The DJH INS ROS Package Documentation

David Hanley, Alex Faustino, David Degenhardt, Tim Bretl

July 6, 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
- A state (orientation, position, and velocity) given a set of IMU measurements (aggregated IMU data) and an initial state estimate.

Aggregated IMU data can be saved or used by a navigation algorithm or for any purpose. The state estimate is the INS output, which can be used as a prior in navigation systems.

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 contains a message (`compute_ins.msg`) created for this package that includes:

- `Header header`
- `float64 time_desired`
- `bool stop_agg`
- `float64 start_time`
- `geometry_msgs/Pose start_pose`
- `geometry_msgs/Vector3 start_vel`
- `std_msgs/Float64MultiArray aggregatedMatrix`

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 `start_pose` variable lists the position and orientation of the IMU at the time `start_time`. The variable `start_vel` lists the velocity of the IMU at

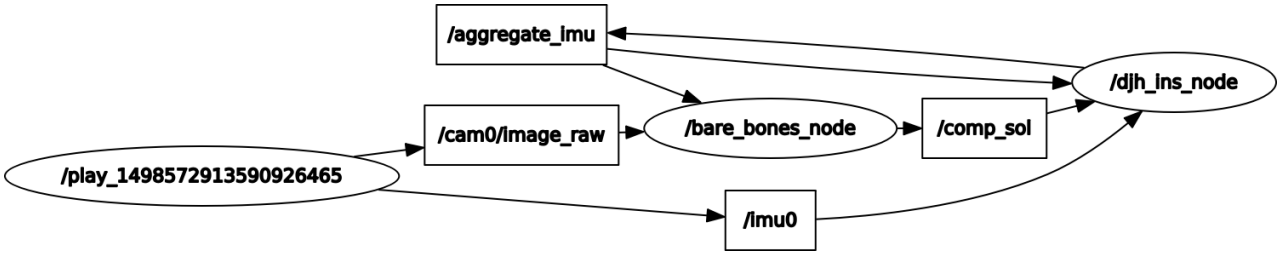

Figure 1: The DJH INS package receives IMU data and a flag to stop aggregating that IMU data in a matrix. That aggregated data is then published and can be used by both the `djh_ins_node` for computing an INS solution or by some other node (in this case `bare_bones_node`) for some other reason.

the time `start_time`. The aggregated matrix is collected in `aggregatedMatrix` and published on a topic called `aggregate_imu`. This topic also consists of a `compute_ins.msg` message. The following C++ code shows how to convert the `aggregatedMatrix` Float64MultiArray 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];
// 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:

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

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

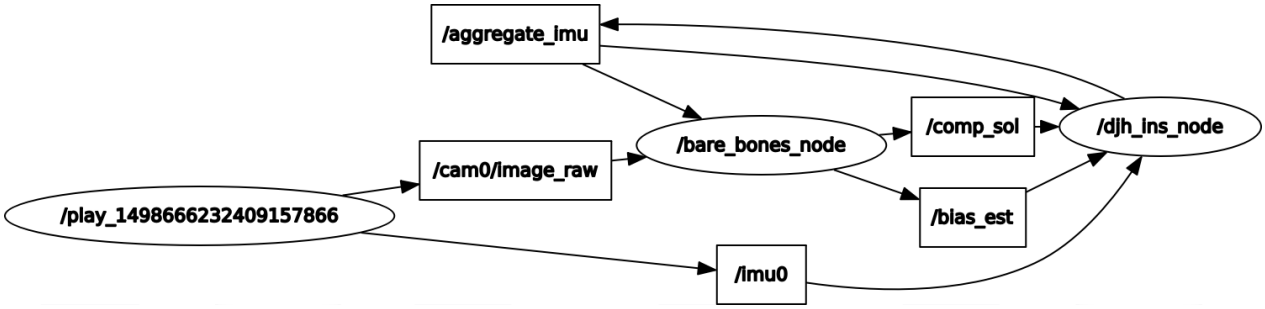

Figure 2: This ROS graph shows all the ROS topics associated with aggregating IMU data and with correcting IMU measurements.

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 1 and 2 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 2. However, now ROS topics relevant for IMU error correction is also included.

