Estimation and control of a simplified single plane drone model

Hari Krishna Hari Prasad
Univeristy of Washington - hkhp@uw.edu

Abstract— **We present a simple discrete time (DT) stabilizing control and estimation for a simplified single plane drone model. A Non-Linear System description is developed which encapsulates the key requirements. Following this, the system is solved for one equilibrium trajectory – hovering and one non-equilibrium trajectory – Vertical climbing. The system is then linearized and discretized for each of those scenarios. Finally, we implement an optimal controller and estimator based on minimum energy principles which work together to maintain the system on its nominal trajectory.**

***Keywords*— discrete, time, control, estimation, hovering, climbing, linearized, discretized, optimal, nominal and trajectory.**

I. Introduction

Drones can be found everywhere these days - from miniature toys to UAVs, UGVs used in humanitarian aid, emergency response, disaster relief, disease control, photography and filmmaking. Their extreme popularity stems from the exponential reduction in cost over the past decade and improvement in more user-friendly programming platforms.

This paper focuses on developing a stabilizing control and estimation algorithm for commercial application to a quadcopter. This is an interesting problem to study since it will help our consumers shoot themselves doing "*cool stuff*" with the quadcopter following a pre-planned trajectory robust to sensor (onboard camera) noise, wind (state disturbance) and other kinds of disturbances. We start off by describing a continuous time non-linear dynamical system (CNL), solve for an equilibrium trajectory – hovering, then proceed to solve for a non-equilibrium trajectory – Vertical Climbing. Then the CNL is discretized and linearized about the two solutions to obtain a Discrete Linear Time Invariant system (DLTI) for hovering and Discrete Linear Time Varying system (DLTV) for climbing. We will then proceed to incrementally obtain a controller, then an estimator and combine the two for both scenarios. These algorithms are developed based on minimum energy principles. Finally, we will be discussing their performance of the developed Discrete Time (DT) controller and estimator from the "*enlightening*" context of their application domain.

II. Model

A. *Continuous Time Non-Linear ODE Control system*

The CNL describes the horizontal, vertical, angular positions and velocities of the quadcopter by solving for the corresponding accelerations. F_a and F_b are the thrusts generated by the Left and Right actuators. θ is the angular position, h is the horizontal position and v is the vertical position of the drone. $\dot{h}$ and $\ddot{h}$ denote the velocity and acceleration of the corresponding state. Hence, we have a total of six states and two inputs. Let us have look at the governing CNL equations:

$$m\ddot{h} = (F_a + F_b)Sin\theta - \beta\dot{h} \tag{1}$$

$$m\ddot{v} = -mg + (F_a + F_b)Cos\theta - \beta\dot{v} \tag{2}$$

$$I\ddot{\theta} = F_b - F_a \tag{3}$$

The equations seem simple enough. The Sine component of the total thrust generates horizontal acceleration whereas the Cosine component generates vertical acceleration. This also tells us that the angular position is measured with respect to the vertical axis and uses a counter clockwise positive convention. Both the acceleration terms have linear constant time damping β with respect to their velocities. Also, the angular acceleration of the drone is described by the difference between the thrusts. Note that there is no angular damping term involved. A single moment of inertia formulation about the center of mass (COM) of the system is used to describe the angular acceleration of the drone.

Hence, this is a simplified model which describes the drone as a point mass and with linear damping terms for spatial acceleration but not for spatial orientation. It also has no knowledge of the fluid flow around the system. This is good enough for our external environment trajectory tracking application where translation of drone is going to happen in a controlled manner along a pre-planned trajectory. The unmodeled non-linear effects aren't required for the basic control and estimator developed and implemented in our application domain.

B. *Schematic of the Model*

The figure shown below summarizes the Model discussion had in the previous section:

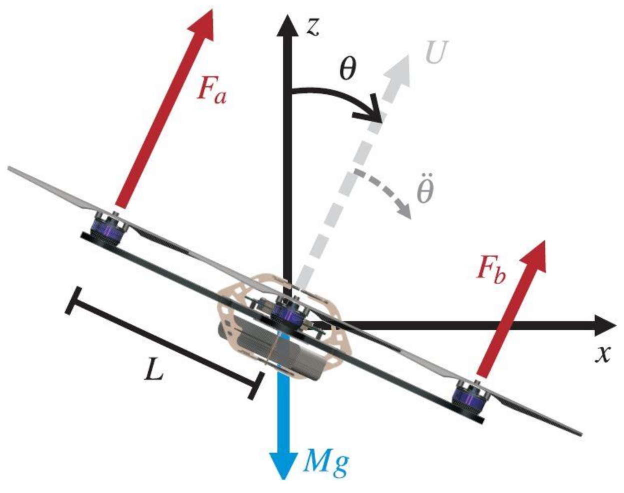

Figure 1. Shows the drone in a specific spatial orientation and acceleration. It also describes the actuator thrusts, and the plane axes.

