ARL-CR-0892 ● AUG 2026

Framework for Implementation and Evaluation of Estimation and Control Methods for Unmanned Aircraft Systems

prepared by Amoolya S. Tirumalai
General Technical Services, LLC
1451 State Route 34, Suite 301
Wall Township, NJ 07727-1615

under contract W911QX-25-F-045

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-CR-0892 ● AUG 2026

Framework for Implementation and Evaluation of Estimation and Control Methods for Unmanned Aircraft Systems

prepared by Amoolya S. Tirumalai
General Technical Services, LLC
1451 State Route 34, Suite 301
Wall Township, NJ

under contract W911QX-25-F-045

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
August 2026	Contractor Report	September 2025	April 2026

4. TITLE AND SUBTITLE
Framework for Implementation and Evaluation of Estimation and Control Methods for Unmanned Aircraft Systems

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
W911QX-25-F-045		
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**

6. AUTHOR(S)
Amoolya S. Tirumalai

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
General Technical Services, LLC
1451 State Route 34, Suite 301
Wall Township, NJ 07727-1615

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
DEVCOM Army Research Laboratory ATTN: FCDD-RLA-PB Adelphi, MD 20783		ARL-CR-0892

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES
ORCID: Amoolya S. Tirumalai, 0000-0003-0388-780X

14. ABSTRACT
A state estimation and control software package was developed that can be used to integrate unconventional sensing modalities, estimation methods, and control methods. The package accounts for asynchronous sensing and control and is implemented in C++ with the Robot Operating System 2 message-passing interface on the PX4 flight stack. The package is evaluated in Gazebo simulation. The modularity of this package is documented with respect to sensor modules, estimators, and controllers. To demonstrate basic capability, the extended Kalman filter for a UAS and a linear-quadratic-integral controller are implemented. The estimator and controller perform well in simulation, successfully allowing trajectory matching for a figure-eight path. Difficulty was encountered when integrating onto physical hardware due to limitations in the PX4 flight stack, necessitating that future work explores other flight stacks, such as Ardupilot.

15. SUBJECT TERMS
Photonics, Electronics, and Quantum Sciences; PEQS; sensor fusion; control; state estimation; unmanned aircraft systems; UAS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR	24

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
Amoolya S. Tirumalai	(520) 691-5307

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

1. Introduction

1.1 Background

Since the early 2000s, there has been much interest in quadrotor unmanned aircraft systems (UAS) from enthusiasts, industry, and military. One could say that a UAS race is underway among various powers around the globe, much like the previous nuclear and space races. UAS are featured prominently in the war between Ukraine and Russia.

Radio-based communications and sensing, including global navigation satellite systems, have been heavily jammed in Ukraine and often cannot be relied upon. This has led to pilots flying drones by wire with long fiber-optic cables. The Defense Advanced Research Projects Agency worked on developing this capability in the early 2000s, and both the Russians and Ukrainians also possess this capability. These UAS are used for reconnaissance as well as in an anti-personnel capacity.

It is also necessary to develop positioning, navigation, and timing capabilities for these UAS. This requires leveraging various methods in estimation and control theory. Within our team in the Applied Photonics, Electronics, and Quantum Sciences Branch at the U.S. Army Combat Capabilities Development Command (DEVCOM) Army Research Laboratory (ARL), we make use of publicly available, open-source methods, principally the PX4[1] flight stack and Robot Operating System 2 (ROS2).[2] PX4 is a result of work from students at Swiss Federal Institute of Technology Zurich; it started in the 2010s as a side project of several students, then became an academic focus and, ultimately, a company that produces an open-source flight stack. It includes various packages for reading sensor data, state estimation, and actuation using standard techniques, such as complimentary filters, extended Kalman filters (EKFs), and proportional–integral–derivative (PID) controllers. A package called EKF2 was completed in the early 2020s, which dramatically improved PX4's estimation capabilities.

EKF2's capabilities are clear and demonstrable; combined with relevant PID controller packages, it holds position very well. However, our team wishes to develop new sensing, estimation, and control capabilities, and PX4's design philosophy presents some obstacles to this goal.

1.2 Objectives

The goals of this work are as follows:

1) Develop a filtering framework where unconventional sensing modalities can be easily integrated, and;
2) Incorporate modularity at the level of the filtering and control algorithm so new approaches can be evaluated or invented.

The initial purpose of the software developed was to test position estimation capabilities with an RF-based ranging system (which will be discussed in a subsequent report). This later grew into the first stated objective. To do this with EKF2, one would have to either fool it by feeding data from the RF ranging system into a data feed for another sensor data type or develop an EKF2 module specifically for the RF ranging data to hold the data, process it, and fuse it with the other sensor data. The first option would be ad hoc, and other topics for position sensor data expect a variety of statistics and outputs from actual sensors that would have to be mimicked, likely degrading performance. The second option would require developing significant familiarity with the EKF2 package specifically. Learning EKF2's intricacies would have taken an effort similar to developing another estimation package and would not have opened additional opportunities for research.