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

$$\tilde{a}_{sf} = (1 + S_a)a_{sf} + M_a a_{sf} + B_{fa} + B_a \tag{2}$$

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]. Therefore, 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

$$q^H = \begin{bmatrix} q_4^{JPL} & -q_1^{JPL} & -q_2^{JPL} & -q_3^{JPL} \end{bmatrix}^T. \tag{3}$$

5 Integration Algorithms and Implementation

To select our INS integration scheme, we conducted a quick survey of methods used in visual-inertial odometry algorithms:

- Fourth-order Runge-Kutta integration [4–7]
- Fourth-order Runge-Kutta integration for attitude and Simpson integration for position and velocity [8]
- Euler integration [9]
- Preintegration [10]

Based on this, we decided to use a standard fourth-order Runge-Kutta method to compute our INS results. Note that for batch optimization methods, preintegration methods are preferable because aggregated IMU measurements do not need to be saved. More over, methods from Savage [11, 12] may be computationally more efficient and may better handle the effects of coning, sculling, and scrolling. However, we leave such methods for future work.

5.1 Gravity Model

The gravity model we use is described in [13]. Gravity magnitude (in m/s^2) determined by latitude is

$$g(0) = 9.780318(1 + 5.3024 \times 10^{-3} \sin^2 L - 5.9 \times 10^{-6} \sin^2 2L) \tag{4}$$

and the magnitude due to height is determined by

$$g(h) = \frac{g(0)}{(1 + h/R_0)^2} \tag{5}$$

where h is the altitude in meters, L is the latitude in radians, and R_0 is the Earth's mean radius in meters. This is applied to a vector a vector as $\mathbf{g} = \begin{bmatrix} 0 & 0 & g(h) \end{bmatrix}^T$ in the NED frame. Finally, we apply the Coriolis effect due to Earth's rotation to compute the entire plumb-bob gravity vector of the IMU frame (l) in the NED frame (n)

$$\mathbf{g}_l^n = \mathbf{g} - \frac{\Omega^2 (R_0 + h)}{2} \begin{pmatrix} \sin 2L \\ 0 \\ 1 + \cos 2L \end{pmatrix} \tag{6}$$

where Ω is the rotation rate of the Earth in radians per second. Note that we created a parameter file called `Earthparam.yaml` which contains the following parameters:

- The latitude of the location
- The altitude of the location
- The Earth's mean radius
- The rotation rate of the Earth.

Thus if the Coriolis effect must be ignored, then the rotation rate of the Earth can be set to zero. It is here also assumed that in the applications in which DJH INS will be used, the latitude or altitude of the system will not change dramatically enough during operation for the gravity model to be significantly impacted.

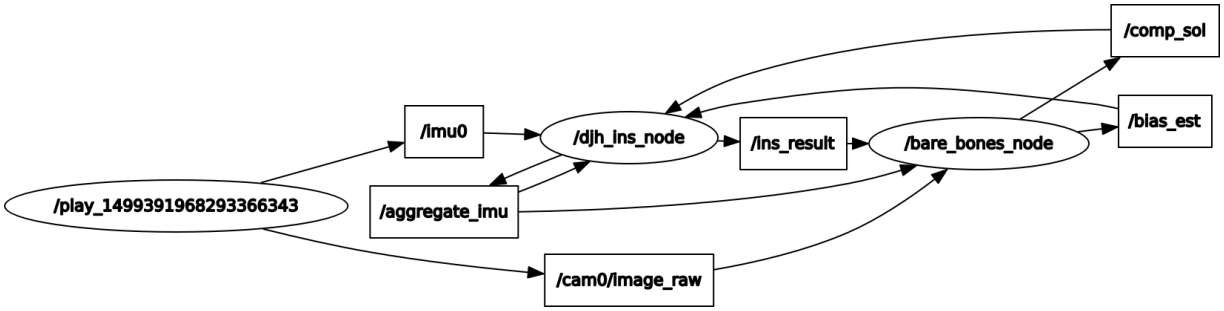

Figure 3: This ROS graph shows all the ROS topics associated with the DJH INS package.

