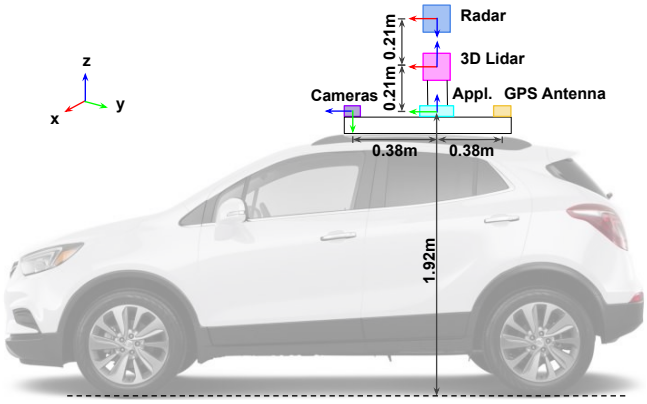
Each sequence is stored as a separate Amazon S3 bucket and follows the same naming convention: `boreas-YYYY-MM-DD-HH-MM` denoting the time that data collection started. Figure 4 provides an overview of the folder structure of each bucket. Note that each bucket includes a `route.html` file which can be used to inspect the driven route in an internet browser. Each route also has an associated video, `video.mp4`, which is also available on YouTube. These videos



(a) Boreas



(b) Top View



(c) Side View

Figure 3: These diagrams of sensor locations and orientations are provided as a quick reference only. We have provided hand-measured and calibrated transformation matrices for several pairs of sensor frames.

can be used to select which sequences are desired for experimentation. Camera images are extracted at 10 Hz, lidar pointclouds at 10 Hz, and radar scans at 4 Hz. Post-processed positioning data is provided at 200 Hz. Please note that users have the option of downloading only the data from a single sensor if desired. Accessing and downloading the dataset is done using the AWS CLI. An example command for downloading only the camera data from a particular sequence is given below:

```
aws s3 sync s3://boreas-2020-11-26-13-58 .  --exclude '*' --include 'camera/*'
```

4 Timestamps

The name of each file corresponds to its timestamp. These timestamps are given as the number of microseconds since January 1st, 1970, in UTC time. The camera was configured to emit a square-wave pulse where the rising edge of each pulse corresponds to the start of a new camera exposure event. The Applanix receiver was then configured to receive and timestamp these event signals. The Velodyne lidar was synchronized to UTC time using a hardwired connection to the Applanix sensor carrying

```
boreas-YYYY-MM-DD-HH-MM
├── applanix
│   ├── camera_poses.csv
│   ├── gps_post_process.csv
│   ├── lidar_poses.csv
│   └── radar_poses.csv
├── calib
│   ├── camera0_intrinsics.yaml
│   ├── P_camera.txt
│   └── T_sens1_sens2.txt
├── camera
│   └── <timestamp>.png
├── lidar
│   └── <timestamp>.bin
├── radar
│   └── <timestamp>.png
├── route.html
└── video.mp4
```

Figure 4: Folder of each sequence and S3 bucket.

NMEA data and PPS signals. The data-recording computer was synchronized to UTC time in the same fashion. The Navtech radar synchronizes its local clock to the NTP time broadcasted on its ethernet subnet. Since the computer publishing the NTP time is synchronized to UTC time, the radar is thereby also synchronized to UTC time.

For camera images, timestamps are provided at the time that exposure started plus half of the total exposure time. For lidar pointclouds, the timestamp corresponds to the temporal middle of a scan. Each lidar point also has a timestamp associated with it, which corresponds to when each point was measured. These point times are given as a number of seconds relative to the middle of the scan. For radar scans, the timestamp corresponds to the middle of the scan: $\lfloor \frac{M}{2} \rfloor - 1$ where M is the total number of azimuths (400). Each scanned radar azimuth is also timestamped in the same format as the filename.

