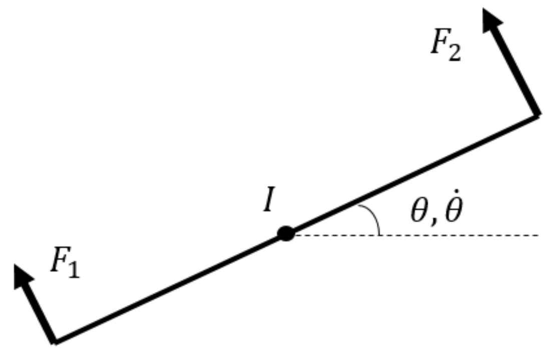
Mark Paluta

AA 290: Quadrotor simulation within PILCO package

This report outlines the quadrotor scenario implementation within the PILCO policy learning software package in MATLAB. It also describes two preliminary incremental rotor scenarios. The code will be made public sometime in the near future.

1. Attitude stabilization in twinrotor

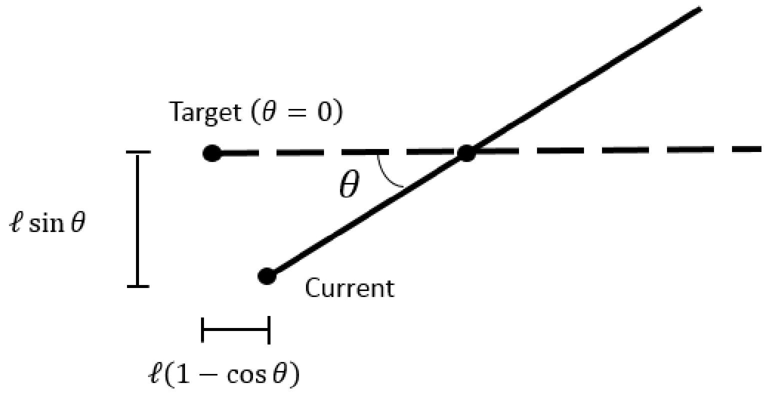
The first scenario developed can be found within the code in a folder named "twinrotor1". This scenario tests the attitude stabilization of a 2 dimensional twinrotor operating in a plane. Since the rotor is modeled as symmetric, gravity will not affect attitude. No position or velocity (x, z, u, w) information is recorded in this simulation. This scenario served to test out the capability of the software to accommodate user-developed systems. This scenario used 2 actuators, representing the propellers. The propellers were modeled as in-plane forces perpendicular to the rotor axis as pictured.



The state describing this system is $[\theta\ \dot{\theta}]$ and in some parts of the algorithm is augmented to $[\theta\ \dot{\theta}\ \ \sin\theta \cos\theta]$ to take advantage of the complex representation of θ. The dynamics of the system are

$$\begin{bmatrix}\theta \\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}\dot{\theta} \\ \dfrac{F_2 - F_1 - b * \dot{\theta}}{I}\end{bmatrix}$$

where b is a damping term. The cost function in many of the predefined scenarios is based on the squared distance between the current position, x, and the target position, x_t.



The distance squared can be expressed in a quadratic form with the following procedure.

$$\begin{aligned} x &= \left[\theta\ \dot{\theta}\ \sin\theta\ \cos\theta\right]^T \\ x_t &= [0\ 0\quad 0\qquad 1\ \ ]^T \end{aligned}$$

$$\begin{aligned} d^2 = \|x - x_t\| &= \ell^2(\sin^2\theta + (1-\cos\theta)^2) \\ d^2 &= \ell^2\sin^2\theta + \ell^2(\cos\theta - 1)^2 \\ d^2 &= (x-x_t)^T Q (x-x_t) \end{aligned}$$

$$\text{where } Q = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \ell^2 & 0 \\ 0 & 0 & 0 & \ell^2 \end{bmatrix}$$

However, a quadratic cost function based on the Euclidean norm is not generalizable to all conceivable scenarios, and even in cases where it can be formed, it may not be easily determinable by inspection as above. As an example, the quadrotor scenario in this report will not record inertial frame position, only body frame states, and it will attribute cost to yaw rate but will not record state data for yaw. This cost function does not allow for attributing cost to a rate without using its respective integrated state.

