MEAM 620, Advanced Robotics, Spring 2013: Project 1 Phase 4

Group 3: Johnathan Balloch,Jialue Huang, Qiong Wang

March 25, 2013

1 Analysis

1.1 Controller

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:

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

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

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

1.2 Trajectory Generator

In actual flying of a quadrotor, we cannot change the velocity suddenly, because the path is not smooth. Functions like sinusoids and polynomials are smooth with continuous derivative. Here, we use a quintic polynomial method as discussed below to generate trajectory.

$x(t)$ is a polynomial from $t = 0$ to $t = T$. We have six known variables here,

$$x(0) = x_0, \dot{x}(0) = \dot{x}_0, \ddot{x}(0) = \ddot{x}_0$$
$$x(T) = x_T, \dot{x}(T) = \dot{x}_T, \ddot{x}(T) = \ddot{x}_T$$

Hence, we need a representation of $x(t)$ with six unknown coefficients in order to construct an unique answer of $x(t)$. We can write as:

$$x(t) = At^5 + Bt^4 + Ct^3 + Dt^2 + Et + F = \begin{bmatrix} t^5 & t^4 & t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \end{bmatrix}$$

We also can write

$$\dot{x}(t) = 5At^4 + 4Bt^3 + 3Ct^2 + 2Dt + E$$
$$\ddot{x}(t) = 20At^3 + 12Bt^2 + 6Ct + 2D$$

Plugging in the value of $x(t), \dot{x}(t), \ddot{x}(t)$ at time $t = 0, t = T$, we can have:

$$\begin{bmatrix} x_0 \\ x_T \\ \dot{x}_0 \\ \dot{x}_T \\ \ddot{x}_0 \\ \ddot{x}_T \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ T^5 & T^4 & T^3 & T^2 & T & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 5T^4 & 4T^3 & 3T^2 & 2T & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 20T^3 & 12T^2 & 6T & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \end{bmatrix}$$

It is easy to find that the six-by-six matrix is invertible, so we can solve the six coefficients to represent $x(t)$ as a polynomial. For any time between $[0, T]$, there are always unique $x(t), \dot{x}(t), \ddot{x}(t)$, which change smoothly. And the $x(t), \dot{x}(t), \ddot{x}(t)$ are also used in the controller.

In this phase, there is no obstacle and the map is empty. Hence, we only have to put each point in `example_waypts` as the start and the end of each small fraction in the whole trajectory.

2 Codes

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 gains of 20 and derivative gains of 5 were found to be well suited for movements as docile as hover to flying at aggressive speeds between multiple

way points (see Section: Results for more on this). 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 0.220 kg, which is heavier than 0.178 kg, the mass given in phases 2 and 3.

2.1 Hover Codes

The current values of yaw, position, and velocity were pulled from the `qd` cell array, 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;

2.2 Single Waypoint Codes

In the single way point code, the execution is largely the same, but the setup is quite different. In the sequence message code an end point is specified, which is then read in to the controller code in its first iteration. This end point is then used by the trajectory generator (see Section 1.2), assuming current position in the first iteration is the starting point, to find a smooth trajectory to that location. After this, the trajectory generator is called each iteration, returning the planned desired position, velocity, acceleration, and total time for the trajectory. If the current time exceeds the planned trajectory time, the quadrotor is commanded into a hover state. Otherwise, it follows its trajectory as intended, finding the desired command accelerations as a function of the difference between the planned state and the current state:

$$acc_{des} = kp_1(r_T - r) + kd_1(\dot{r}_T - \dot{r}) + acc_T;$$

where $x_t, \dot{x}_t$, and $\ddot{x}_t$ are the planned position, velocity, and acceleration. Beyond this, the controller functions the same as the hover controller.

2.3 Multi Waypoint Codes

The multi-waypoint controller functions the same as the single way point controller with the exception of one aspect; instead of specifying just an end point in the sequence message, a series of endpoints are specified. The trajectory planner finds a smooth path between, and including, all of these points, and then the controller behaves the same as for one waypoint.

3 Results

In the following figures, the blue line represents the desired position caculated by the trajectory generator with the given path. To ensure the quadrotor stay relatively horizontal while flying. We limit the acceleration along x, y, z direction, so the performance of the quadrotor will be stable.

Figure 1 is the 3D view of example 1. Figure 2, Figure 3, and Figure 4 are the results of 3 different views for example 1. The largest acceleration for this example is $0.3m^2/s$.

Figure 5 to Figure 8 are the results from example 2 with a largest acceleration of $1m^2/s$ and Figure 9 to Figure 12 are also the results from example 2 but with a higher acceleration limitation of $3m^2/s$.

