The University of Texas at Austin
Aerospace Engineering and Engineering Mechanics
Cockrell School of Engineering

October 2023

USRC F23 – DESIGN REVIEW #1

374L: Vincent Spada, Mahi Juthani, W.J. Morrison
374K: Dao Ton-Nu, Jose Rodriguez, Ryan Mok, Evan Sayer, Azeem Bhaiwala

MISSION INTRODUCTION

ASE 374K/L
CAPSTONE DESIGN SEQUENCE
PROFESSOR: ADAM NOKES
TEACHING ASSISTANT: APOORVA KARRA

TEXAS DRONE ESTIMATION LAB
A NASA USRC MISSION

TEXAS DRONE ESTIMATION LAB - FALL 2023 TEAM

Mahi Juthani

W. J. Morrison

Vincent Spada

Ryan Mok

Dao Ton-Nu

Evan Sayer

Azeem Bhaiwala

Jose Rodriguez

Need Statement

Develop drones capable of providing real-time trajectory data to a ground control computer and develop an EKF based data processing and visualization system.

GOALS

1. Manufacture drone prototypes capable of manual flight.
2. Develop real-time data acquisition, processing, and visualization system for simulated and actual data.
3. Incorporate an Extended Kalman Filter for state estimation and uncertainty quantification.
4. Develop testing equipment and a flight test plan; learn about flight test safety.
5. Admin: collaborate effectively, integrate new members, complete key deliverables, spend grant funding, plan for the future.

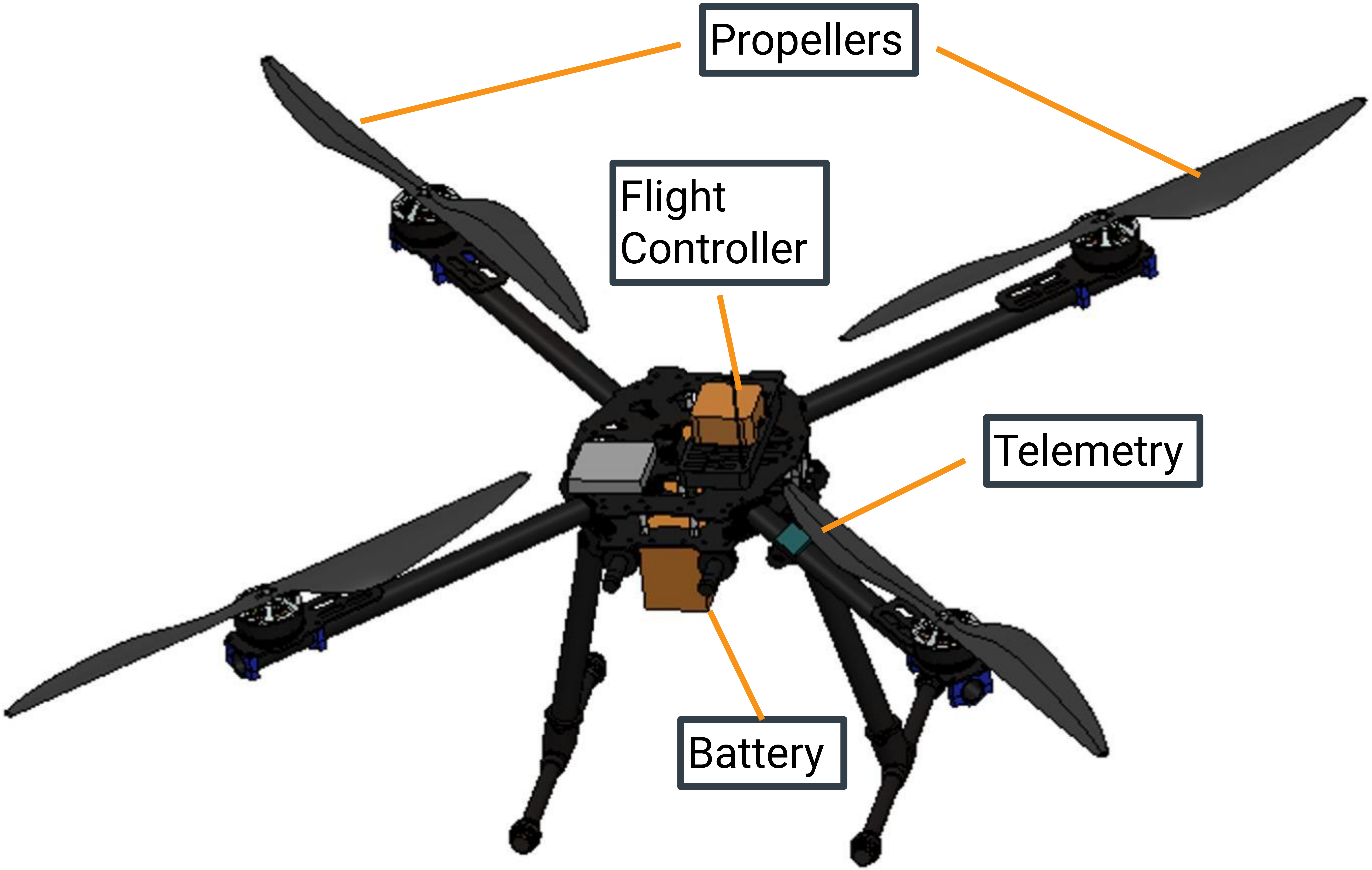


Figure 1: Manual flight prototype.

TABLE OF CONTENTS

Admin Updates - Vincent

Dates

1. **September 11:** NASA update (Steven Holz)
2. **Spend funds by end of October:** Grant details negotiated
3. **September 25:** New members joined
   - **Collaboration:** members choose what they want to work on each meeting based on what needs to be done. We all meet in the same room for roughly the same time.
4. **Funds:** $12,000 spent for hardware, $9,000 marked for laptops, $22,000 remaining
5. **Deliverables for October 30:** update website

Semester Metrics

1. Completed first data acquisition
2. Prewritten MATLAB UKF to ROS integrated EKF
3. No real-time hardware to multiple designs
4. Zero simulation to full quadcopter simulation
5. Zero ROS for real-time to working ROS
6. Zero data visualization to multiple designs
7. 1 drone to at least 5 custom drones

September flight test

Fall 23 Budgeting - Mahi

Components ordered

1. Hardware components for 4 drones have been ordered

Component	Amount	Cost	Total cost
Holybro X500 V2 ARF Drone frame	4	284.99	1139.96
M8N GPS	4	89.99	359.96
Extra Flight controller wires	2	18.99	37.98
Anti-static soldering mat	2	20.99	41.98
Solder wire	2	8.99	17.98
Pixhawk 6X flight controller baseboard	4	89.99	359.96
Pixhawk 6X module	4	389.99	1559.96
Lipo batteries	4	59.99	239.96
Pixhawk cable set	4	21.99	87.96
Telemetry short range module	4	76.9	307.6
Power connector	4	29.99	119.96
Herelink radio controller	2	899	1798
Herelink extra air unit	2	500	1000
Jetson nano developer kit	4	179.99	719.96
Zed X camera w/ GMSL2 card	4	998	3992
		Total	11783.22

Drone frame kit: Holybro S500 ARF V2

The Holybro S500 ARF V2 Frame Kit comes with a frame, landing gear, motors, ESC’s and propellers.

improvements in this drone frame:

1. Pre-installed ESC and PDB: This means we will have less soldering to do ourselves which reduced chances of shoddy soldering job affecting performance.

2. Mount for NVIDIA Jetson Nano companion computer: Since we are using that exact Jetson model, it's helpful to have a pre-existing mount.

3. Depth camera mount: In our case, for the Zed X camera

Pixhawk 6X flight controller

Holybro Pixhawk 6X Flight computer

Flight Controller: This is the brain of the drone, responsible for stabilizing and controlling the aircraft's movements in response to user inputs and external factors.

1. Recommended FC with our chosen drone frame,
2. Vibration isolated triple redundant IMU

Radio Controller

Herelink radio controller and air-unit

Integrated radio controller and ground station.

Industry standard for radio controllers.

Herelink allows RC control and telemetry data to be transmitted up to 20 km between the air unit (on the drone) and controller

GPS

Holybro DroneCAN M8N GPS

We want GPS for 2 reasons:

1. Keep our error ellipsoid from expanding too quickly
2. Program autonomous missions for future sensor testing efforts or data collection

Once installed, our groundstation allows us to enable/disable GPS

NVIDIA Jetson

NVIDIA Jetson Orin Nano Developer Kit

NVIDIA Jetson Nano: The NVIDIA Jetson Nano is a small, high-performance computer In a drone, it can be used for onboard processing of complex algorithms, such as our kalman filter algorithm for error ellipsoids.

ZED X mini Stereo Camera

ZED X mini Stereo Camera

Switched from the Zed 2 Stereo Camera to the ZED X mini for weight and size reduction.

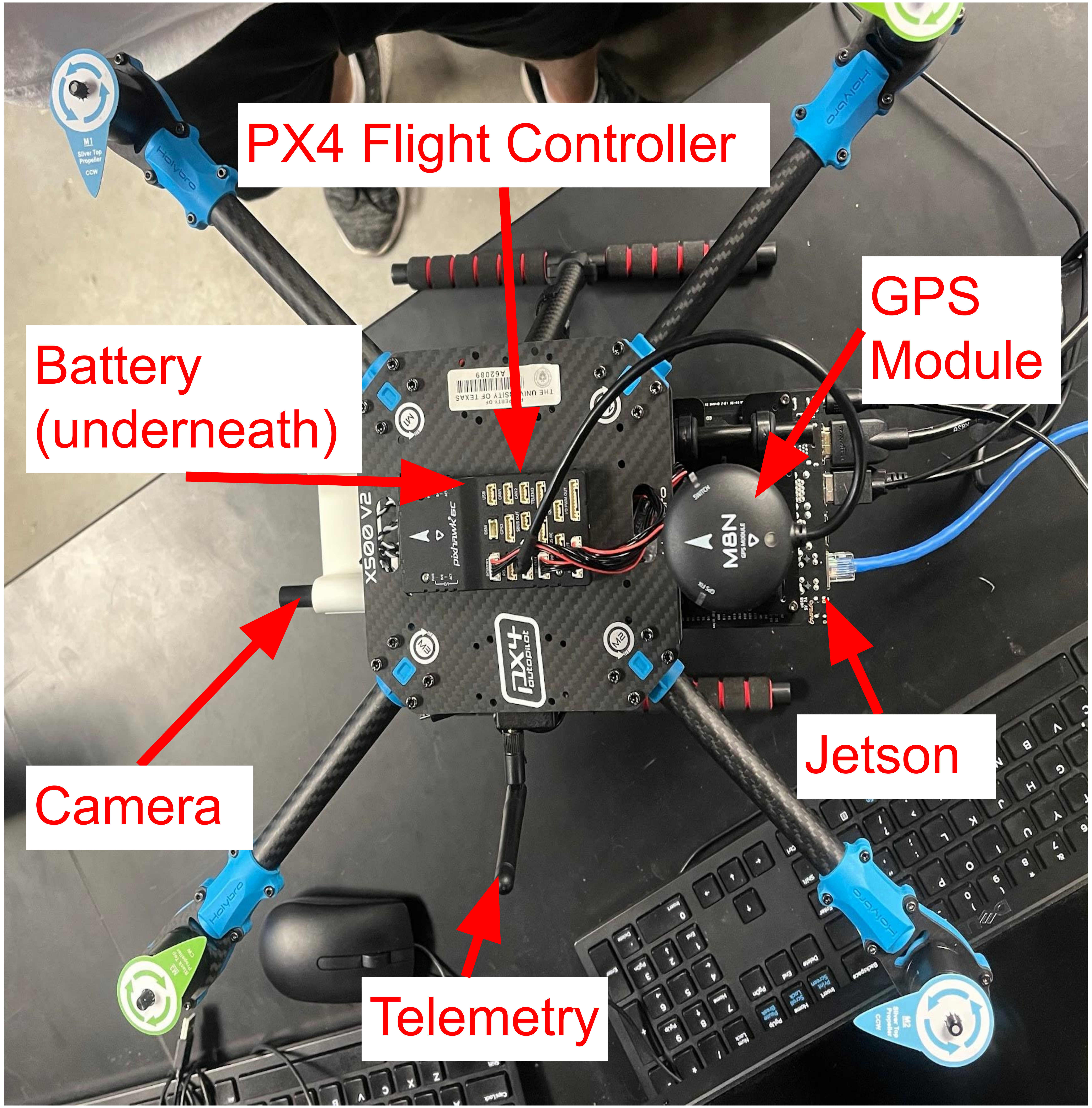
Stereo cameras are used for depth sensing, it also has an onboard IMU that we can use to confirm our position estimate from our error ellipsoid.

