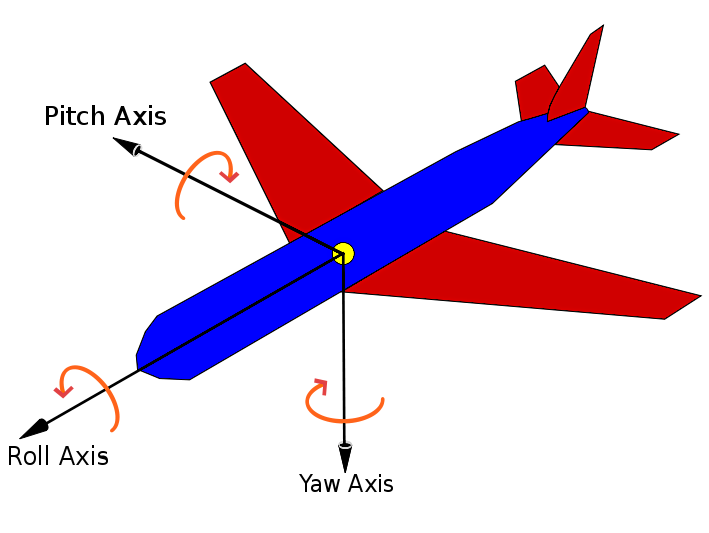
Technology Review

Our flight control system has one goal: to keep a quadrotor helicopter in level flight at all times despite environmental factors, gravity, and flight path. To these ends, the quadrotor’s flight system will be equipped with several sensors streaming data to our program. Currently, our sensor inputs include three 1-axis gyroscopes, one 3-axis accelerometer, and one 3-axis magnetometer.

From a software engineering perspective, this project has two computationally intensive elements, filtering the sensor data and processing the cleaned data. Our goal is to calculate the most accurate correction possible while staying with in our time constraints and the microprocessor’s computational abilities. The correction vector will be measures in degrees and we will be correcting tilt (pitch), roll, and yaw (see diagram).



There are numerous ways to filter and process data. However, in order to discuss data filtering techniques, we must first understand the type of noise we expect from the sensors. In this case, the sensors will have high frequency noise from the electronics and, most likely, some outlier readings as well as a certain measurement error margin from the sensor.

Beginning with data filtering, high frequency noise is trivial to filter with a low pass filter. Low pass filters are so common; there really isn’t an alternative. What does vary is the implementation. Low pass filters can be implemented in either hardware or software. In hardware, a low pass filter is a simple resistor-capacitor circuit. In this circuit, a high capacitance capacitor will reduce the amount of high frequency noise while the filter’s cutoff frequency is determined by the time constant, RC. A software low pass filter would take the signal’s Fourier transform over some time then use data from frequencies lower than the cutoff frequency.

Filtering outliers and system error is much more complex both mathematically and computationally. Kalman filter are the most commonly used data filter in this realm. Kalman filters use the data from all sensors to reduce mean squared error, creating a more accurate overall solution. Furthermore, Kalman filters are recursive in nature and can be used to estimate past and future states using feedback control. Basically the filter makes some estimate in time then corrects with noisy measurements. Once the filter is constructed, adding covariance to the model and tweaking values allows hefty customization until the filter works smoothly.

Again, while using a Kalman filter is industry standard for filtering, there are different implementations and different varieties of Kalman filters. The discrete Kalman filter analyzes measurements at discrete points in time by assuming the process noise covariance and measurement noise covariance are constant. Similarly, we could use an extended Kalman filter or unscented Kalman filter if we chose to represent the quadrotor’s system with a nonlinear model. The unscented Kalman filter is another non-linear model using deterministic sampling points with the unscented transform to more accurately capture the mean and covariance. Along with any of these Kalman filters, we could also use fixed-interval smoothers (there are many) to further clean the data.

Finally, we can use PID (proportional–integral–derivative) to calculate the final position change. PID uses previous positions with a feedback loop to calculate the “error” or change between the previous and new positions. While PID isn’t precisely a filter, it allows us to adjust the flight system’s sensitivity to prevent overshoot and uncontrollable oscillations as the quadrotor tries to correct for the overshoot. PID can be improved given some model of the system or with further smoothing algorithms.

In summary, we have some fixed modules we should use. Other options are deprecated compared to a Kalman filter and PID base but there are numerous implementation options. At a bare minimum we need to use PID and a low pass filter. The low pass filter would eliminate most noise while PID would prevent overcorrection. In our case, we decided to use hardware low pass filters since they are very effective, small and cheap at our power level, and don’t require microprocessor time (our main limiting factor). The PICCOLO microprocessor we’re using is fast enough to run a discrete Kalman filter and maybe PID. Depending on load balancing, basic PID calculations may be on the PICCOLO or on the Flight Controller’s Xtmega though a smaller Kalman filter with PID on the PICCOLO would be better for modularity and would keep PID on a microprocessor with accelerated floating point. While the extended Kalman filter could improve the output data quality, it would also require more processing power than we have available.