5 Conventions

In this document, we follow the convention used at UTIAS for describing rotations, translations, and transformation matrices. $\mathbf{r}_a^{ba}$ refers to the translation from frame a to frame b as measured in frame a. $\mathbf{C}_{ba} \in SO(3)$ is a 3x3 rotation matrix such that $\mathbf{r}_b = \mathbf{C}_{ba}\mathbf{r}_a$, $\mathbf{C}^T\mathbf{C} = \mathbf{I}$ and det$\mathbf{C} = 1$. Transformations are then defined as 4x4 homogeneous transformation matrices: $\mathbf{T}_{ba} = \begin{bmatrix} \mathbf{C}_{ba} & \mathbf{r}_b^{ab} \\ \mathbf{0}^T & 1 \end{bmatrix} \in SE(3)$.

Note 1: $\mathbf{T}_{ba}$ is the transformation from frame a to frame b. Note the order of the indices here are the same as for translations and rotations.

Note 2: $\mathbf{T}_{ba}$ transforms a vector a $\mathbf{r}_a^{ca}$ into $\mathbf{r}_b^{cb}$ ($\mathbf{r}_b^{cb} = \mathbf{T}_{ba}\mathbf{r}_a^{ca}$). Transformations like this can be used to transform lidar points into the camera's frame and so on. See (Barfoot, 2017) [1] for more details.

6 File Formats

Lidar pointclouds are stored in a binary format to minimize storage requirements. Each point has six fields: $[x, y, z, i, r, t]$ where (x, y, z) is the position of the point with respect to the lidar, i is the intensity of the reflected infrared signal, r is ID of the

laser that made the measurement, and t is a timestamp measured in seconds relative to the middle of the scan. The following code snippet can be used to convert a binary file into a numpy array (N, 6):

```
import numpy as np
def load_lidar(path):
    return np.fromfile(path, dtype=np.float32).reshape((-1, 6))
```

Raw radar scans are 2D polar images: M azimuths x R range bins. We follow Oxford's convention and embed timestamp and encoder information into the first 11 columns (bytes) of each polar radar scan. The first 8 columns represent a 64-bit integer, the UTC timestamp of each azimuth. The next 2 columns represent a 16-bit unsigned integer, the rotational encoder value. The encoder values can be converted into azimuth angles in radians with: azimuth $=$ encoder $* \frac{\pi}{2800}$. The next column is a *valid* bit which we do not use but preserve for compatibility with the Oxford format. For convenience, we also provide a pre-computed cartesian representation of each radar scan with a width of 640 pixels and a resolution of 0.2384 m/pixel.

All camera images are rectified such that a simple projection matrix can be used to project lidar points onto an image.

Each sensor frame's position information is stored in the associated `applanix/sensor_poses.csv` file, in the following format:

$t, x, y, z, v_x, v_y, v_z, r, p, y, \omega_z, \omega_y, \omega_x$

where t is the UTC timestamp in microseconds that matches the file name, $\mathbf{r}_e^{se} = [x\ y\ z]^T$ is the position of the sensor s with respect to the ENU origin frame as measured in the ENU frame, $\mathbf{v}_e^{se} = [v_x\ v_y\ v_z]^T$ is the velocity of the sensor with respect to the ENU frame, (r, p, y) are the roll, pitch, and yaw angles which can be converted into the rotation matrix between the sensor frame and the origin frame. $\boldsymbol{\omega}_s^{se} = [\omega_z\ \omega_y\ \omega_x]^T$ are the angular velocities of the sensor with respect to the ENU frame as measured in the sensor frame. The pose of the sensor frame is then: $\mathbf{T}_{es} = \begin{bmatrix} \mathbf{C}_{es} & \mathbf{r}_e^{se} \\ \mathbf{0}^T & 1 \end{bmatrix} \in SE(3)$ where $\mathbf{C}_{es} = \mathbf{C}_1(\text{roll})\mathbf{C}_2(\text{pitch})\mathbf{C}_3(\text{yaw})$ as described in Section 10.3.

