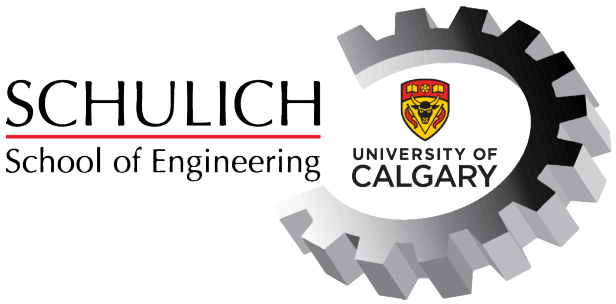
University of Calgary

Final Project
ENGO 623: Inertial Navigation

Adam Smith (30031453)

April 3, 2023



1 Introduction

In this project, we implement a Python module for mechanization of IMU data in the local level frame. The module performs alignment for a user-specified time and then begins computing navigation information. Errors in attitude, position, and velocity are tracked throughout the mechanization and reported at the end.

2 Mechanization Module Implementation

2.1 The INSMechanization class

Mechanization is implemented in the `INSMechanization` class in the file `mechanization.py`. This module is written using the scientific computing package *numpy* to allow for easy array operations. To simplify computation, all calculations are performed in SI base or derived units. As such, the module requires all input information to be in SI base or derived units.

To initialize the mechanization module, create an instance of the `INSMechanization` class, passing in the following arguments:

- `h0` — Float
  Initial height in meters above the ellipsoid. Required.
- `lat0` — Float
  Initial latitude in radians. Required.
- `long0` — Float
  Initial longitude in radians. Required.
- `accel_bias` — Float or Callable
  Accelerometer bias in m/s^2. If callable, it should compute the accelerometer bias based on the local gravity. Default: 0
- `gyro_bias` — Float
  Gyroscope bias in rad/s. Default: 0
- `accel_sf` — Float or Numpy Array
  Accelerometer scale factor. Default: 0
- `gyro_sf` — Float or Numpy Array
  Gyroscope scale factor. Default: 0

- `accel_no` — Numpy Array
  Accelerometer non-orthogonality matrix in radians. Default: 3×3 zero matrix

- `gyro_no` — Numpy Array
  Gyroscope non-orthogonality matrix in radians. Default: 3×3 zero matrix

- `vrw` — Float
  Velocity random walk of the accelerometer in $\mathrm{m/s}^{\frac{3}{2}}$. Default: 0

- `arw` — Float
  Angle random walk of the gyroscope in $\mathrm{rad/s}^{\frac{1}{2}}$. Default: 0

- `accel_corr_time` — Float
  Accelerometer correlation time in seconds. Default: 0

- `gyro_corr_time` — Float
  Gyroscope correlation time in seconds. Default: 0

- `accel_bias_instability` — Float or Callable
  Accelerometer bias instability in $\mathrm{m/s}^2$. If callable, it should compute the accelerometer bias based on the local gravity. Default: 0

- `gyro_bias_instability` — Float
  Gyroscope bias instability in rad/s. Default: 0

- `alignment_time` — Float
  Time duration in seconds to perform alignment before beginning navigation. Default: 0

The mechanization can then be run using the `process_measurement` method of the `INSMechanization` class. This method accepts a single argument `measurement`, which represents a single measurement from the IMU and is a one-dimensional numpy array of length 7 consisting of the current time step in seconds, the x-, y-, and z-gyroscope measurements in rad/s, and the x-, y-, and z-accelerometer measurements in $\mathrm{m/s}^2$.

At any time, navigation parameters can be obtained through the `get_params` method. This method returns the time, position, velocity, and attitude information as computed by the mechanization module at the most recent time step, as well as whether or not alignment has completed. If the argument `degrees` is set to true, values in radians will be converted to degrees before being returned.

The following sections describe the implementation of the mechanization class in more detail.