Hardware Outlook - Ryan

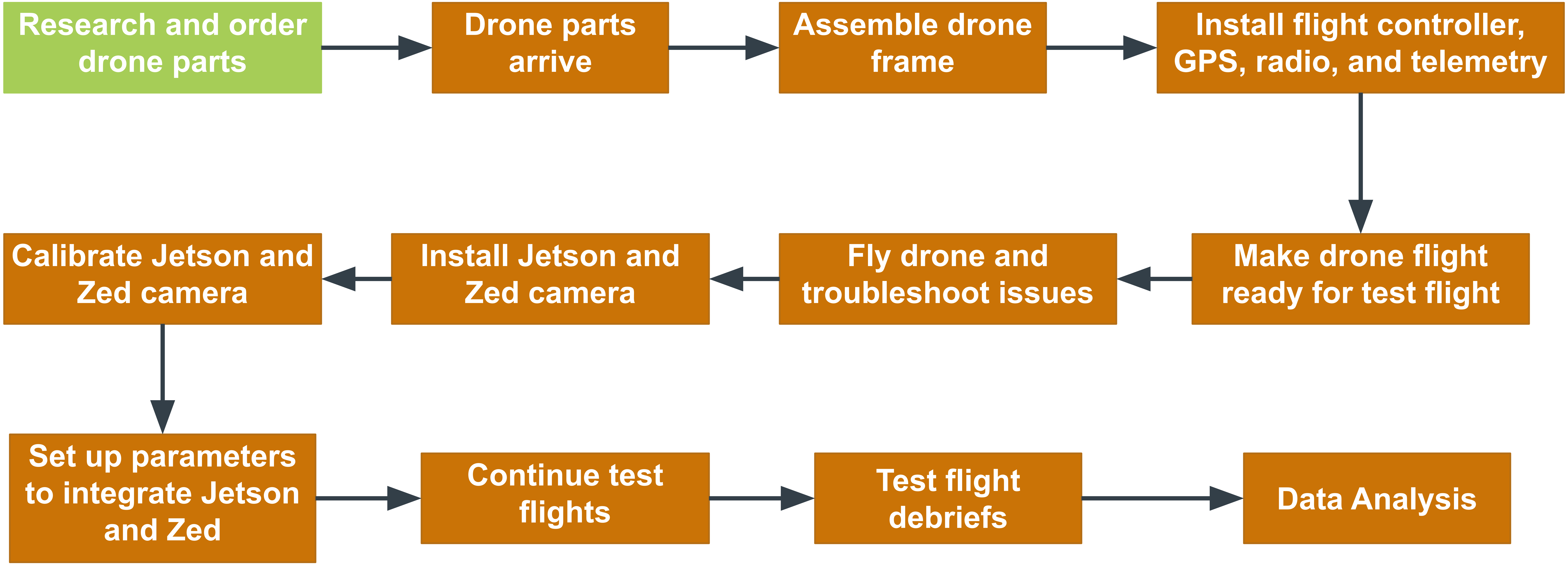
New Drone Hardware Configuration

The old drone design is being revamped to allow for the addition of new sensors and different drone designs.

Ryan



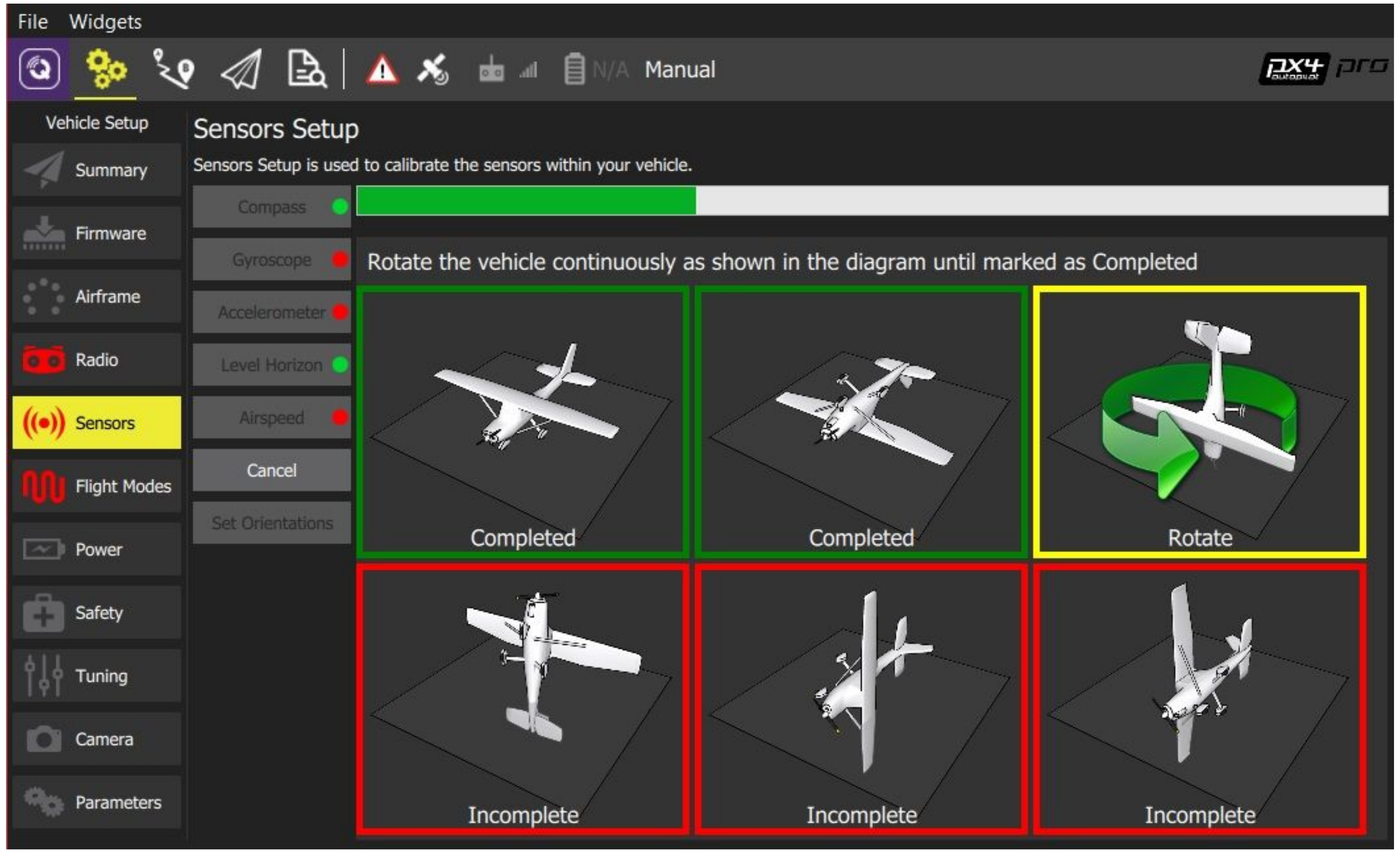
New Drone Process Flowchart and Flight Checklist



Ryan

Hardware Pre-Flight Checklist

1. Charge battery and install on drone.
2. Calibrate drone using QGroundControl (see right).
3. Calibrate radio to set threshold throttle limits.
4. Set up kill switch (kills power to motors).
5. Set up flight modes and arm/disarm buttons.
    a. Manual, altitude, position
6. Install propellers.
7. Place drone in cage and fly!



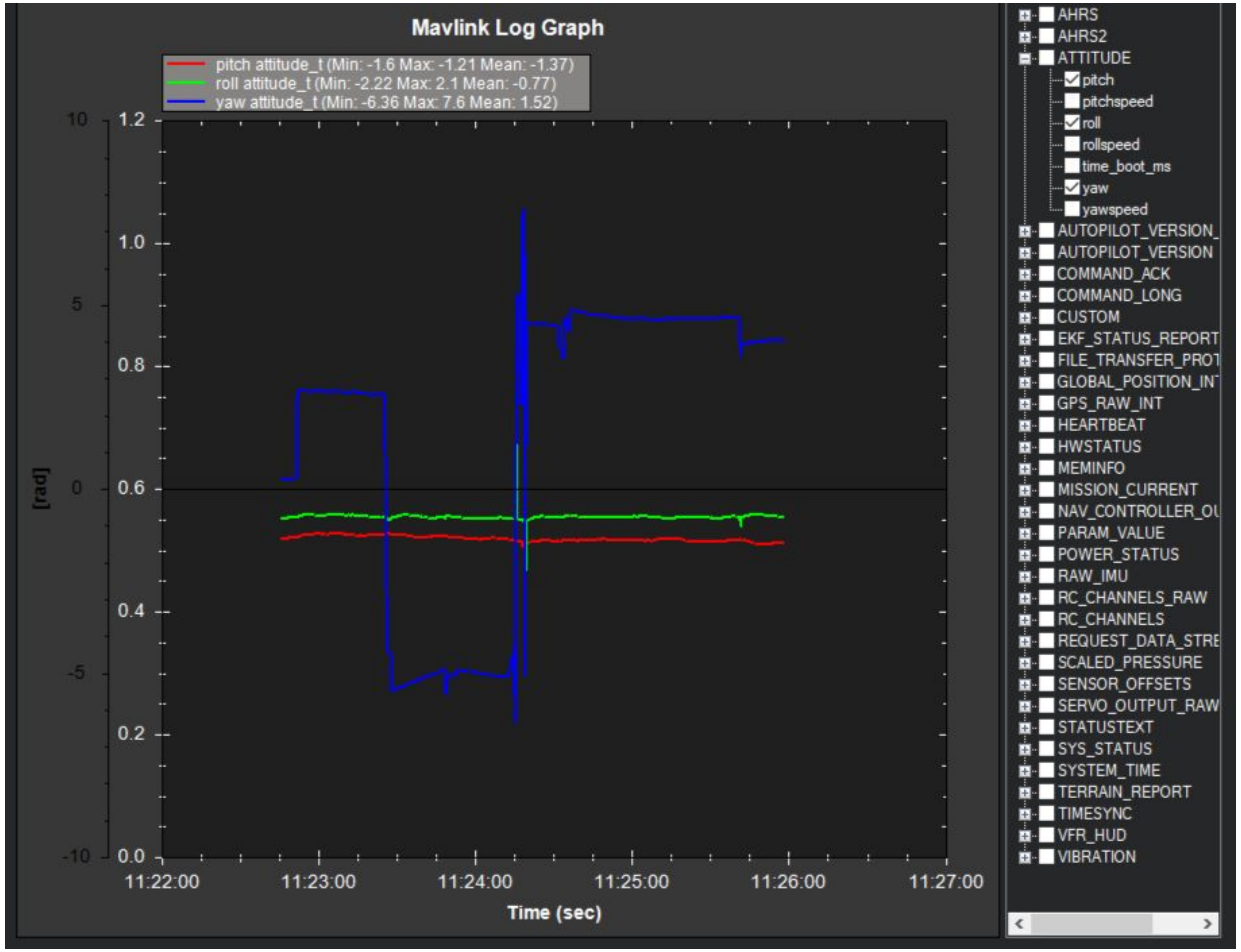
QGC Calibration View

Ryan

Data Acquisition - Jose

Data Acquisition

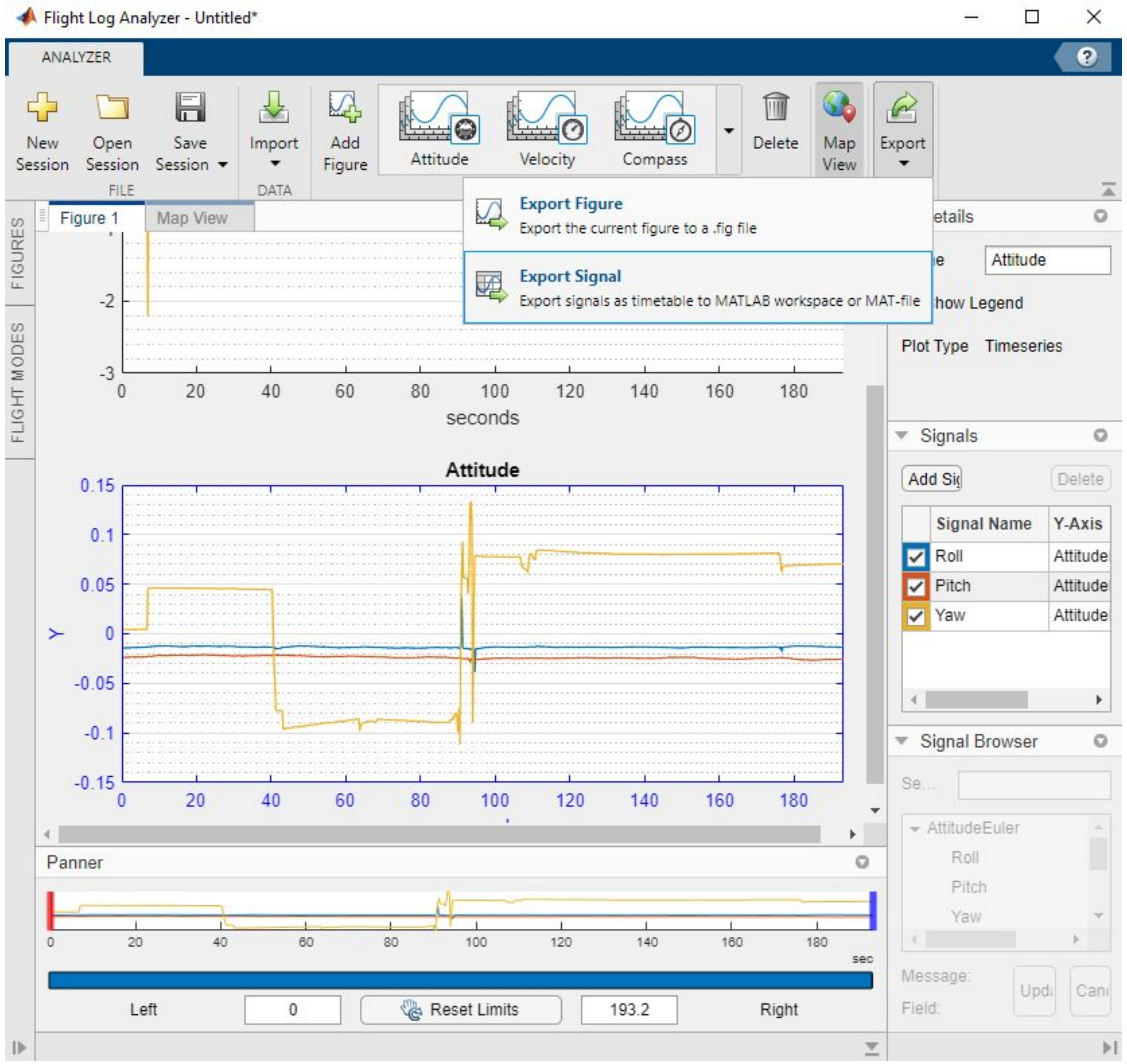
- **Data types:**
  - Mission Planner:
    - tlog file from telemetry
    - bin file on SD card
  - Reviewing log:
    - Data graphs
    - Starting parameters only
    - Does not show timetables