The three views of example 3 are showed in Figure 14, Figure 15 and Figure 16. The 3D view of example 3 is showed in Figure 13.

The plots give us clear views for the desired trajectory and the real path recorded by Vicon. The quadrotor follows the blue path quite well. In example 2, we tried a higher speed by giving the controller a higher acceleration. Although the real path does not match the desired trajectory that good in the faster case, it still finish the flying quite stable. So, our controller is qualified for a flying at high speed.

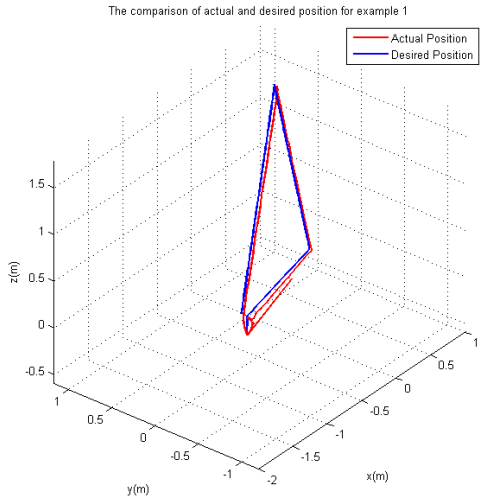

Figure 1: 3D view for example 1

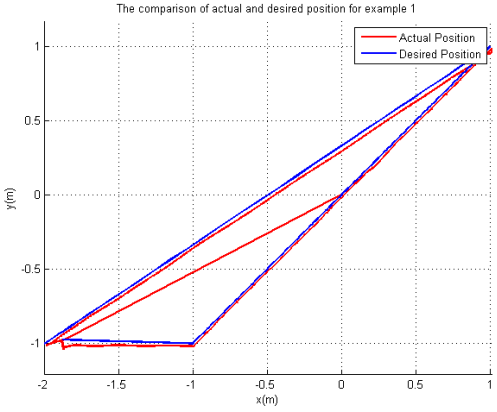

Figure 2: Results of example1 in XY plane

