MEAM 620, Advanced Robotics, Spring 2013: Project 2 Phase 3

Group 3: Jonathan Balloch, Jialue Huang, Qiong Wang

May 1, 2013

1 Analysis

1.1 Review: Calculating Pose

Pose calculation is based on linear homogeneous method in project 1 phase 1.

Given position in world frame and calibrated image coordinates

$$\text{Given:}(X, Y, x, y)_{i=1,2,\ldots,n}$$

We can find R, T using 4 points. We have equations

$$\begin{cases} x_i = \frac{r_{11}X_i + r_{12}Y_i + T_1}{r_{31}X_i + r_{32}Y_i + T_3} \\ y_i = \frac{r_{21}X_i + r_{22}Y_i + T_2}{r_{31}X_i + r_{32}Y_i + T_3} \end{cases}$$

which is equivalent to

$$\begin{cases} x_i X_i r_{31} + x_i Y_i r_3 2 + x_i T_3 - X r_{11} - Y r_{12} - T_1 = 0 \\ y_i X_i r_{31} + y_i Y_i r_3 2 + y_i T_3 - X r_{21} - Y r_{22} - T_2 = 0 \end{cases}$$

the equivalent matrix form is

$$\begin{pmatrix} -X_1 & 0 & x_1X_1 & -Y_1 & 0 & x_1Y_1 & -1 & 0 & x_1 \\ 0 & -X_1 & y_1X_1 & 0 & -Y_1 & y_1Y_1 & 0 & -1 & y_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -X_n & 0 & x_nX_n & -Y_n & 0 & x_nY_n & -1 & 0 & x_n \\ 0 & -X_n & y_nX_n & 0 & -Y_n & y_nY_n & 0 & -1 & y_n \end{pmatrix} \begin{pmatrix} r_{11} \\ r_{21} \\ r_{31} \\ r_{12} \\ r_{22} \\ r_{32} \\ T_1 \\ T_2 \\ T_3 \end{pmatrix} = 0$$

Let

$$A = \begin{pmatrix} -X_1 & 0 & x_1X_1 & -Y_1 & 0 & x_1Y_1 & -1 & 0 & x_1 \\ 0 & -X_1 & y_1X_1 & 0 & -Y_1 & y_1Y_1 & 0 & -1 & y_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -X_n & 0 & x_nX_n & -Y_n & 0 & x_nY_n & -1 & 0 & x_n \\ 0 & -X_n & y_nX_n & 0 & -Y_n & y_nY_n & 0 & -1 & y_n \end{pmatrix}$$

Again apply SVD to solve this question

$$U\Sigma V^T = svd(A)$$

Take the last column of V and normalize it to get r_1, r_2 and T. r_3 is easily computed by

$$r_3 = r_1 \times r_2$$

So the solution is

$$R = \begin{pmatrix} r_1 & r_2 & r_3 \end{pmatrix}$$
$$T = \begin{pmatrix} T_1 & T_2 & T_3 \end{pmatrix}$$

1.2 Review: Controller for Quadrotor

As in phase 2, because all of the motions of the quadrotor are sufficiently close to a hover state (e.g., the roll and the pitch less than or equal to $\frac{\pi}{12}$) we can use a linear approximation of the quadrotor's command desired accelerations and roll and pitch angles.

Starting with Newton's Equations of Motion:

$$m\ddot{r} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ \sum F \end{bmatrix} \tag{1}$$

where m is the mass, $\ddot{r}$ is the acceleration vector, g is gravity, R is the rotation matrix from the normal propeller frame into the world frame, and $\sum F$ is the sum of the forces from each propeller.

Linearizing (1) this we get:

$$\ddot{r} = \begin{bmatrix} g(\triangle\theta \cos\psi_0 + \triangle\phi \sin\psi_0) \\ g(\triangle\theta \sin\psi_0 - \triangle\phi \cos\psi_0) \\ \frac{1}{m}u_1 - g \end{bmatrix} \tag{2}$$

Because of the hover approximation we can assume that $\triangle\theta \approx \theta - \theta_0$ and $\triangle\phi \approx \phi - \phi_0$, and therefore that

$$\begin{aligned} \ddot{r}_1 &= g(\theta^{des} \cos\psi_0 + \phi^{des} \sin\psi_0) \\ \ddot{r}_2 &= g(\theta^{des} \sin\psi_0 - \phi^{des} \cos\psi_0) \\ \ddot{r}_3 &= \frac{1}{m}u_1 - g \end{aligned} \tag{3}$$

We can form a PD controller by scaling the vector of accelerations by the error that is proportional to and derivative of the state x:

$$\ddot{r}^{des} = k_d(\dot{r}_T - \dot{r}) + k_d(\dot{r}_T - \dot{r}) + \ddot{r}_T \tag{4}$$

where the subscript T represents 'trajectory', which for the case of hover is simply 0, for the original state. Thrust, the first control output, was found in grams of thrust, as opposed to newtons as in phase 2. Given that, we were able to find desired accelerations in all 3 degrees of freedom, and the desired Euler angles.

