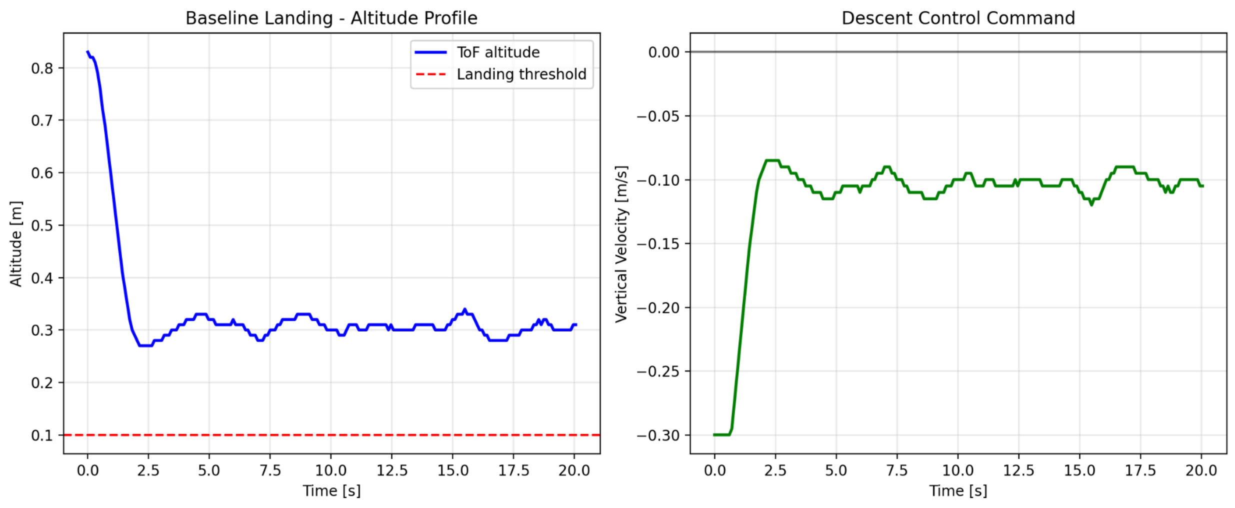
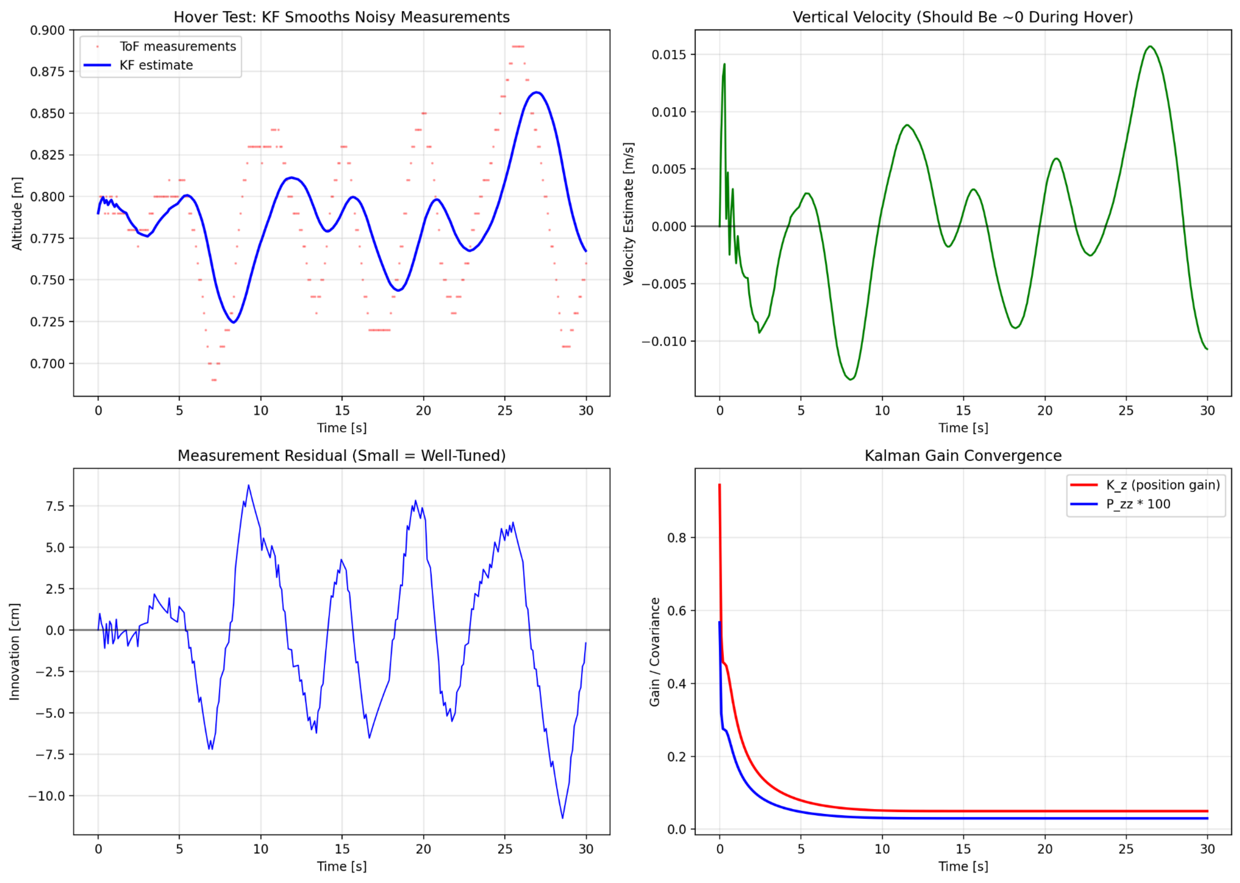
PHASE 4: Comparative Analysis

