16.30/31 Feedback Control Systems

Lab 3: Resilient Landing Under Sensor Attack

Issued: 11/14/2025 Due: 12/03/2025

Mission Briefing

Navy plans to implement AeroTech Dynamic altitude control (Lab 1) and trajectory tracking (Lab 2) systems. Now comes the final challenge, where company needs to demonstrate resilience against cyber-physical attacks on critical sensors.

Intelligence reports indicate adversaries can inject false readings into altitude sensors during drone operations. Traditional control systems that blindly trust sensor data become vulnerable to such attacks, potentially causing crashes during critical maneuvers. The Navy needs proof that your system can maintain safe operation even when primary sensors are compromised.

Your approach? A Kalman filter that fuses sensor measurements with physics-based predictions. When sensors become unreliable, the filter can switch to prediction-only mode, using the drone's motion model to maintain accurate state estimates. While competitors add expensive redundant hardware, you'll demonstrate that intelligent estimation algorithms provide cost-efficient cyber-attack resilience.

Lab Overview

This three-week lab progresses through increasingly sophisticated validation:

1. **Phase 1 - Baseline System:** Implement simple landing using direct sensor feedback to establish why naive approaches fail under attack
2. **Phase 2 - Filter Validation:** Design and validate a 1D Kalman filter through hover tests and controlled attack scenarios
3. **Phase 3 - Mission Integration:** Execute complete navigation mission where sensor compromise is emulated when visual feedback is lost
4. **Phase 4 - Analysis:** Quantify system resilience and compare performance across all scenarios

You're provided with `Lab3-Starter-Code.zip` containing:

- `utils/apriltag_utils.py` - AprilTag detection and positioning utilities (PROVIDED)
- `utils/kalman_filter.py` - Kalman filter template with TODO markers
- `phase1_baseline_landing.py` - Baseline landing template
- `phase2_kalman_filter.py` - Filter validation framework
- `phase3_landing_attack.py` - Integrated mission template

Phase 1: Baseline Landing System 15 points

Objectives

Establish baseline performance using direct sensor feedback without filtering. This demonstrates why naive sensor-based control becomes vulnerable when measurements are compromised.

Tasks

1. Mount AprilTag vertically at 1.0-1.5m height. Mark landing target 1.5m forward from tag on ground.
2. Takeoff 2m from AprilTag. Use tag for horizontal (x,y) positioning to hover above landing target.
3. Descend straight down using **only ToF sensor feedback** with simple proportional control. No horizontal corrections during descent.
4. Touch down when ToF altitude falls below 0.1m threshold.

Control Strategy

Implement simple proportional control for altitude:

$$v_z = K_p(z_{\text{target}} - z_{\text{ToF}})$$

where z_{ToF} is the direct reading from the Time-of-Flight sensor (no filtering, no validation). Choose initial $K_p \approx 0.5$ and if needed change for stable descent.

Student Implementation Tasks

1. Reuse Lab 2 coordinate transformations to position above landing target using AprilTag
2. Implement P-controller for descent phase using direct ToF feedback
3. Test and record landing accuracy and touchdown velocity
4. Explain what happens if ToF sensor reports altitude 0.5m higher than truth?

Performance Metrics

Metric	**Target**
Landing accuracy	Within 15 cm of target center
Touchdown velocity	Less than 0.3 m/s downward
Time to land	Record actual duration

Landing accuracy measured as Euclidean distance from target center at touchdown.

Deliverables

1. `baseline_landing.py` - Complete implementation with P-control descent
2. `baseline_data.csv` - Logged altitude, velocity commands, and positions
3. `baseline_metrics.txt` - Landing accuracy, touchdown velocity, timing
4. `baseline_analysis.txt` - Two-sentence explanation of crash mechanism under attack, with mathematical example showing how false sensor reading leads to wrong control action

Phase 2: Kalman Filter Design & Validation 35 points

Objectives

Design, implement, and validate a 1D Kalman filter for altitude estimation. Test filter performance under three conditions: (1) normal operation with trutful sensors, (2) hover validation for noise characterization, and (3) resilience testing under simulated sensor attacks of varying duration.

Part A: Filter Design (10 points)

System Model

Implement a discrete-time double integrator for the altitude axis at sampling period $\Delta T = 0.1$ s:

$$\mathbf{x}_k = \begin{bmatrix} z \\ v_z \end{bmatrix}, \quad \mathbf{x}_{k+1} = \mathbf{A}_d \mathbf{x}_k + \mathbf{B}_d u_k + \mathbf{w}_k$$

System matrices:

$$\mathbf{A}_d = \begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix}, \quad \mathbf{B}_d = \begin{bmatrix} \Delta T^2/2 \\ \Delta T \end{bmatrix}$$

Measurement model (ToF sensor observes altitude only):

$$y_k = \mathbf{H}\mathbf{x}_k + v_k, \quad \mathbf{H} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

Process noise $\mathbf{w}_k \sim \mathcal{N}(0, \mathbf{Q})$ and measurement noise $v_k \sim \mathcal{N}(0, R)$.