Neither option was satisfactory, as they led to either a degraded performance or lack of future directions. Therefore, it was decided to write the necessary packages.

A bonus to this package is that it only uses PX4 for reading sensors and processing throttle and rotation inputs. Thus, these packages are agnostic to what flight software is used (Betaflight, Ardupilot, etc.), as long as that software has basic input/output (IO) capabilities.

The secondary objective was to structure the code in a modular way so that new algorithms can be implemented. The code is currently in a good position to develop and implement filters and controllers that more accurately deal with the nonlinearities in the UAS dynamics or include finite-horizon or adaptive capabilities. Safety, trust, and robustness certificates can be developed and implemented on both the estimator and controller.

1.3 Literature Review

Interest in UAS has steadily grown since the early 2000s when powerful, portable computing hardware began to become increasingly available, greatly increasing the

variety of state estimators and (digital) controllers that can be implemented and flown on real systems.

ROS2[2] (and its predecessor, ROS) is a widespread middleware and message-passing framework used in experimental robotics to interact with hardware and data streams. Its ability to manage multirate IO and data processing makes it very useful for sensing, estimation, decision-making, and actuation.

PX4[1] consists of a messaging structure called uORB that behaves in a similar way to messages in ROS2, but is lighter-weight, more specialized, and faster for certain data types, such as data from the accelerometer and gyroscope. It also implements a complete flight control system that can control many types of vehicles, including fixed-wing aircraft, helicopters, multicopters, and ground vehicles. The estimation capabilities in PX4 were dramatically augmented by the completion of the EKF2 package in the early 2020s. This package is a modern implementation of an EKF and a state predictor, including a variety of additional functionalities that ensure healthy, accurate state estimates.

On the control side, PX4 uses industry-standard PID controller cascades. The PID controller focuses on control for the velocity. Outputs of this controller feed into acceleration and orientation cascades. The acceleration cascade is converted into a required throttle, which is then sent to PID controllers and mixers for the motors. The orientation cascade begins with a PID controller for orientation that outputs a rotation rate, which has its own controller that then feeds into the controllers and mixers for the motors.

This work seeks to develop a framework where more modern and performant estimation and control methods can be integrated into flight stacks and evaluated. At the time of this writing, this work consists of conventional methods that can be built into more sophisticated ones.

1.3.1 State Estimation

This subsection reviews various implementations of estimators that interface with PX4 and similar flight stacks.

Lee and Kim[3] report an invariant EKF that uses techniques from differential geometry to obtain an EKF that captures nonlinear behaviors better than conventional EKFs. The performance of their filter exceeded that of the local position estimator (an estimator predating the EKF2) in PX4 and took advantage of optical flow. Study of their filter was performed strictly in simulation. Bstaoros[4] uses motion capture (indoor) and optical flow (indoor and outdoor) as part of an adaptive EKF, which is implemented successfully on real hardware. The adaptation

is based on a convex combination of matrices that depends on outer products containing innovations and gain matrices. Brommer et al.[5] document an EKF with a covariance update that only propagates correlated errors. The filter estimates both vehicle states as well as sensor intrinsics that are used in the estimation model. The sensor states often are not correlated to each other, so when an estimation model is built around the assumption that they are, in fact, correlated, computations are wasted and errors are introduced by these spurious correlations. Limiting the update step to only those correlations that are real leads to improvements in both computation time and estimation quality.

1.3.2 Control

This subsection reviews various controller implementations on open-source UAS flight stacks.

Appapogu[6] uses linear–quadratic regulator (LQR) and model predictive control (MPC)-based controllers on quadrotors in GPS-denied environments, as well as virtual potential field-based obstacle avoidance. The algorithms were implemented for UAS and unmanned ground vehicles and executed with ROS-based application programming interfaces. Bonazza[7] discusses MPC on a PX4-based platform. Amadi[8] reports an adaptive controller with a Pixhawk in the PX4 flight stack. The authors track disturbances and errors to automatically tune the built-in PX4 PID cascade parameters.

This work implements conventional techniques in estimation and control, albeit in a way that allows them to be augmented with more sophisticated capabilities, such as those identified here. This can be developed into a full estimation and control system that can be used to prove certain guarantees for safety, robustness, trust, and stability.

2. Quadrotor UAS Dynamics

This section describes the estimation dynamic model and controller dynamic model.