Jose

Data Acquisition

- **Data Acquisition:**
  - Matlab UAV toolbox:
    - flight log analyzer
    - import tlog, ulog, or mat data
    - export timetables as mat data
  - Mat data converted to data needed for estimation

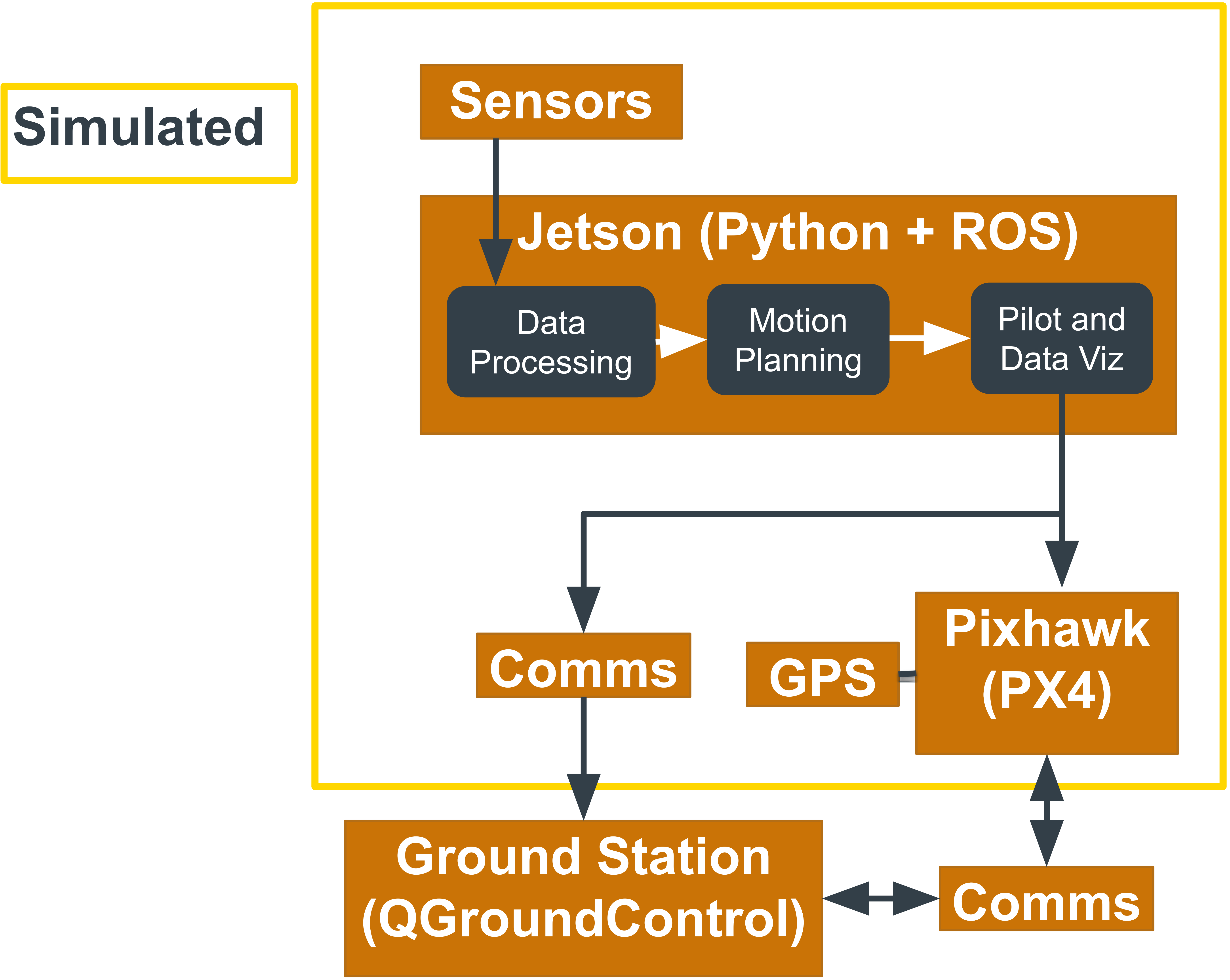


Jose

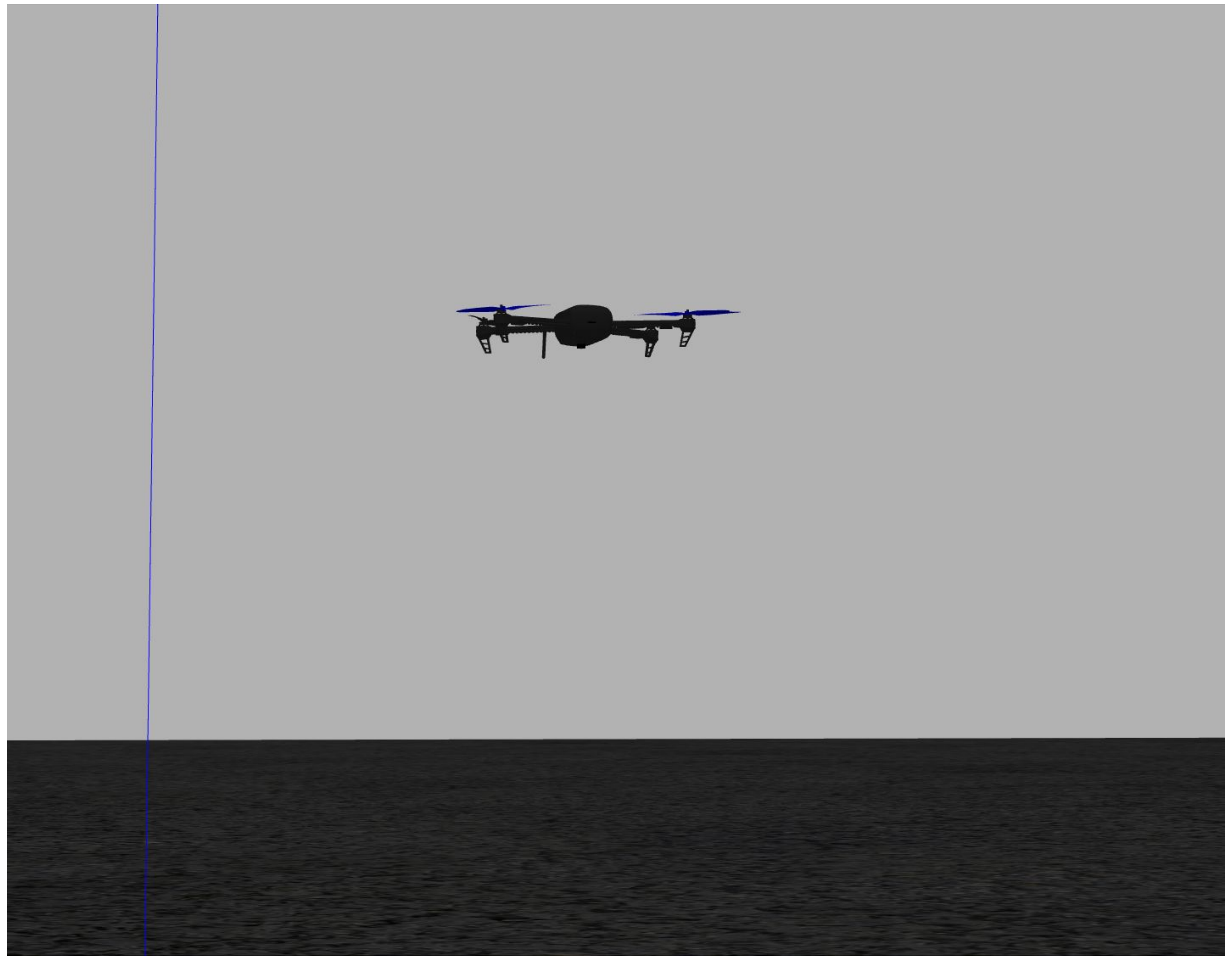
PX4 Simulation - Azeem

Simulation

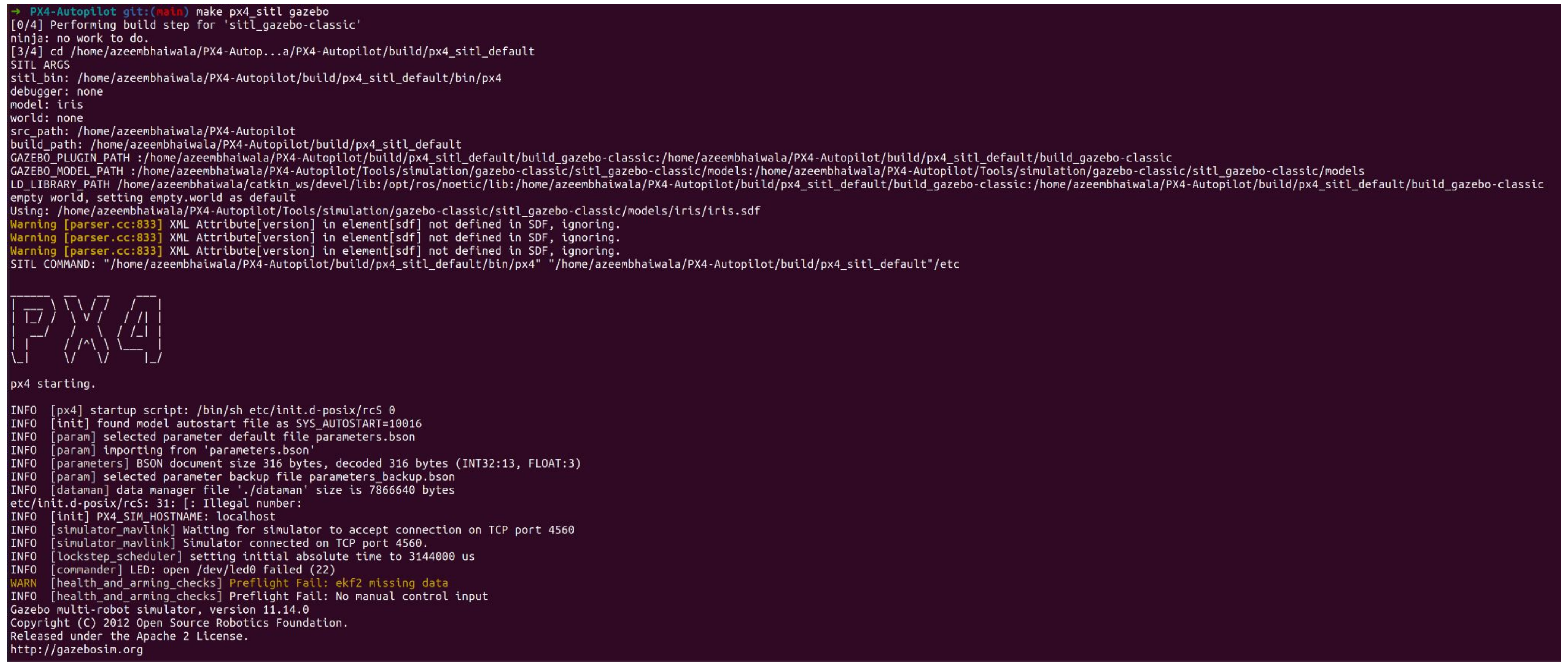
- **Software-In-The-Loop**
  - Simulated controller running PX4, with generic drone frame
  - Simulates sensor inputs, disturbances, and environment using Gazebo
  - Uses real control loops
  - Simulates comms with ROS and connects to QGC