The figure also illustrates the position of the actuators from the COM. This information is not relevant from our modelling, control and estimation point of view for the CNL at hand.

III. ANALYSIS

Our next step involves finding an equilibrium trajectory of this system. We would love to balance first before moving. Naturally, the first thing that comes to mind is hovering which is the most important stage after developing a mathematical model. Before we proceed, let us use some known parameters to simplify the CNL:

$$\ddot{h} = (F_a + F_b)Sin\theta - 0.1\dot{h} \tag{4}$$

$$\ddot{v} = -g + (F_a + F_b)Cos\theta - 0.1\dot{v} \tag{5}$$

$$\ddot{\theta} = 100(F_b - F_a) \tag{6}$$

We want our numerical simulation and system parameters to be as close to the real system as possible. Hence, using the parameters reported to us by the mechanical design team, we have the above simplified CNL equation.

Mass of the drone – *m* = 1Kg

Moment of Inertia about COM – *I* = 0.01Kgm^2

Linear Damping term – β = 0.1Kg/s

Ignoring uncertainties in actuator thrusts and other real world non-linearities, we can see that for a drone to hover, both actuator thrusts have to be equal and must balance the weight of the system or acceleration due to gravity *g*. Hence, each thrust has to be *g*/2. Also, θ has to be zero making the drone vertically upright.

A. Linearization of CNL about Hovering Equilibrium

In order to develop and work with a DT controller and estimator we need to discretize this CNL system and Linearize it about the Hovering equilibrium. Linearization is a first order approximation of the system dynamics around the equilibrium/fixed point. It will help us analyse the system in that neighbourhood.

The first step involves taking first order partial derivatives of each equation (4), (5) and (6) with respect to our states and inputs. After doing this we obtain the following generalization for each of those states:

Jacobian of Equation $\ddot{\theta}$:

$$\frac{\partial\ddot{\theta}}{\partial F_b} = 100; \frac{\partial\ddot{\theta}}{\partial F_a} = 100;$$

Jacobian of Equation $\ddot{h}$:

$$\frac{\partial\ddot{h}}{\partial\theta} = (F_a + F_b)Cos\theta; \frac{\partial\ddot{h}}{\partial\dot{h}} = -0.1;$$

$$\frac{\partial\ddot{h}}{\partial F_b} = Sin\theta; \frac{\partial\ddot{h}}{\partial F_a} = Sin\theta;$$

Jacobian of Equation $\ddot{v}$:

$$\frac{\partial\ddot{v}}{\partial\theta} = -(F_a + F_b)Sin\theta; \frac{\partial\ddot{v}}{\partial\dot{v}} = -0.1;$$

$$\frac{\partial\ddot{v}}{\partial F_b} = Cos\theta; \frac{\partial\ddot{v}}{\partial F_a} = Cos\theta;$$

B. State-Space for the Linearization

A general state-space representation approach is used to represent our Linearized system. If we have a column vector of states *X* and column vector of inputs *U*, we can represent the time derivative of our states $\ddot{X}$ in the following form:

$$\ddot{X} = AX + BU$$

This is the input equation where the *A* is the system matrix which contains the dynamical relationships between states and their derivatives. *B* is the input matrix which tells how U affects the state derivatives.

We have a similar output equation for that, we have to linearize the sensor information. The camera onboard the drone gives us information in the form of Sines and Cosines of the elevation angle. Our outputs are *y1* and *y2*:

$$y1 = \frac{h}{(v^2 + h^2)^{\frac{1}{2}}}; \; y2 = \frac{v}{(v^2 + h^2)^{\frac{1}{2}}}$$

Jacobian for Output 1:

$$\frac{\partial y1}{\partial h} = \frac{v^2}{(v^2 + h^2)^{\frac{3}{2}}}; \frac{\partial y1}{\partial v} = \frac{-hv}{(v^2 + h^2)^{\frac{3}{2}}}$$

Jacobian for Output 2:

$$\frac{\partial y2}{\partial h} = \frac{-hv}{(v^2 + h^2)^{\frac{3}{2}}}; \frac{\partial y2}{\partial v} = \frac{h^2}{(v^2 + h^2)^{\frac{3}{2}}}$$

State Space for the output side is given by:

$$Y = CX + DU$$

In our case, there is no feedforward matrix and hence *D* = 0. Here, *C* is the Sensor/Output matrix which linearly operates on *X* to give *Y.*

$$X = (\theta \quad h \quad v \quad \dot{\theta} \quad \dot{h} \quad \dot{v})'$$

$$\ddot{X} = (\dot{\theta} \quad \dot{h} \quad \dot{v} \quad \ddot{\theta} \quad \ddot{h} \quad \ddot{v})'$$

In the above equations we have our State and State derivative vector.

Our inputs also contain the acceleration due to gravity as a third uncontrolled input *g*. This directly affects our vertical acceleration, but isn't a function of any of our system variables:

$$U = (F_a \quad F_b \quad g)'$$

Also, we have to substitute $\theta = 0$ to obtain the CLTI.