2.1 Estimation Model

2.1.1 Estimator Dynamics

For time indices $t \in \mathbb{N}_0$, the non-negative integers, consider state $p_t, v_t, a_t, \omega_t \in \mathbb{R}^3$ the position, velocity, measured acceleration, and measured angular velocity, and the orientation quaternion $q_t \in \mathbb{Q}_1$, a unit quaternion. These evolve according to dynamics:

$$p_{t+1} = p_t + Tv_t + w_t^p$$

$$v_{t+1} = v_t + T\boldsymbol{R}(q_t)(a_t - b_t^a) + w_t^v$$

$$q_{t+1} = q_t + \frac{T}{2} q_t \odot (\omega_t - b_t^\omega) + w_t^q$$

with $T > 0$ the filter update period and the accelerometer and gyroscope biases b_t^a, b_t^ω each subject to dynamics of type:

$$b_{t+1}^i = b_t^i - k^i T b_t^i + w_t^{b,i}$$

with $k^i > 0$, resulting in a first-order Markov process that tends to zero rather than a random walk. This allows the filter to "forget" about biases that are not "reinforced" by observations. All processes denoted w_t^i are white-noise processes of suitable dimension.

2.1.2 Observation Models

There are several observation models, one for each sensor:

$$y_t^{GPS} = \begin{pmatrix} p_t \\ v_t \end{pmatrix}$$

$$y_t^{Mag} = \boldsymbol{R}(q_t) m^{earth}$$

$$y_t^{Baro} = p_t^z$$

where $\boldsymbol{R}(\cdot)$ is the rotation matrix associated to its quaternion argument. m^{earth} is the Earth's magnetic field, assumed to be constant in the flight area. The barometer's pressure and temperature measurement are transformed in software into an altitude, and it is simply assumed that the result is a direct measurement of the altitude to greatly simplify the measurement model.

These equations are linearized and used together with the linearization of the estimation model to construct the prediction and correction steps for an EKF (there is no control input in the estimation model):

$$\hat{x}_t = f(\tilde{x}_{t-1})$$

$$\widehat{\boldsymbol{P}}_t = \boldsymbol{A}(\tilde{x}_{t-1})\widetilde{\boldsymbol{P}}_{t-1}\boldsymbol{A}^\top(\tilde{x}_{t-1}) + \boldsymbol{G}_{t-1}$$

$$z_t = y_t - h(\hat{x}_t)$$

$$\boldsymbol{S}_t = \boldsymbol{C}(\hat{x}_t)\widehat{\boldsymbol{P}}_t\boldsymbol{C}^\top(\hat{x}_t) + \boldsymbol{\Gamma}_{t-1}$$

$$\boldsymbol{K}_t = \widehat{\boldsymbol{P}}_t \boldsymbol{C}^\top(\hat{x}_t)\boldsymbol{S}_t^{-1}$$

$$\tilde{x}_t = \hat{x}_t + \boldsymbol{K}_t z_t$$

$$\widetilde{\boldsymbol{P}}_t = \big(\boldsymbol{I} - \boldsymbol{K}_t \boldsymbol{C}(\hat{x}_t)\big)\widehat{\boldsymbol{P}}_t$$

where $\hat{x}$ is the prior state estimate, $\hat{\boldsymbol{P}}$ is a linear estimator of the prior state covariance matrix, $\boldsymbol{G}$ is the process noise, z is the residual, the matrix $\boldsymbol{S}$ is a linear estimator of the residual covariance, $\boldsymbol{\Gamma}$ is the measurement noise, $\boldsymbol{K}$ is the Kalman gain matrix, $\tilde{x}$ is the posterior state estimate, and $\tilde{\boldsymbol{P}}$ is a linear estimator of the posterior state estimate covariance. $f(\cdot)$ is the dynamics model, and $h(\cdot)$ is the observation model, and the other matrices are as follows:

$$\boldsymbol{A}(x) = \begin{pmatrix} \boldsymbol{I}_3 & T\boldsymbol{I}_3 & \boldsymbol{0}_{3\times4} & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{I}_3 & T\boldsymbol{\nabla}_q[\boldsymbol{R}(q)(a-b^a)] & -T\boldsymbol{R}(q) & 0_{3\times3} \\ \boldsymbol{0}_{4\times3} & \boldsymbol{0}_{4\times3} & \boldsymbol{I}_3 + \frac{T}{2}\boldsymbol{\Omega}(\omega - b^\omega) & \boldsymbol{0}_{4\times3} & -\frac{T}{2}\boldsymbol{Q}(q) \\ \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times4} & (1-Tk^a)\boldsymbol{I}_3 & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times4} & \boldsymbol{0}_{3\times3} & (1-Tk^\omega)\boldsymbol{I}_3 \end{pmatrix}$$

where $\boldsymbol{\Omega}(\omega)$ is the skew-symmetric matrix that induces the right quaternion product with the pure quaternion $(0, \omega)$, $\boldsymbol{Q}(q)$ is a matrix consisting of the last three columns of the skew-symmetric matrix that induces the left quaternion product with respect to q and