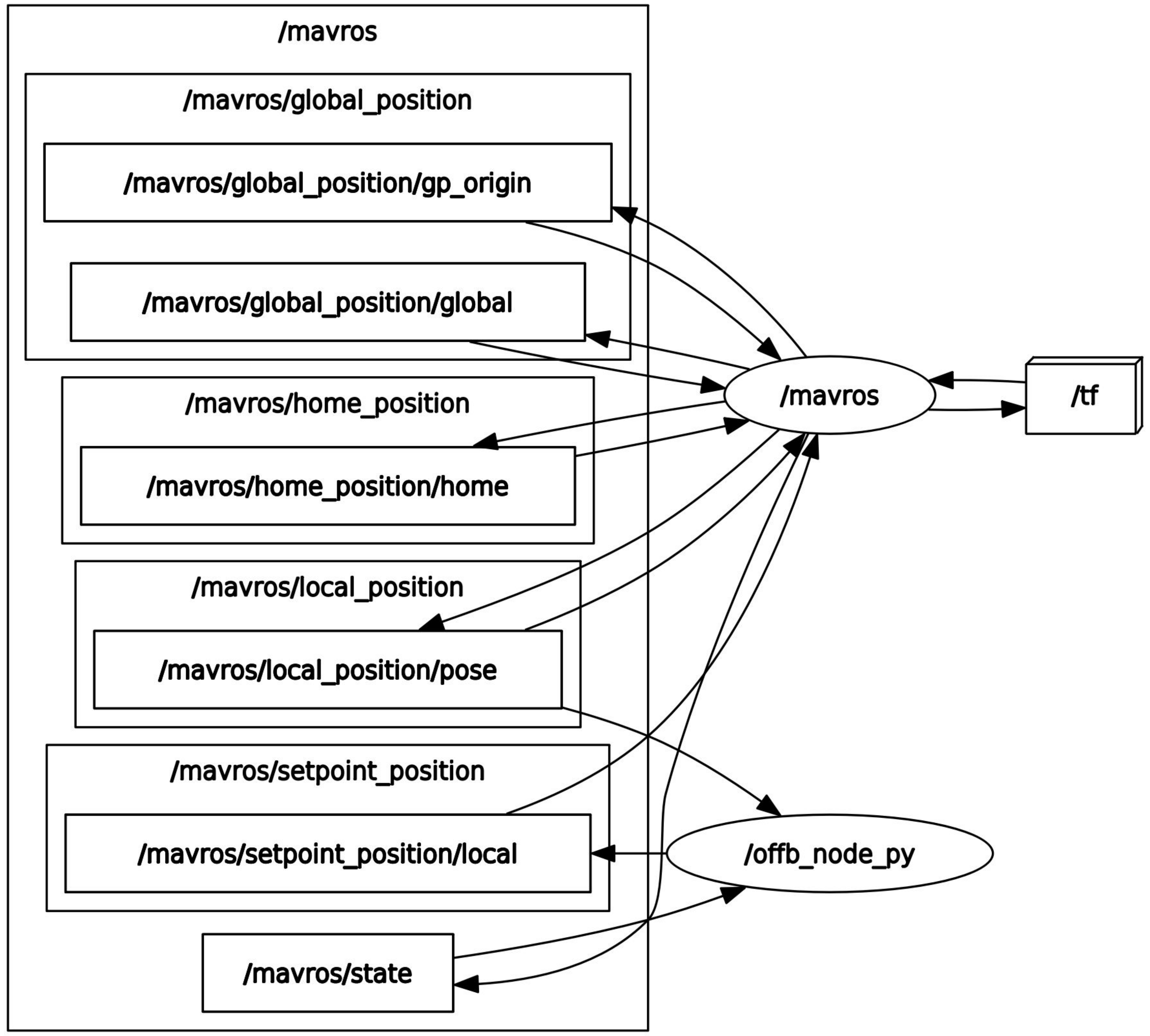


Hardware software integrations where some parts are within the scope of hardware and some should be provided by sims and estimation.



Simulated drone in gazebo simulation





MAVROS connections diagram

Simulation

- **Extracting Data from Sim**
  - To investigate how we could get data from the real drone, we first did it in the simulation using Python.
  - Used mavros subscriber links to get real-time position data from the simulated drone

offb_node.py

home > azeembhaiwala > catkin_ws > src > offboard_py > scripts > offb_node.py > ...

```python
#! /usr/bin/env python3

"""
 * File: offb_node.py
 * Stack and tested in Gazebo Classic 9 SITL
"""


import rospy
from geometry_msgs.msg import *
from mavros_msgs.msg import *
from mavros_msgs.srv import CommandBool, CommandBoolRequest, SetMode, SetModeRequest

current_state = State()

current_pose = Pose()

def state_cb(msg):
    global current_state
    current_state = msg

#new
def pose_cb(msg):
    global current_pose
    current_pose = msg.pose.position

if __name__ == "__main__":
    rospy.init_node("offb_node_py")

    state_sub = rospy.Subscriber("mavros/state", State, callback = state_cb)

    local_pos_pub = rospy.Publisher("mavros/setpoint_position/local", PoseStamped, queue_size=10)

    #new
    local_pos_sub = rospy.Subscriber("mavros/local_position/pose", PoseStamped, pose_cb)

    rospy.wait_for_service("/mavros/cmd/arming")
    arming_client = rospy.ServiceProxy("mavros/cmd/arming", CommandBool)

    rospy.wait_for_service("/mavros/set_mode")
    set_mode_client = rospy.ServiceProxy("mavros/set_mode", SetMode)


    # Setpoint publishing MUST be faster than 2Hz
    rate = rospy.Rate(20)

    # Wait for Flight Controller connection
    while(not rospy.is_shutdown() and not current_state.connected):
        rate.sleep()

    pose = PoseStamped()
```

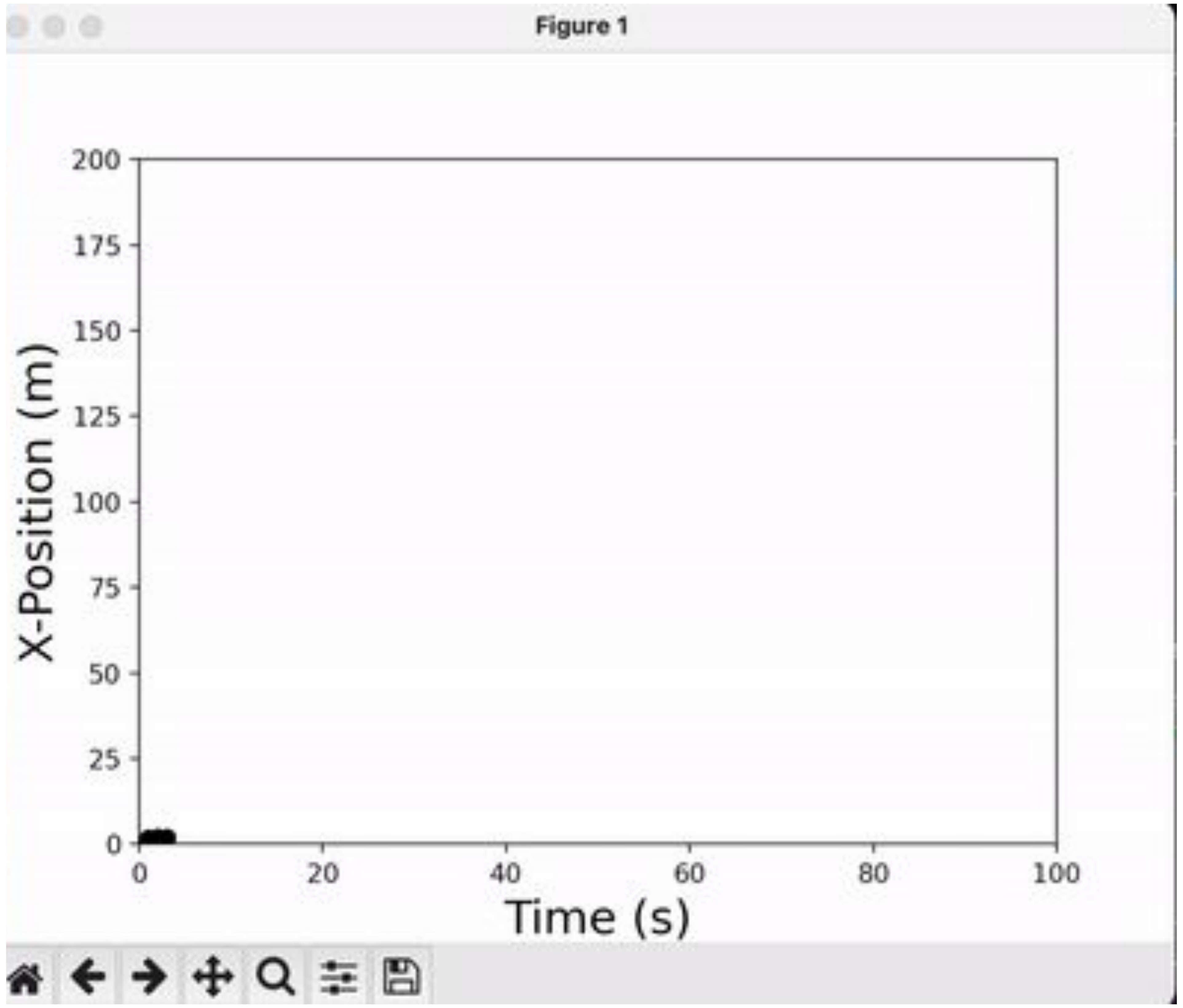
```python
while(not rospy.is_shutdown()):
    if(current_state.mode != "OFFBOARD" and (rospy.Time.now() - last_req) > rospy.Duration(5.0)):
        if(set_mode_client.call(offb_set_mode).mode_sent == True):
            rospy.loginfo("OFFBOARD enabled")

        last_req = rospy.Time.now()
    else:
        if(not current_state.armed and (rospy.Time.now() - last_req) > rospy.Duration(5.0)):
            if(arming_client.call(arm_cmd).success == True):
                rospy.loginfo("Vehicle armed")

            last_req = rospy.Time.now()
    #new
    rospy.loginfo(current_pose.z)

    local_pos_pub.publish(pose)

    rate.sleep()
```

```
[INFO] [1698085700.631991]: 1.9794886112213135
[INFO] [1698085700.681875]: 1.9800797700881958
[INFO] [1698085700.731825]: 1.979805827140808
[INFO] [1698085700.781663]: 1.9794824123382568
[INFO] [1698085700.831712]: 1.979361653327942
[INFO] [1698085700.881896]: 1.9795454740524292
[INFO] [1698085700.931651]: 1.9801539182662964
[INFO] [1698085700.981808]: 1.9804906845092773
[INFO] [1698085701.031904]: 1.9812999963760376
[INFO] [1698085701.081853]: 1.981763243675232
[INFO] [1698085701.131728]: 1.982410192489624
[INFO] [1698085701.181918]: 1.9825834035873413
[INFO] [1698085701.231714]: 1.9830780029296875
[INFO] [1698085701.281910]: 1.9833555221557617
[INFO] [1698085701.331913]: 1.9841479063034058
[INFO] [1698085701.381867]: 1.9845844507217407
[INFO] [1698085701.431782]: 1.985502004623413
[INFO] [1698085701.481880]: 1.986132025718689
[INFO] [1698085701.531848]: 1.986053705215454
[INFO] [1698085701.582061]: 1.9859437942504883
[INFO] [1698085701.631739]: 1.9858952760696411
[INFO] [1698085701.681899]: 1.9860705137252808
[INFO] [1698085701.731681]: 1.9867432117462158
[INFO] [1698085701.781974]: 1.9871230125427246
[INFO] [1698085701.831878]: 1.9881565570831299
[INFO] [1698085701.881613]: 1.9888299703598022
[INFO] [1698085701.931829]: 1.9895918369293213
[INFO] [1698085701.981806]: 1.9899969100952148
[INFO] [1698085702.031782]: 1.990868091583252
[INFO] [1698085702.081910]: 1.9913616180419922
[INFO] [1698085702.131591]: 1.9925137758255005
[INFO] [1698085702.181924]: 1.9929438829421997
[INFO] [1698085702.231859]: 1.9939210414886475
[INFO] [1698085702.281985]: 1.9944050312042236
[INFO] [1698085702.331837]: 1.9946223497390747
[INFO] [1698085702.382205]: 1.9945704936981201
[INFO] [1698085702.431978]: 1.9946296215057373
[INFO] [1698085702.481971]: 1.9947750568389893
[INFO] [1698085702.531945]: 1.995481252670288
[INFO] [1698085702.581930]: 1.9959743022918701
[INFO] [1698085702.631930]: 1.9968156814575195
[INFO] [1698085702.682054]: 1.99722158908844
[INFO] [1698085702.731823]: 1.9990177154541016
[INFO] [1698085702.781955]: 2.0000035762786865
[INFO] [1698085702.831912]: 2.0018603801727295
[INFO] [1698085702.881958]: 2.0026979446411133
[INFO] [1698085702.931840]: 2.004333019256592
[INFO] [1698085702.982013]: 2.0047850608825684
[INFO] [1698085703.031897]: 2.0052995681762695
[INFO] [1698085703.081869]: 2.005551338195801
[INFO] [1698085703.131854]: 2.00693678855896
[INFO] [1698085703.181967]: 2.0077524185180664
[INFO] [1698085703.231971]: 2.0092034339904785
[INFO] [1698085703.281999]: 2.0097994804382324
[INFO] [1698085703.331843]: 2.0110104084014893
[INFO] [1698085703.382195]: 2.011500835418701
```

Data Visualization - Vincent

matplotlib

- **Easy plotting in Python:**
  - Compatible directly with ROS1, EKF pipeline
  - Multiple options:
    - For loop updating frames
    - FuncAnimation
  - Surface plots available
- **Moving forward:** plotting ellipsoids