2.2 Alignment

When a measurement is received in the `process_measurement` method, we check whether or not alignment has completed by checking the `alignment_complete` flag, which is initially set to False. If alignment is not complete, the `align` method is called, passing in the current measurement. The align method then uses the current timestamp (adjusted, if necessary, such that the first measurement received always has time 0) and the user-specified alignment time to determine whether or not alignment will be complete in this iteration. If not, align then calls the `compensate_errors_and_compute_params` method (described in section 2.3) to compensate for deterministic errors in the accelerometer and gyroscope measurements. The compensated values are then added to running totals, and a counter describing the number of times the `align` method has been called is incremented. No further computation occurs.

Once `align` determines that alignment is complete, it sets the `alignment_complete` flag to True, then computes the mean of the measurements received during the alignment time by dividing the totals by the number of iterations. The roll, pitch, and azimuth of the IMU are then calculated via

$$r = -\text{sign}(f_z)\sin^{-1}\frac{f_x}{g}$$

$$p = \text{sign}(f_z)\sin^{-1}\frac{f_y}{g}$$

$$A = \tan^{-1}\frac{-\omega_x}{\omega_y}$$

where f and ω are the mean accelerometer and gyroscope measurements during alignment, respectively, and g is the local gravity. These values are used to compute the rotation of the IMU w.r.t. the LLF as

$$R_b^l = \begin{pmatrix} \cos A\cos r + \sin A\sin r\sin p & \sin A\cos p & \cos A\sin r - \sin A\cos r\sin p \\ \cos A\sin r\sin p - \sin A\cos r & \cos A\cos p & -\sin A\sin r - \cos A\cos r\sin p \\ -\cos p\sin r & \sin p & \cos p\cos r \end{pmatrix}.$$

The quaternion describing this rotation is the calculated using the elements of R_b^l:

$$Q = \begin{pmatrix} (r_{32} - r_{23})/4q_4 \\ (r_{13} - r_{32})/4q_4 \\ (r_{21} - r_{12})/4q_4 \\ q_4 \end{pmatrix}$$

where

$$q_4 = \frac{1}{2}\sqrt{1 + r_{11} + r_{22} + r_{33}}$$

This quaternion is then divided component-wise by its Euclidean norm, which allows us to recompute the rotation matrix R_b^l and ensure it is orthogonal. R_b^l can be recovered from the normalized quaternion via

$$R_b^l = \begin{pmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_2 - q_3q_4) & 2(q_1q_3 + q_2q_4) \\ (q_1q_2 + q_3q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2q_3 - q_1q_4) \\ 2(q_1q_3 - q_2q_4) & 2(q_2q_3 + q_1q_4) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{pmatrix}.$$

Once alignment is complete and the initial attitude is determined, the `align` method will raise an error if called again.

2.3 Compensating for deterministic errors and computation of Earth parameters

The `compensate_errors_and_compute_params` method compensates for deterministic errors in the accelerometer and gyroscope measurements and computes the Earth parameters at the current latitude. Deterministic errors in the accelerometer measurements are corrected by accounting for the bias, scale factor, and non-orthogonality of the accelerometer. The corrected acceleration $\hat{f}$ is given by

$$\hat{f} = (I + S_f + N_f)^{-1}(f - b_f)$$

where f is the raw accelerometer measurement, b_f is the accelerometer bias, S_f is the scale factor matrix of the accelerometer, and N is the non-orthogonality matrix of the accelerometer. Similarly, deterministic errors in the gyroscope measurement are corrected via

$$\hat{\omega} = (I + S_\omega + N_\omega)^{-1}(\omega - b_\omega).$$

The local gravity g is calculated using the formula

$$g = a_1(1 + a_2 \sin^2\phi + a_3 \sin^4\phi) + (a_4 + a_5 \sin^2\phi)h + a_6h^2$$

derived by Heiskanen and Moritz (1967), where ϕ is the current latitude, h is the height above the ellipsoid, and the a_i are coefficients given by GRS 80.