Kalman Filter Equations

Implement the standard two-step Kalman filter.

Prediction step (always runs):

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{A}_d \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_d u_{k-1} \tag{1}$$

$$\mathbf{P}_{k|k-1} = \mathbf{A}_d \mathbf{P}_{k-1|k-1} \mathbf{A}_d^T + \mathbf{Q} \tag{2}$$

Update step (skipped in prediction-only mode):

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}^T(\mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^T + R)^{-1} \tag{3}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(y_k - \mathbf{H}\hat{\mathbf{x}}_{k|k-1}) \tag{4}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_{k|k-1} \tag{5}$$

The innovation $\nu_k = y_k - \mathbf{H}\hat{\mathbf{x}}_{k|k-1}$ represents measurement surprise.

Parameter Selection

Tune measurement noise covariance R and process noise covariance $\mathbf{Q}$. Use sensor characteristics from Lab 1. ToF sensor typically has $\sigma \approx 0.01$ m, so $R = 0.01^2$. For process noise covariance, which accounts for model uncertainty, start with:

$$\mathbf{Q} = \text{diag}([0.001^2, 0.01^2])$$

Position uncertainty (0.001 m) accounts for unmodeled dynamics. Velocity uncertainty (0.01 m/s) accounts for rapid changes between samples.

Student Implementation Tasks

1. Complete `predict()` method in `kalman_filter.py`
2. Complete `update()` method with innovation calculation
3. Implement `set_mode()` to enable/disable measurement updates
4. Run simulation test: `python kalman_filter.py` to verify implementation
5. Justify your $\mathbf{Q}$ and R choices based on Lab 1 sensor data

Part B: Hover Validation (15 points)

Test Protocol

Hover at constant altitude for 30 seconds while logging at 10 Hz:

- Raw ToF measurements (noisy sensor data)
- KF altitude estimates (filtered data)
- Innovation sequence ν_k
- Kalman gain evolution K_k
- Covariance diagonal elements P_{zz}, $P_{v_z v_z}$

During hover, acceleration command $u_k = 0$, so filter tracks constant altitude despite measurement noise.

Student Analysis Tasks

1. Calculate noise reduction: $\sigma_{\text{ToF}}/\sigma_{\text{KF}}$
2. Compute innovation RMS and verify it's consistent with R
3. Plot gain convergence and identify settling time
4. Assess whether $\mathbf{Q}$ and R are well-tuned (adjust if needed)

Part C: Attack Resilience Test (10 points)

Test Protocol

Execute controlled descent from 1.5m to 0.5m target altitude. During descent, inject simulated sensor attacks of three durations:

1. 2-second attack
2. 5-second attack
3. 10-second attack

Attack emulation: Add +0.5m bias to ToF readings during attack window. KF immediately switches to prediction-only mode (no measurement updates), relying purely on physics-based dead reckoning.

Attack Response Strategy

When attack detected (or emulated):

1. **Immediately** call `kf.set_mode(prediction_only=True)`
2. Continue running `predict()` at each timestep
3. **Skip** all `update()` calls
4. Filter becomes pure integrator: $\hat{z}_{k+1} = \hat{z}_k + \Delta T \cdot \hat{v}_z$
5. Estimate drifts over time as model uncertainty accumulates

Performance Metrics

For each attack duration, compute:

- Maximum altitude error during attack
- Mean altitude error during attack
- Final drift at end of attack window
- Pass/fail against 15 cm threshold

Success criterion: Maximum error < 15 cm during attack window.

Student Analysis Tasks

1. Plot altitude error vs. time for all three attack durations
2. Determine maximum safe attack duration (error stays under 15 cm)
3. Analyze error growth pattern: Linear? Quadratic? Explain based on double-integrator dynamics
4. Discuss how $\mathbf{Q}$ tuning affects prediction-only performance (tradeoff analysis)
5. Recommend operational limits and explain how long can your system safely operate without sensor updates?

Deliverables

1. `kalman_filter.py` - Complete implementation with predict/update/mode switching
2. `kf_simulation.png` - Simulation results (normal and attack scenarios)
3. `kf_tuning.txt` - Justification for $\mathbf{Q}$ and R based on Lab 1 data
4. `hover_test_data.csv` - 30-second hover telemetry
5. `hover_test_plots.png` - Four-panel figure showing measurements vs. estimates, velocity, innovation, gain convergence
6. `attack_test_2s.csv`, `attack_test_5s.csv`, `attack_test_10s.csv` - Attack test data
7. `attack_comparison.png` - Comparative plots showing error vs. attack duration
8. `attack_analysis.txt` - Completed analysis answering questions about attack tolerance, error growth, and system limitations

Phase 3: Blind Navigation Mission 40 points

Objectives

Execute a complete 3D navigation mission that emulates sensor attack when visual feedback is lost. Demonstrate that Kalman filter prediction can maintain safe altitude control even when measurements become unavailable.