$$\boldsymbol{C}(x) = \begin{pmatrix} \boldsymbol{I}_3 & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times4} & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{I}_3 & \boldsymbol{0}_{3\times4} & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{\nabla}_q[\boldsymbol{R}(q)m^{earth}] & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} \\ (\hat{e}_3^3)^\top & 0_3^\top & 0_3^\top & 0_3^\top & 0_3^\top \end{pmatrix}$$

where the terms $\boldsymbol{\nabla}_q[\boldsymbol{R}(q)(a-b^a)], \boldsymbol{\nabla}_q[\boldsymbol{R}(q)m^{earth}]$ are omitted for brevity. These are the Jacobian with respect to the quaternion argument of rotations applied to the bias-free acceleration and Earth's magnetic field, respectively. Here, the vector $(\hat{e}_n^j)$ is the j-th standard unit vector for n-dimensional Euclidean space. Obviously, the biases are not observed directly, but the bias estimates receive information from the sensors through coupling in the dynamics and covariance matrices.

This predictor-corrector is implemented as C++ header file and interfaces with the PX4 flight stack through ROS2. Numerous quality-of-life features are also included, such as functionalities for rejecting spurious sensor data, and detecting and correcting errors in the covariance matrix. Tuning was performed manually.

2.2 Controller Model

A slightly different model is used for the controller. Control is performed at the level of thrust and body rotation rates. The model has the same states (excluding sensor biases, as they are not controlled), but some slightly different assumptions are made on the dynamics:

$$p_{t+1} = p_t + Tv_t$$

$$v_{t+1} = v_t + T\left(\boldsymbol{R}(q_t)\begin{pmatrix}0_2\\u_1\end{pmatrix} + g\right)$$

$$q_{t+1} = q_t + \frac{T}{2}q_t \odot \begin{pmatrix}u_2\\u_3\\u_4\end{pmatrix}_t$$

where $T > 0$ denotes the control update frequency. The model is then linearized about level flight, which is the quaternion $q = (1,0,0,0)$ and the control $u = \left(-||g||, 0,0,0\right)$ with arbitrary position and velocity. The first component of the quaternion must be deleted since in linearization about this operating point, its dynamics are always static. Its deletion facilitates conventional LQR-based control methods, which require system stabilizability for the LQR problem to be feasible. The quaternion q_t is replaced by the Euler angles θ_t. The states are augmented with integral compensation terms to become: $\xi = \left(x^\top, v^\top, \theta^\top, x^{i,\top}, \theta^{i,\top}\right)^\top$ resulting in the dynamics

$$\xi_{t+1} = \bar{\boldsymbol{A}}\xi_t + \bar{\boldsymbol{B}}u_t$$

where

$$\bar{\boldsymbol{A}} = \begin{pmatrix}\boldsymbol{A} & \boldsymbol{0}_{9\times6}\\ -\boldsymbol{C}T & \boldsymbol{I}_6\end{pmatrix}$$

$$\boldsymbol{A} = \begin{pmatrix}\boldsymbol{I}_3 & T\boldsymbol{I}_3 & \boldsymbol{0}_{3\times3}\\ \boldsymbol{0}_{3\times3} & \boldsymbol{I}_3 & -T||g||\boldsymbol{A}_{vq}\\ \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{I}_3\end{pmatrix}$$

$$\boldsymbol{A}_{vq} = \begin{pmatrix}0 & -1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 0\end{pmatrix}$$

$$-\boldsymbol{C} = \begin{pmatrix}\boldsymbol{I}_3 & 0_{3\times3} & 0_{3\times3}\\ 0_{3\times3} & 0_{3\times3} & \boldsymbol{I}_3\end{pmatrix}$$

$$\bar{\boldsymbol{B}} = \begin{pmatrix}\boldsymbol{B}\\ 0_{6\times4}\end{pmatrix}$$

$$\boldsymbol{B} = \begin{pmatrix} 0_3 & \mathbf{0}_{3\times3} \\ \hat{\mathrm{e}}_3^3 & \mathbf{0}_{3\times3} \\ 0_3 & \mathbf{I}_3 \end{pmatrix}.$$