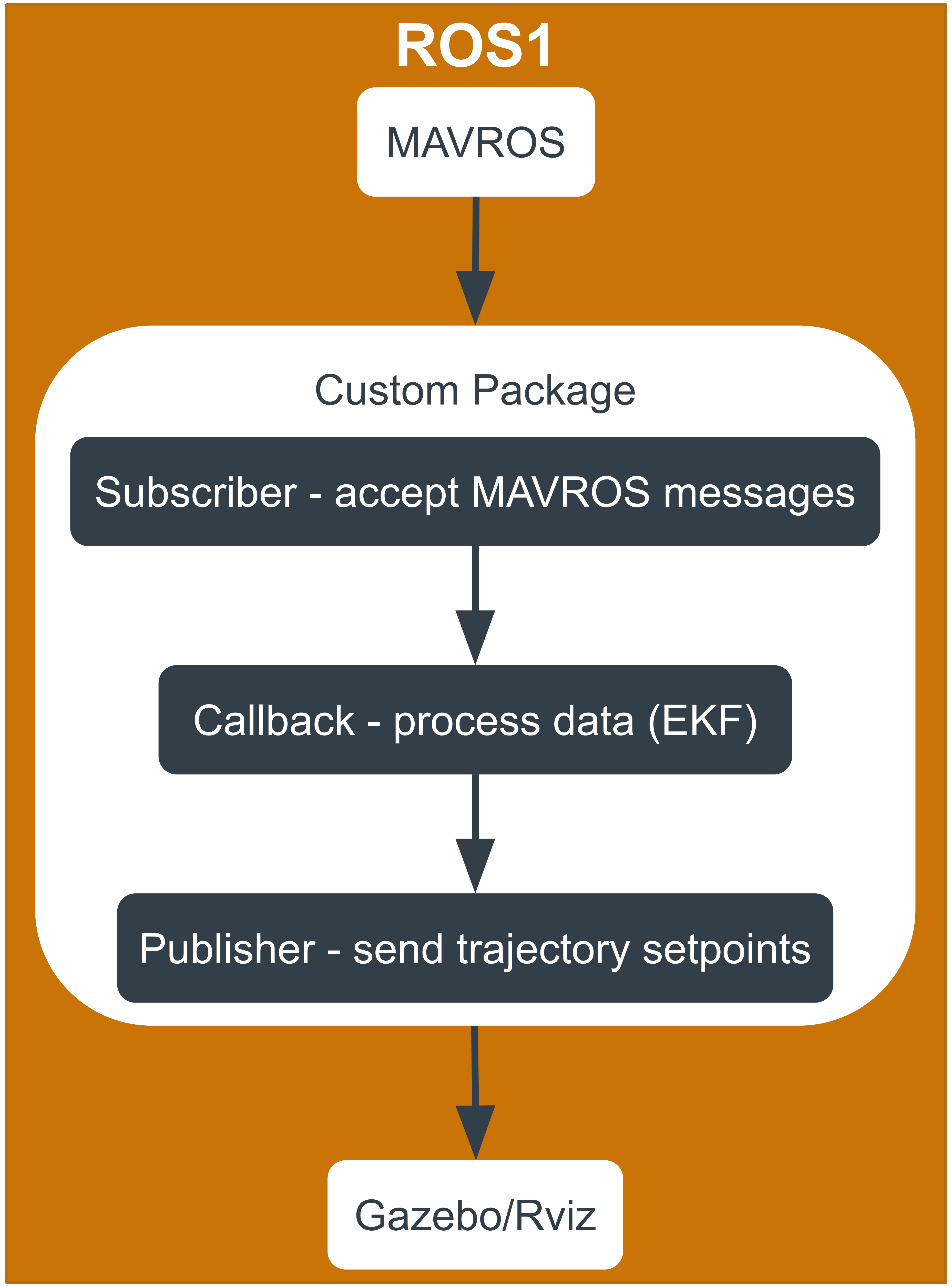


Example trajectory animation made using random trajectory and matplotlib

Gazebo or Rviz

- **Issues:**
  - Need robot description (.urdf, .sdf file) - output by SolidWorks?
  - Need robot inertia
- **Gazebo:**
  - Already able to send real-time trajectory commands
  - Need to update .urdf
- **Rviz:**
  - Faster, cheaper

Vincent

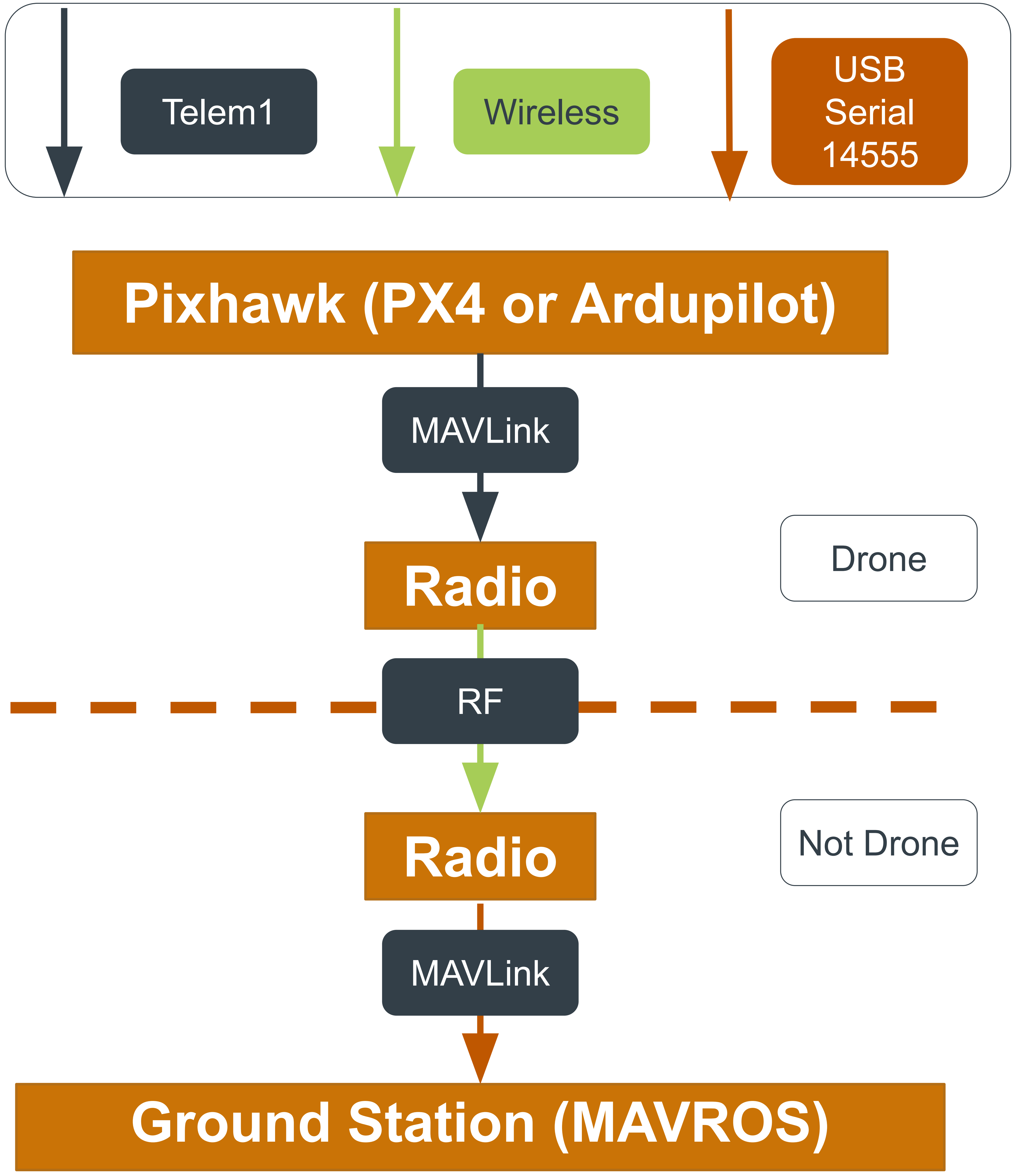


Data visualization pipeline from MAVROS to Gazebo/Rviz. Valid for simulation and actual data.

Real-Time Data Acquisition - Vincent

Telemetry Radio Based

- **Pros:**
  - Hardware is easy to use and cheap
  - Adequate sampling rate
- **Cons:**
  - Difficult to set-up
  - All of Oct. 16 spent troubleshooting
- **Moving forward:**
  - Test connection to MAVROS, MAVProxy
  - Drivers on Linux

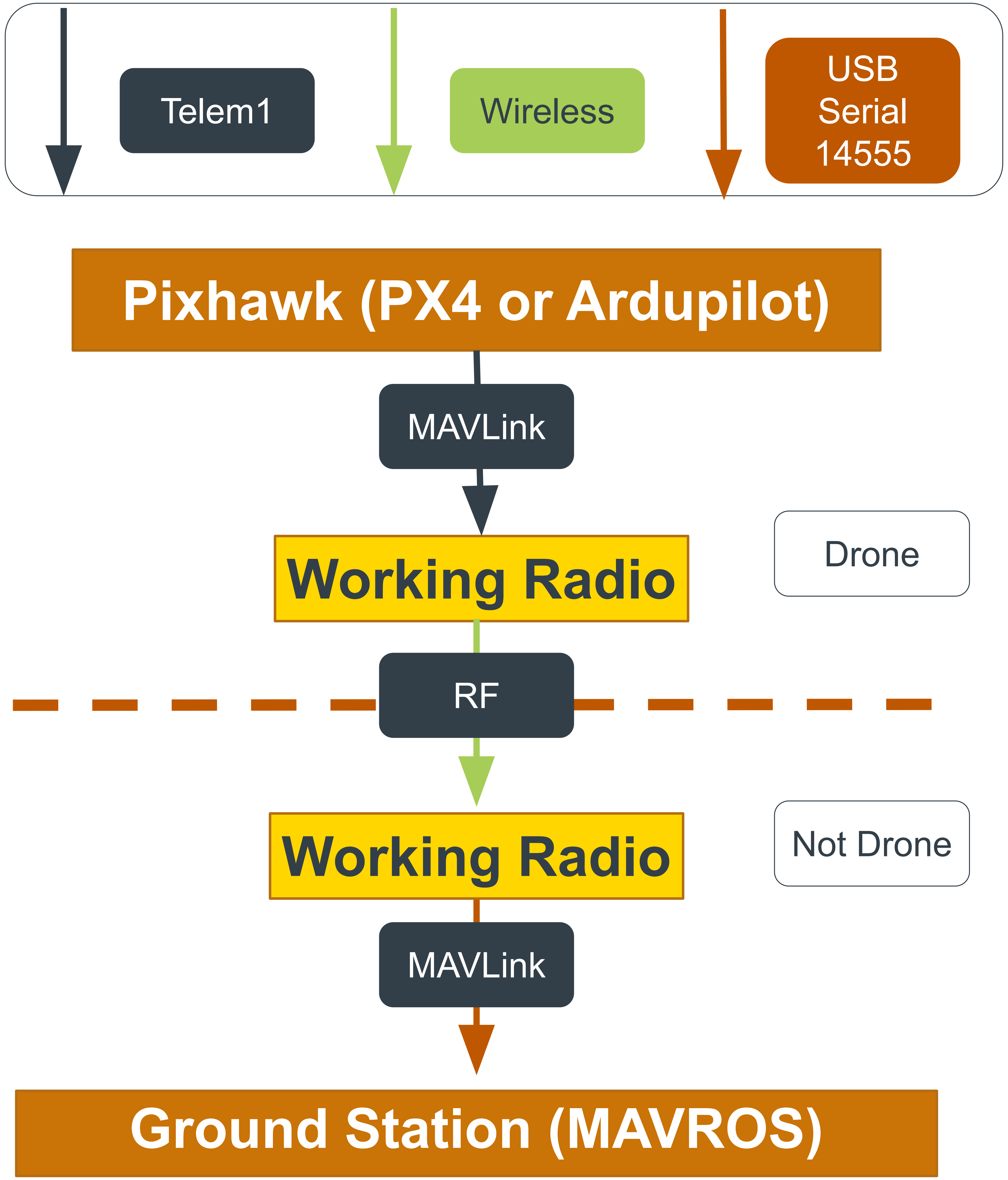


Telemetry radio based real-time communications.