5.2 Equations of Motion

The equations of motion we use to describe the motion of the IMU are

$$\dot{q} = \frac{1}{2}\Omega(\tilde{\omega})q \tag{7}$$

$$\dot{p} = v \tag{8}$$

$$\dot{v} = C(q)\tilde{a}_{sf} + \mathbf{g}_l^n \tag{9}$$

where q is the orientation, p is the position, and v is the velocity of the IMU in the NED frame. The matrix $\Omega(\tilde{\omega})$ is the appropriate 4×4 matrix created from corrected gyroscope measurements $\tilde{\omega}$. The 3×3 matrix $C(q)$ is the rotation matrix generated from the quaternion q.

5.3 Runge Kutta Integration and Run Time

Expressing our equations of motion as $\dot{x} = f(x)$, we compute a classical Runge-Kutta solution as

$$k_1 = f(x_i) \tag{10}$$

$$k_2 = f(x_i + \frac{\Delta t}{2}k_1) \tag{11}$$

$$k_3 = f(x_i + \frac{\Delta t}{2}k_2) \tag{12}$$

$$k_4 = f(x_i + \Delta t k_3) \tag{13}$$

$$x_{i+1} = x_i + \frac{\Delta t}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \tag{14}$$

where x_i is the state at step i and Δt is the time between step i and $i+1$. We update the state estimate by solving the Runge-Kutta equations between each IMU measurement between our start and finish time stamps.

The INS result is publish on a ROS topic called `ins_result` as a `compute_ins.msg` message. In this topic, the `start_time` and `time_desired` are both set to the `time_desired` used to compute the INS solution. The final full INS ROS package graph is shown in Figure 3.

Note that we tested our algorithm on the EuRoC dataset [14]. This has a camera that provides images at 20 Hz and an IMU that provides measurements at 200 Hz. We request an INS solution every time we receive an image. It requires on our test computer (which uses an i7 processor with 16 GB of RAM) approximately 0.002 seconds to compute an INS solution. Note that this is faster than the frame rate of the camera and that IMU data collection does not stop while we compute and INS solution (since these are different processes). Note also that if we received a faster camera, we would have fewer IMU measurements, so the runtime of the INS solution will be much faster (since the Runge-Kutta calculation requires the bulk of the time).

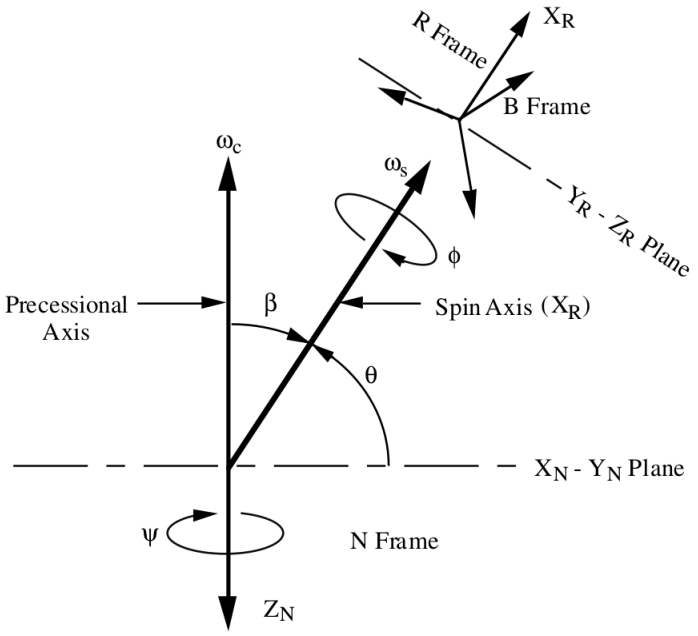

Figure 4: The spin-cone test model.

