The DJH INS ROS Package Documentation

David Hanley, Alex Faustino, David Degenhardt, Tim Bretl

June 30, 2017

Abstract

The purpose of this article is to document the approach to the DJH inertial navigation system (INS) ROS package. This package, we anticipate, will be used in a variety of other navigation systems. For example, we design this so that it can be easily used in a visual-inertial odometry system or a magnetic positioning system.

1 Introduction

The DJH INS ROS package is designed to provide several potentially useful computations to a user as IMU data is received. These include:

- IMU data aggregated into sets of Eigen matrices
- blah blah blah

2 The IMU Aggregator

One of the functions of the DJH INS is that an INS solution is only computed when requested by the `comp_sol` topic. This topic is a message created for this package that includes:

- `Header header`
- `float64 time_desired`
- `bool stop_agg`

The `time_desired` variable is the time for which an INS solution is desired. The `stop_agg` variable is switched to `true` when it is desired to stop aggregating the data (presumably to then compute an INS solution at `time_desired`). As the system is running if IMU data is collected with a timestamp at or after `time_desired`, then that data is saved for use in a matrix with a later `time_desired`. The aggregated matrix is published on a topic called `aggregate_imu`. This aggregated IMU data is published as Float64 vector standard message in ROS. The following C++ code shows how to convert that message into a regular n-by-7 Eigen matrix.

```
/*------- Receive and reform aggregated IMU Matrix -------*/
// Define a temporary std vector for the aggregated IMU
// message data
vector<double> vec = msg->data;
// Compute the number of rows in the aggregated matrix
int sz = vec.size()/7;
// Create pointer and store memory address of first vector
   element
double* ptr = &vec[0];
```

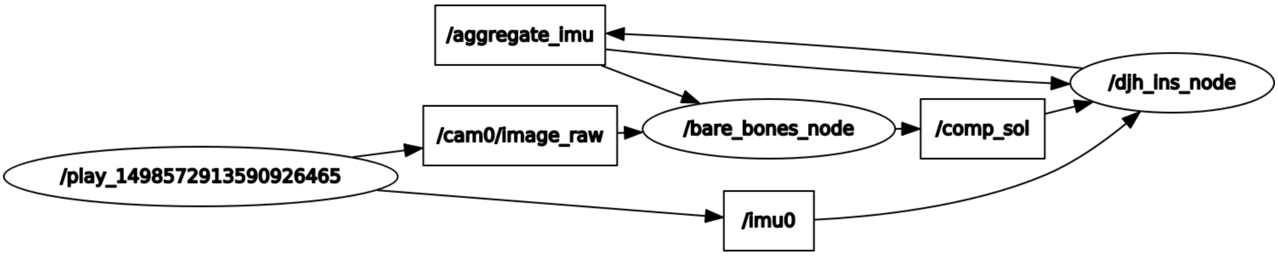

Figure 1: The DJH INS package receives IMU data and a flag to stop aggregating that IMU data in a matrix. That aggregated data is then published and can be used by both the djh_ins_node for computing an INS solution or by some other node (in this case bare_bones_node) for some other reason.

```
// Note: MatrixX7d is defined in aggregator.h
Map<MatrixX7d>agg_mat(ptr,sz,7);
/*----- End receive and reform aggregated IMU Matrix -----*/

// Print Results
cout << "------------------------------------------------------------\n";
cout << agg_mat << endl;
cout << "------------------------------------------------------------\n";
```

The structure of the resulting aggregated matrix is as follows:

$$\begin{bmatrix} timestamp_1 & accel_{x1} & accel_{y1} & accel_{z1} & gyro_{x1} & gyro_{y1} & gyro_{z1} \\ timestamp_2 & accel_{x2} & accel_{y2} & accel_{z2} & gyro_{x2} & gyro_{y2} & gyro_{z2} \\ ... & ... & ... & ... & ... & ... & ... \end{bmatrix} \tag{1}$$

Since the DJH INS package is a ROS node, it can interface with some navigation algorithm through ROS topics. Using bare_bones_node as an example navigation node (such as a visual-inertial odometry code), the aggregated IMU data can interface with it as shown in Figure 1.