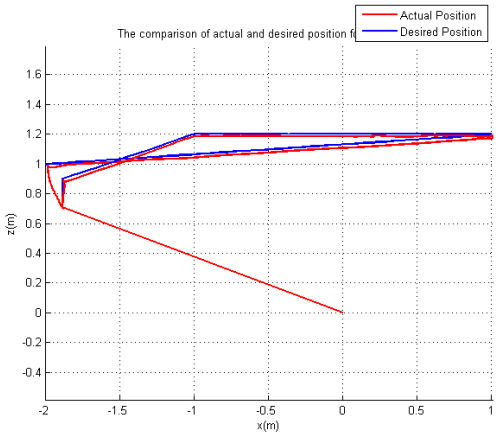

Figure 3: Results of example1 in XZ plane

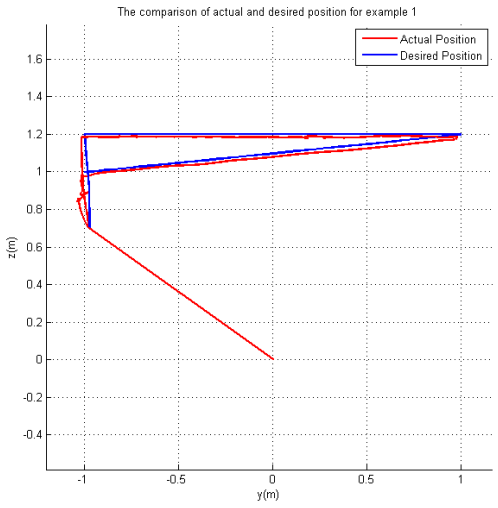

Figure 4: Results of example1 in YZ plane

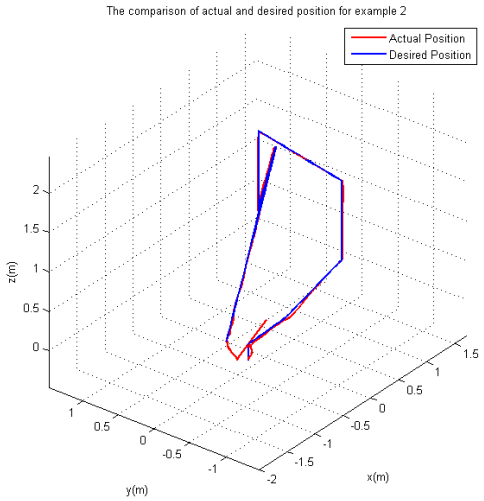

Figure 5: 3D view for example 2 with a lower acceleration

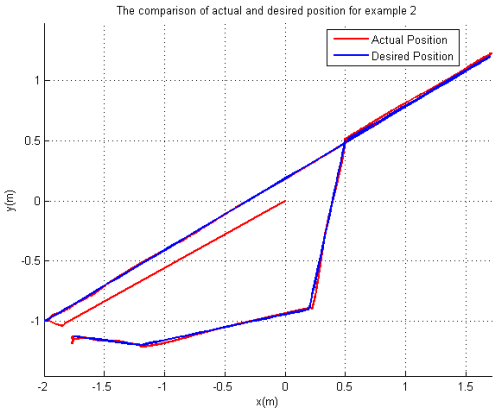

Figure 6: Results of example2 in XY plane with a lower acceleration

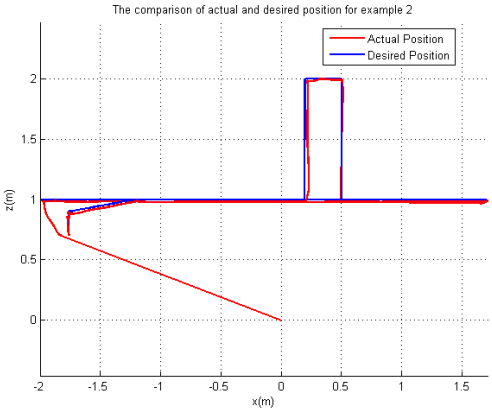

Figure 7: Results of example2 in XZ plane with a lower acceleration

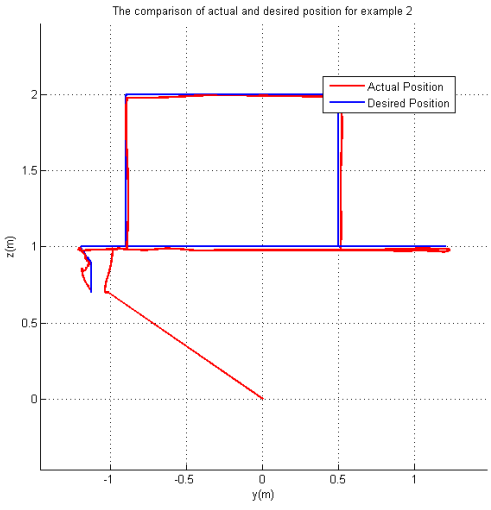

Figure 8: Results of example2 in YZ plane with a lower acceleration

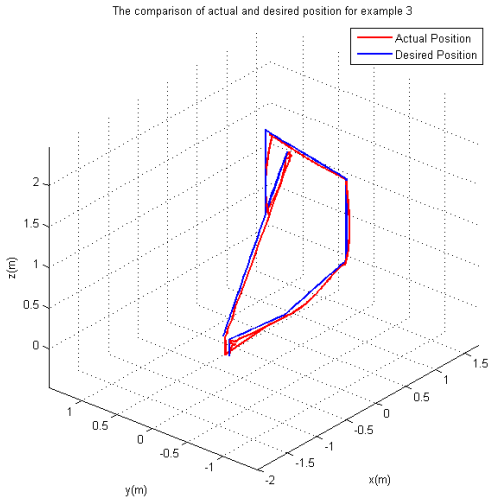

Figure 9: 3D view for example 2 with a higher acceleration