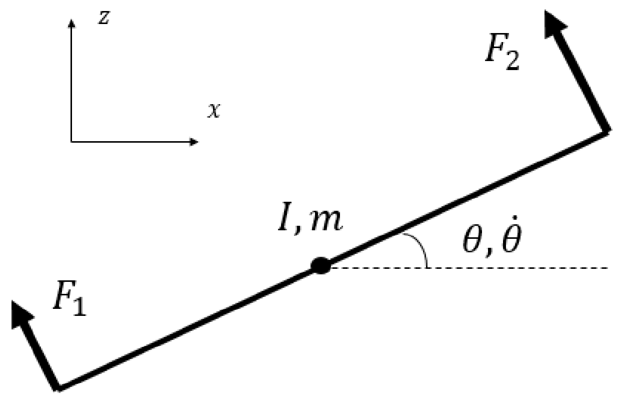
Fortunately, Euclidean distance need not be used for the cost. Any positive semidefinite matrix for Q that will drive the state vector to the target state vector can be chosen. The most basic way to ensure these conditions is with the identity matrix for Q. This methodology proved successful in the twinrotor1 scenario. It rapidly approached the target state (aided by the fact that this attitude-only system is linear, and all values of theta are stable). Furthermore, the complex representation of the state is redundant and will be driven to its target if theta also goes to its target. Thus, for twinrotor1, a Q matrix of

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

is sufficient to account for all state variables. This result of diagonal Q, neglecting the complex angle states, will be generalizable to all future scenarios. A further improvement can be made by scaling diagonal elements. This was shown to lessen the chance of saturating of the cost function (and thus losing information) and also can be used to force faster convergence of particular state variables.

2. Velocity stabilization in twinrotor

The next scenario developed was “twinrotor2”. This system built upon twinrotor1, stabilizing the planar velocities in addition to the attitude. This scenario is stabilized within the body frame, with no regard to absolute position in an inertial reference frame. This scenario used a convention of z pointing up, but this would be reversed later in the quadrotor simulation to a more conventional reference frame.



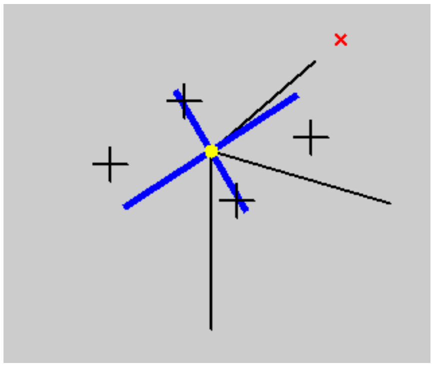
Its dynamics are defined as

$$\begin{bmatrix} \theta \\ \dot{\theta} \\ \dot{x} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} \dot{\theta} \\ \dfrac{F_2 - F_1 - b * \dot{\theta}}{I} \\ \dfrac{1}{m}(-F_1 - F_2)\sin\theta \\ \dfrac{1}{m}(F_1 + F_2)\cos\theta \end{bmatrix}$$

The dynamics of twinrotor2 do not include gravity, and thus when it stabilizes to $\dot{z} = 0$, this is really a freefall; alternatively, there could be a secondary controller which would enable a hover. This latter method is more helpful to imagine, since the propeller thrusts can then be thought of as delta thrusts from a trimmed condition. This same idea of a secondary controller and thrusts as deviations from a trim condition will be used in the development of the quadrotor simulation.

3. Quadrotor simulation

The third and final scenario was named "quadrotor". This scenario models the quadrotor as an x-shaped rotor with four actuators at the tip of each arm. A screen capture is shown below.