3 The IMU Model and Corrector

The elements of the aggregated matrix are corrected for fixed scale factors (S_g and S_a), cross-coupling effects (M_g and M_a), and biases (B_{fg} and B_{fa}). These parameters (since they are fixed) are set in a parameter file called `IMUmodel.yaml`. We also assume that the navigation algorithm can potentially estimate some bias online (B_g for the gyroscopes and B_a for the accelerometers). Therefore, we have set up a subscriber to a ROS topic called `bias_est` which contains a `std_msgs::Float64MultiArray` message. The first three elements of the message are assumed to correspond to the x, y, and z-axis accelerometer biases respectively. The second three elements of the message are assumed to correspond to the x, y, and z-axis gyroscope biases respectively. Equations 2 and 3 show how we use measurements from the gyroscopes, ω, and accelerometers, a_{sf}, to compute corrected gyroscope, $\tilde{\omega}$, and accelerometer, $\tilde{a}_{sf}$, data. Figure 2 shows the same information as Figure 3. However, now ROS topics relevant for IMU error correction is also included.

$$\tilde{\omega} = (1 + S_g)\omega + M_g\omega + B_{fg} + B_g \tag{2}$$

$$\tilde{a}_{sf} = (1 + S_a)a_{sf} + M_a a_{sf} + B_{fa} + B_a \tag{3}$$

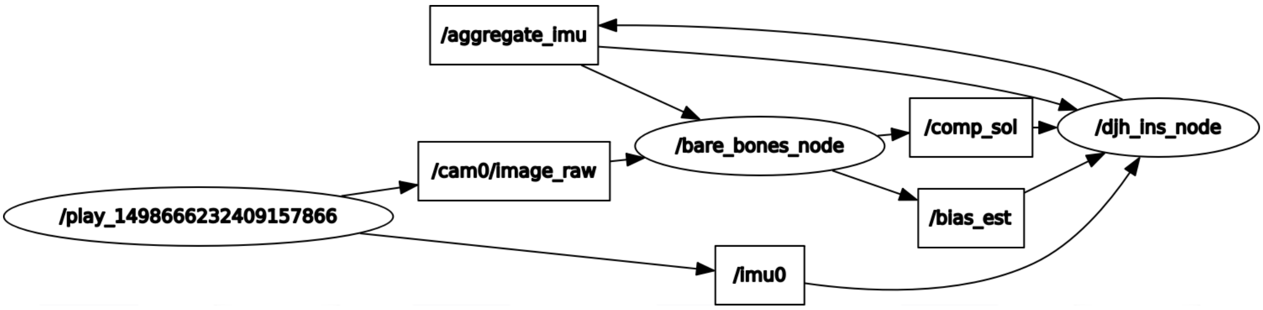

Figure 2: This ROS graph shows all the ROS topics associated with aggregating IMU data and with correcting IMU measurements.

4 Quaternion Math Library

Our INS code makes significant use of the C++ library Eigen (`http://eigen.tuxfamily.org`). However, we do not make use of Eigen's quaternion functionalities. This is because we use a JPL convention for quaternions. More information on the JPL convention can be found at [1–3]. There we create our own quaternion math class call `QuatMath` which performs the following functions on JPL-style quaterions:

1 Normalize the Quaternion

2 Compute the Angle of Rotation

3 Perform Quaternion Multiplication

4 Compute the Quaternion Inverse

5 Create a Rotation Matrix from a Quaternion

6 Convert Rotation Matrix to a Quaterion.

Note that to convert from a JPL quaternion to a Hamilton quaternion one must simply apply the following transformation

$$q^H = \begin{bmatrix} q_4^{JPL} & -q_1^{JPL} & -q_2^{JPL} & -q_3^{JPL} \end{bmatrix}^T. \tag{4}$$

5 Integration Algorithms and Implementation

6 Conclusion

blah blah blah [4–6]

References

[1] N. Trawny and S. I. Roumeliotis, "Indirect Kalman Filter for 3D Attitude Estimation: A Tutorial for Quaternion Algebra," University of Minnesota Department of Computer Science and Engineering, Tech. Rep. 2005-002 Rev. 57, March 2005.

[2] T. Barfoot, J. R. Forbes, and P. T. Furgale, "Pose Estimation Using Linearized Rotations and Quaternion Algebra," *Acta Astronautica*, vol. 68, pp. 101–112, 2011.

[3] B. Wie, *Space Vehicle Dynamics and Control*, 2nd ed., ser. Education Series. Reston, VA: American Institute of Aeronautics and Astonautics, Inc., 2008.

[4] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "On-Manifold Preintegration for Real-Time Visual-Inertial Odometry," *IEEE Transactions on Robotics*, vol. 33, no. 1, pp. 1–19, February 2017.

[5] ——, "IMU Preintegration on Manifold for Efficient Visual-Inertial Maximum-a-Posteriori Estimation," in *Proceedings of the Robotics: Science and Systems (RSS)*, Sapienza University of Rome, July 2015.

[6] K. Eckenhoff, P. Geneva, and G. Huang, "High-Accuracy Preintegration for Visual-Inertial Navigation," in *Proceedings of the Workshop on the Algorithmic Foundations of Robotics (WAFR)*, San Francisco, CA, December 2016.