To perform trajectory tracking, a reference state ξ_t^d and a reference control u_t^d are constructed, which are assumed to satisfy

$$\xi_{t+1}^d = \bar{\boldsymbol{A}}\xi_t^d + \bar{\boldsymbol{B}}u_t^d$$

and define the error state $e_t = \xi_t - \xi_t^d$ and error control $\nu_t = u_t - u_t^d$, which gives error system

$$e_{t+1} = \bar{\boldsymbol{A}}e_t + \bar{\boldsymbol{B}}\nu_t$$

and solve the LQR problem on the error system:

$$\mathcal{J}(\nu) = \sum_{t=0}^{\infty} e_t^\top \boldsymbol{Q} e_t + \nu_t^\top \boldsymbol{R}\nu_t$$

for positive-definite matrices $\boldsymbol{Q}, \boldsymbol{R}$, which is done by constructing the discrete-time Hamiltonian matrix and identifying its eigenvectors to construct the solution to the algebraic Riccati equation.[9] The gain matrix is obtained from the solution to the Riccati equation $\boldsymbol{\Pi}$:

$$\boldsymbol{K}^* = (\boldsymbol{R} + \boldsymbol{B}^\top \boldsymbol{\Pi} \boldsymbol{B})^{-1} \boldsymbol{B}^\top \boldsymbol{\Pi} \boldsymbol{A}$$

resulting in a control of form:

$$u_t^* = -\boldsymbol{K}^*\left(\xi - \xi_t^d\right) + u_t^d$$

from which the thrust and rotation rate controls are extracted and clamped to satisfy operating constraints and input to the rate controller in PX4 (or any other flight control system). Since level flight is assumed, the reference control is the control at the operating point defined initially, $u = \left(-||g||, 0,0,0\right)$. Tuning was performed manually.

3. Implementation Modularity

3.1 Filter Modularity

The implementation of the filter is divided into a number of C++ structs. These contain member functions and data types to perform various tasks. The most important structs are the sensor structs, which contain member variables and

functions to store data and perform computations for the EKF. The basic skeleton of these structs is always the following:

```
struct Sensor
{
    bool sensor_updated;
    double update_time;
    double var1;
    double var2;
    double measurement;

    void query(double &update_time, double sensor_in)
    {
        sensor_updated = false;
        if (update_time != previous_time)
        {
            previous_time = update_time;
            sensor_updated = true;
            measurement = sensor_in;
        }
    }
    double idealMeasurement(const VectorXd &state) const
    {
        /// model of sensor here...
    }
    MatrixXd jacobianX(const VectorXd &state) const
    {
        /// Jacobian w.r.t. state of sensor model here...
    }
    /// other member functions ...
};
```

To add a new sensor, one needs to add variables to store data from the relevant data stream, whatever other variables and functions, and populate the function for the measurement model and for the Jacobian. Then, these need to be added to the update step of the filter. Other than sensors, different (Kalman-like) filtering algorithms can be implemented using the sketch of the following struct:

```
struct Estimator
{

    // initialize member variables and various member functions

    // compute Jacobian
    MatrixXd jX(const EstimState &current_state,
                              const SensorMeasurements &measurements, double dt)
    {

    // Implement Jacobian of drift w.r.t. state. Depending on formulation
    // (standard Euclidean, invariant, etc.) the form of the Jacobian changes.

        return Fd;
    }
    MatrixXd makeSensorNoiseMatrix()
    {
        // set up sensor noise matrix, usually it is square and diagonal
        return R;
    }
    MatrixXd makeProcessNoiseMatrix()
    {
        // set up process noise matrix, again usually square and diagonal
        return Q;
    }
    Derivative computeIMU(const EstimState &state,
                                  const Vector3d &measured_accel,
```

```
                                const Vector3d &measured_gyro)
    {


        // this function is where the dynamics are implemented. Different types
        // can be used, such as manifold-based equivalents of
        // of a given system's dynamics
        return deriv;
    }

    void integrateEulerIMU(EstimState &state, const &Derivative &deriv,
                           double dt)
    {
        // The Euler method which propagates the IMU readings.
        // Can be replaced with better integrator, but that will change the
        // DT dynamics model, making the Jacobian much more complicated.
    }
    void predict(Estimator &estim, SensorMeasurements &meas, double dt)
    {
        // The actual prediction step.
    }

    // helper functions for innovation steps here

    void stepEstim(Estimator &estim, SensorMeasurements &meas,
                   Accelerometer &acc, Gyroscope &gyro, GPS &gps,
                   Magnetometer &mag, Barometer &baro, OpticalFlow &flow,
                   Radio &radio, double dt)
    {
        predict(estim, meas, dt);

        // perform state update step for your given algorithm here

    };
```

3.2 Controller Modularity

The controller is a single struct that contains all relevant member variables and functions.

```
class Controller {

private:
  // various private variables
  bool is_initialized = false;
  MatrixXd K;
  double dt;



public:
  // put required public variables here

  bool initialize(double dt_in) {
    // various controller initialization code, e.g. PID parameters, or code needed
    // to compute LQR solution, or whatever controller you want

    is_initialized = true;
    return is_initialized;
  }

  ControlCommand update(const DroneState &state, const DroneState &target) {
    // put whatever code you need to compute a control output
    return cmd;
  }

};
```

There are two important functions in this code: the initialization and the control update. Currently, this code is only structured for modularity in terms of controllers whose parameters are set or calculated once, but modifications can be made easily to incorporate adaptive methods or finite-horizon methods that need to be updated regularly.