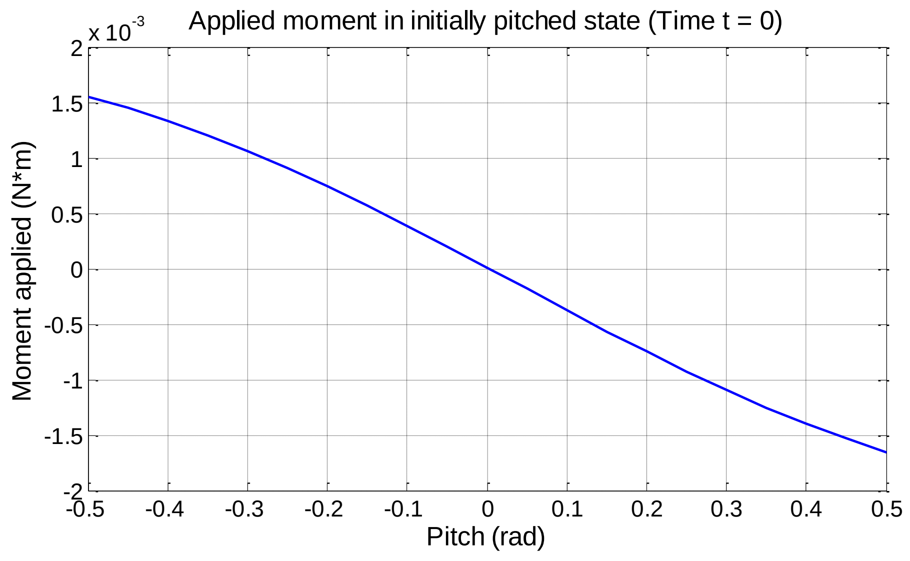


The blue arms define the rotor. The three black lines define the yawed intermediate coordinate system, with the red x indicating the forward x-direction. The four black plus-signs lie in the yawed x-y-plane and indicate target points for the rotor, when its pitch and roll are forced to zero. Although the animation is shown using an intermediate rotated frame, the dynamics are all computed within the fully rotated body frame.

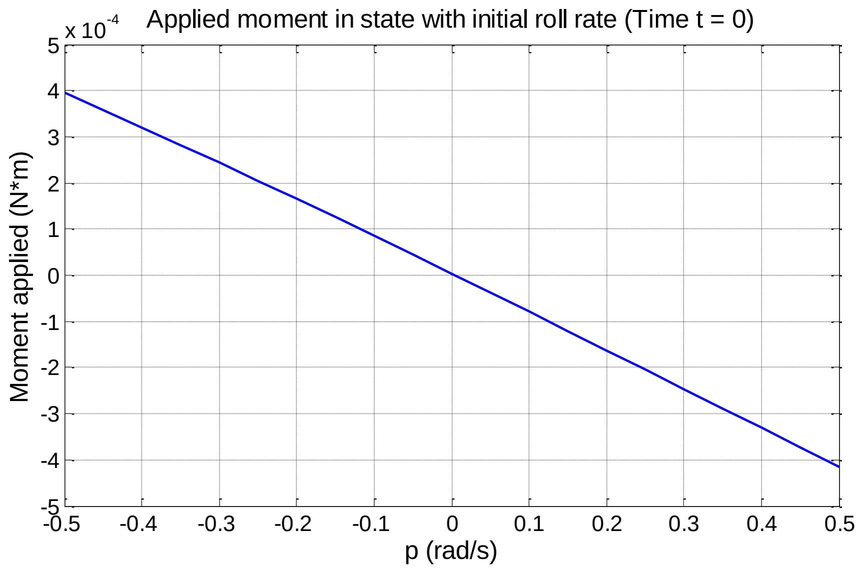
The state for calculating cost in this scenario was chosen to be $[\phi\ \theta\ p\ q\ r\ u\ v]^T$. Thus, no cost is attributed to w. It is assumed that a secondary controller would be in effect to control this state, with the other 7 states controlled through delta thrusts. This state vector was also appended with complex representations of ϕ and θ when convenient, and with w in the dynamics calculations, since the other states depend on the value of w. These dynamics are outlined below.

