Report

Aerial Robotics Kharagpur — Controls Team

January 12, 2026

Abstract

This report presents the design, implementation, and debugging process of a custom Extended Kalman Filter (EKF) and PID-based offboard control pipeline for a PX4 multirotor UAV using ROS 2 and PX4 SITL. The objective of this work is to deeply understand state estimation, control, and PX4 offboard behavior rather than replace PX4's internal EKF and controllers. The report documents the system architecture, mathematical formulation, implementation details, mistakes encountered, and lessons learned. Due to PX4 Offboard mode constraints, the full autonomous flight execution could not be achieved, and the system remained in `nav_state = 14 (POSCTL)` during simulation. The work is therefore validated up to estimator correctness, controller logic, and Offboard interface design.

1 Introduction

Reliable autonomous flight requires accurate state estimation and robust feedback control. Modern autopilots such as PX4 provide highly optimized estimators and controllers; however, understanding and reimplementing these systems is critical for research, advanced control, and development.

This project focuses on:

- Designing a custom EKF using raw IMU and altitude measurements
- Implementing a cascaded PID controller for altitude regulation
- Interfacing with PX4 using ROS 2 Offboard mode

All development and validation are performed in PX4 SITL using Gazebo and ros2(humble).

2 System Architecture

2.1 High-Level Architecture

```
PX4 SITL
  IMU (sensor_combined)
  Local Position (vehicle_local_position_v1)


ROS 2 EKF Node
  Prediction
  Measurement updates
  Covariance propagation
  Publishes /ekf_data
```

```
ROS 2 PID Offboard Node
  Cascaded altitude PID
  Attitude setpoints
  Offboard heartbeat


PX4 Commander + Attitude Controller
```

2.2 Design Philosophy

PX4 remains responsible for:

- Angular rate control
- Motor mixing
- Safety and failsafes

The external ROS 2 nodes provide:

- State estimation
- High-level control commands
- Attitude stabilization

This separation ensures safety while enabling experimentation.

3 Coordinate Frames and Conventions

PX4 uses strict **NED / FRD** conventions:

Quantity	Frame	Convention
Position Z	NED	Positive down
Thrust	Body (FRD)	Negative Z
Roll / Pitch	Body	Right-hand rule

Failure to respect these conventions leads to silent instability.

4 Estimation Module

4.1 State Vector

The EKF estimates the following state:

$$\mathbf{x} = \begin{bmatrix} \phi & \theta & r & z & v_z \end{bmatrix}^T$$

Where:

- ϕ : roll angle (rad)
- θ : pitch angle (rad)
- r : yaw rate (rad/s)
- z : altitude (NED, positive down)
- v_z : vertical velocity (m/s)

4.2 Prediction Model

Using IMU inputs:

$$\phi_{k+1} = \phi_k + p_k \Delta t \tag{1}$$
$$\theta_{k+1} = \theta_k + q_k \Delta t \tag{2}$$
$$r_{k+1} = r_k \tag{3}$$
$$z_{k+1} = z_k + v_{z,k} \Delta t \tag{4}$$
$$v_{z,k+1} = v_{z,k} \tag{5}$$

4.3 State Transition Jacobian

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

4.4 Covariance Prediction

$$\mathbf{P}^- = \mathbf{F}\mathbf{P}\mathbf{F}^T + \mathbf{Q}$$

$\mathbf{Q}$ represents process uncertainty and is tuned empirically.

5 EKF Implementation Details

This section explains the EKF implementation at the code level and maps each block of code to its corresponding estimation concept. The EKF is intentionally written from scratch to expose all internal steps clearly.

5.1 IMU Data Abstraction

The `ImuData` dataclass acts as a clean interface between PX4 ROS messages and the EKF logic:

- Decouples estimation logic from ROS message formats
- Stores sensor values, time deltas, and metadata together
- Makes EKF testable independently of ROS

```
@dataclass
class ImuData:
    accel: Tuple[float, float, float]
    gyro: Tuple[float, float, float]
    accel_dt: float
    gyro_dt: float
    timestamp: float
    accelerometer_clipping: int
    gyro_clipping: int
```

Key learning: EKF code becomes significantly more readable and debuggable when sensor handling is abstracted cleanly.