1. **How did the Kalman filter improve performance over baseline even without attacks? Consider noise reduction, velocity estimation, and control smoothness.**

Without spoofing, the Kalman filter still gives noticeably better controlled descent than baseline. This is shown in the hover test results. As shown in the plots, KF reduces sensor noise, leading to a smoother altitude estimate and smoother control. Additionally, the vertical velocity estimation is centered around 0 m/s, with very small and slow varying oscillations. This demonstrates KF gives a usable velocity signal. Additionally, the bottom plots show that the innovation (residual) is shrinking towards 0, and Kalmain gain convergence shows the filter is stabilizing and smoothing over time. KF appears to dampen effects of noise which leads to more stable control inputs. The baseline landing error was 13.04, and touchdown velocity was 0.105 m/s. The time to land was also within 20s. So the baseline does meet the mission performance, but the noisy raw ToF used causes larger altitude. As shown in the left plot, the ToF measurement is noisy and bouncing, and altitude does not settle cleanly. On the right, the fluctuations are between -0.085 and -0.105, causing continuous micro corrections and there is never a smooth hover state. This jitter is greatly attenuated in the KF version of control.



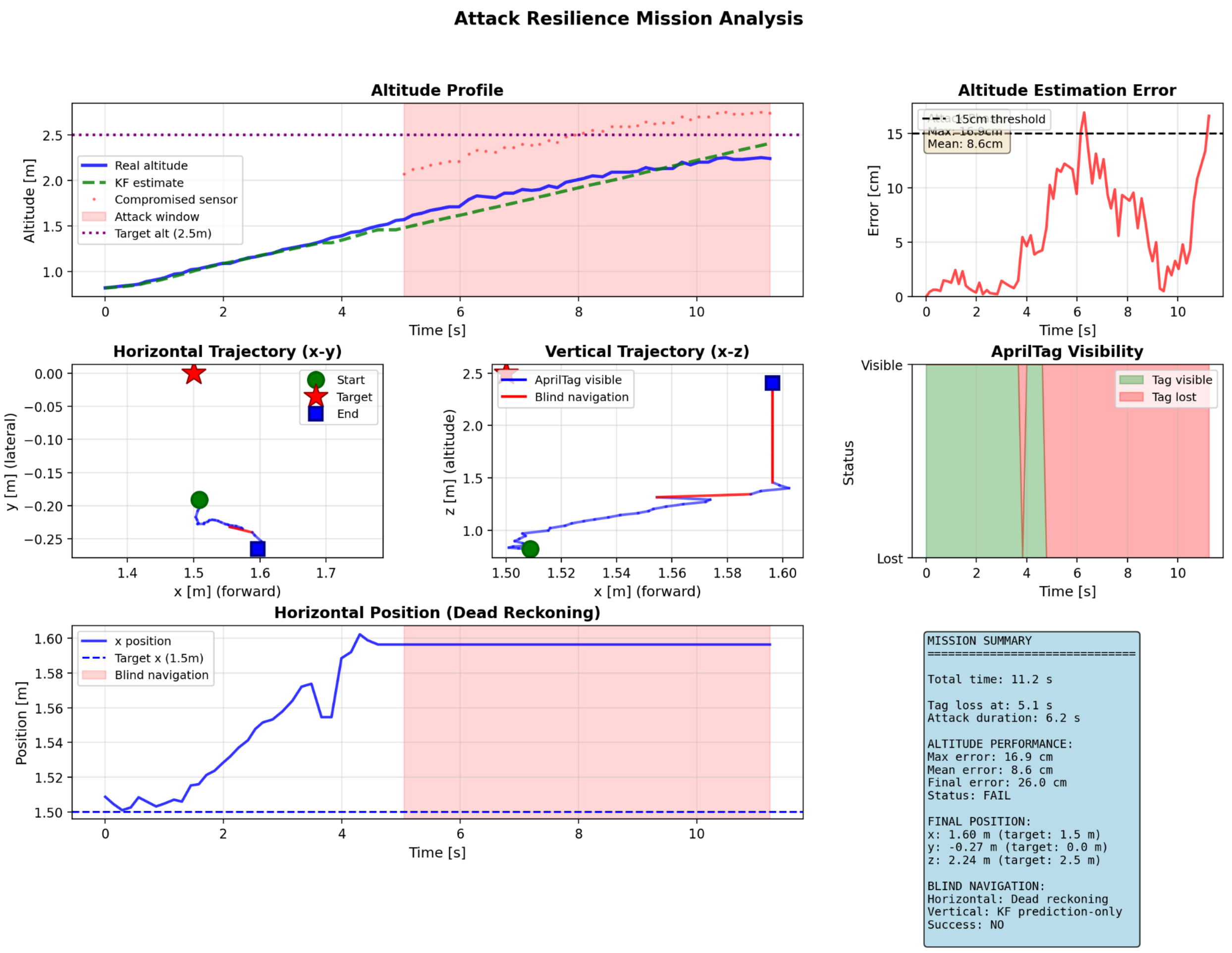


2. **Why can the drone still navigate safely using only predictions without measurements? Explain the role of the accurate motion model and state estimation.**

This is because the Kalman filter has an accurate, physics-based model which predicts how altitude evolves over time, even without sensor data $(z_1 = z_0 + v_{z,0}\Delta T + \Delta T^2/2 * u)$ & $(v_{z,1} = v_{z,0} + \Delta T * u)$ where u is input acceleration. In a relatively short time frame, the model remains mostly accurate because the model captures real dynamics. This ability to generate an accurate motion model along and stable velocity estimate allow it to dead-reckon altitude for a few seconds which allows the state estimate to remain close to the true motion without any sensor updates. Some position and velocity uncertainty in the model is expected due to mismatch with real dynamics and is brought in using Q.

3. **During Phase 3 blind navigation, how much did the altitude estimate drift? Was this drift acceptable? What factors limit how long you can operate without sensor updates?**