The radii of curvature of the Earth in the prime vertical N and meridian M directions

are then computed as

$$N = \frac{a}{\sqrt{1 - e^2 \sin^2 \phi}}$$

$$M = N \cdot \frac{1 - e^2}{1 - e^2 \sin^2 \phi}$$

where e is the eccentricity of the Earth and a is the semi-major axis.

2.4 Angular velocity compensation

After alignment is complete and deterministic errors are corrected, the next step is to compensate for the rotation of the Earth and the movement of the local level frame. This is necessary since the gyroscope axes in the North and up directions observe part of the Earth's rotation, and the motion of the local level frame causes a change in orientation of the local level frame axes. Due to these effects, the rotation of the LLF w.r.t. the inertial frame as seen in the body frame is given by

$$\omega_{il}^b = R_l^b \begin{pmatrix} \frac{-V^n}{M+h} \\ \frac{V^e}{N+h} + \omega_e \cos\phi \\ \frac{V^e \tan\phi}{N+h} + \omega_e \sin\phi \end{pmatrix}$$

where ω_e is the rotation rate of the Earth and V^e and V^n are the velocities of the LLF in the East and North directions, respectively. Multiplying this by the current time step Δt gives the angular changes θ_{il}^b corresponding to ω_{il}^b. The angular changes of the body w.r.t. the LLF can then be computed as

$$\theta_{lb}^b = \theta_{ib}^b - \theta_{il}^b = \omega \cdot \Delta t - \theta_{il}^b$$

where ω is the corrected gyroscope measurement.

The `angular_velocity_compensation` method performs the above calculation and returns θ_{lb}^b as a numpy array.

2.5 Attitude integration

After angular velocity compensation, attitude is then updated in the `attitude_integration` method. First, the quaternion update matrix U is computed as

$$U = \begin{pmatrix} 0 & \Delta\theta_z & -\Delta\theta_y & \Delta\theta_x \\ -\Delta\theta_z & 0 & \Delta\theta_x & \Delta\theta_y \\ \Delta\theta_y & -\Delta\theta_x & 0 & \Delta\theta_z \\ -\Delta\theta_x & -\Delta\theta_y & -\Delta\theta_z & 0 \end{pmatrix}$$

where the $\Delta\theta_i$ are the components of θ_{lb}^b as returned by the `angular_velocity_compensation` method. The quaternion Q describing the rotation of the body w.r.t. the LLF can then be updated as

$$Q_t = Q_{t-1} + \frac{1}{2} U Q_{t-1}$$

and, if necessary, normalized by dividing component-wise by its Euclidean norm. Using the method described in section 2.2, the updated orthogonal rotation matrix R_b^l can then be computed from Q. The updated roll, pitch, and azimuth can be determined from R_b^l via

$$r = \tan^{-1} \frac{-r_{31}}{r_{33}}$$

$$p = \tan^{-1} \frac{r_{32}}{\sqrt{r_{12}^2 + r_{22}^2}}$$

$$A = \tan^{-1} \frac{r_{12}}{r_{22}}.$$

2.6 Velocity and position integration

The final step to each mechanization iteration is to update the LLF velocity and position. This is performed in the `v_and_r_integration` method. The acceleration measurements provided by the accelerometer not only contain the acceleration in the body frame, but also the accelerations caused by gravity and the Coriolis effect. Thus, these values need to be removed from the accelerometer measurements before updating the navigation parameters.