Hence, the general state space representation for the CLTV for constant $\theta = 0$ is given by:

$$A_{CLTV} = \begin{matrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ F_a + F_b & 0 & 0 & 0 & -0.1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.1 \end{matrix} \tag{7}$$

$$B = \begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 100 & -100 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & -1 \end{matrix} \tag{8}$$

We can see that our B matrix is constant irrespective of our applied inputs. Finally, Our C matrix is of the form:

$$C_{CLTV} = \begin{matrix} 0 & \dfrac{v^2}{(v^2+h^2)^{\frac{3}{2}}} & \dfrac{-hv}{(v^2+h^2)^{\frac{3}{2}}} & 0 & 0 & 0 \\ 0 & \dfrac{-hv}{(v^2+h^2)^{\frac{3}{2}}} & \dfrac{h^2}{(v^2+h^2)^{\frac{3}{2}}} & 0 & 0 & 0 \end{matrix} \tag{9}$$

Let us say the drone is hovering at around (10,5) in the x,z coordinate system. Also, we know our thrusts for hovering to be $g/2$ each. Therefore, we compute our CLTI A and C matrices:

$$A_{CLTI} = \begin{matrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ g & 0 & 0 & 0 & -0.1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.1 \end{matrix} \tag{10}$$

$$C_{CLTI} = \begin{matrix} 0 & 0.0179 & -0.0358 & 0 & 0 & 0 \\ 0 & -0.0358 & 0.0716 & 0 & 0 & 0 \end{matrix} \tag{11}$$

C. *CLTV Trajectories*

Next, we want to identify a non-equilibrium trajectory which is relevant to our "*amazing"* application domain.

The users will need to change the distance to the camera depending purely on their whim/requirements. Hence, if we have a trajectory which disturbs the drone from hovering at a specific position and takes it to a different vertical position and back to hovering, it would be ideal.

There are multiple ways to do this. If we want the lowest or the most fuel-efficient way, then Parametric Optimization would be an option.

Here, we have used a different approach. From the problem specification above we have the following boundary conditions. S denotes the state at the current time step and t denotes the time-horizon/final time:

$$\theta_s = \dot{\theta}_s = \ddot{\theta}_s = 0 \quad \forall \quad 0 \leq s \leq t \tag{12}$$

$$h_s = \dot{h}_s = \ddot{h}_s = 0 \quad \forall \quad 0 \leq s \leq t \tag{13}$$

Ideally, nothing should change our drone's orientation and horizontal position characteristics. Then we move on the actual boundary conditions on the vertical position, velocity and acceleration. Let us say for the sake of simplicity we start from rest on ground:

$$v_0 = \dot{v}_0 = \ddot{v}_0 = 0 \tag{14}$$

$$v_t = 10;\ \dot{v}_t = \ddot{v}_t = 0 \tag{15}$$

Since we have six boundary conditions, we choose to approximate the vertical position with a 5th order polynomial in time *s:*

$$v_s = a_0 + a_1 s + a_2 s^2 + a_3 s^3 + a_4 s^4 + a_5 s^5 \tag{16}$$

Plugging in our boundary conditions and choosing our time horizon $t = 5$ seconds, final vertical position $v_t = 10$m, we have our vertical states and inputs as function of time s:

$$v_s = 0.8s^3 - 0.24s^4 + 0.0192s^5 \tag{17}$$

$$\dot{v}_s = 2.4s^2 - 0.96s^3 + 0.096s^4 \tag{18}$$

$$\ddot{v}_s = 4.8s - 2.88s^2 + 0.384s^3 \tag{19}$$

$$F_a = F_b = \frac{g}{2} + 2.4s - 1.32s^2 + 0.144s^3 + 0.0048s^4 \tag{20}$$

Just to summarize, the continuous time input description will take our drone from rest to hovering condition 10m above the starting point.

The plot shown below will clearly demonstrate this non-equilibrium trajectory:

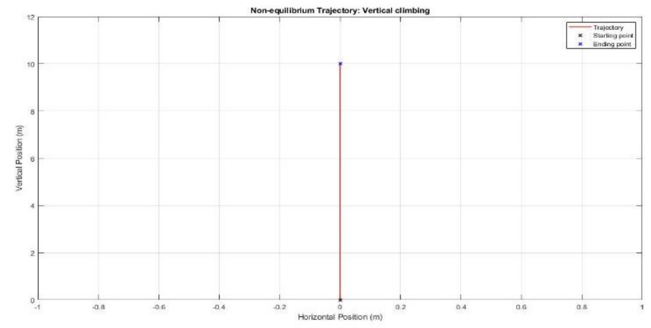

Figure 2. Shows the trajectory taken by the drone from rest at (0,0) to hovering at (0,10).