7 Calibration

Camera intrinsics are calibrated using MATLAB's camera calibrator [2] and are recorded in `camera0_intrinsics.yaml`. Images located under `camera/` have already been rectified and as such, the intrinsic parameters can be ignored for most applications. The rectified camera matrix $\mathbf{P}$, stored in `P_camera.txt`, can then be used to project points onto the image plane. To do this, we first obtain the pose of the lidar frame, $\mathbf{T}_{el}$, and the pose of the camera frame, $\mathbf{T}_{ec}$, for their respective timestamps as described in Section 6. Each point in the lidar frame can then be transformed into the camera frame using the following transformation: $\mathbf{x}_c = \mathbf{T}_{ec}^{-1}\mathbf{T}_{el}\mathbf{x}_l$, where $\mathbf{x}_l = [x\ y\ z\ 1]^T$ is the homogeneous vector describing the coordinates of a point in the lidar frame. The projected image coordinates are then obtained using the following equation:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \mathbf{D}\,\mathbf{P}\,\frac{1}{z}\,\mathbf{x}_c \tag{1}$$

$$\text{where } \mathbf{D} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix},\ \mathbf{P} = \begin{bmatrix} f_u & 0 & c_u & 0 \\ 0 & f_v & c_v & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

The extrinsic calibration between the camera and lidar is obtained using MATLAB's camera to LIDAR calibrator [3]. The results of this calibration are demonstrated in Figure 5. To calibrate the extrinsics between the lidar and radar, we use correlative

scan matching via the Fourier Mellin transform [4]. Several lidar-radar pairs are collected while the vehicle is stationary in different positions. The final transform estimate is obtained by averaging over several measurements. The extrinsics between the lidar and IMU (Applanix reference frame) were obtained by using Applanix's in-house lidar-to-IMU calibration tool. Their tool outputs this relative transform as a by-product of a batch optimization aiming to estimate the most likely vehicle path given a sequence of lidar pointclouds and post-processed GPS/IMU measurements. Extrinsic calibrations are provided as 4x4 homogeneous transformation matrices.

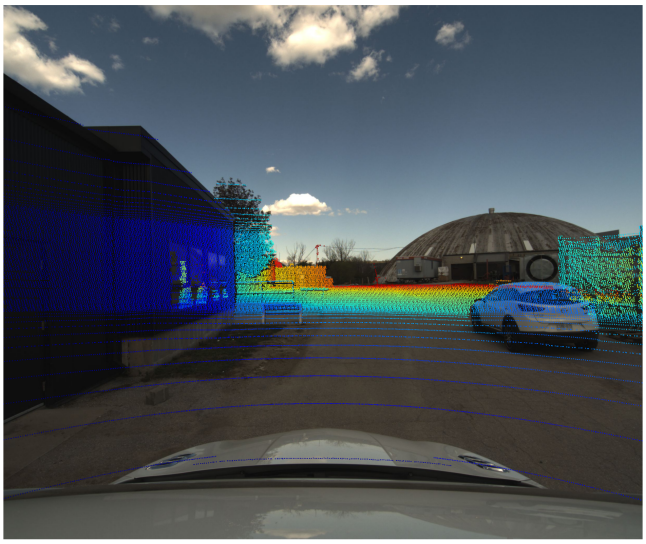


Figure 5: Lidar points projected onto a camera image using the camera-lidar calibration. Lidar points are colored based on their distance from the vehicle.

8 Applanix Measurement Frames

Raw GPS position measurements are provided as Latitude, Longitude, and Altitude (**LLA**). These measurements are provided in the **WGS84** (World Geodetic System 1984 Revision) standard. "It comprises a standard coordinate frame for the Earth, a standard spheroidal reference surface (the *datum* or *reference ellipsoid*) for raw altitude data, and a gravitational equipotential surface (the geoid) that defines the nominal sea level." [5]

The Applanix has two virtual measurement frames: the **Applanix reference frame** and the **vehicle frame**, and one physical sensor frame: the **IMU frame**. The orientation of the vehicle frame is defined as x-forwards, y-right, and z-down. The Applanix reference frame will be used to provide position and orientation data with respect to the origin. Nominally, the Applanix reference frame and vehicle frame have the same position and orientation (x-forwards, z-down). However, if desired, the Applanix reference frame can be assigned a **mounting angle** which differs from the vehicle frame orientation. We do this for Boreas in order to obtain an Applanix reference frame with x-right, y-forwards, and z-up as shown in Figure 6.