Vincent

Back-up Design #1 - New Radio

- **Buy a radio that works:**
  - ($65) Holybro SiK Telemetry - on order
  - ($239) RFD 900+ Bundle - enhanced capability, pixhawk support
    - One version was tested on Oct. 16
    - MAVLink compatible
- **Cons:**
  - Wait time, no guarantee

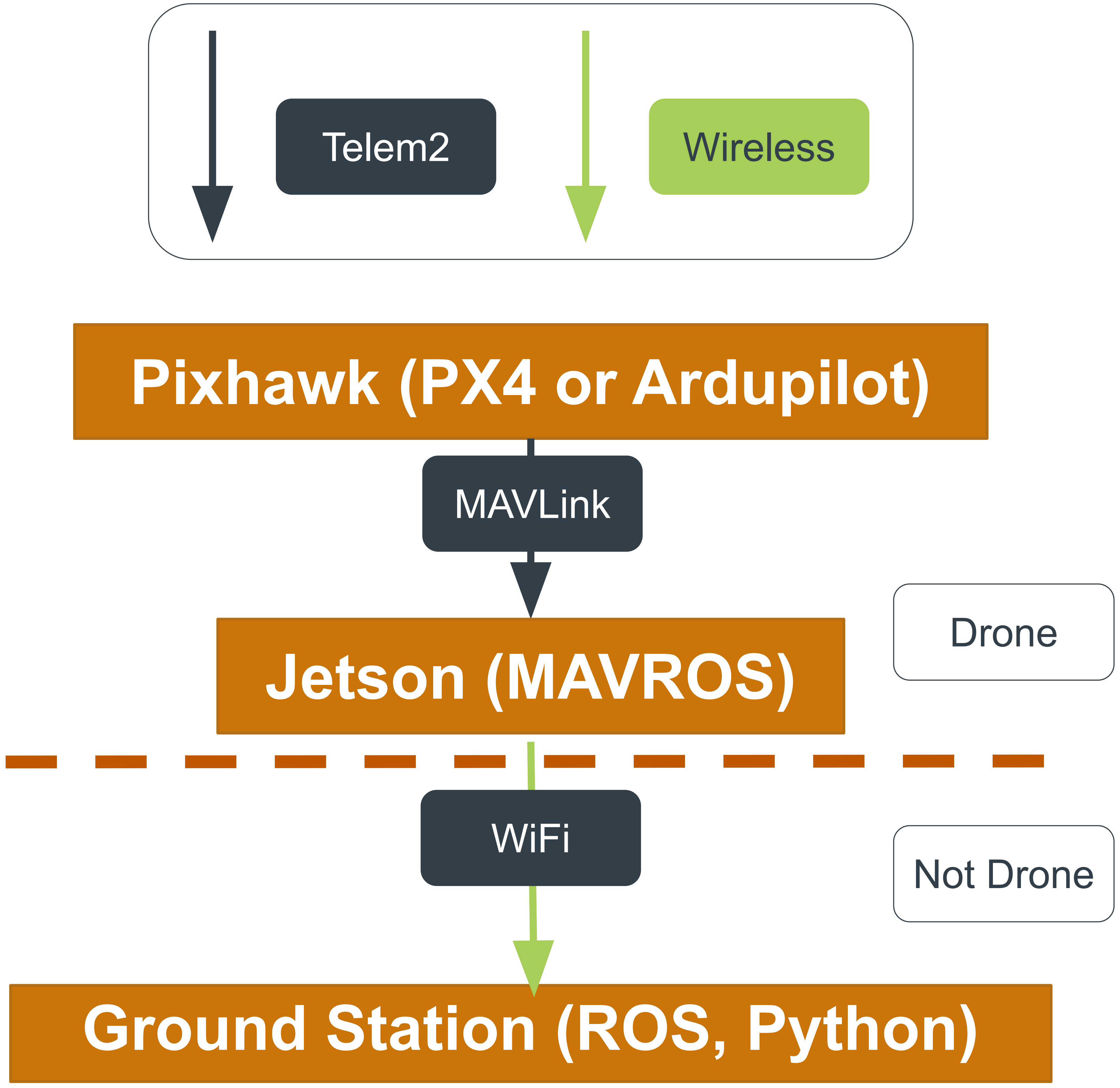


Telemetry radio based real-time communications with working radios.

Vincent

Back-up Design #2 - Jetson

- **Pros:**
  - Needed for VIO eventually
  - Decent documentation
  - Implemented on new drones anyways
- **Cons:**
  - Hardware design - powering, airframe layout
- **Moving forward:**
  - Research setup

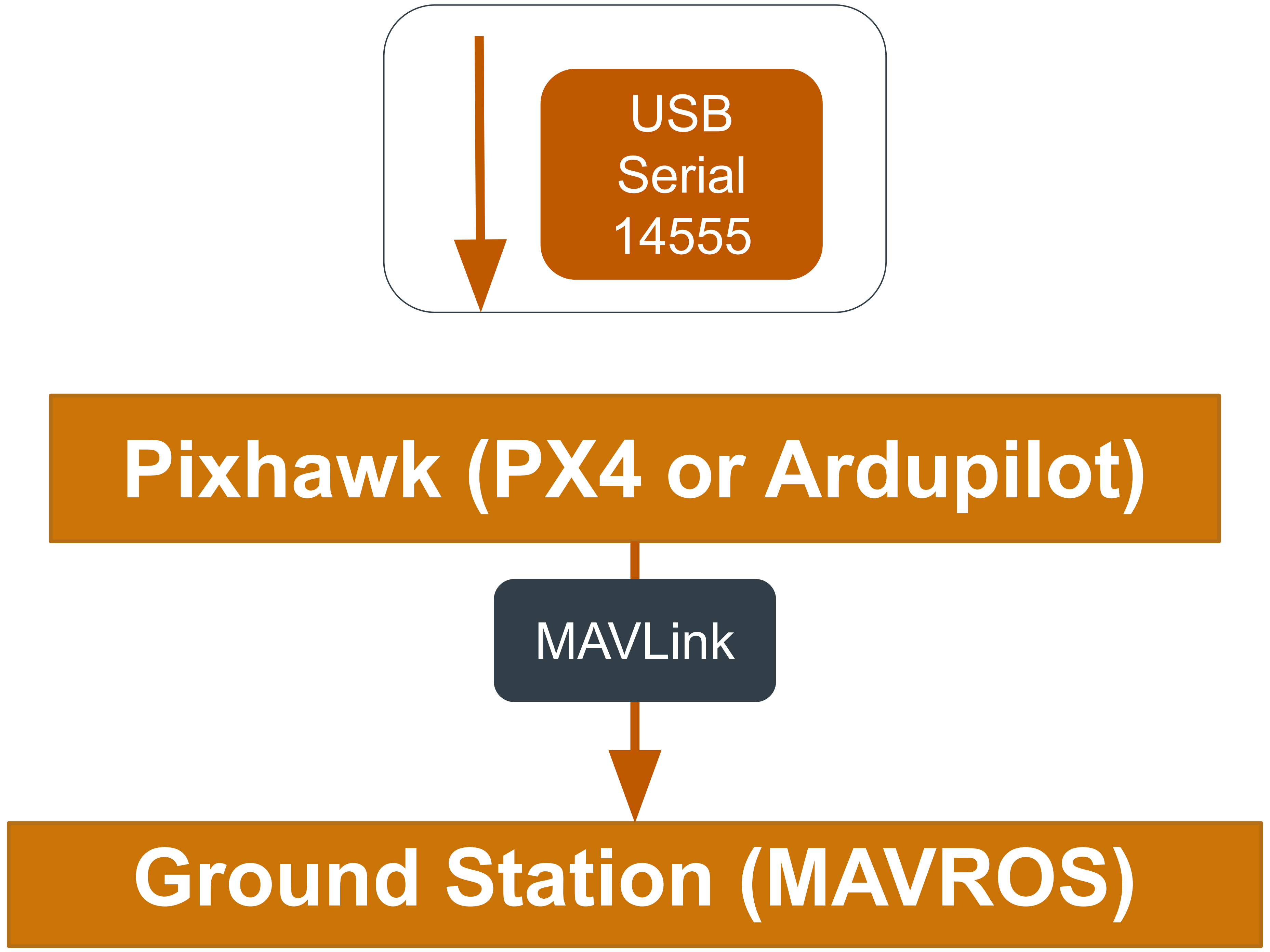


Jetson based real-time communications.

Vincent

Back-up Design #3 - HITL

- **Why:**
  - Proves we can connect software to hardware and acquire data in real-time using MAVROS
  - Useful for Jetson setup
  - Fast and easy? There is documentation
- **Cons:**
  - Not functional
- **Possible plan:** conduct HITL while radios are on order and/or Jetson is setup



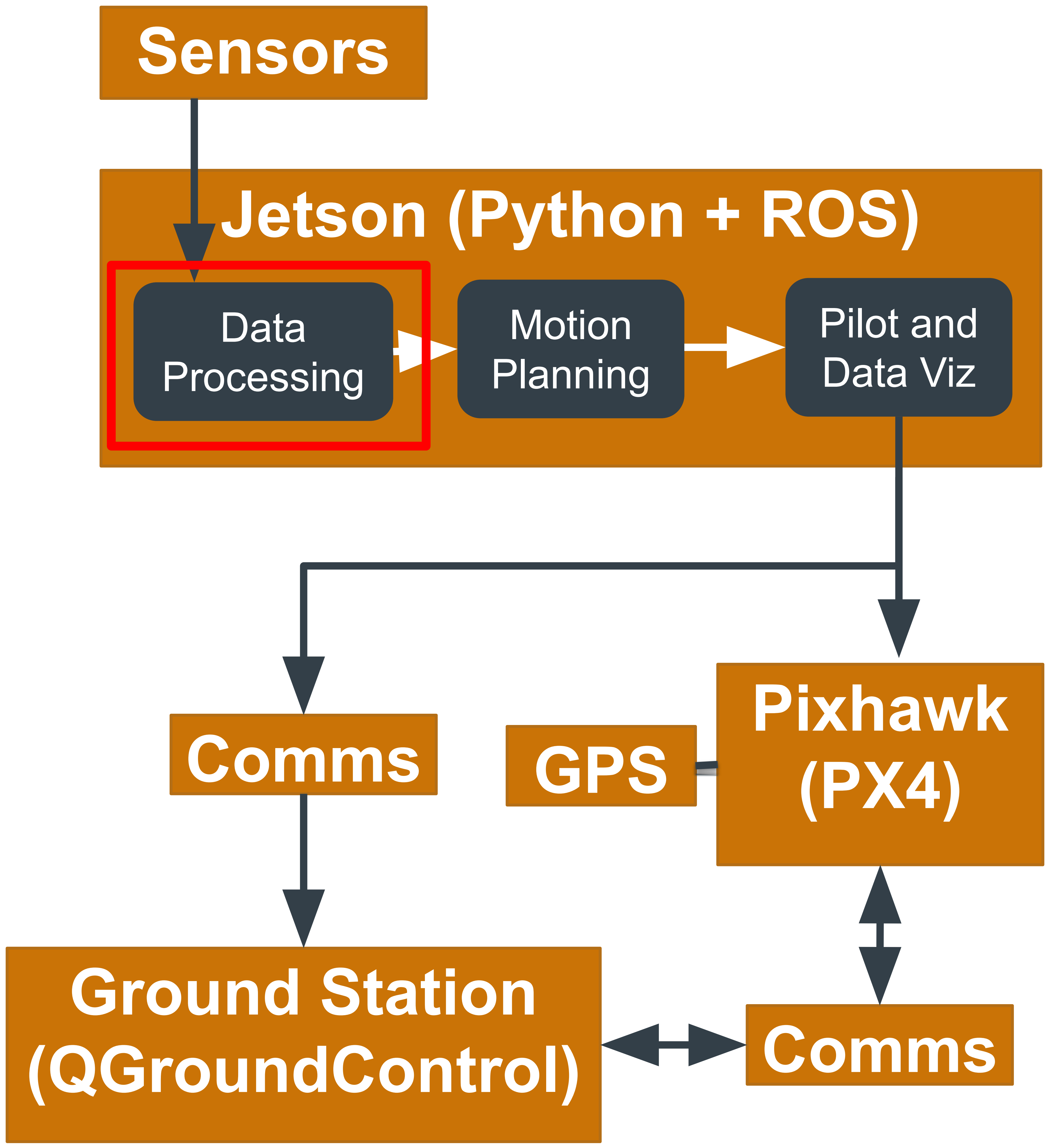
Hardware in the loop testing for real-time data acquisition.

Vincent

Pose Estimation

The Estimation Pipeline

- Pose Estimation is the prediction of an object's 3D position and orientation based on sensor data
- Our pose estimation utilizes an Extended Kalman Filter (EKF)
- Codebase:
  - Largely written in Python
  - Testing done with Matlab
  - Git tracked

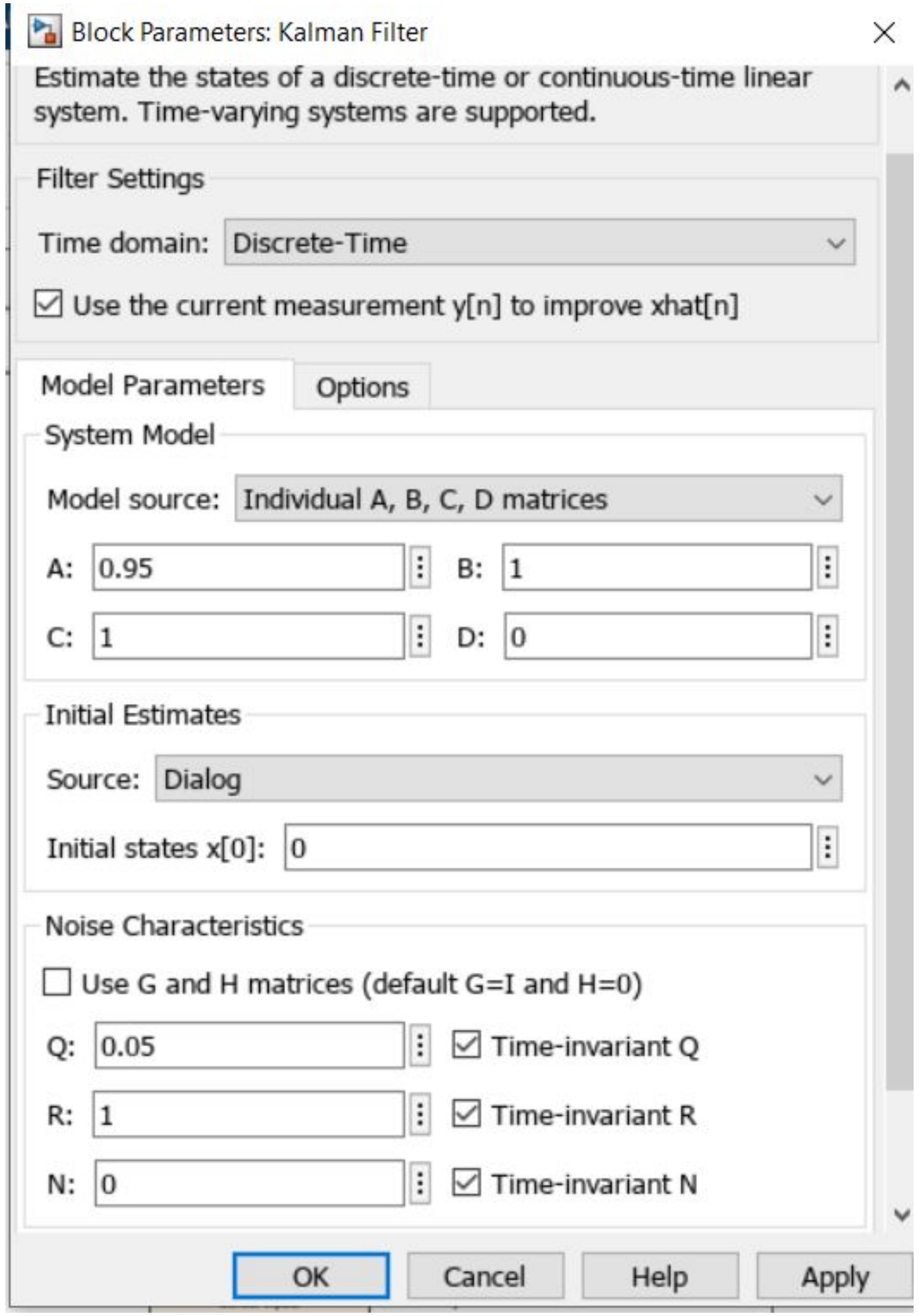


Hardware software integrations where some parts are within the scope of hardware and some should be provided by sims and estimation.

