GyroLibNav - Matlab Integrated Navigation Systems Library

GyroLibNav is the library for Matlab that includes the following simple algorithms of integrated navigation systems:

- Three-wheeled robot - it is assumed that it's equipped with: odometers, that measure angular rates of each of the two wheels; azimuth gyroscope, that measures angular rate around vertical axis of the robot; GPS receiver, that was installed on some known horizontal lever relative to the center of wheel axis;
- Quadrocopter, equipped with the inertial measurement unit - IMU and GPS;
- Foot-mounted navigation system using only IMU and ZUPT updates.

Axes definitions. Navigation systems work in a local, horizontal frame - so called NED frame (North-East-Down) with X axis pointing to the North, Y axis to the East, and Z pointing down. To greatly simplify the algorithms we assume that the NED is a non-rotating inertial frame, which is not true in reality, but as long as we are using pretty noisy MEMS inertial sensors and we do not move far from the starting position to be able to observe the Earth's curvature - we are fine.

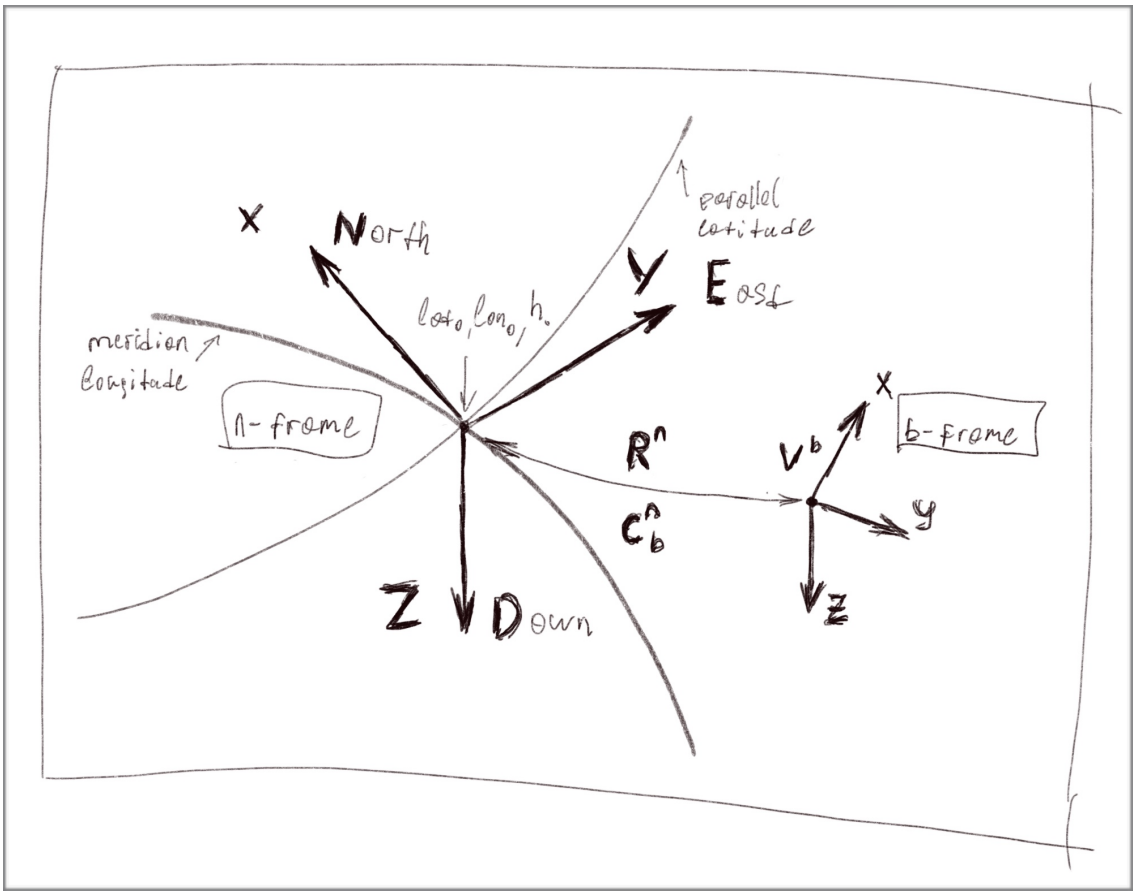
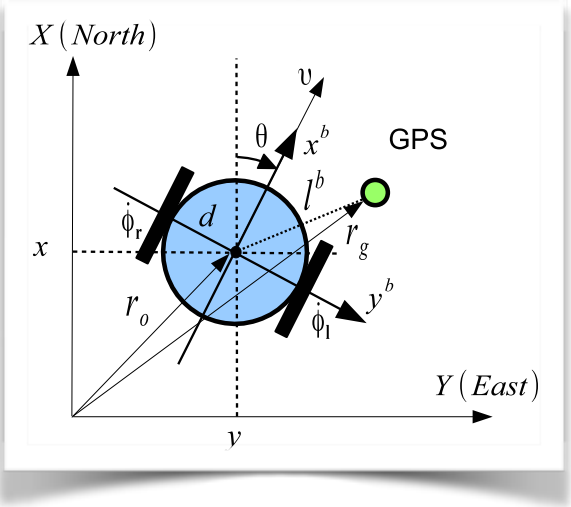