Finally, Input and vertical acceleration variation with time s is given by:

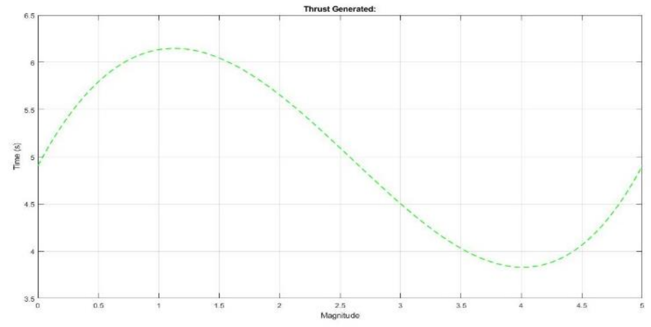

Figure 3. Shows the variation of input thrusts with time which is approximated by a 4th order polynomial function in time *s*.

IV. COMPUTATION

This section addresses the task of designing an optimal least energy-based controller and estimator. We start off by choosing a time horizon *t* = 5 seconds with step size of 0.01 seconds. This will give us a total of 501 states including the initial state. Looking at it from the frequency point of view, we can say that the controller and estimator runs at 100Hz.

A. *Linear Quadratic Regulator (LQR)*

We will start off by specifying a Quadratic cost function which penalizes deviation of state more than input energy. It is of the following form:

$$J_\tau(X,U) = \frac{1}{2}X_t^T P_t X_t + \frac{1}{2}\sum_{s=\tau}^{t-1} X_s^T Q_s X_s + U_s^T R_s U_s \tag{21}$$

Here, P_t is the cost associated with the final state. Q_s is the cost associated with each state. The subscript *s* denotes that these cost functions can be time varying. This means that we are free to change our control strategy at any given time, but for our case we keep constant running and terminal costs. The matrices *P*, *Q* and *R* are positive definite and having the quadratic structure makes the cost function convex in nature. Hence, we can develop these nice Quadratic cost-based control algorithms which find the local minima optimal inputs of the cost landscape thereby saving fuel. We then proceed to use Bellman's principle to obtain an optimal control policy. The Optimal cost U_s^* and Optimal cost J_s^* at time s, is given by:

$$U_s^* = -K_s X_s \tag{22}$$

$$J_s^* = \frac{1}{2}X_s^T P_s X_s \tag{23}$$

The State-feedback matrix K_s and Optimal input coefficient matrix P_s is found out recursively by applying Bellman's approach to the last step in the time horizon. They are given by:

$$K_s = (B_s^T P_{s+1} B_s + R_s)^{-1} B_s^T P_{s+1} A_s \tag{24}$$

$$P_s = (A_s - B_s K_s)^T P_{s+1}(A_s - B_s K_s) + K_s^T R_s K_s + Q_s \tag{25}$$

It is crucial to note that our third input is acceleration due to gravity which affects our vertical acceleration. Hence, we can't really include it in our LQR computation.

Therefore, we have only considered the 6x2 block of the 6x3 *B* matrix and the cost in *R* matrix corresponding to the third input will be zero. By doing this, we obtain a 2x6 state feedback law matrix K_s which gives us our two optimal thrust values. As usual the third input is preset at *g* and remains undisturbed. Let us verify our control law by simulating the DLTI system at an initial condition away from the origin.

The simulation for the linearization is run for 5 seconds at 100Hz. We obtain the following results:

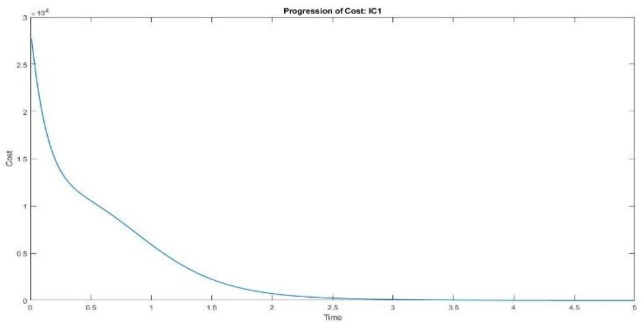

Figure 5. Shows decaying cost with time *s*.

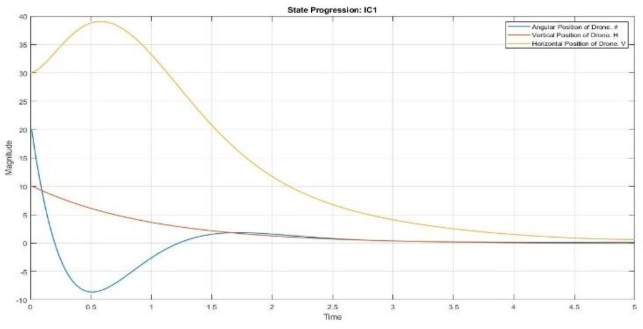

Figure 6. Shows the decay of states with time. The controller drives the linearized system dynamics to the origin.