4. Evaluation in Simulation

This work successfully implemented an EKF and LQR controller for trajectory tracking in the PX4 flight stack in Gazebo simulation. A state machine for takeoff and flight, return to home, and landing phases was developed to send setpoints corresponding to various behaviors. There is some overshoot in takeoff to the flight height, but it is quickly corrected, and the trajectory matching for the flight phase is generally good, accurate to within 0.5 m. Tracking errors appear to originate in the state estimator and are propagated through the controller into the drone's state. The error does not lead to any wild oscillation or instability; it appears to mostly be propagated as an offset from the desired position, as shown in Figures 1–3.

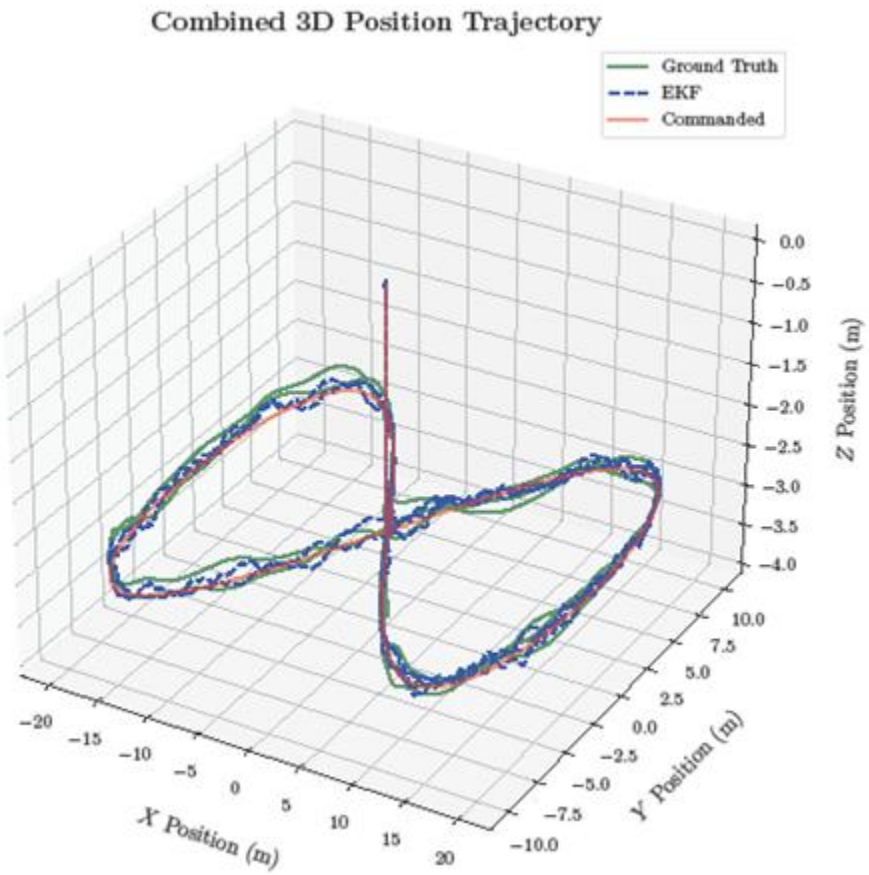

Figure 1. 3D plot of flight. The position is estimated by the EKF given in Section 2.1. The controller in Section 2.2 transforms estimates and reference position, velocity, and orientations into thrust and rotation rate commands fed into PX4's rate controller.