$$\dot{\phi} = p + (q\sin\phi + r\cos\phi)\tan\theta$$
$$\dot{\theta} = q\cos\phi - r\sin\phi$$
$$\dot{p} = \left(I_{zz}L + I_{xz}N - \left\{I_{xz}\left(I_{yy} - I_{xx} - I_{zz}\right)p + \left[I_{xz}^2 + I_{zz}\left(I_{zz} - I_{yy}\right)\right]r\right\}q\right) \Big/ \left(I_{xx}I_{zz} - I_{xz}^2\right)$$
$$\dot{q} = \frac{1}{I_{yy}}\left[M - \left(I_{xx} - I_{zz}\right)pr - I_{xz}\left(p^2 - r^2\right)\right]$$
$$\dot{r} = \left(I_{xz}L + I_{xx}N - \left\{I_{xz}\left(I_{yy} - I_{xx} - I_{zz}\right)r + \left[I_{xz}^2 + I_{xx}\left(I_{xx} - I_{yy}\right)\right]p\right\}q\right) \Big/ \left(I_{xx}I_{zz} - I_{xz}^2\right)$$
$$\dot{u} = X/m - g\sin\theta + rv - qw$$
$$\dot{v} = Y/m + g\sin\phi\cos\theta - ru + pw$$
$$\dot{w} = Z/m + g\cos\phi\cos\theta + qu - pv$$

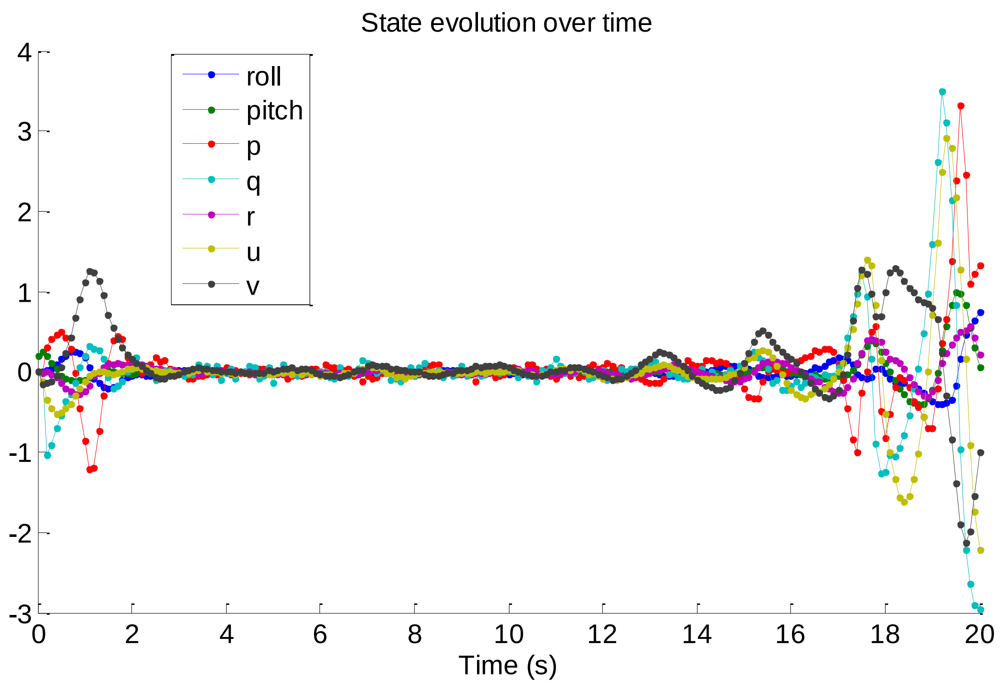
Given the dynamics and cost structure defined above, a policy was computed at 10 Hz (both the controller and the dynamics) for 25 trials of 2 seconds each—10 of random actuation and 15 of controlled actuation. Random trials were found to be very helpful for the policy at high angles and rates, since the controlled trials spend much of their time near equilibrium. With this methodology, the controller developed closely resembled a PD controller. The following figure shows the applied pitching moment at various pitches (all other states are zero).



Similar phenomena are found in roll, p, and q. Rolling moment versus p is plotted below.