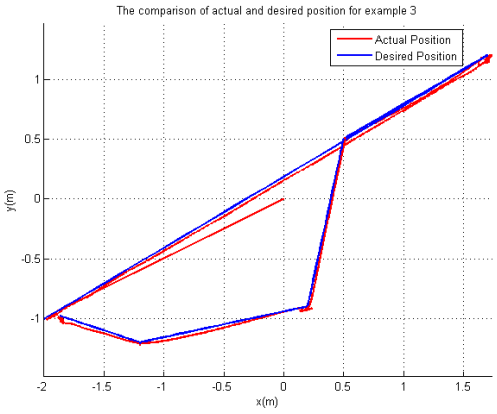

Figure 10: Results of example2 in XY plane with a higher acceleration

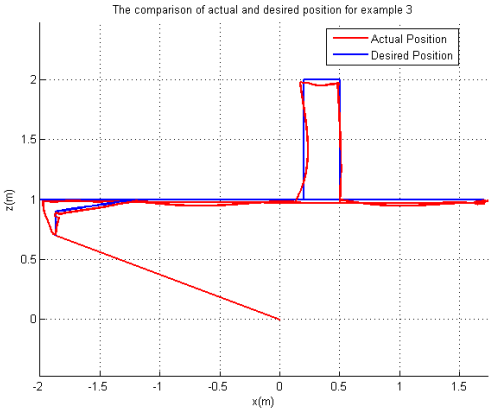

Figure 11: Results of example2 in XZ plane with a higher acceleration

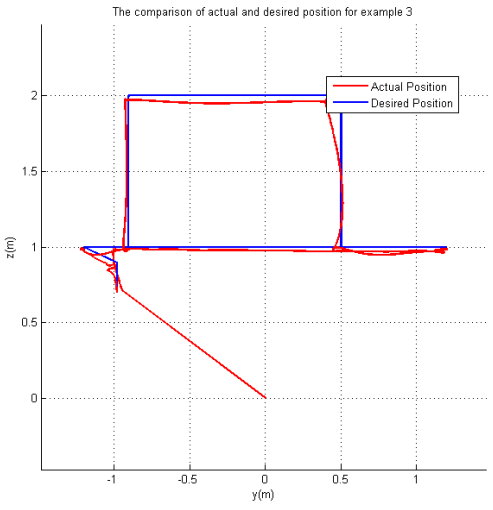

Figure 12: Results of example2 in YZ plane with a higher acceleration

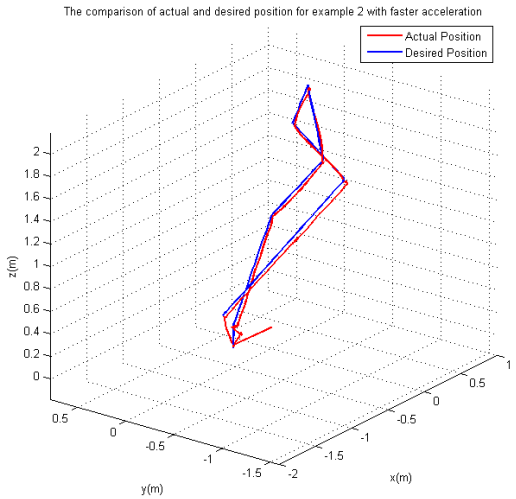

Figure 13: 3D view for example 3

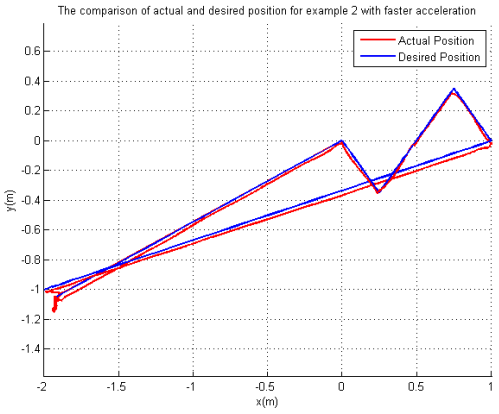

Figure 14: Results of example1 in XY plane

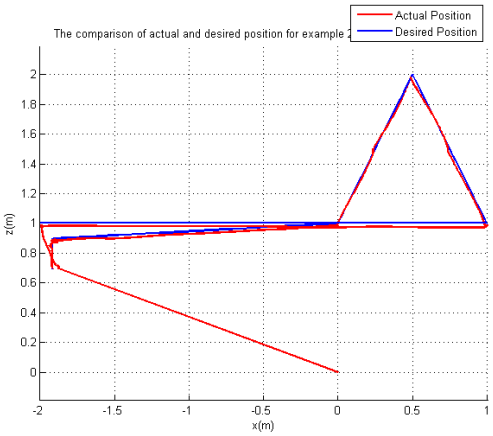

Figure 15: Results of example1 in XZ plane

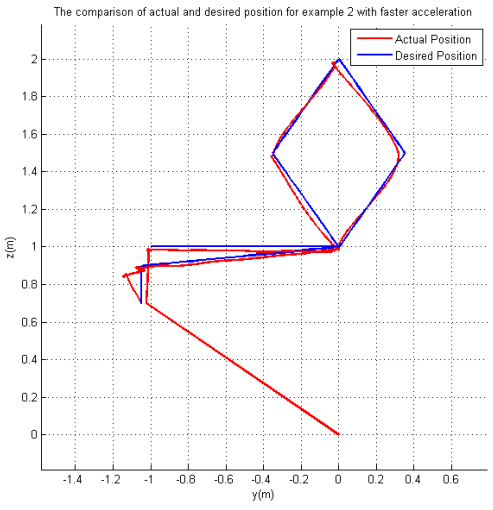

Figure 16: Results of example1 in YZ plane