5.2 EKF Step Function Overview

The EKF logic is implemented inside a single function:

- Input: previous state $\mathbf{x}$, covariance $\mathbf{P}$, IMU data, optional altimeter data
- Output: updated state $\mathbf{x}$ and covariance $\mathbf{P}$

This mirrors the mathematical EKF structure exactly:

1. Prediction
2. Covariance propagation
3. Accelerometer update (roll, pitch)
4. Gyro yaw-rate update
5. Altimeter update (when available)

5.3 Prediction Step (IMU Integration)

```
x_predicted = np.array([
    phi + p * gyro_dt,
    theta + q * gyro_dt,
    r,
    z + vz * accel_dt,
    vz + (az + g) * accel_dt
])
```

Explanation:

- Roll and pitch are propagated using gyro integration
- Yaw rate is modeled as a random walk
- Altitude is propagated using vertical velocity
- Vertical velocity uses accelerometer Z with gravity compensation

Important design note: Accelerometers measure *specific force*, not pure acceleration. Explicit gravity compensation is therefore required.

5.4 Covariance Prediction

```
p_pred_var = F @ p_var @ F.T + Q
```

Explanation:

- **F** propagates uncertainty through system dynamics
- **Q** models unmodeled effects such as noise and bias
- Covariance grows during prediction and shrinks during correction

Key learning: EKF memory lives in **P**, not in sensor history.

5.5 Accelerometer Measurement Update

```
h_acc = np.array([
    [g * np.sin(theta)],
    [-g * np.sin(phi)]
])
```

Explanation:

- Uses gravity direction to correct roll and pitch
- Only valid during low linear acceleration (hover assumption)
- Prevents long-term gyro drift

Critical mistake avoided: Using old state values instead of predicted values breaks EKF consistency.

5.6 Yaw Rate Update

Yaw rate is directly observable from the gyro and updated separately:

```
y_r = np.array([[r_meas - r_pred]])
```

Reasoning:

- Prevents yaw-rate drift
- Keeps yaw dynamics well-bounded
- Simplifies the estimator without sacrificing stability

5.7 Altimeter Update

Altimeter data arrives at a lower rate and is applied conditionally:

```
if alt is not None:
    z_meas, vz_meas, z_reset = alt
```

Explanation:

- Only altitude is corrected
- Vertical velocity is indirectly improved through covariance coupling
- Measurement is consumed once and then cleared

Key learning: EKF does not overwrite states — it softly corrects them based on confidence.

6 Measurement Models

6.1 Accelerometer Update (Roll & Pitch)

Accelerometer measures gravity direction:

$$a_x = g\sin(\theta) \tag{6}$$
$$a_y = -g\sin(\phi) \tag{7}$$

Measurement Jacobian:

$$\mathbf{H}_{acc} = \begin{bmatrix} 0 & g\cos\theta & 0 & 0 & 0 \\ -g\cos\phi & 0 & 0 & 0 & 0 \end{bmatrix}$$

6.2 Altimeter Update

$$z_{meas} = z \quad , \quad \mathbf{H}_{alt} = [0\ 0\ 0\ 1\ 0]$$

6.3 Kalman Update Equations

$$\mathbf{y} = \mathbf{z} - \mathbf{h}(\mathbf{x}^-) \tag{8}$$
$$\mathbf{S} = \mathbf{H}\mathbf{P}^-\mathbf{H}^T + \mathbf{R} \tag{9}$$
$$\mathbf{K} = \mathbf{P}^-\mathbf{H}^T\mathbf{S}^{-1} \tag{10}$$
$$\mathbf{x} = \mathbf{x}^- + \mathbf{K}\mathbf{y} \tag{11}$$
$$\mathbf{P} = (\mathbf{I} - \mathbf{K}\mathbf{H})\mathbf{P}^- \tag{12}$$

7 Control Module

7.1 Why Cascaded PID

Altitude dynamics are second-order. Direct thrust PID causes oscillations. Cascaded control separates slow and fast dynamics.

7.2 Outer Loop: Position to Velocity

$$v_{z,sp} = K_z(z_{sp} - z)$$

7.3 Inner Loop: Velocity to Thrust

$$u = K_p e + K_i \int e dt + K_d \dot{e}$$

$$T = T_{hover} + u$$

With saturation:

$$T_{min} \leq T \leq T_{max}$$

Roll and pitch are held at zero; PX4 handles stabilization.