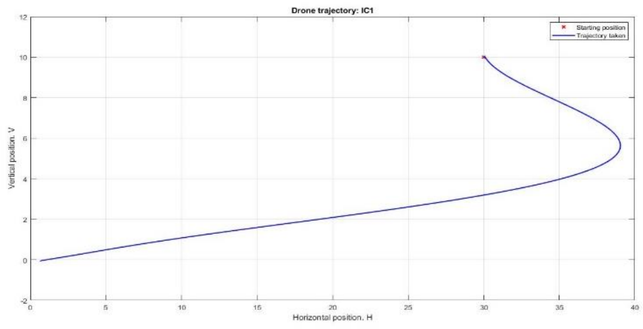

Figure 7. Shows the trajectory of the Linearized DLTI system being taken to the origin – Stabilizing control of LQR.

From the above plots the stabilizing nature of LQR is pretty clear. It drives the Linearized dynamics to zero. This effectively means that, if the drone is off the nominal trajectory the stabilization controller brings it right back on track.

There is one very important catch though. The controller results show above assumes it knows all the states of the system. This isn't realistically possible since the camera only gives us Sines and Cosines of the elevation angle. Hence, the Estimator tries to minimize "*estimation energy*" assuming it knows something about the *nature* of the sensor error and state disturbance.

B. Steepest Descent

We now take a short detour to briefly discuss and compare the performance of steepest-descent-based controller to the controller developed in the previous section.

For this section we've taken a very short time horizon of 0.02 seconds which amounts to two timesteps. Hence, we have two randomized inputs based on which our terminal cost at *t* = 0.02 is evaluated. Then we perturb initial slightly and rerun the simulation to obtain the terminal cost. This way, we can get an idea as to effect of the input 1 on the terminal cost.

The gradient computed this way can be used to modify our initial input and this can be thought of as a single descent in the cost landscape. Realistically speaking, we would have to do multiple descends until we hit the local minimum for a particular descent. This is really tedious in that context. But this method is robust in a way that it can be applied to any complicated non-linear system and it would work. Some of the results are shown below:

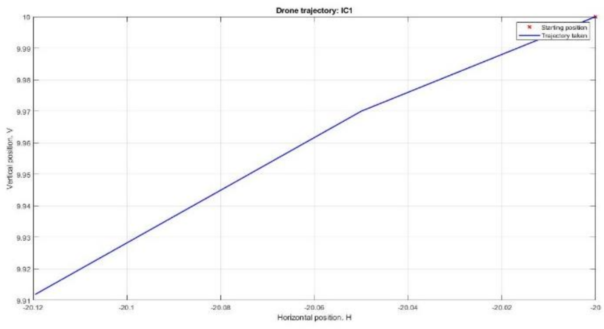

Figure 8. Shows the decay of states with time. The LQR based controller drives the states towards origin for two timesteps.

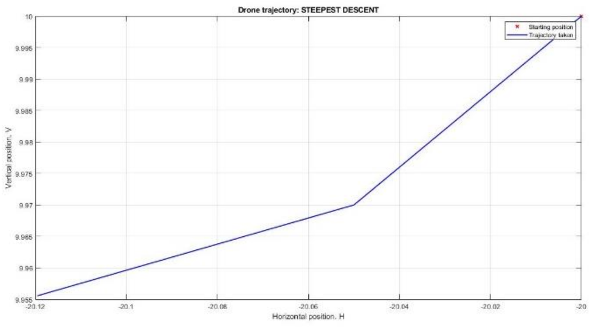

Figure 9. Shows the decay of states with time. The SD based controller drives the states towards origin for two timesteps.

We can see that both the controllers in the given time frame, try to accomplish the same goal – which is come up with optimizing inputs to reach the origin by minimizing input energy. By construction, we know that the inputs developed by the LQR is optimal as each u^* is a local minimum. Even after bunch of descends, since there is some difference between the trajectories generated by SD and LQR, it is safe to say that the SD is giving us sub-optimal results and isn't feasible from the context of our application. It also takes way more time to compute compared to the DT-LQR implemented in the previous section.

C. Linear Quadratic Estimator (LQE)

There are a few important assumptions which should be discussed initially. The sensor/output noise along with the external disturbance to the state at each timestep *s* is assumed to be Gaussian and white. Meaning, there is no correlation between the noise and disturbance, and noises between two different time steps. Hence, these noises and disturbances have zero mean and diagonal covariance matrices for now. The MATLAB command *mvnrnd* is used to generate such signals.