It was decided that for simplicity to keep $\psi_{des} = 0$ at all times. Unlike in phase 2 where the desired moments in all three rotational DOF were the second, third, and four control output, in phase 4 the control output is in the form of $\langle\, thrust\ roll\ pitch\ yaw \,\rangle_{desired}$. Given this information, and equations (3) and (4), we find the remaining 3 control output to be:

$$u_1 = (\ddot{r}_3^{des}\frac{m}{g} + m)1000 \tag{5}$$

$$u_2 = \phi^{des} = \frac{1}{g}(\ddot{r}_1^{des} \cos\psi_0 + \ddot{r}_2^{des} \sin\psi_0) \tag{6}$$

$$u_3 = \theta^{des} = \frac{1}{g}(\ddot{r}_1^{des} \sin\psi_0 + \ddot{r}_2^{des} \cos\psi_0) \tag{7}$$

1.3 Extended Kalman Filter (EKF) Algorithm

In a linear system, we define the relationship between the state and the measurement and control as,

$$\begin{aligned} X_t &= AX_t + BU_t + \epsilon \\ Z_t &= CX_t + \delta \end{aligned}$$

In this representation of the system, the state at a given time is X_t, the measurement for that time is Z_t, the control is U_t, and the two error functions $\epsilon = \mathcal{N}(0, R)$ and $\delta = \mathcal{N}(0, Q)$ are each defined by a Guassian.

The algorithm for the Linear Kalman Filter is as follows:

Given $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$, find μ_t, Σ_t :

The "Prediction" Step

$\bar{\mu}_t = A\mu_{t-1} + Bu_{t-1} + \epsilon$

$\bar{\Sigma}_t = A\Sigma_{t-1}A^T + R$

The "Correction" or "Measurement" Step

$K_t = \bar{\Sigma}_t C^T (C\Sigma_t C^T + Q)^{-1}$ (This is a quantity called the *Kalman Gain*)

$\mu_t = \bar{\mu}_t K_t (z_t - C\bar{\mu}_t)$

$\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$

Where Q and R are the deterministic parameter covariance matrices that were defined in the model. The Extended Kalman Filter is an extension of this so that it can be applied to behavior that is nonlinear, but that can be approximated by being linearized locally. Because the quadrotor rarely behaves in a strictly linear fashion, we have to locally approximate its behavior at each point as linear. This is essentially the principle behind the EKF.

In the EKF We define the state and measurement as

$$X_t = g(X_{t-1}, U_t, \epsilon_t)$$
$$Z_t = h(X_t, \delta_t)$$

where functions g and h are the functions that relate the state to the control and the measurement recursively. In the case of part 1, where we are estimating pose solely from vision and the body frame velocity, g is defined as

$$g = \begin{bmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + R(\phi,\theta,\psi)\left(\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} + \epsilon\right)\Delta t \\ \texttt{Rot2RPY}(R(\phi,\theta,\psi) - \Delta R(\omega_x, \omega_y, \omega_z, \Delta t) \end{bmatrix}$$

Where $X_t = (x, y, z, \phi, \theta, \psi)^T$, and $h = (x, y, z, \phi, \theta, \psi)^T$. The linearization of these nonlinear functions are done using Jacobians (as matrix extentions of the Taylor approximation $f(x) \approx f(a) + f'(a)(x - a)$:

$$G_t = \frac{\partial g}{\partial \vec{x_{t-1}}}\bigg|_{y=0} \qquad L_t = \frac{\partial g}{\partial \vec{\epsilon_t}}\bigg|_{y=0} \qquad H_t = \frac{\partial h}{\partial \vec{x_t}}\bigg|_{y=0}$$

These jacobians are difficult to calculate and are typically found, as they were in this project, using a program like Matlab.