For our 3rd trial, the altitude estimate drifted by a maximum of 16.9 cm with a mean error of 8.6 cm from the actual ToF sensor. The mean was acceptable (<10 cm) but the maximum was just barely over the requirement (<15 cm). The final error for this trial was 26.0 cm but for our 4th trial, it was 7.0 cm. This requirement (<10 cm) could be met based on tuning of the gains, tolerances, and clipping of velocity magnitudes. Overall, the drift was nearly acceptable during blind navigation for our trials with some exceptions.



```
MISSION SUMMARY
==============================

Total time: 11.2 s

Tag loss at: 5.1 s
Attack duration: 6.2 s

ALTITUDE PERFORMANCE:
Max error: 16.9 cm
Mean error: 8.6 cm
Final error: 26.0 cm
Status: FAIL

FINAL POSITION:
x: 1.60 m (target: 1.5 m)
y: -0.27 m (target: 0.0 m)
z: 2.24 m (target: 2.5 m)

BLIND NAVIGATION:
Horizontal: Dead reckoning
Vertical: KF prediction-only
Success: NO
```

It would not be recommended to operate in this blind navigation mode for too long as the altitude drift for most of the trials appeared to worsen quadratically near the end of the run as seen from the error growth pattern identified in phase 2. Blind-navigation is inherently limited due to all the effects of the drone-environment aerodynamic, avionics, and propulsion system not being perfectly captured by our model. Although the Kalman filter can do a decent job of making up for this by using the double integrator system model, it is not configured to account for any changes in acceleration due to either the proportional gain control or from the environment (i.e, downwash, air circulation). The first mismatch specifically caused the KF estimate to create a linear slope for the altitude estimate even when the proportional controller would slow down the drone near 2.5m. The plots from phase 3 show the real trajectory falling off as the proportional gain decreased while the KF estimate just remained linear all the way to 2.5m. This mismatch could be improved by implementing additional code to provide the kf.predict() function with a non-zero input ($u \neq 0$), where u is equal to some estimate of the acceleration caused by changing of the velocity command due to the proportional controller. Adding many robust estimates and measurements of the drone's pose through sensors and more intricate system models will prove to make navigation of the drone more resilient to attacks and complex environments.

