Use three different transformations to get a final one. Basically, rotate around the z-axis, and compute the transformation matrix (inertial to frame 1, or I->f1). Then repeat this for the y and x-axis. What this does is change values from one frame of reference to another. Usually (I would imagine always), one frame is unmoving (such as the earth frame of reference). So in our example, you want to change the accelerometer data from the body frame of reference to the earth frame so it’s easier for a person to understand, or change it from the earth reference to the body reference in order for the machine (quadcopter) to understand the information. Why would we do this? Well, let’s say you want to go 5 meters/sec in the x direction. To do this, the ‘copter has to tilt around the y-axis (pitch). As it starts to move, there will be an acceleration on the both the z-axis and x-axis. This is in the body frame of reference. Using the transformation matrix, the acceleration in the x-direction relative to the earth can be easily found.

The most common place this is used (so far) is to remove gravity from the accelerometer’s readings. As the accelerometer changes orientation, gravity is going to have a different effect on the axes. To remove this, gravity and the accelerometer readings need to be put into same frame.

The rotation matrices below are basically taking the angle and using that to scale the initial frames value. Since and similarly for sine, we can find the new value based on the change in angle, then that gets multiplied by the hypotenuse (which is the reading of the initial frame’s value).

Rotation about the z-axis from the inertial frame to intermediate frame 1

Rotation about the y-axis from intermediate frame 1 to intermediate frame 2

Rotation from the intermediate frame 2 to the body frame of reference

Need to put this all together to get a get a rotation matrix that takes you from the inertial frame of reference to the body frame of reference.

Change from the inertial frame to the body frame

Change from the body frame to the inertial frame

Dead Reckoning

Dead reckoning is the method of using inertial measurements to calculate the kinetic information of the system. That is, use an accelerometer to calculate the velocity by integrating it. Then integrating that you get position.

As time goes on the error for velocity will accumulate linearly (v=mx+b sort of thing), but for position, the error accumulates in a non-linear fashion (p=a2+c). Also, when you integrate, you should get a constant value that represents is usually found using initial conditions. But because this is technically a nonlinear system, we can’t know this value, so there will be an error introduced. As can be seen in the one figure, the accelerometer reading error is less than 0.05, but the velocity and position still diverge quickly.

The gyro isn’t really a huge deal for this. Because we only need to integrate once, there isn’t too much of an error, and the readings are pretty stable. The only issue is in the long term because of drift.

Filtering

For now, a recursive filter is being used to clean up the accelerometer signal. Since the accelerometer is very noisy, it needs that cleaning up. It is also an attempt to improve the dead reckoning. Because of the noise, the accelerometer needs a good filter in order to accurately get the velocity/position, especially since gravity needs to be filtered out.

Because filters can be complicated and computationally intensive quickly, a simple running average filter was used. This greatly helped to reduce the noise within the accelerometer and it’s relatively easy to implement.

Ideally, a Kalman filter would be used. This is a filter that uses the least sum of squares and is recursive. It basically predicts the value you’re filtering, then uses the known error rate of your sensor(s) to generate a covariance matrix. From there, the sensors are read, and an error is calculated and a gain applied. However, due to the fact that this system is nonlinear (nonlinear because the system could change at any time), and the Kalman filter can be a little complicated for a beginner, it hasn’t been implemented yet. First it’s being tested in Matlab (to accommodate for the matrix math since the filter is done using matrices).

Calibration

Calibrating these instruments can be annoying. They have drift and unless the unit is absolutely still and level. If it’s not there will be some error in the calibration. This calibration means that the values for the gyro and accelerometer are read in and averaged (right now it’s done 2000 times). Then this value is subtracted whenever the IMU is read. However, there is still come error in the accelerometer (less than 0.05 m/s/s, though), and overall it is creating issues and needs to be solved.

Also, after a while drift needs to be dealt with. To do this, the unit needs to stop in a known position, and re-calibrate. One possible solution is to use two IMUs. One calibrates itself, while the other is used for measurements, or something like that.