The EKF algorithm is therefore as follows:

Given $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$, find μ_t, Σ_t :

The "Prediction" Step

$$\bar{\mu}_t = g(X_{t-1}, U_t, (\epsilon_t = 0))$$
$$\bar{\Sigma}_t = G\Sigma_{t-1}G^T + LRL^T$$

The "Correction" or "Measurement" Step

$$K_t = \bar{\Sigma}_t H^T (H\Sigma_t H^T + Q)^{-1} \qquad \text{(This is a quantity called the } \textit{Kalman Gain}\text{)}$$
$$\mu_t = \bar{\mu}_t K_t (z_t - h(\bar{\mu}_t))$$
$$\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$$

1.4 Extended Kalman Filter Applied to Offline Dataset

The results of plots for the offline EKF filterfor part 1 are shown in Figure 1 - 2. The results of the offline EKF filter for part 2 are shown in Figure 3 - 4.

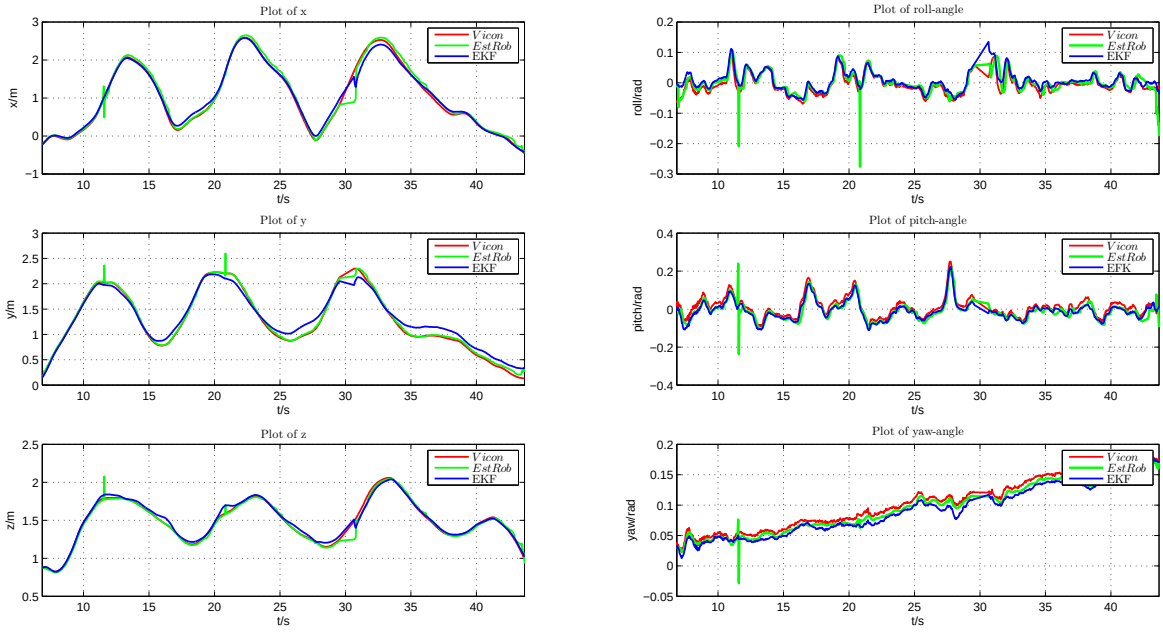

Figure 1: Estimated Position and Orientation with EKF applied to camera data

2 Codes

The programming of the controllers was largely a modification of the same codes from Project 1: Phase 4, integrated with the EKF and the pose data extracted from images of the April Tag mat done in Project 2: Phase 1.

Because the PD controller for the desired angles and thrust was dependent on the assumption of a near-hover state, the controller was very similar for all three controllers. Proportional (k_p) gains of 3,3, and 2 and derivative (k_d) gains of 1.5, 1.5, and 1 were used, as suggested, for prototyping the controller. After the gains were tuned it was found that gain values of $k_p = [3\ 3\ 3]$ and $k_d = [2.5\ 2.5\ 2]$ was best suited for both hover and faster speeds between multiple way points (see Section: Results for more on this) while