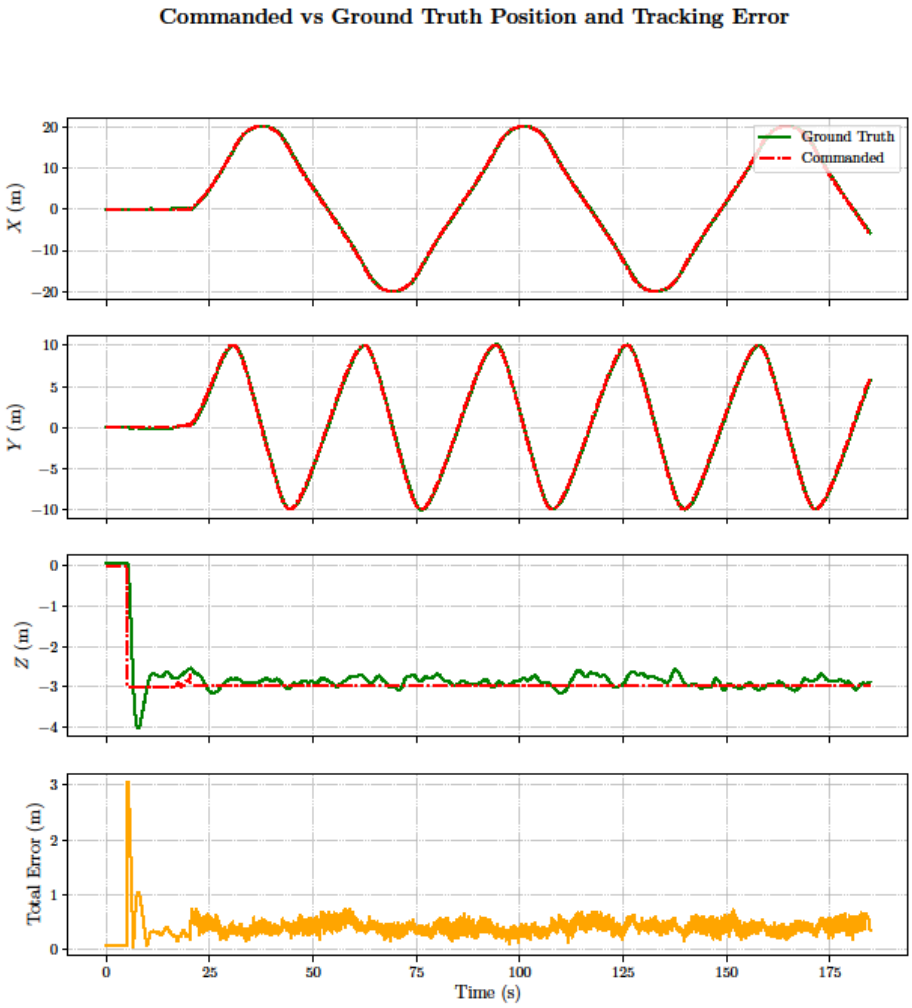

Figure 2. Tracking results. The command setpoints are fed with the state estimates to the linear-quadratic-integral controller. The tracking error is generally below 1 m and is mostly due to fluctuations in altitude. The source of that error in altitude seems to originate from the state estimator.

