ESE 650 - Learning in Robotics

PROJECT 2
Orientation Tracking

Varsha Shankar
vasha@seas.upenn.edu

The stages outlined below are according to the order that my implementation follows :

Interpreting IMU Data
The bias in the accelerometer data and the rate gyroscope data are calculated by computing the average of the first 500 data points in the readings. This is because for this period of time, the platform is kept stable and hence the bias of the instruments is assumed to be very close to the readings themselves. The bias as calculated from the data sheet for the IMU is not the same as this calculated value, but this was expected. The x and y axes of the accelerometer data is flipped and this is corrected for by negating the Ax and Ay values.
The gyro readings once converted into angular velocity are further converted into rotations in radians by multiplying it 1/frequency, the frequency being 100Hz.

Conversion to Quaternions
The gyro rotations from the previous stage are converted into quaternions using MATLAB's angle2quat() and subsequently normalised using quatnormalize(). These gyro readings are used later on to update the state in the Process Model.
The first accelerometer reading is converted into a quaternion using the followings equations:

$$Roll = atan2(\frac{-A_x}{A_z}) \qquad Yaw = 0 \qquad Pitch = atan2(\frac{-A_x}{\sqrt{(A_y)^2 + (A_z)^2}})$$

Overview
My UKF implementation keeps track of four states in the form of a quaternion. The four states represent the orientation as given the accelerometer. The process model updates the current state by using the gyro reading which specifies a rotation which the platform has gone through. Once the process is updated to reflect the new predicted orientation, a measurement update is made using the true 'g' vector. This is compared to the observed accelerometer reading from the IMU. The Kalman gain computed from this deviation is used to correct the mean and covariance of the current state and leads to the next state.

Naive Model
Beginning with an initial state corresponding to the first quaternion representation of the accelerometer reading, I calculated a set of simplistic states using :

$q_{\omega\Delta t} = gyro\ rotation\ at\ time\ t-1$
$q_t = q_{t-1} q_{\omega\Delta t}$

These states are referred to in my code as well as in this report as ‘simple states’. Apart from plotting these simple states side by side the states from the UKF model , I use this state to bootstrap the selection of the mean of the transformed sigma points.

Process Model

The first state is initialised as the first quaternion obtained from the accelerometer reading.

Sigma Points

For each time t of the model, sigma points are calculated using P (the state covariance) and Q (the process noise covariance) and the previous state. This is done by performing a Cholesky decomposition of (P+Q)*n followed by a quaternion multiplication of the previous state and each column of the said decomposition.

The sigma points are then transformed to the projected sigma points $\{Y_i\}$ by a quaternion multiplication with $q_{\omega\Delta t}$.

Estimation of the mean of the projected sigma points

The mean of Y_i = $\hat{x_k^-}$ is estimated iteratively by computing error vectors between the estimated mean of the previous iteration and each of the sigma points from $\{Y_i\}$. I have used 100 iterations on this gradient descent. The iteration process of the gradient descent begins with an initial guess of the mean. I have used q_t from the naive model as the initial guess. This allows convergence in a fewer number of iterations. The error vectors $\{\vec{e_i}\}$ from the last iteration of this step are used subsequently for the computation of the covariance.

Estimation of the state covariance

The covariance of the state is computed using the error vectors from the mean finding step $\{\vec{e_i}\}$ as the column vectors of a matrix whose covariance is equal to that of the state. Thus,

$W_i = \{\vec{e_i}\}$ and $\bar{P}_k = cov(W_i)$

This concludes the Process model. The next step is to refine our estimates of the state mean and covariance using a measurement update.

Measurement Model

The measurement model involves projecting the projected sigma points $\{Y_i\}$ into $\{Z_i\}$. This is done by rotating the ground truth ‘g’ vector (0,0,1) into the coordinate of the tracker system. Therefore we perform a rotation using MATLAB’s quatrotate().

The $\{Z_i\}$s are a set of vectors that represent ‘g’ in our platform’s coordinate frame. The mean of the $\{Z_i\}$s is calculated as $\bar{z_k}$.

Innovation

The innovation is defined as the difference between the actual measured value of the observation (the accelerometer reading at time t = $\bar{z_k}$) and $\bar{z_k}$.

$$\bar{v_k} = \bar{z_k} - \bar{z_k}$$

Uncertainty in measurement

The uncertainty in the predicted mean $\bar{z_k}$ is given by the covariance of the set of $\{Z_i\}$s. Also, we cannot completely trust our sensor data. This leads to its own noise covariance. The uncertainty in measurement thus needs to factor into the measurement model and this is taken together by combining the above two covariances into the measurement noise covariance given by :

$$P_{vv} = P_{zz} + R$$

Cross correlation

The cross correlation P_{xz} between the noise in the state W_i and the noise in the measurement Z_i-$\bar{z_k}$

Kalman Gain

The Kalman Gain is found as follows:

$$K_k = P_{xz} * P_{vv}^{-1}$$

Using the Kalman Gain, the estimated mean is updated.