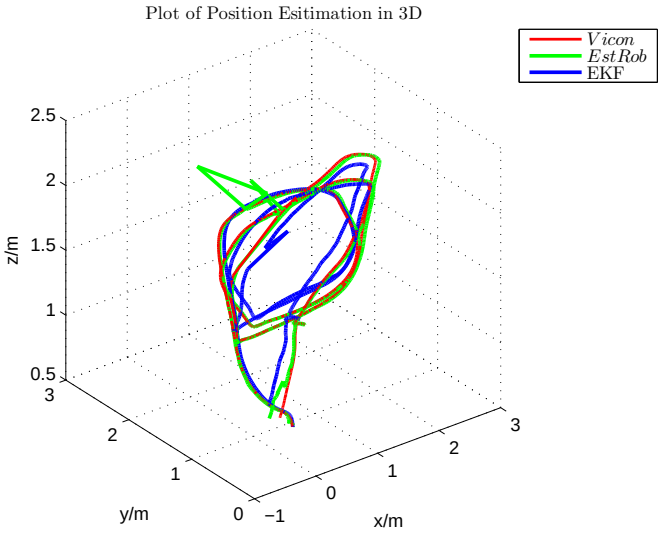

Figure 2: Estimated 3D Position with EKF applied to camera data

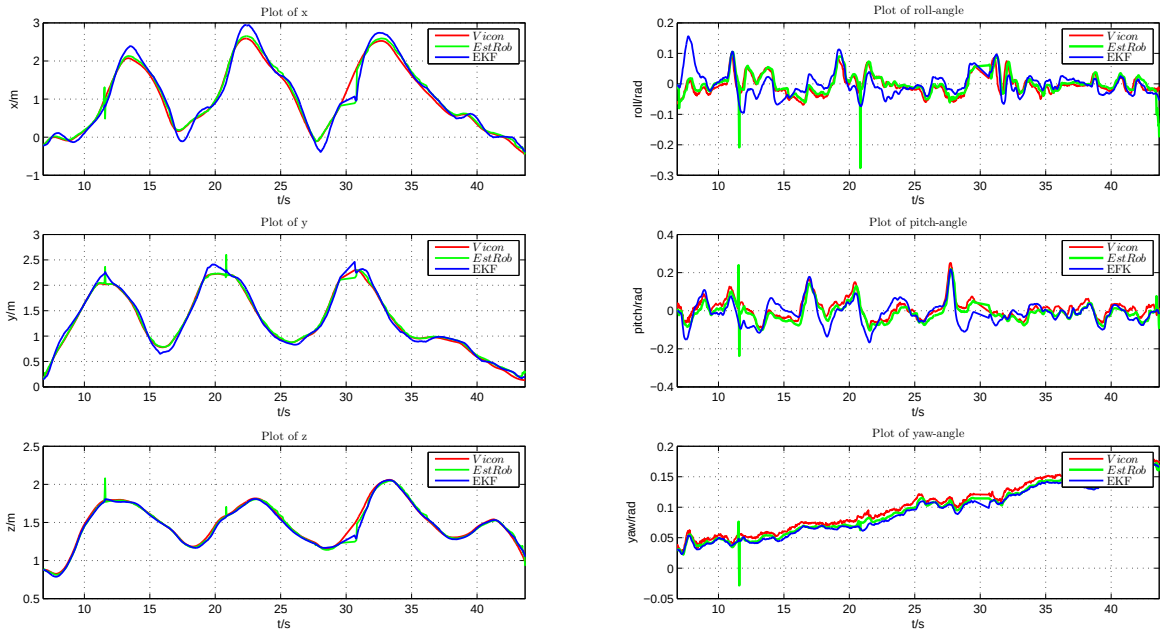

Figure 3: Estimated Position and Orientation with EKF applied to IMU data

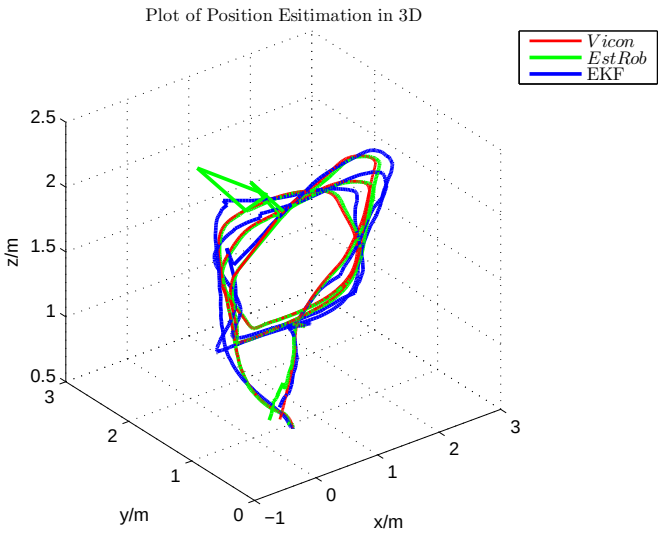

Figure 4: Estimated 3D Position with EKF applied to IMU data

remaining robust to occurances of no image data. This gain could have been tuned slightly more if there was more time, but was suitable for flying the quad precisely and robustly. To set up each controller, a time `t_start = GetUnixtime` was declared as the beginning of the time the controller was running. Additionally, a gravity constant was declared, as well as a constant quadrotor mass of 0.232 kg, which is heavier than 0.220 kg, the mass found in Project 1: phase 4.