Fig. 1 Coordinate frames

Sensors measurements. The IMU includes three-axis accelerometer and gyroscope with mutually orthogonal axes, corresponding to the body frame of reference: x axis is pointing “forward”; z axis - “downwards”; y axis completes the right orthogonal system of the axes.
Gyroscopes measure positive angular rates being rotated counter-clockwise (if one watches from the end of the corresponding axis). Here we actually assume that the gyroscope triad measures the increments of the attitude angles during the computer cycle. Accelerometer “measures” the gravity force (it actually doesn’t - it can only measure active forces, and when it sits on the table it measures the force with which the table acts on the accelerometer body preventing it from falling right through the lid of the table). Here we also assume that we measure velocity increments during the computer cycle. We are using this increments because almost any decent modern IMU do this for you onboard - it collects data with high sampling rate, integrate it and provide you with increments of angles and velocities. Gyroscopes angular rates measurements are in rad/sec (angle increments in radians), accelerometers measurements are in m/sec^2 (velocity increments in m/sec). GPS measures coordinates of receiver’s antenna in NED frame (in meters) and velocity vector in the same frame (in m/sec). Of course in real life GPS measures altitude, longitude and altitude or coordinates of the receiver in ECEF (Earth-Centered, Earth-Fixed) frame, but it’s really easy to transform these coordinates to local NED frame using **geo2ned** and **ecef2geo** functions supplied with GyroLibNav. Odometers of the three-wheeled robot measure wheel angular rates in rad/sec.
Order of rotations. The standard ZYX order is used - first we rotate counter-clockwise around the z axis, then around the new y axis and finally around the new x axis of the b-frame.

Three-wheeled robot

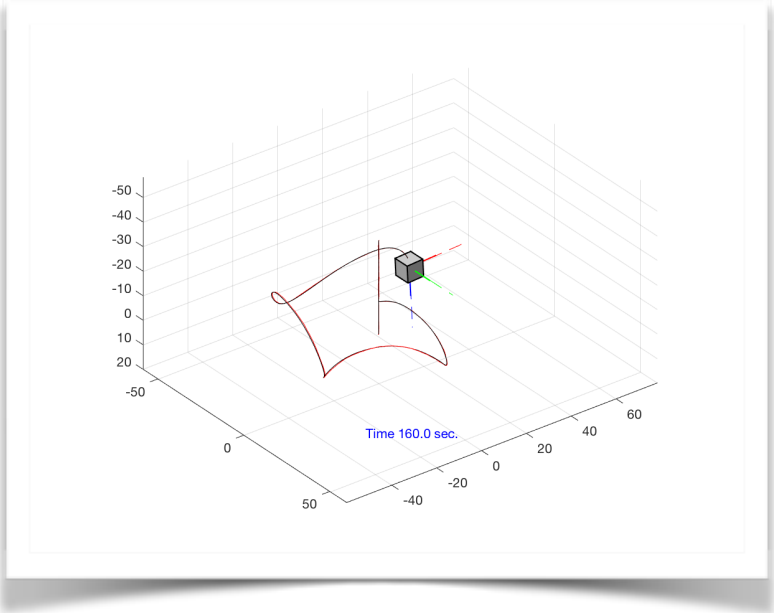
We need to calculate X and Y coordinates of the three-wheeled robot, its velocity and heading angle (see figure below). As a set of measurements we have: angular rates from wheel odometers; heading angle increment from azimuth gyroscope; velocity and position measurements from the GPS. All measurements are assumed to be made with errors: wheels radiuses are known imprecisely; gyroscope measurement include the bias; GPS coordinates and velocities are noisy. So, besides the coordinates, velocity and heading of three-wheeled robot we also need to estimate and compensate these errors. Traditional approach with complementary Kalman filter is used. See the code of **test_threewheel.m** function for details.



To run three-wheeled robot navigation example first load the test data **threewheel.mat** to Matlab workspace and then run the following:
test_threewheel(Rn_ref, Vb_ref, Euler_ref, dThe_ref, dt);

Quadrocopter

We need to calculate 3D position, velocity and three attitude angles of the quadrocopter relative to the NED frame. As a set of measurement we have: angles and velocities increments from the IMU; coordinates and velocities from the GPS. All the measurements are assumed to be made with errors: accelerometers and gyroscopes are noisy and have different biases for each of the channels; GPS reading are noisy. Angles and velocities from the IMU are fed to the strapdown navigation system algorithm. We use traditional approach with complementary Kalman filter to estimate and compensate IMU errors. For the details of realization see the code of **test_quad.m** function.



To run the quadrocopter navigation example load the **quad.mat** test data first and then run the following:
test_quad(dThe_ref, dUps_ref, Rn_ref, Vb_ref, Euler_ref, g, dt);

Foot-mounted navigation system

Here we calculate the coordinates, velocities and the attitude of the human foot. We strapped the 6-axis MEMS IMU to it. The reason for this placement is that this is the only part of human body that doesn't move for some short periods of time even when we walk or run. There is this tiny moment when we can be sure that the velocity of the

foot is essentially zero, we can detect this moment and use this in a framework of ZUPT update - or pseudo-measurement for the Kalman filter. For the details see the code of **test_foot.m** function.

To rune the foot-mounted navigation system example load the foot.mat test data that includes the real-life recording from FMT1100 6-axis MEMS IMU and then run the following: **result = test_foot(dThe, dUps, g, dt);**

Reference simulations of INS-GPS integrated system

GyroLibNav library also includes couple functions that allow you to synthesize the reference dataset for IMU and GPS readings and then simulate how the integrated system with IMU and GPS works on this dataset. Reference data is generated by calling the following function:
[Rn_ref, Vn_ref, Euler_ref, dThe_ref, dUps_ref, dt, gn, Rn0, Vn0, Cbn0] = reference_spinrocksize;
This generates the analytically exact reference for the motion of the solid body using SPIN-ROCK-SIZE algorithm by Paul G. Savage. You can then test how the IMU-GPS integrated system works by calling the following function:
test_spinrocksize(dThe_ref, dUps_ref, Rn_ref, Vn_ref, Euler_ref, Rn0, Vn0, Cbn0, gn, dt);