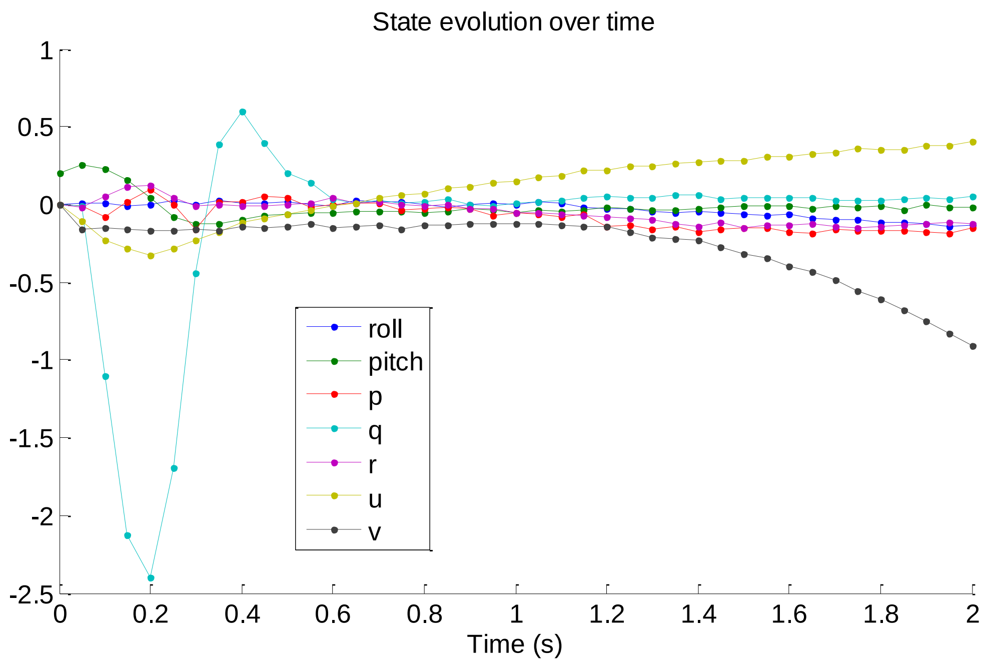


The evolution of the full state over time, given an initial pitch of 0.2 radians can be found in the following figure.



This figure illustrates a few important features. First, the controller has not perfectly figured out how to decouple the pitch and roll dynamics. As seen above, the roll rate is heavily utilized in correcting an initially pitched state. Furthermore, it can be seen that although the system appears to be near stability for about 10 seconds, eventually, a slow and unstable mode gets excited and causes the system to blow up.

An additional experiment at 20 Hz was run with the same learning experience of 25 trials to see whether this would allow the controller to capture the faster dynamics of the system, which may be in some way exciting this slow and unstable mode. Admittedly, 20 Hz still will leave a large deal of dynamics uncaptured, but the computations rapidly become more expensive at high frequencies, so computations at 50 or more Hertz quickly became impractical without a faster machine. An evolution with the same initial state of 0.2 radians of pitch is shown below. Note the much shorter time scale.



Interestingly, this higher frequency controller decoupled pitch and roll much more effectively. Other initial states such as initial roll still exhibited a reasonable degree of coupling. Also of note, the dynamics blow up much more quickly here, so there are very clearly faster dynamic modes not being captured at 10 Hz, and probably not even at 20 Hz.

4. Future work

Going forward, it would be wise to run the algorithm with much higher frequency to capture all of the high frequency modes in the dynamics. Unfortunately, this quickly becomes computationally intensive and was impractical on a laptop. It would also be ideal to adapt the code to allow the controller to operate at a higher frequency than the dynamics in order to handle the fast dynamic modes. If these results proved promising, the next step for this project would be to input flight data from any quadrotor flight log to see how well that developed controller matches the one developed above. This would illustrate whether the dynamics used in the simulation do in fact match those onboard a real system. The final step is then to apply a developed controller onboard the Parrot drone with Paparazzi installed. The drone seemed crash resistant and could drop from several feet in a hover position with no damage. Its battery lasted about 5-10 minutes which would be sufficient for recording data.

5. Conclusion

The structure of the simulation should allow for integration with a physical platform effectively. The simulated dynamics can be replaced with logged data, and the policy must be translated to a usable form for the drone. The PILCO algorithm was shown to be able to handle high dimensionality problems at low frequencies with a reasonable degree of success, given that its learning experience (50 seconds) was fairly limited. That being said, these low frequency experiments don't fully exploit the inherent instabilities found in a quadrotor dynamic system, and high frequency tests still need to be run to verify that it can control such dynamics.