2.1 EKF Initialization and Parameters

In the first part of the project, the setup of the EKF is as it was layed out in the **Analysis** section. For that EKF, the Q and R matrices were found by tuning the offline data. Using trial and error, and the knowledge that a smaller variance value in the covariance makes the filter trust the measurement corresponding to that variable more, the covariance we chose to use for this part was,

$$R = diag([0.10.10.10.10.10.1]); Q = diag([0.30.30.30.10.10.05]);$$

For part 2, the full state was,

$$X_t = (x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, ba_x, ba_y, ba_z, b\phi, b\phi)^T$$

Where the "b" terms referred to the biases of the error from the IMU. In part 2, the Q and R were found to be

$$R = diag([0.10.10.10.10.10.10.10.10.1]); Q = diag([0.050.050.050.030.030.03]);$$

the generally low covariances show that we mostly trusted the measurement, but the larger covariances in Part 2 show we trusted the variances less when depending on the IMU

2.2 Hover Codes

The values of yaw, position, and velocity were pulled from X_{est}, which was the EKF final approximation for the state, and the gain values k_p and k_d were set as 1×3 vectors. Using a vectorized implementation of (5) above, the desire linear accelerations were found. The trust was found to be

$$F = 1000((acc_{des}(3) + 1)\frac{m}{g}).$$

The desired roll and pitch were exactly as the are shown in (6), dependent on the current yaw ψ, and the desired yaw was always set to 0;

2.3 Waypoint Codes

In the way point code, end point (for single) or way points are specified in the sequence message, which is then read in to the controller code in its first iteration. The same trajectory generator from Project 1 is used to calculate desired states along the given path, and it is these desired state with which the EKF is compared at each iteration. Once within a margin of the last point, the controller commands a hover state about that location.

3 Results

In the following figures, red curves are from vicon data, which indicates the true ground. Green curves represent the estimated pose caculated by the algorithm developed in phase 1 project 2 with the tags on the map. Blue curves are the results after applying EKF.

To ensure the quadrotor stay relatively horizontal while flying. We limit the acceleration along x, y, z direction in our trajectory generator, so the performance of the quadrotor will be stable. Here the acceleration is limited up to $0.5m^2/s$. Meanwhile, since the X-Y plane of the map is shifted relative to the world vicon frame, a offset of (1.38, 1.28, 0) is added to vicon data.

Figure 5 is the 3D view of the quadrotor going from the center of the map to point (1.5, 1.5, 1.5). Figure shows the pose estimation compare to the vicon data.

$$\text{Error Standard Deviation of Camera Data} = \begin{cases} 0.0124 \\ 0.0071 \\ 0.0048 \\ 0.0035 \\ 0.0031 \\ 0.0016 \end{cases}$$

$$\text{Error Standard Deviation of EKF Data} = \begin{cases} 0.0123 \\ 0.0043 \\ 0.0032 \\ 0.0034 \\ 0.0022 \\ 0.0013 \end{cases},$$

where the first six components are repectively for position and orientation.

Figure 7 shows the tracking error versus time in `student_control_GoToWayPt`.