In the numerical computation, the noise and state disturbance are computed at every timestep since we have knowledge of the Covariance of these two quantities ahead of time. This is then run through the dynamics of a Deterministic system:

$$X_{s+1} = A_s X_s + B_s U_s + \delta_s \; ; \; E[\delta_s] = 0; \; Cov[\delta_s] = \bar{R}_s^{-1} \tag{26}$$

$$Y_s = C_s X_s + \eta_s; \; E[\eta_s] = 0; \; Cov[\eta_s] = \bar{Q}_s^{-1} \tag{27}$$

Here, $\bar{R}_s^{-1}$ and $\bar{Q}_s^{-1}$ are the covariance matrices of the state disturbance δ_s and observation error η_s which are Gaussian multivariable random vectors with zero mean and zero correlation in this case (White!!). Essentially, we are trying to minimize the estimation energy which has the following form:

$$Estimation\ Error = \sum_{s=0}^{t} \eta_s^T \bar{Q}_s \eta_s + \delta_s^T \bar{R}_s \delta_s \tag{28}$$

The Estimator dynamics which runs in parallel to the deterministic systems tries to estimate the state based on the *noisy* observation. A prediction of the state at the current timestep $\tilde{X}_s$ and the expected observation $\tilde{Y}_s$. This is then compared to the observation obtained through the Kalman Gain L_s and the state prediction is updated to a state estimate $\widehat{X_s}$. In the simulation domain, this estimate is going to be close to the actual state. It is *important* to note that there is ***no*** stabilizing control at work.

Finally, we can have a look at the estimator dynamics to see how our Estimate is being computed:

State and Observation prediction:

$$\tilde{X}_s = A_{s-1}\widehat{X_{s-1}} + B_{s-1}U_{s-1} \tag{29}$$

$$\hat{Y}_s = C_s \tilde{X}_s \tag{30}$$

Computation of the Kalman Gain L_s:

$$\widetilde{\Theta}_s = A_{s-1}\Theta_{s-1}A_{s-1}^T + \bar{R}_s \tag{31}$$

$$L_s = \widetilde{\Theta}_s C_s^T \left(\bar{Q}_s + C_s \widetilde{\Theta}_s C_s^T\right)^{-1} \tag{32}$$

$$\Theta_s = (I - L_s C_s)\widetilde{\Theta}_s (I - L_s C_s)^T + L_s \bar{Q}_s L_s^T \tag{33}$$

The above equation first calculates the prediction of the state covariance $\widetilde{\Theta}_s$ based on the state covariance Θ_{s-1} from the previous step. Then the Kalman gain L_s is computed which is followed by updating the state covariance prediction to estimate Θ_s.

Now, let us look at some of the results obtained by running the LQE Estimator at 100Hz:

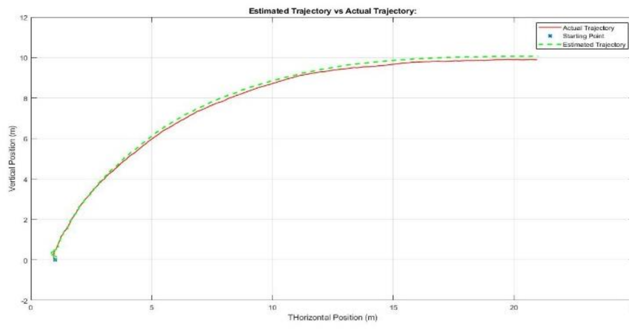

Figure 10. Shows the deviated trajectory taken by the drone. The Estimator does really well with an approximated Horizontal and vertical position close to the *actual* position of the drone.

From the results, we can see that since there is no stabilizing control, the horizontal position of the drone is deviates far away from the starting coordinate. The Estimation obtained has an error of about 10-20cms which is well within the bounds of our application.

D. *LQR and LQE:*

Finally, we have reached stage where we can implement the Estimator and Controller together on our DLTI system. The key difference is that in this case we actually don't know our actual states and have access only to the state estimates obtained from the Estimator. Hence, the modified feedback control is of the form:

$$U_s^* = -K_s\widehat{X_s} \tag{34}$$

In the above equation, $\widehat{X_s}$ is the estimate obtained from the LQE/ Kalman filter. One thing we can be sure of is that the performance isn't going to be as great as the ones obtained in the individual cases as seen in the previous sections. Both the states and observations are Gaussian random variables with increasing Variance in time *s* since the Feedback control is based on the state estimate which is prone to *disturbance* and observation which is *noisy*. Hence, our estimate and control might get worse with time. The estimator and controller run at 100Hz. Let us look at the results obtained:

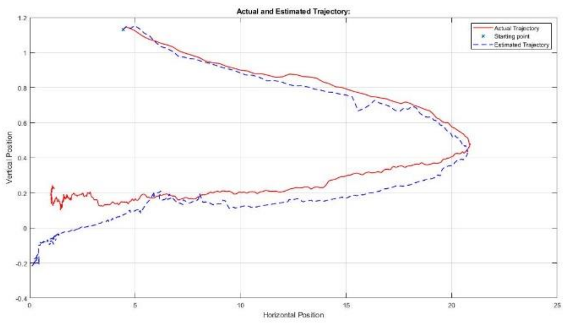

Figure 11. Shows the stabilizing trajectory taken by the drone – Controller + Estimator frequency - 100Hz