The correction for the Coriolis effect is given by $-2(\Omega_{ie}^l + \Omega_{el}^l)V^l$ where V^l is the velocity of the LLF. Ω_{ie}^l and Ω_{el}^l are the skew-symmetric representations of

$$\omega_{ie}^l = \begin{pmatrix} 0 \\ \omega_e \cos\phi \\ \omega_e \sin\phi \end{pmatrix}$$

and

$$\omega_{le}^{l} = \begin{pmatrix} \frac{-V^n}{M+h} \\ \frac{V^e}{N+h} \\ \frac{V^e \tan\phi}{N+h} \end{pmatrix}$$

respectively. The static method `vec_to_skew` converts these vectors to matrices, and then the true acceleration in the LLF is computed as

$$a^l = R_b^l f^b - 2(\Omega_{ie}^l + \Omega_{el}^l)V^l + g^l$$

where f^b is the corrected accelerometer measurement and $g^l = \begin{pmatrix} 0 & 0 & -g \end{pmatrix}^T$ is the gravitational acceleration in the LLF. The velocity increment ΔV^l in the LLF is then obtained by multiplying this acceleration with the time step, and velocity is then updated via

$$V_t^l = V_{t-1}^l + \frac{1}{2}\left(\Delta V_{t-1}^l + \Delta V_t^l\right)$$

where $\Delta V_0^l = 0$.

Finally, position of the LLF is then updated. The updated longitude λ and latitude ϕ are computed as

$$\lambda_t = \lambda_{t-1} + \frac{1}{2}\left(\frac{V_{t-1}^e + V_t^e}{(N+h)\cos\phi}\right)\Delta t$$

and

$$\phi_t = \phi_{t-1} + \frac{1}{2}\left(\frac{V_{t-1}^n + V_t^n}{M+h}\right)\Delta t$$

and the updated altitude is given by

$$h_t = h_{t-1} + \frac{1}{2}\left(V_{t-1}^u + V_t^u\right)\Delta t.$$

2.7 Summary of mechanization implementation

The steps to mechanization with the `INSMechanization` class can be summarized as follows:

1. Instantiate an instance of the `INSMechanization` class, passing in the initial position, error parameters, and alignment time.

2. Call the `process_measurement` method with each IMU update, passing in the timestamp and accelerometer and gyroscope measurements. This method functions as follows:

i. Call the `compensate_errors_and_compute_params` method to correct for deterministic errors in the accelerometer and gyroscope measurements and compute Earth parameters at the current latitude.

ii. If alignment is not complete, call the `align` method and return. This keeps track of the mean acceleration and rotation rate. On the last iteration during the alignment time, the rotation of the body w.r.t. the LLF is computed and `align` is no longer called.

iii. Compensate the angular velocity measurement for the rotation of the Earth and the movement of the LLF and compute the angular changes of the body. This is performed by the `angular_velocity_compensation` method.

iv. Update the rotation matrix and quaternion describing the rotation of the body w.r.t. the LLF, and compute the updated roll, pitch, and azimuth with the `attitude_integration` method.

v. Call `v_and_r_integration` to compensate the acceleration for the Coriolis effect and the acceleration due to gravity, and update the velocity and position of the LLF.

3. Obtain navigation parameters at any time by calling the `get_params` method.

2.8 Accessory methods

Several other methods are provided in the `INSMechanization` class to facilitate computation. They are described as follows:

- `@staticmethod gravity(lat, h)` — Returns the gravitational field strength in $\mathrm{m/s^2}$ given the latitude in radians and the height above the ellipsoid in meters.
- `@classmethod radii_of_curvature(cls, lat)` — Returns the radii of curvature of the Earth in the prime vertical N and meridian M directions given the latitude in radians.
- `@staticmethod vec_to_skew(v)` — Returns the skew-symmetric representation of a the input vector as a numpy ndarray.
- `@staticmethod get_rotation_matrix(r, p, A)` — Returns the rotation matrix of the body w.r.t. the LLF as a numpy ndarray given roll, pitch, and azimuth in radians.
- `@staticmethod matrix_to_normalized_quaternion(R)` — Converts a 3×3 rotation matrix into a normalized quaternion and returns it as a numpy array.

- `@staticmethod quaternion_to_matrix(q)` — Converts a normalized quaternion into a rotation matrix and returns it as a numpy array.

3 References