Note 1: The position of the Applanix reference frame and vehicle frame are defined with respect to the IMU frame. The IMU frame is the physical mounting position and orientation of the IMU sensor on the robot. For simplicity, we choose to have our virtual Applanix reference frame in the same position as the IMU frame.

Note 2: Post-processed position, orientation, velocity, linear acceleration, and angular velocity are provided about the Applanix reference frame. For raw data (`gps_raw.csv`, `imu.csv`), the linear acceleration is provided about the IMU frame which has x-forwards, y-right, and z-down.

9 Applanix Position Data

Raw GPS position data is referred to as such because it is the output of the Applanix system as it was received in real-time. No post-processing is applied. As such, the accuracy can vary from 5-20 cm of error with real-time corrections or up to 3.0 m without real-time corrections. Locally, the positioning accuracy may be good enough for testing perception algorithms like object tracking, but this accuracy is insufficient for benchmarking odometry algorithms or constructing a map. If you play back a rosbag that was collected while driving, the `\navsat\odom` topic contains the raw position information.

We convert from latitude-longitude-altitude into a metric coordinate frame. **UTM** (Universal Transverse Mercator) coordinates are used for this purpose. UTM divides the earth into 60 zones and projects each to a plane as the basis for its coordinates [6]. Converting from latitude-longitude to UTM outputs a metric value for Easting and Northing.

For our origin frame, we use x-east, y-north, z-up which is abbreviated to **ENU**. GPS position, orientation, and velocity information is reported for the Applanix reference frame with respect to ENU as shown in Figure 6.

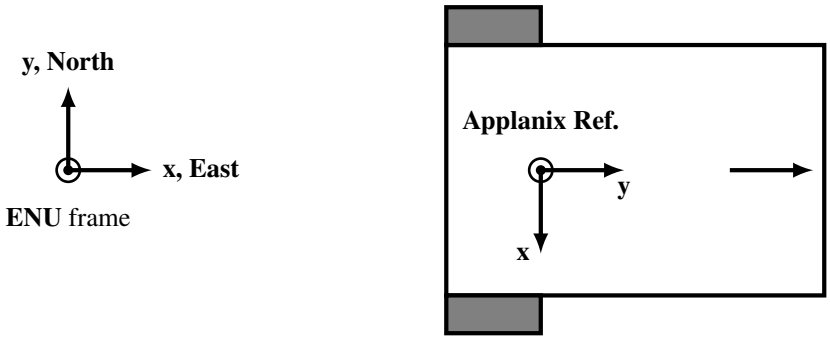


Figure 6: Applanix Reference Frame and ENU (East-North-Up).

Note 1: Not all GPS data sources use the same datum for reporting altitude. If you stick to producing and consuming data from the exact same sensor, then you shouldn't have any issues. However, if you want to localize within a map created by a different company's sensor, you may run into problems. You may end up with a static z offset between the reference frames. The easiest way to check for is to move your sensor to a known location in the map and then check the difference in z values.