4. **From Phase 2 attack tests, how does estimation error scale with attack duration (2s vs. 5s vs. 10s)? Is it linear, quadratic, or other? Explain based on double-integrator dynamics and process noise.**

The response seems somewhat quadratic. As shown in the plot generated from the data, you can see the max drift vs attack duration appears to have a quadratic trend. At 2s, the max error is 5cm, at 5s it is 22.1cm, and at 10s it is 79.3cm.From 2 to 5s, the max error 4.4x. From 5 to 10s, the ratio 3.6x. A linear model would anticipate that at 5s the error is around 12.5, and that at 10s it is approximately at 25. The error grows much faster than linear, hinting that the behavior is instead quadratic.

This behavior makes sense, as due to the double integrator, we find that variance should grow like t^3.Therefore RMS error would scale to t^3/2. This is faster than linear, but slower than pure quadratic, which explains why the integration causes the error to blow up faster with time. Based on the form of B_d, the altitude z relates to input u by t^2/2 which also explains the quadratic relationship.

5. **If the attack magnitude was 1.0m instead of 0.5m, would your Phase 3 mission still succeed? What about 2.0m? At what point does the approach break down and why?**

The attack detection of this algorithm was not explicitly explored or made robust. The attack was simulated by injecting a magnitude of error into the real ToF sensor reading, while the ToF reading was treated as the truth. In reality, this distinction would not be as clear. Since in our case we simulated the attack by injection of a fake attack reading when the AprilTag was lost, the algorithm knew to switch to prediction-only mode so attack magnitude would be irrelevant to our case unless the real ToF reading equaled the attacked reading (attack magnitude = 0). In short, our phase 3 mission would still succeed regardless of the magnitude.

This approach breaks down if a real attack makes the ToF sensor biased by adding some error or noise and we are not aware of when it happens. If we do know exactly when the attack occurs, this is not an issue since we will be able to enter prediction-only mode where the sensors are not trusted at all for navigation, and re-enter normal KF mode making updates using sensor data when the attack is done.

To detect an attack, more advanced methods would be needed like potentially assuming large spikes in altitude readings from the ToF sensor that depart from the KF estimate are attacks to the ToF sensor, and the attack is over when the sensor reading returns close to the current KF estimate. In this way, we could prioritize stability in pose estimation and control and assume departures of a certain magnitude are either attacks or noise not to be trusted until they return to what the KF currently thinks the drone pose is. Innovation could be used as the trigger for measurement surprise to detect attacks and enter prediction-only mode. If the drone remains in update mode, the innovation is used to stabilize the drone against sensor noise by trusting the system model and measurements partially. The proportion of Q and R adjusts how much the KF trusts sensor measurements vs the physics model. Tuning Q and R proportions can make the update mode more resilient to attacks but it still should not be used as the only source of attack detection.

6. **How does Q selection affect the tradeoff between normal operation and prediction-only performance? What happens if you decrease Q to improve attack tolerance?**

Q controls how much the Kalman filter trusts the motion model versus the measurements, specifically the proportion of Q and R will adjust this trust relationship. If Q is larger, the filter assumes more model uncertainty and thus trusts measurements more and responds more quickly to real disturbances. Meanwhile a smaller Q means the filter believes the motion model is accurate and trusts prediction more. For the prediction only operation, a smaller Q means the estimates drift more slowly as there is less uncertainty injected at each timestep, which improves attack tolerance. However there is a tradeoff. First, decreasing Q causes the filter to be less responsive to real dynamics when the measurements do return. Innovation shrinks artificially as well, because the filter is underestimating uncertainty. Therefore, the flight performance can worsen by causing an increase in overshoot and disturbance rejection gets worse.

Therefore, reducing Q improves prediction-only drift behavior but hurts the normal operation because the filter is overconfident in the model and slow to correct from the sensor data.