6 Test Cases

6.1 The Spin-Cone Truth Model

As a first test case, we use the Spin-Cone Model described by [18]. This model tests the attitude integration algorithms against a known closed-form solution. This models spins about one axis and precesses about another as shown in Figure 4. The ground truth is determined by Euler angle equations

$$\phi = (\omega_s - \omega_c \cos\beta)\, t + \phi_0 \tag{15}$$

$$\theta = \frac{\pi}{2} - \beta \tag{16}$$

$$\psi = -\omega_c t. \tag{17}$$

We assume that $a_{sf} = 0$. The gyroscope measurements ω_{iB}^B are determined by the equations

$$\omega_{iB}^R(t) = \begin{bmatrix} \omega_s t \\ \frac{\omega_c \sin\beta}{\omega_s - \omega_c \cos\beta}(\cos\phi - \cos\phi_0) \\ \frac{-\omega_c \sin\beta}{\omega_s - \omega_c \cos\beta}(\sin\phi - \sin\phi_0) \end{bmatrix} \tag{18}$$

$$\omega_{iB}^B(t) = C_R^B \omega_{iB}^R(t). \tag{19}$$

Here parameters ω_s, ω_c, β, ϕ_0, and the rotation matrix C_R^B are chosen by the user. The closed form solution is described by the equations

$$C_R^N = \begin{bmatrix} c_\theta c_\psi & -c_\phi s_\psi + s_\phi s_\theta c_\psi & s_\phi s_\psi + c_\phi s_\theta c_\psi \\ c_\theta s_\psi & c_\phi c_\psi + s_\phi s_\theta s_\psi & -s_\phi c_\psi + c_\phi s_\theta s_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix} \tag{20}$$

$$C_B^N = C_R^N C_B^R. \tag{21}$$

7 Future Work

In the future, we would like to make significant changes to this package. First, we would like to add preintegration as part of our packag [15–17]. Second, we would like to include as many performance analysis test cases (as described in [18]) as possible.

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] F. M. Mirzaei and S. I. Roumeliotis, "A kalman filter-based algorithm for imu-camera calibration: Observability analysis and performance evaluation," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1143–1156, Oct 2008.

[5] J. A. Hesch, D. G. Kottas, S. L. Bowman, and S. I. Roumeliotis, "Consistency analysis and improvement of vision-aided inertial navigation," *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 158–176, Feb 2014.

[6] N. Keivan and G. Sibley, "Asynchronous adaptive conditioning for visualinertial slam," *The International Journal of Robotics Research*, vol. 34, no. 13, pp. 1573–1589, 2015. [Online]. Available: http://dx.doi.org/10.1177/0278364915602544

[7] J. Kelly and G. S. Sukhatme, "Visual-inertial sensor fusion: Localization, mapping and sensor-to-sensor self-calibration," *The International Journal of Robotics Research*, vol. 30, no. 1, pp. 56–79, 2011. [Online]. Available: http://dx.doi.org/10.1177/0278364910382802

[8] M. Li and A. I. Mourikis, "High-precision, consistent ekf-based visual-inertial odometry," *The International Journal of Robotics Research*, vol. 32, no. 6, pp. 690–711, 2013. [Online]. Available: http://dx.doi.org/10.1177/0278364913481251

[9] E. S. Jones and S. Soatto, "Visual-inertial navigation, mapping and localization: A scalable real-time causal approach," *The International Journal of Robotics Research*, vol. 30, no. 4, pp. 407–430, 2011. [Online]. Available: http://dx.doi.org/10.1177/0278364910388963

[10] R. Mur-Artal and J. D. Tards, "Visual-inertial monocular slam with map reuse," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 796–803, April 2017.

[11] P. G. Savage, *Strapdown Analytics*, 2nd ed. Maple Plain, MN: Strapdown Associates, Inc., 2007, vol. 1.

[12] ——, *Strapdown Analytics*, 2nd ed. Maple Plain, MN: Strapdown Associates, Inc., 2007, vol. 2.

[13] D. Titterton and J. Weston, *Strapdown Inertial Navigation Technology*, 2nd ed., ser. IET Radar, Sonar, Navigation and Avionics. Reston, VA: The Institution of Engineering and Technology, 2004, vol. 17.

[14] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016.

[15] 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.

[16] ——, "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.

[17] 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.

[18] P. G. Savage, "Performance analysis of strapdown systems," NATO Science and Technology Organization, Tech. Rep. RTO-EN-SET-116-2009-10, March 2009.