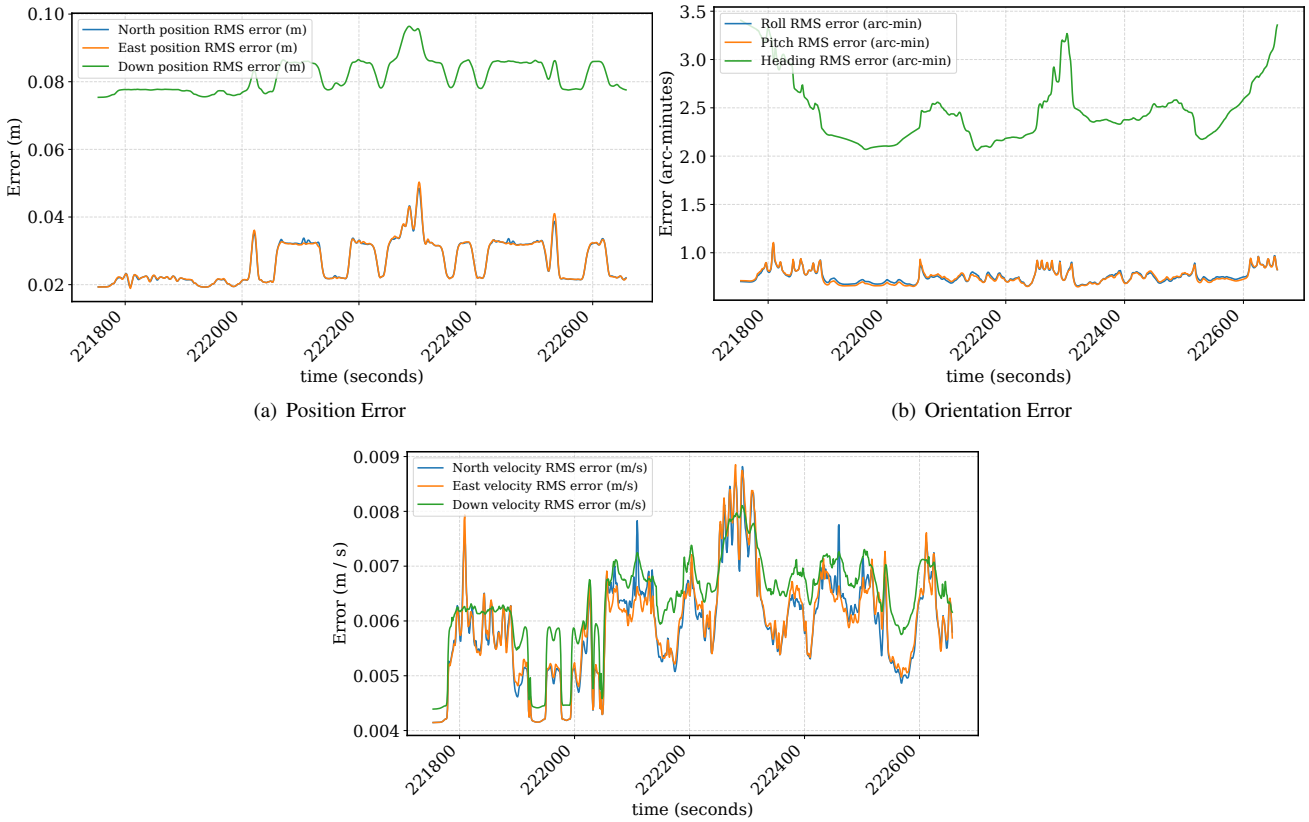
Note 2: When using ENU, the heading is zero when the vehicle is pointing North, and a positive heading is defined when the vehicle rotates counter-clockwise.

We Applanix's proprietary POSPac suite to obtain post-processed results. The POSPac suite uses all available measurements (GPS, IMU, wheel encoder) and performs a batch optimization using an RTS smoother to obtain the most accurate position, orientation, and velocity information at each time step. The RMS position error is typically between 5 and 20 cm. However, this accuracy can change depending on the atmospheric conditions and the visibility of satellites. This accuracy can also change throughout the course of a sequence. For detailed information on the position accuracy of each sequence, we have provided a script, `plot_processed_error.py`, which produces plots of position, orientation, and velocity error vs. time. Examples of these plots are provided in Figure 7. It should be noted that these numbers represent global positioning accuracy, whereas the relative positioning accuracy between temporally proximal measurements will likely be better.