8 PID Controller Implementation

This section explains the PID-based offboard controller and how it safely interfaces with PX4.

8.1 PID Controller Class

```
class PIDController:
    def __init__(self , kp , ki , kd , limit = None):
```

Design choices:

- Integral and derivative are computed explicitly
- Optional output limit prevents windup and saturation
- Controller is reusable for roll, pitch, and velocity

Key learning: Saturation limits must exist even in simulation.

8.2 Cascaded Altitude Control Logic

```
vz_r = (self.z_r - z) * self.z_kp
vz_error = vz_r - vz
thrust_correction = self.vz_pid.update(vz_error, dt)
```

Explanation:

- Outer loop: position → velocity
- Inner loop: velocity → thrust
- Mimics PX4 internal controller architecture

This separation significantly improves stability and tuning robustness.

8.3 Thrust Safety Clamp (Critical)

```
thrust_cmd = max(min(thrust_cmd, 0.8), 0.1)
```

Why this is essential:

- Prevents motor saturation
- Avoids free-fall due to zero thrust
- Protects against EKF spikes or bad transients

Interpretation:

- Lower bound ensures lift is always generated
- Upper bound ensures PX4 retains control authority

This acts as a **software safety layer**.

8.4 Offboard Mode Management

```
offboard.attitude = True
self.offboard_pub.publish(offboard)
```

Explanation:

- Continuous heartbeat is mandatory
- PX4 exits OFFBOARD immediately if heartbeat stops

8.5 Arming and Mode Switching

```
self.send_vehicle_command(
    VehicleCommand.VEHICLE_CMD_DO_SET_MODE,
    param1=1.0,
    param2=6.0
)
```

Key learning:

- Arming and OFFBOARD must be requested repeatedly
- PX4 silently ignores invalid command sequences

8.6 Attitude and Thrust Command

```
att.q_d = self.euler_to_quat(roll_cmd, pitch_cmd, 0.0)
att.thrust_body = [0.0, 0.0, -thrust_cmd]
```

Important detail:

- Thrust is applied in FRD body frame
- Negative Z corresponds to upward thrust

Failure to respect this convention causes silent failure.

9 PX4 Offboard Control Logic

PX4 requires three continuous streams:

1. `VehicleCommand` — authority
2. `OffboardControlMode` — heartbeat
3. `VehicleAttitudeSetpoint` — commands

Failure of any stream causes immediate failsafe exit.

10 Major Mistakes and Lessons Learned

Mistake	Lesson Learned
Using accelerometer as true acceleration	Accelerometers measure specific force
Ignoring frame conventions	Sign errors silently destabilize systems
Uninitialized EKF state	ROS callbacks are asynchronous

11 Results and Current Status

11.1 What Was Successfully Achieved

- Stable EKF execution with bounded covariance
- Correct fusion of IMU and altimeter data
- Physically consistent roll, pitch, altitude, and vertical velocity estimates
- Functional cascaded PID control logic
- Continuous publication of PX4 Offboard heartbeat and setpoints

11.2 Simulation Limitation Observed

During PX4 SITL testing, the vehicle remained in `nav_state = 14 (POSCTL)` and did not transition successfully into full OFFBOARD control. As a result, autonomous flight execution could not be completed.

This behavior indicates that although:

- EKF estimation was functioning correctly

- PID control commands were generated correctly
- Offboard messages were published

there exist unresolved PX4 Offboard mode constraints related to mode switching, command sequencing, or safety checks.

11.3 Validation Scope

The work presented in this report is therefore validated up to:

- Estimator design and implementation
- Controller logic and safety constraints
- ROS 2–PX4 Offboard interface structure

Full autonomous flight execution remains part of future work.

12 Conclusion

This project demonstrates:

- Correct EKF formulation and implementation
- Proper handling of multi-rate sensor fusion
- Sound PID controller design with safety constraints
- Clear understanding of PX4 Offboard architecture and failure modes

Although full OFFBOARD flight execution could not be achieved due to the vehicle remaining in `nav_state = 14`, the work provides a strong and correct foundation for autonomous control. Most importantly, it highlights that **timing, interfaces, mode sequencing, and safety logic matter as much as control and estimation equations**. .

13 Future Work

- Debug and resolve PX4 Offboard mode transition from POSCTL to OFFBOARD
- Horizontal state estimation
- MPC-based control
- Hardware deployment