We can see that estimator's performance is slightly worse compared to the standalone case. There is about 40cms of error margin by the very end. But looking at the stabilizing trajectory there is lot more disturbance closer to the origin. Increasing the estimator and controller frequency to 1000Hz, and reducing the magnitude of the state and observation disturbance/noise we have:

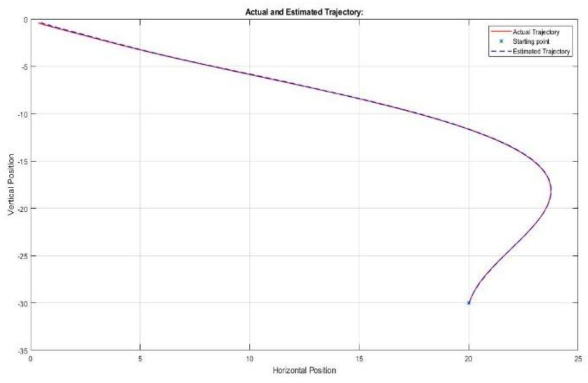

Figure 12. Shows the stabilizing trajectory taken by the drone – Controller + Estimator frequency - 1000Hz

We can see that there is much less disturbance in the trajectory with the control and estimation implemented being near perfect (with miniscule error of the order of a few centimetres). This is an extremely ideal case where we feed the exact starting point to the initial state estimation. This result is just shown for illustration of what happens when we increase the control and estimation frequency and initialize our estimator the actual state.

V. Application

This section deals with the application of our controller and estimator algorithm to stabilizing the system about a nominal trajectory which we solved of in the initial sections – Hovering (equilibrium trajectory) and Vertical Climbing (Non-equilibrium trajectory).

A. *Hovering Application:*

The nominal inputs for the Hovering are of the form g/2 each. We run the nominal trajectory and the DLTI system in parallel to approximate the behaviour of the CNL. At the end of the time horizon, behaviour of the CNL system can be approximated as follows:

$$X_{CNL} = X_{nom} + X_{DLTI} \tag{35}$$

$$U_{CNL} = U_{nom} + U_{DLTI} \tag{36}$$

Hence, we will now be trying to stabilize our drone prone to state disturbances and observation noise at every timestep *s* for Hovering equilibrium. For a time-horizon of *t* = 5 seconds and controller-estimator frequency of 100Hz, we obtained the following results:

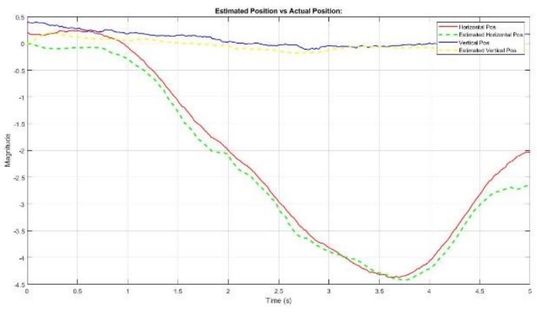

Figure 13. Shows the estimated states vs the actual states.

The above plot shows us the convergence of the estimated state value to the actual state within one or two time-steps.

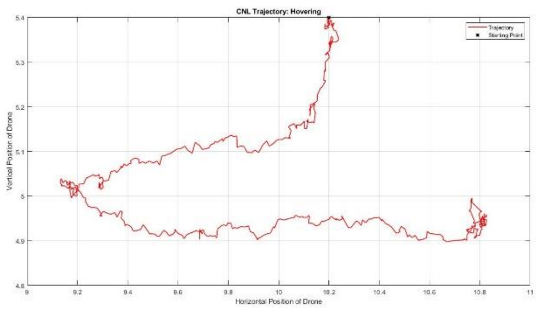

Figure 14. Shows the CNL trajectory taken by the drone. Frequency – 100Hz.

The system experiences continuous disturbance to all its states which is unrealistic. But the system still manages to stay within 50cms of the nominal vertical position and a metre of nominal horizontal position. Since, the covariances are randomized each trial run gives a different result, but all computed results share error bounds not greater than a metre in spatial position. Increasing the frequency of controller-estimator to 1000Hz led to better results:

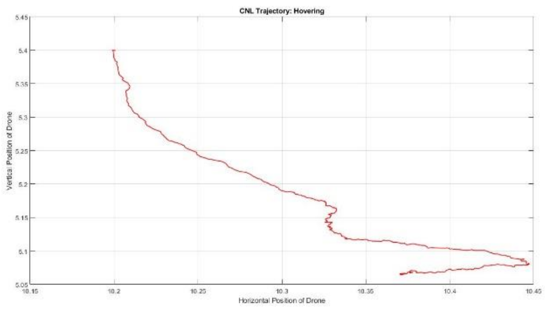

Figure 15. Shows the CNL trajectory taken by the drone. Frequency – 1000Hz.

We can see that the deviation is way less. It is hard to derive concrete results from these runs. The error bounds remain as mentioned above. The next section involves making the state disturbance more realistic to the application domain. There is always some amount of observation noise in every timestep *s.* External disturbance isn't that frequent.