By default, post-processed position, orientation, and velocity information is reported as the Applanix reference frame with respect to NED. We convert this to be with respect to ENU. The post-processed output and errors are stored in `sbet.out` and `smrmsg.out` respectively. Note that the heading, and velocity reported is with respect to a local frame that contains a "wander angle". To fix this, we first need to extract the true heading:

$$\Psi_{true} := \Psi_{wander} - \alpha \tag{3}$$

Ψ_{wander} is the heading that's reported in the post-processed output. α is also reported as the "wander angle". Note that we



(a) Position Error

(b) Orientation Error

(c) Velocity Error

Figure 7: RMS position, orientation, and velocity error for the post-processed data from the `boreas-2021-09-07-09-35` sequence.

also need to transform the velocities from this wander frame into what we actually want:

$$V_N = V_x \cos\alpha - V_y \sin\alpha \tag{4}$$
$$V_E = -V_x \sin\alpha - V_y \cos\alpha \tag{5}$$

Note that our desired linear velocities in ENU are then $V_x := V_E$, $V_y := V_N$, and V_z remains unchanged. In order to convert the orientation which is reported with respect to NED into an orientation reported with respect to ENU, we use the following conversions which are explained more thoroughly in sections 10.3, 10.4.

$$\mathbf{C}_{ned,applanix} = \mathbf{C}_1(\text{roll}_{ned})\mathbf{C}_2(\text{pitch}_{ned})\mathbf{C}_3(\text{heading}_{ned}) \tag{6}$$
$$\mathbf{C}_{enu,ned} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{7}$$
$$\mathbf{C}_{enu,applanix} = \mathbf{C}_{enu,ned}\mathbf{C}_{ned,applanix} \tag{8}$$

We can then use our conversion in section 10.4 to convert $\mathbf{C}_{enu,applanix}$ into heading$_{enu}$, pitch$_{enu}$, roll$_{enu}$. Finally, to convert the UTM Northing and Easting into a position in ENU, we use the following assignments:

$$x_{enu} := \text{Easting} \tag{9}$$
$$y_{enu} := \text{Northing} \tag{10}$$
$$z_{enu} := \text{Altitude} \tag{11}$$

10 Useful Conversions

10.1 Quaternion to 3x3 Rotation

GPS Raw poses are reported as a ROS quaternion $\mathbf{q} = [q_x\ q_y\ q_z\ q_w]^T$ and a position $\mathbf{r}_e^{re} = [x\ y\ z]^T$. $\mathbf{r}_e^{re}$ describes the position of the Applanix reference frame with respect to the ENU frame, as reported in the ENU frame. It is useful to convert this into a 4x4 homogeneous transformation matrix. We will denote this 4x4 transformation matrix from the Applanix reference frame r to the ENU frame e as [1]:

$$\mathbf{T}_{er} = \begin{bmatrix} \mathbf{C}_{er} & \mathbf{r}_e^{re} \\ \mathbf{0}^T & 1 \end{bmatrix}. \tag{12}$$

In order to obtain the 3x3 rotation matrix $\mathbf{C}_{er}$, we use the following formula for working with the ROS quaternion [1]:

$$\mathbf{C}_{er} = (\eta^2 - \boldsymbol{\xi}^T\boldsymbol{\xi})\mathbf{1} + 2\boldsymbol{\xi}\boldsymbol{\xi}^T - 2\eta\boldsymbol{\xi}^\wedge, \tag{13}$$

where $\boldsymbol{\xi} = [q_x\ q_y\ q_z]^T$, $\eta = q_w$, and $(\cdot)^\wedge$ is the skew-symmetric operator [1]:

$$\mathbf{u}^\wedge = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}^\wedge := \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix}. \tag{14}$$

10.2 3x3 Rotation to Quaternion

Using the axis-angle representation for a rotation, we can represent a rotation by a unit-length axis $\mathbf{a}$ and a single rotation angle ϕ, 4 parameters and 1 constraint ($\mathbf{a}^T\mathbf{a} = 1$). Note that $\mathbf{C}\mathbf{a} = \mathbf{a}$ which implies that $\mathbf{a}$ is a (unit-length) eigenvector of $\mathbf{C}$. The angle can be found by exploiting the trace of a rotation matrix [1]:

$$\phi = \cos^{-1}\Big(\frac{tr(\mathbf{C}) - 1}{2}\Big) + 2\pi m \tag{15}$$

A unit-length quaternion can then be generated using the following formulas:

$$\boldsymbol{\xi} = \mathbf{a}\sin\frac{\phi}{2} \tag{16}$$

$$\eta = \cos\frac{\phi}{2} \tag{17}$$

$$\mathbf{q} = \begin{bmatrix}\boldsymbol{\xi}\\ \eta\end{bmatrix} \tag{18}$$

Note that the space of quaternions is a double cover of SO(3), so each 3x3 rotation matrix maps to two quaternions: $\pm\mathbf{q}$.