$$\hat{x_k} = quatmultiply(\bar{x_k}, \ convert2quat(K_k * \nu_k))$$

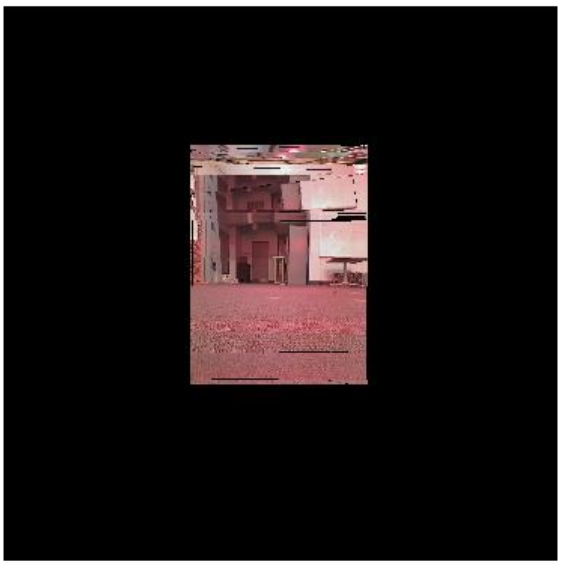
Again, using the Kalman Gain, the estimated covariance is updated

$$P_k = \bar{P_k} - K_k * P_{vv} * K_k^T$$

This concludes the measurement update step. The state is now corrected for.

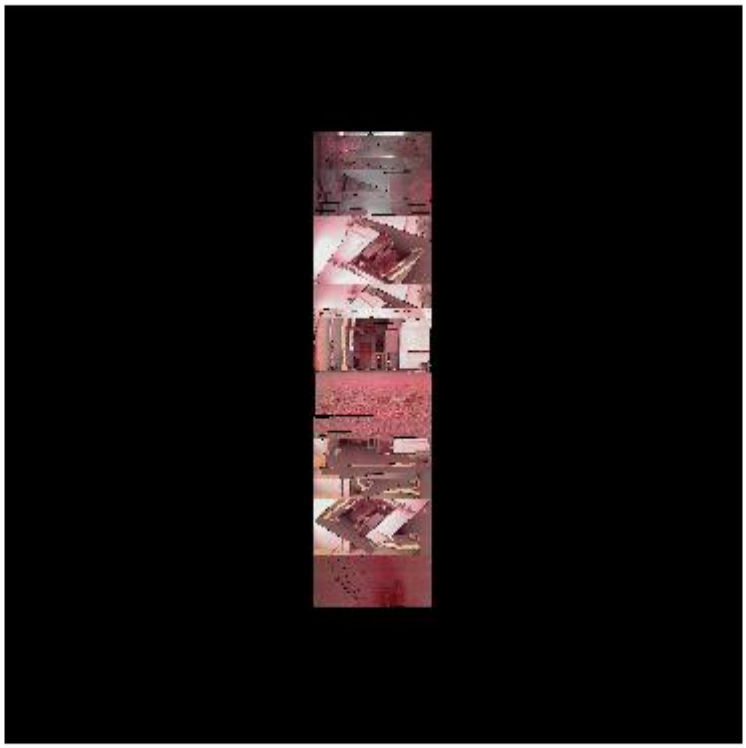
Mosaic Creation

Using the states obtained from the IMU data, a mosaic of the corresponding camera images are built. The first step in this however was to identify for each camera image, the timestamp that most closely corresponds to the IMU data's timestamp. The appropriate state is identified and used to plot the image on a large 'canvas'. The state quaternion is converted into its corresponding Euler angles. The pitch is used to position the image vertically in the correct spot whereas the roll is used to rotate the image to its correct orientation. Some example mosaics are shown below:

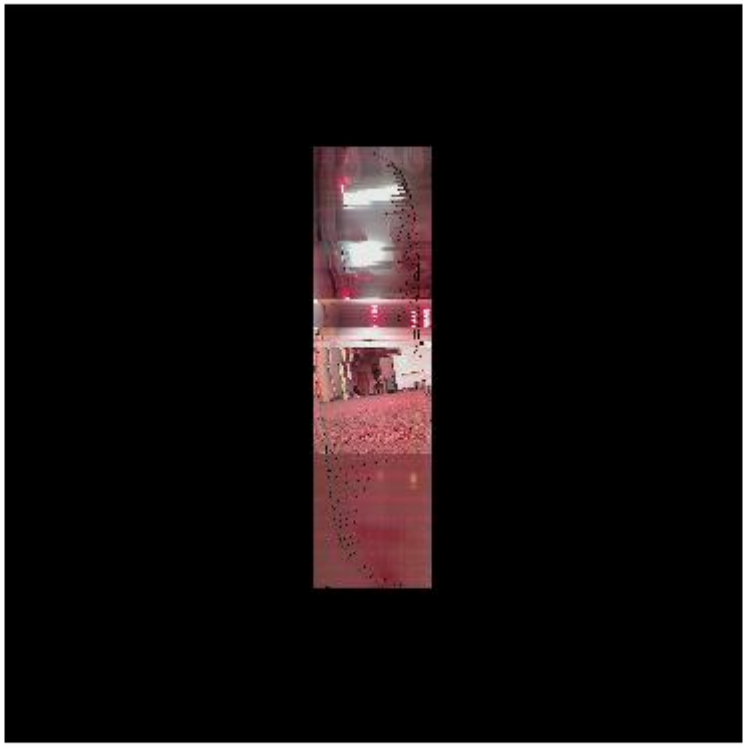


Data Set 8

Data Set 1 with naive model



Data Set 1 with UKF



Analysis

The results of the naive model show that using only the gyro, the rotations are stable, but not always necessarily correct. However with my UKF model, at the extremes of the rotation, the rotations get slightly out of control. Unfortunately, I wasn't able to find the time to implement a learning algorithm to learn the noise covariances. But I did try playing around with the values manually and noticed variations in the rotations being caused by them. Further, the covariances that I have currently in my code seem to give the best performance that I could achieve with my code.