Mission Scenario

Navigate from starting position to a high-altitude target at (1.5, 0.0, 2.5) meters in the tag frame. During ascent to this altitude, the AprilTag will naturally be lost from the drone's field of view. At this moment, we emulate a cyber-physical attack by injecting +0.5m bias into ToF sensor readings, simulating adversarial sensor behavior.

The mission tests your system's ability to:

1. Navigate using AprilTag while available
2. Detect loss of visual feedback
3. Switch to blind navigation mode immediately
4. Complete mission using dead reckoning and KF predictions
5. Reach target altitude safely despite compromised sensors

Blind Navigation Strategy

When AprilTag is lost (typically when climbing above tag height):

Horizontal control (x, y):

- Remember last known AprilTag position
- Command velocities toward target based on last known state
- Accept that horizontal accuracy degrades without feedback

Vertical control (z):

- Switch KF to prediction-only mode
- Emulate sensor attack by injecting +0.5m bias into ToF readings
- Control descent using KF altitude estimate $\hat{z}$, **not** sensor reading

Mission Phases

1. Takeoff and establish visual contact with AprilTag. Initialize KF with current altitude.
2. Navigate toward target (1.5, 0.0, 2.5)m using AprilTag guidance. Monitor tag detection continuously.
3. When AprilTag lost for 3+ consecutive frames:
   - Print "TAG LOST - ATTACK INITIATED"
   - Save last known position
   - Switch KF to prediction-only: `kf.set_mode(prediction_only=True)`
   - Begin injecting +0.5m bias into ToF readings (emulating attack)
4. Continue to target using:
   - Horizontal: Dead reckoning from last known position
   - Vertical: KF predictions (ignoring biased measurements)
5. When altitude estimate reaches $2.5m \pm 0.05m$, command landing sequence.

Student Implementation Tasks

1. Integrate KF into mission control loop (template mostly provided)
2. Implement tag loss detection (3 consecutive lost frames)
3. Implement attack emulation trigger when tag is lost
4. Implement mode switching: normal to prediction-only
5. Tune horizontal and vertical control gains (K_p values)
6. Implement mission state machine: approach → climb → blind nav → arrival

Performance Metrics

Metric	**Symbol**	**Target**
Max altitude error	$e_{z,\max}$	< 15 cm during blind nav
Mean altitude error	$e_{z,\text{mean}}$	< 10 cm during blind nav
Final altitude error	$\lvert z_{\text{final}} - z_{\text{target}} \rvert$	< 10 cm
Blind navigation duration	t_{blind}	Record actual time
Mission success	-	Reach target without crash

Altitude errors computed as $|z_{\text{ToF\_real}} - \hat{z}_{\text{KF}}|$ during blind navigation phase.

Deliverables

1. `landing_attack.py` - Complete mission implementation with attack emulation
2. `attack_mission.csv` - time, phase, true altitude, reported altitude, KF estimate, positions, commands, KF mode flags
3. `mission_analysis.png` - Comprehensive multi-panel plot showing:
   - Altitude profile (real vs. estimated vs. compromised sensor)
   - Altitude error over time
   - 3D trajectory visualization
   - AprilTag visibility timeline
   - Phase annotations
4. `mission_report.txt` - Performance summary including all metrics, timeline, and pass/fail assessment

Phase 4: Comparative Analysis 10 points

Answer the following in 1-2 sentence each:

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

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

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?

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.

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?

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?

Deliverables

1. `analysis.pdf` - Thoughtful answers to all six analysis questions (1 page)

Mission Debrief

Outstanding work! You've demonstrated that intelligent estimation algorithms provide resilience against cyber-physical attacks without expensive hardware redundancy. Your Kalman filter maintains safe operation by switching to physics-based predictions when sensors are compromised, a capability that distinguishes your system from vulnerable baseline approaches.

Compress everything into `Lab3-[name1]-[name2].zip` for Canvas submission.

Reflection Questions (Not Graded)

1. How would you extend this approach to handle attacks on multiple sensors simultaneously?

2. Could you detect the attack automatically without being told? What would you monitor?

3. How would you design an attack detector that triggers prediction-only mode automatically?

4. What if the attack was a slowly increasing ramp instead of a sudden bias?

5. How would you handle complete loss of both altitude sensors and AprilTag?

The Navy contract is yours! You've proven that AeroTech's estimation algorithms provide unmatched resilience against cyber-physical threats. Well done!