10.3 Yaw, Pitch, Roll to 3x3 Rotation

The post-processed GPS provides the heading (yaw), pitch, and roll of the Applanix reference frame with respect to NED which we convert into yaw, pitch, roll values with respect to ENU. To convert this into a 3x3 rotation matrix, we use the following formula based on the Tait-Bryan sequence of rotations [7]:

$$\mathbf{C}_{er} = \mathbf{C}_1(\text{roll})\mathbf{C}_2(\text{pitch})\mathbf{C}_3(\text{yaw}), \tag{19}$$

where C_1, C_2, C_3 are the principal rotation matrices about x, y, and z respectively:

$$\mathbf{C}_1(\theta_1) = \begin{bmatrix}1 & 0 & 0\\ 0 & \cos\theta_1 & \sin\theta_1\\ 0 & -\sin\theta_1 & \cos\theta_1\end{bmatrix} \tag{20}$$

$$\mathbf{C}_2(\theta_2) = \begin{bmatrix}\cos\theta_2 & 0 & -\sin\theta_2\\ 0 & 1 & 0\\ \sin\theta_2 & 0 & \cos\theta_2\end{bmatrix} \tag{21}$$

$$\mathbf{C}_3(\theta_3) = \begin{bmatrix}\cos\theta_3 & \sin\theta_3 & 0\\ -\sin\theta_3 & \cos\theta_3 & 0\\ 0 & 0 & 1\end{bmatrix} \tag{22}$$

$$\tag{23}$$

10.4 3x3 Rotation to Yaw, Pitch, Roll

$$c_y = \sqrt{\mathbf{C}(2,2)^2 + \mathbf{C}(1,2)^2} \tag{24}$$

if $c_y > \epsilon$ (where ϵ is chosen to be small such as 1×10^{-15}) then

$$\text{yaw} = \text{atan2}(\mathbf{C}(0,1), \mathbf{C}(0,0)) \tag{25}$$

$$\text{pitch} = \text{atan2}(-\mathbf{C}(0,2), c_y) \tag{26}$$

$$\text{roll} = \text{atan2}(\mathbf{C}(1,2), \mathbf{C}(2,2)) \tag{27}$$

else if $c_y \leq \epsilon$,

$$\text{yaw} = \text{atan2}(-\mathbf{C}(1,0), \mathbf{C}(1,1)) \tag{28}$$
$$\text{pitch} = \text{atan2}(-\mathbf{C}(0,2), c_y) \tag{29}$$
$$\text{roll} = 0 \tag{30}$$

References

[1] T. D. Barfoot, *State estimation for robotics*. Cambridge University Press, 2017.

[2] "Single camera calibrator app," https://www.mathworks.com/help/vision/ug/single-camera-calibrator-app.html, accessed: 2020-12-13.

[3] "Lidar and camera calibration," https://www.mathworks.com/help/lidar/ug/lidarcameracalibrationexample.html, accessed: 2020-12-13.

[4] "radar to lidar calib," https://github.com/keenan-burnett/radar_to_lidar_calib, accessed: 2020-12-13.

[5] "WGS84," http://wiki.gis.com/wiki/index.php/WGS84, accessed: 2020-11-23.

[6] "Universal transverse mercator coordinate system," https://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system, accessed: 2020-11-24.

[7] "Euler angles: Tait-Bryan angles," https://en.wikipedia.org/wiki/Euler_angles#Tait%E2%80%93Bryan_angles, accessed: 2020-11-24.