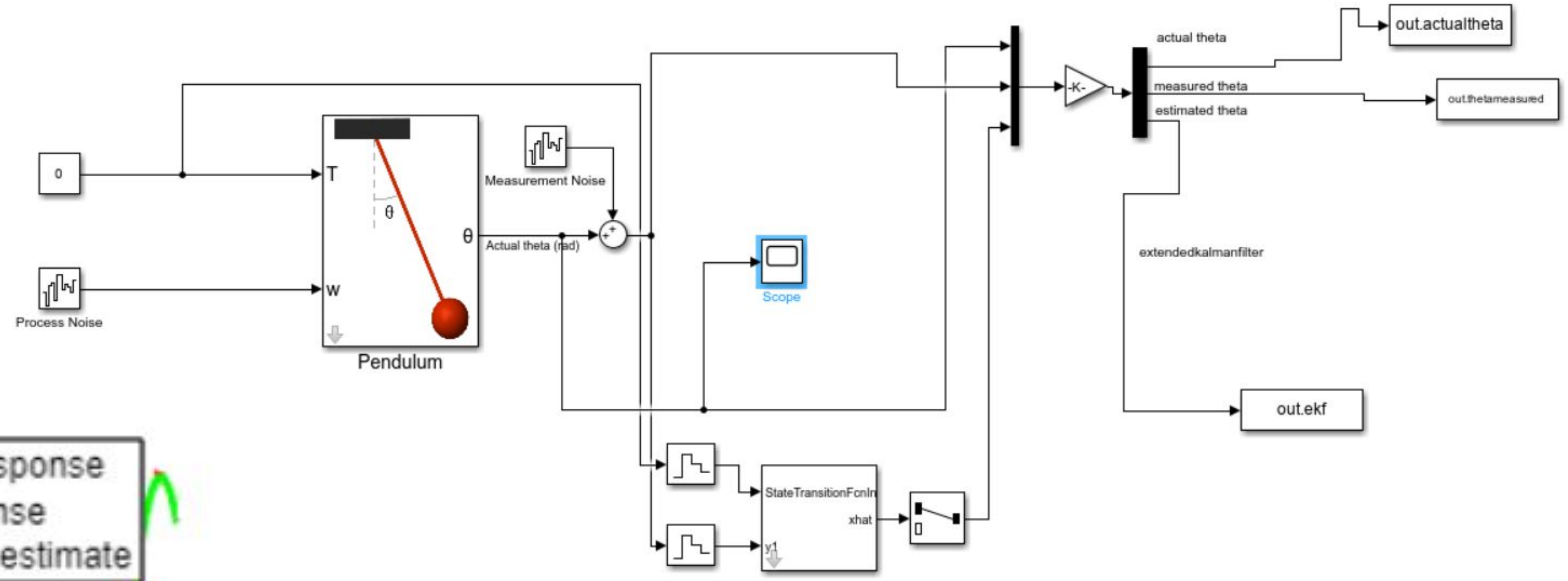
The Kalman Filter

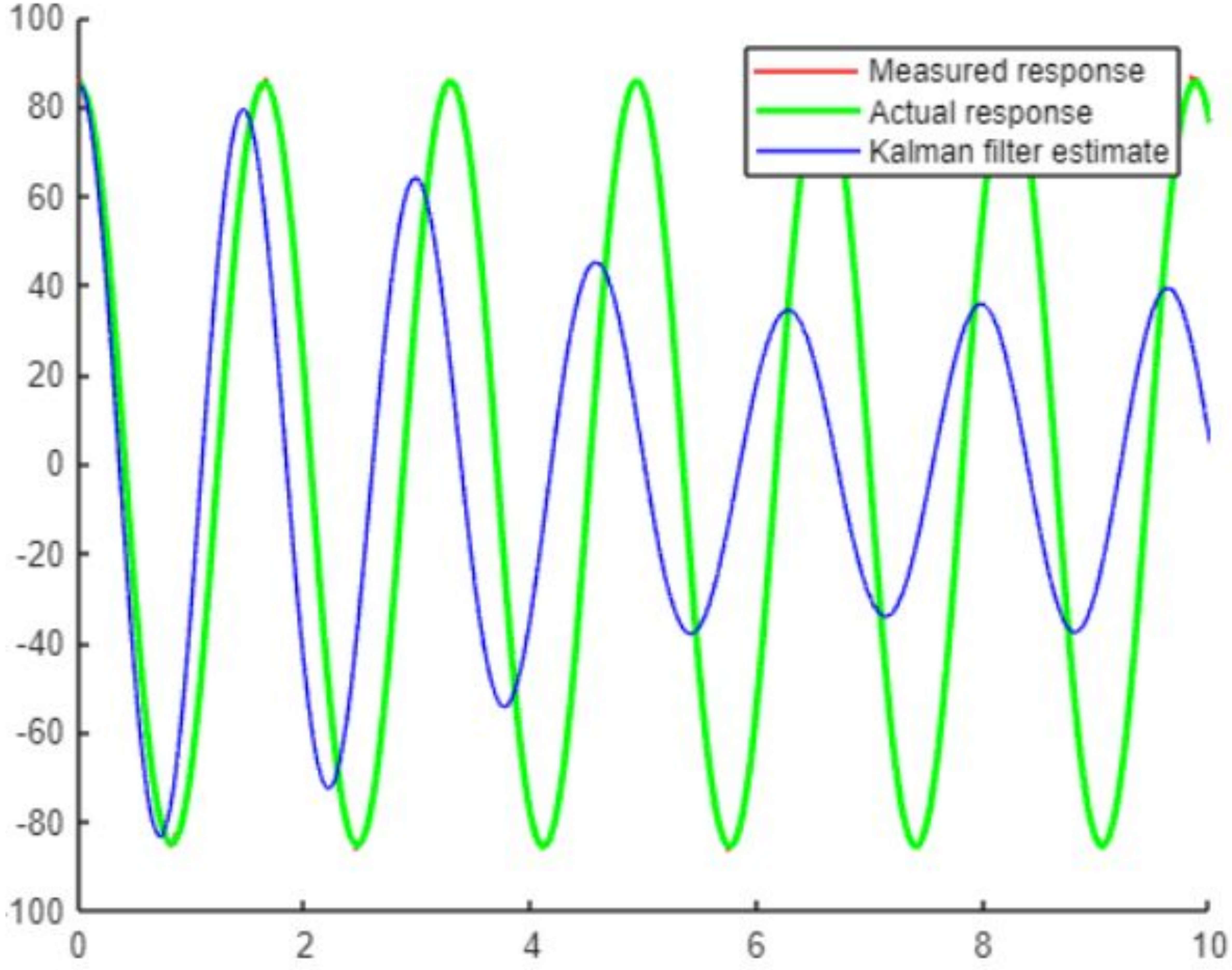
An Extended Kalman Filter (EKF)

- A Kalman Filter is a filter that takes a less than perfect dynamic model and noisy measurements to provide very accurate state estimations for a system. This filter only works for linear systems.
- Two covariance matrices: Q and R (process and measurement noise respectively)
- An EKF can be used in non-linear systems but requires more computation. The general concept is the same, and the Q and R covariance matrices are still very important.



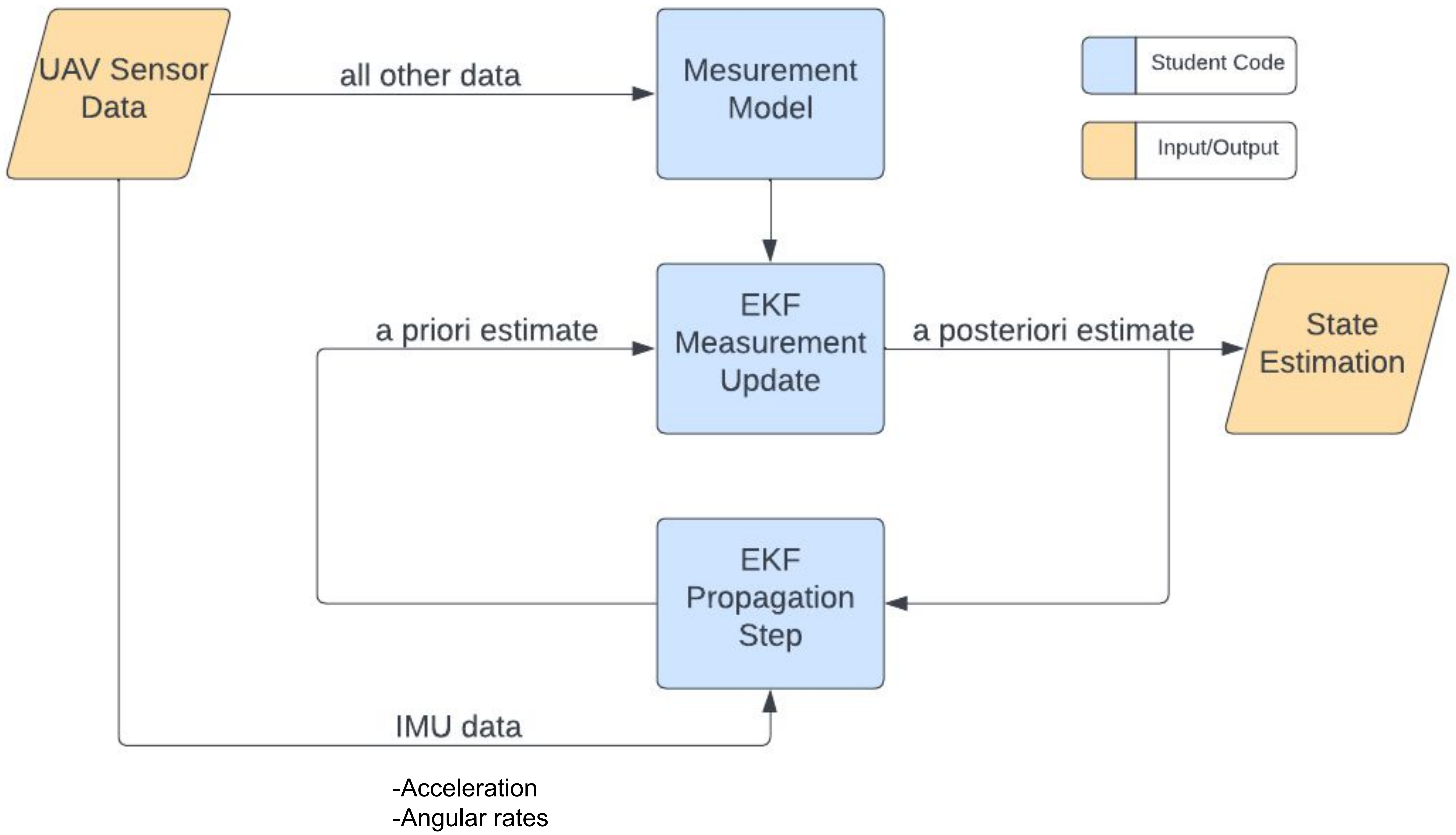
Efficacy of an EKF





- Previous groups proved the effectiveness of a Kalman filter
- Applied a Kalman filter to a pendulum to estimate the oscillation in the system