B. *Hovering application: Non-uniform state disturbance:*

Different kinds of state disturbances were tried. From the application point of view, wind disturbance usually offsets the angular position and spatial accelerations. They might not have a direct effect on angular and spatial position at the same timestep.

Sparse state disturbances didn't work with the recursive LQE algorithm with the Kalman gain being undefined after a certain timestep. The Covariance matrix always has to be positive definite. Hence, most of the trials on implementing more realistic state disturbances ended not working with our recursive LQE algorithm. Finally, non-zero mean state-disturbance was implemented. This can be assumed as a prolonged gust of wind affecting all the states in a similar manner:

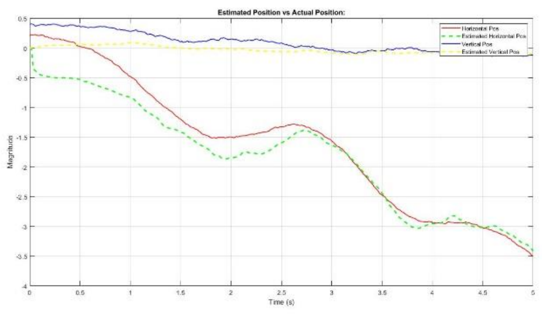

Figure 16. Shows estimated states vs actual states – non-zero mean state disturbance.

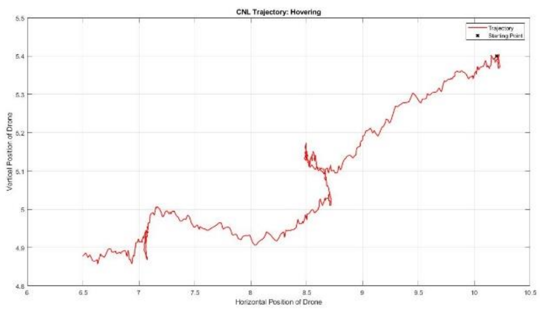

Figure 17. Shows the CNL trajectory taken by the drone. Frequency – 100Hz.

The above CNL trajectory plot shows that the system remains within 50cms of the nominal vertical height, but with a constant wind, its horizontal position is offset by 3-4 metres. It is redundant to check with sparse state disturbance since we know that the LQR stabilizes the offset in a finite number of steps.

C. *Vertical climbing:*

As a bonus we tried the same estimator-controller combo on our non-equilibrium trajectory. The system doesn't follow the nominal trajectory. This is because the LQR has barely had time to compensate for state-disturbances coming in at every timestep. But the drone reached the nominal vertical position by the end of the time horizon:

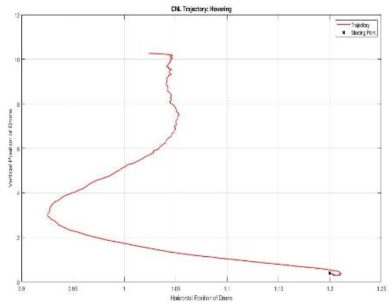

Figure 16. Shows the CNL trajectory taken by the drone. Frequency – 100Hz. Vertical climbing.

The continuous state disturbance offsets the system from starting at (1,0). By the end of the time-horizon the drone very close to the nominal spatial position (10,1) with negligible velocities and acceleration as required to resume hovering operation.

VI. Discussion

The results obtained in the numerical simulation have been encouraging if not conclusive. Some of the important observations include:

- The standalone operation of the controller and estimator is close to ideal and can never be achieved on a real system.
- The LQR takes a finite number of steps to stabilize the offset from the nominal trajectory. State disturbance to all states at every timestep isn't stabilizable and realistic.
- The DLTI approximation for the hovering approximation seems to be inaccurate. If the spatial position of the drone drifts from the nominal position our State-space description has to be updated to form a DLTV system. This will probably give is a closer approximation of the actual dynamic process.
- From the DLTI approximation, and simulations run we can see that the horizontal states are only indirectly controlled through the angular position at timestep *s* for the trajectories developed in our analysis. More trajectories which involve horizontal translation need to be explored.

The above steps highlight the important observations and conclusions made from the two trajectories probed. Some of the current work involve:

- Tuning the LQR to converge to the nominal position within a certain number of steps.
- Probing better representations for more realistic state disturbance formulation. A starting point would be to set the state disturbance to occur periodically in a certain number of steps and then tune the operational sensitivity of the LQR to get a better feel for the application we are dealing with.
- Not really able to comment on the requirement of a more aggressive controller/estimator. Even then, conclusive remarks can be made only after testing them out on actual hardware.

VII. Bibliography

[1] S. Lupashin, A. Schöllig, M. Sherback and R. D'Andrea, "A simple learning strategy for high-speed quadrocopter multi-flips - IEEE Conference Publication", *Ieeexplore.ieee.org*, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/5509452. [Accessed: 22- Mar- 2019].