References

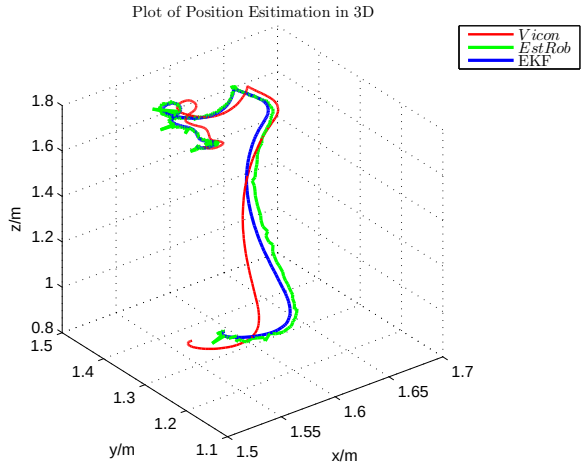

Figure 5: 3D view of GoToWaypt

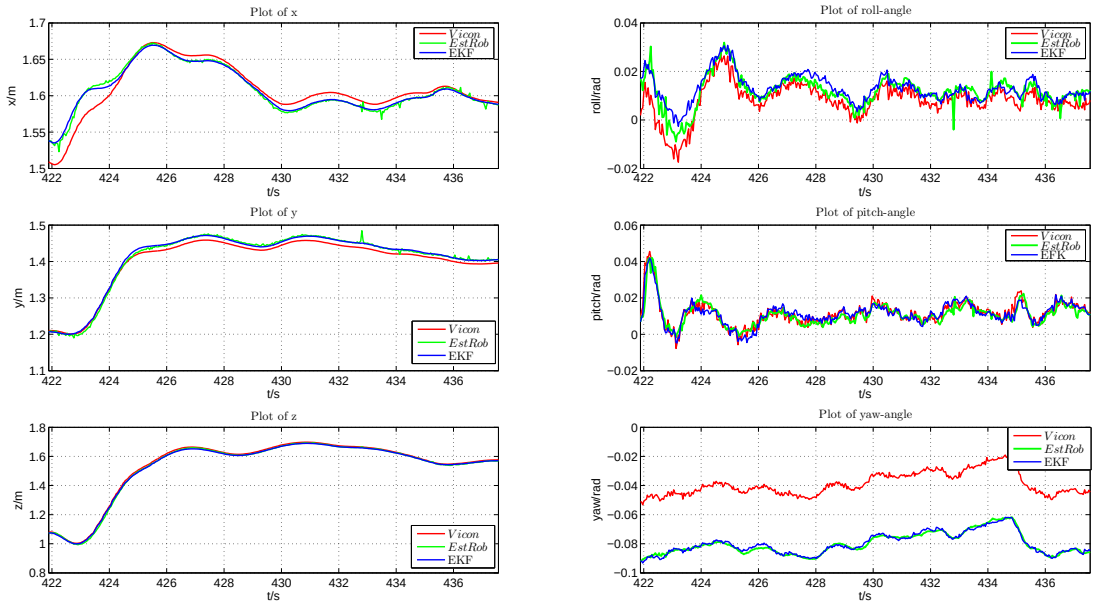

Figure 6: Results of GoToWaypt in position and orientation

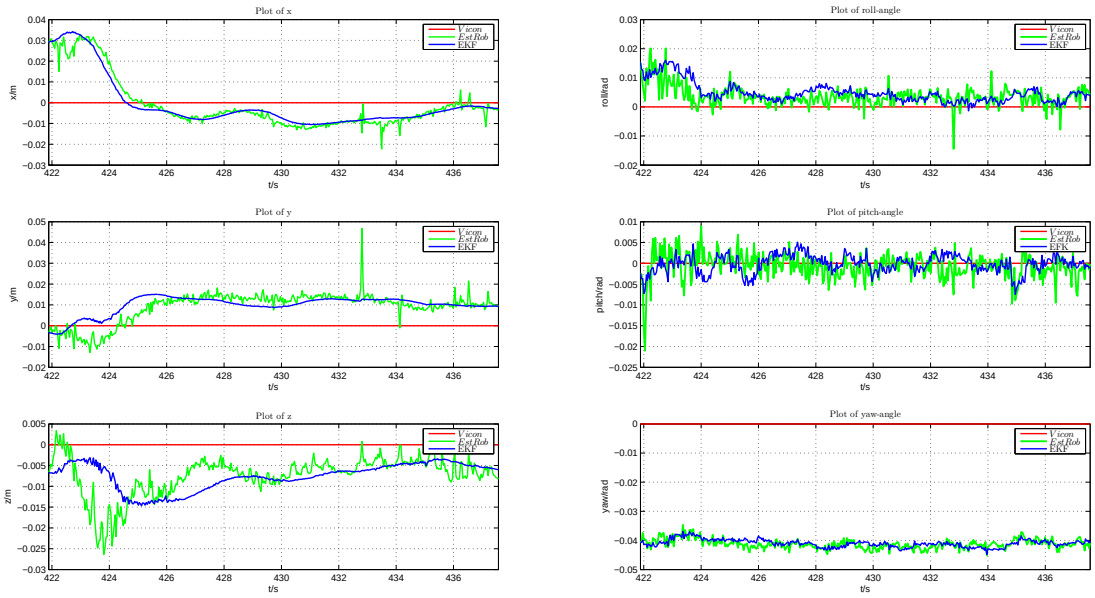

Figure 7: Error versus time(comparing to vicon)