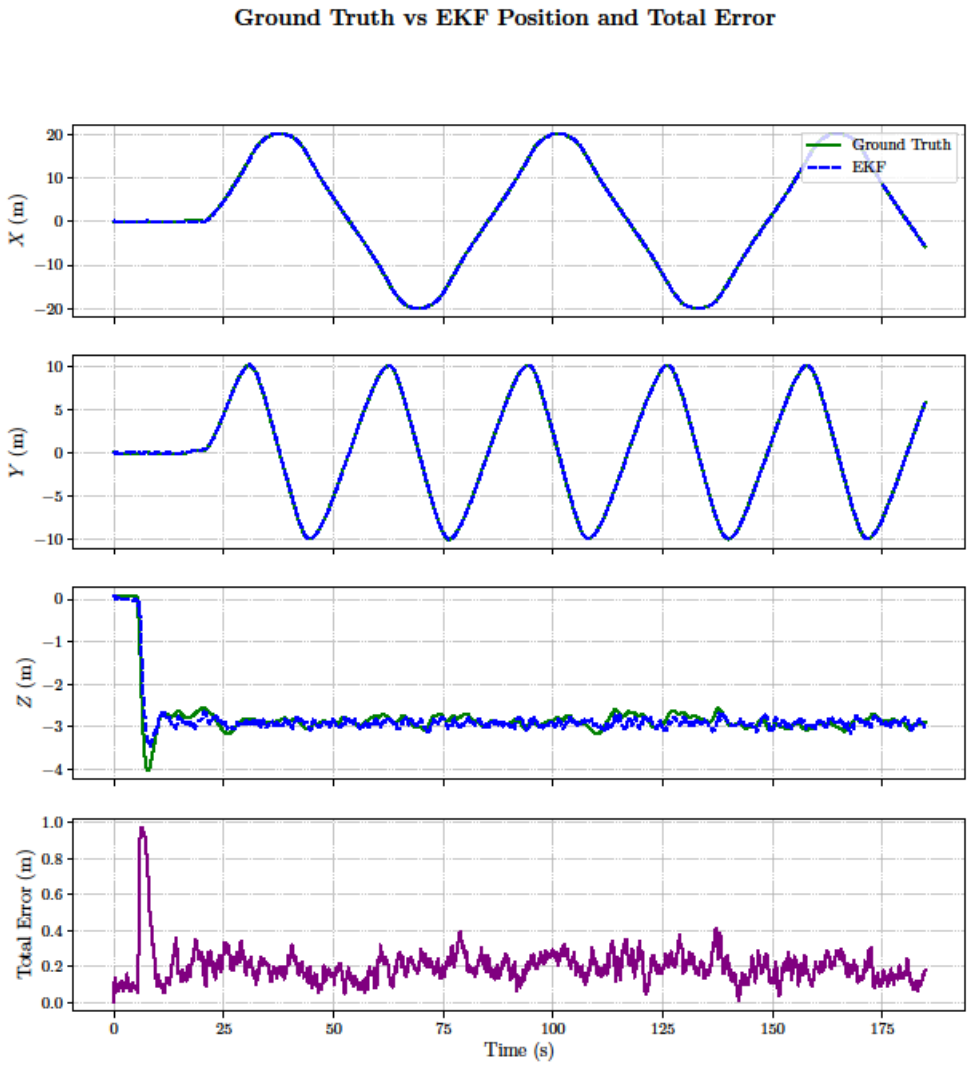

Figure 3. Estimation results. The state reflects the ground truth very well, generally accurate within 0.5 m. Most of that error seems to come from the altitude estimates.

5. Conclusion

A state estimator and controller package was developed and successfully evaluated in simulation. The implementation leaves room for new sensors to be integrated in a convenient way, and for new estimation and control methods to be implemented in place of those currently developed.

Substantial difficulty was encountered with the PX4 flight stack when attempting to transition the controller from simulation to physical implementation. Initial attempts used a ModalAI Starling 2 Max platform equipped with a VOXL2. The hardware is capable of running this estimator and controller at a high rate, but PX4 has numerous safety features and fail-safes that present some obstacles. Many of these fail-safes are tied to sensor and estimator data streams.

The approach via PX4 worked in simulation since there are always reliable, simulated data streams; hence, the EKF2 is able to produce accurate estimates immediately, so no fail-safes are triggered.

In the attempts to transition to real-world flight, numerous fail-safes were triggered in the arming process, apparently due to issues in the various data streams. It is not exactly clear what causes these issues, but the result is rapid chattering between offboard flight and altitude hold modes caused by PX4 fail-safes and the controller's interface nodes, leading to totally uncontrollable flight.

Workarounds were attempted, but it does not seem possible to fully disable these fail-safes in current versions of PX4. To make progress in this area, it is necessary to either restructure the controller software interface with PX4 or use different flight control software.

6. References

1. Meier L, Honegger D, Pollefeys M. PX4: a node-based multithreaded open source robotics framework for deeply embedded platforms. Proceedings of 2015 IEEE International Conference on Robotics and Automation (ICRA); 2015 May 26–30; Seattle, WA. IEEE; 2015. p. 6235–6240. https://doi.org/10.1109/ICRA.2015.7140074

2. Macenski S, Foote T, Gerkey B, Lalancette C, Woodall W. Robot Operating System 2: design, architecture, and uses in the wild. Sci Robot. 2022;7(66):eabm6074. https://doi.org/10.1126/scirobotics.abm6074

3. Lee T, Kim HJ. Invariant Kalman filter application to optical flow based visual odometry for UASs. In: 2019 International Conference on Unmanned Aircraft Systems (ICUAS); 2019 June 11–14; Atlanta, GA. IEEE; 2019. p. 1473–1478. https://doi.org/10.1109/ICUAS.2019.8798313

4. Bstaoros K. Kalman filtering for UAS navigation using optical flow [master's thesis]. Politecnico di Milano; 2020. https://hdl.handle.net/10589/153128

5. Brommer C, Jung R, Steinbrener J, Weiss S. MaRS: a modular and robust sensor-fusion framework. IEEE Robot Autom Lett. 2021;6(2):359–366. https://doi.org/10.1109/LRA.2020.304319

6. Appapogu RD. Autonomous navigation in GPS denied environments using MPC and LQR with potential field based obstacle avoidance [master's thesis]. Colorado School of Mines (US); 2019. https://hdl.handle.net/11124/172896

7. Bonazza MC. Implementation of adaptive nonlinear model predictive control on a PX4-enabled quad-rotor platform [master's thesis]. University of Padua; 2023. https://hdl.handle.net/20.500.12608/46069

8. Amadi CA. Design and implementation of model predictive control on Pixhawk flight controller [master's thesis]. Stellenbosch University; 2018. http://hdl.handle.net/10019.1/105078

9. Anderson BDO, Moore JB. Optimal control: linear quadratic methods. Prentice-Hall; 1990.

List of Symbols, Abbreviations, and Acronyms

3D	Three-Dimensional
ARL	Army Research Laboratory
DEVCOM	U.S. Army Combat Capabilities Development Command
EKF	Extended Kalman Filter
GPS	Global Positioning System
IO	Input/Output
LQR	Linear–Quadratic Regulator
MPC	Model Predictive Control
PID	Proportional–Integral–Derivative
RF	Radio Frequency
ROS	Robot Operating System
UAS	Unmanned Aircraft Systems

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DEVCOM ARL FCDD RLB CB TECH LIB