Our Extended Kalman Filter Structure



a priori $\hat{x}(k+1|Z^k)$

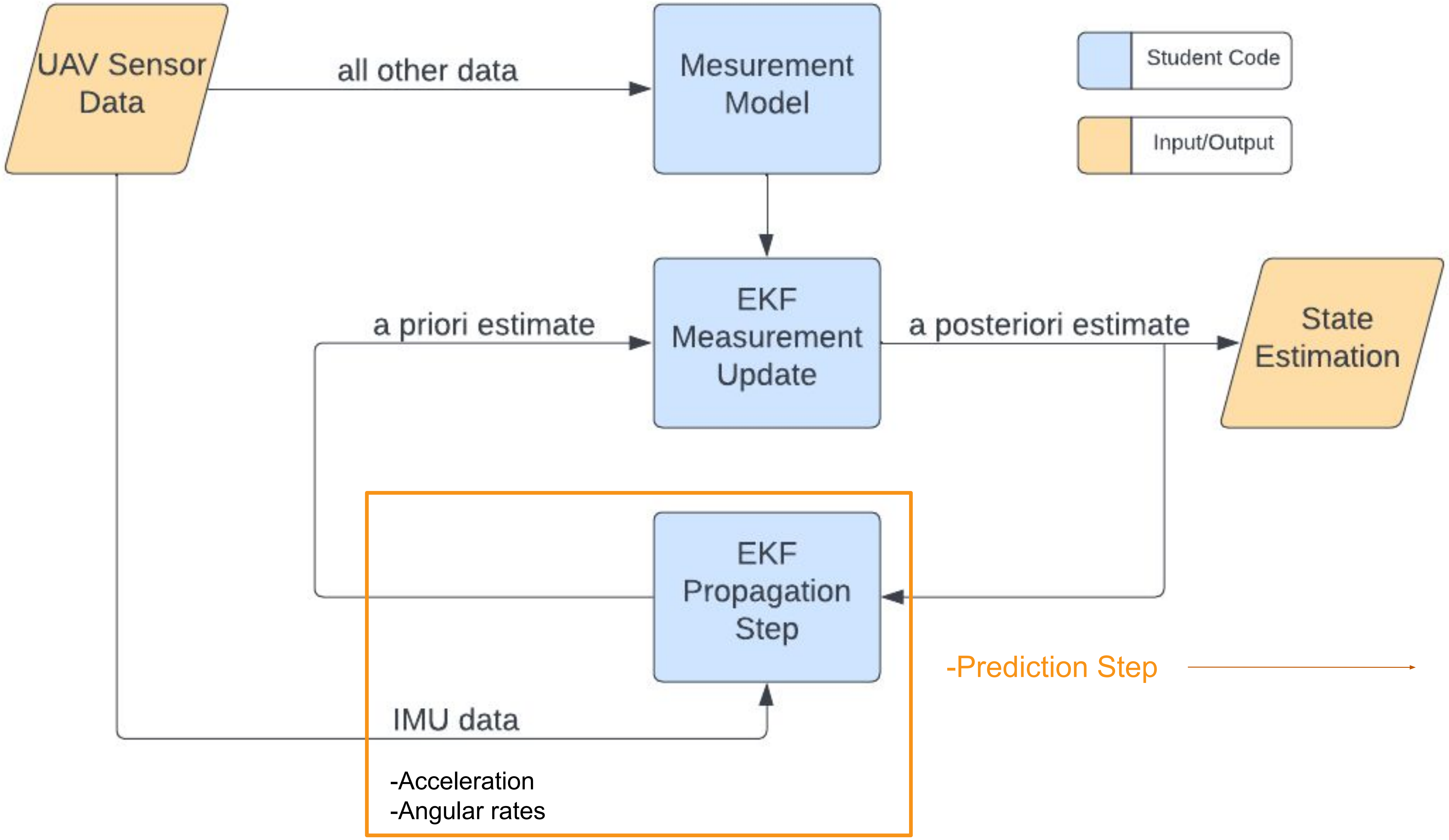
a priori $P(k+1|Z^k)$

a posteriori $\hat{x}(k|Z^k)$

a posteriori $P(k|Z^k)$

Current Extended Kalman Filter Progress

-Current Working State: No Measurement Model (set to 0)



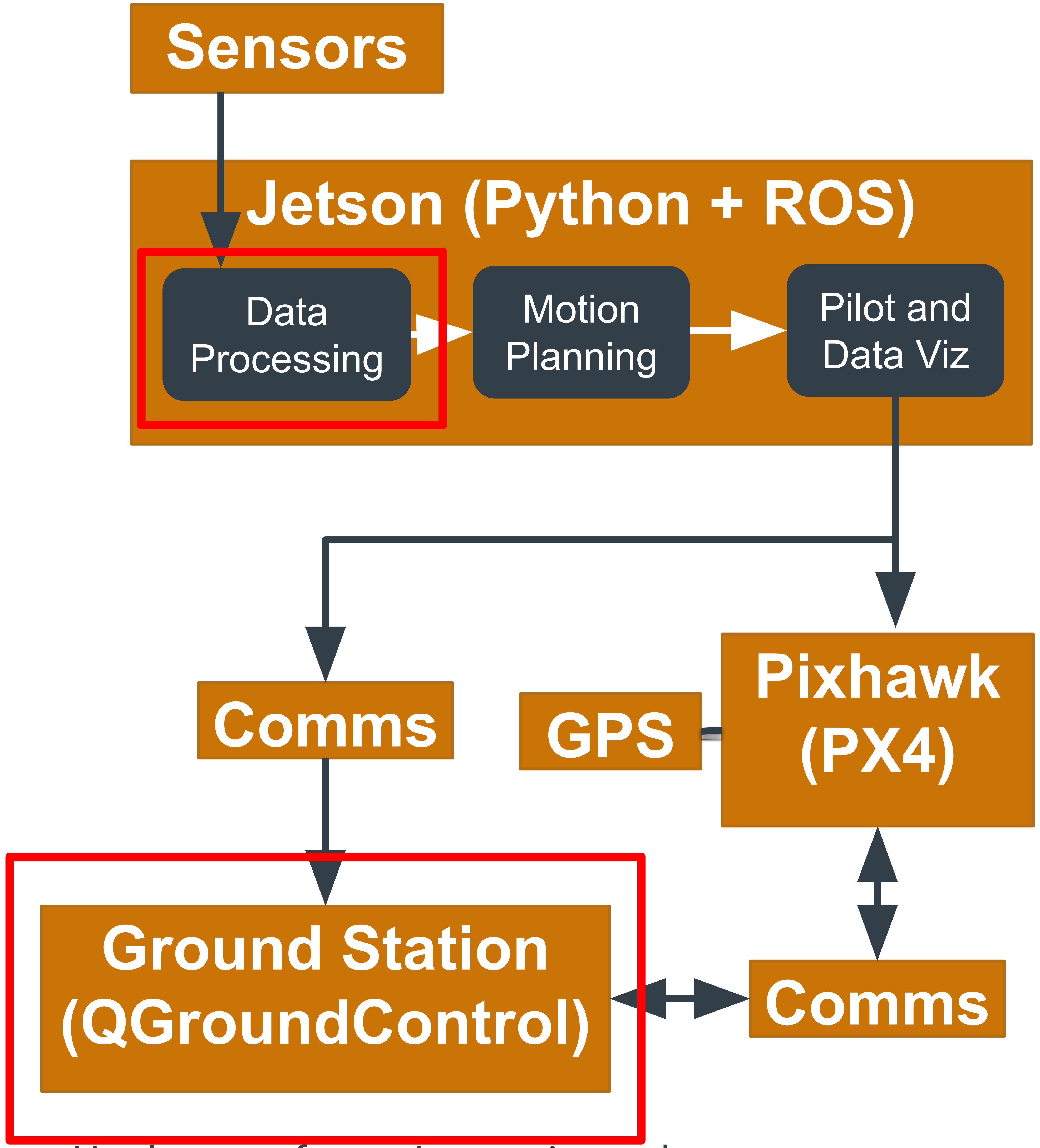
a priori $\hat{x}(k+1|Z^k)$
a priori $P(k+1|Z^k)$

a posteriori $\hat{x}(k|Z^k)$
a posteriori $P(k|Z^k)$

-Prediction Step → See how the error covariance matrix evolves with just the prediction step active.

Extended Kalman Filter cont.

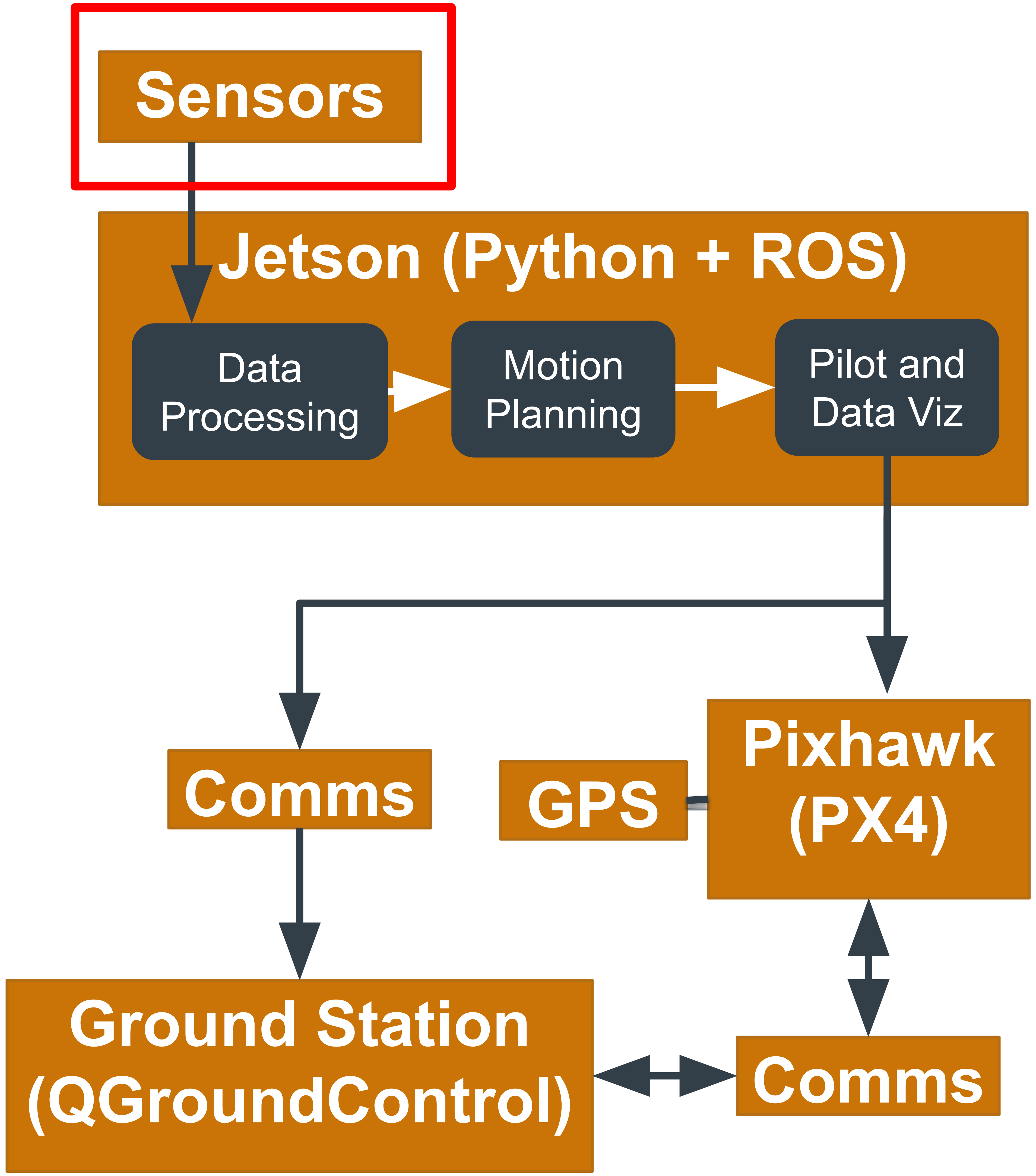
- **Goal: Real time kalman filtering onboard the quadcopter**
  - Python implementation
  - Integration with ROS channels
- **Current Implementation: Offboard post processing on our ground based computers**



Hardware software integrations where some parts are within the scope of hardware and some should be provided by sims and estimation.

Extended Kalman Filter cont.

- **Plan to start exploring sensor options for integration into the measurement update of the EKF**
  - Magnetometer
  - Barometer
  - Optical Flow
  - Optical Expansion



Hardware software integrations where some parts are within the scope of hardware and some should be provided by sims and estimation.

State Dynamics in the Kalman Filter

State Dynamics and Model Replacement

Our state transition function is the non-linear $f(\cdot)$:

$$\mathbf{x}(k+1) = f(\mathbf{x}(k), \mathbf{u}(k)) \tag{1}$$

where $\mathbf{x}(k) = [\mathbf{r}, \mathbf{v}, \mathbf{e}]^T$ and $\mathbf{u}(k) = [\boldsymbol{\omega}_g, \mathbf{f}_a]^T$ (for now ...)

Under the assumption of a sufficiently small time step Δt and a model replacement architecture, we can get a good propagation to $x(k+1)$ by Euler type integration:

$$\begin{aligned} \mathbf{r}(k+1) &= \mathbf{r} + \Delta t \mathbf{v} + \frac{1}{2}\Delta t^2 \mathbf{a}_a \\ \mathbf{v}(k+1) &= \mathbf{v} + \Delta t \mathbf{a}_a \\ \mathbf{e}(k+1) &= \mathbf{e} + \Delta t \dot{\mathbf{e}}_g \end{aligned} \tag{2}$$

How do we relate the measurements from the accelerometer and gyroscope to the acceleration and Euler angle derivative in our state propagation function?

State Dynamics and Model Replacement

To get the acceleration in the inertial reference frame we use the current Euler angle estimate:

$$\mathbf{f}_a^I = \mathbf{DCM}[\mathbf{e}(k)]\mathbf{f}_a \tag{3}$$

The force felt by the accelerometer always includes the local gravity g, so to find the actual acceleration felt by the body we need to subtract out the local force of gravity:

$$\mathbf{a}_a^I = \mathbf{f}_a^I - [0, 0, g]^T \tag{4}$$

We find the Euler angle derivative from the gyroscope data by the following equations.

$$\dot{\mathbf{e}}(k) = \mathbf{S}[\mathbf{e}(k)]\boldsymbol{\omega}_g \tag{5}$$

The matrix $\mathbf{S}[\mathbf{e}(k)]$ is dependent on the Euler angle sequence chosen to represent the quads attitude. In our case the 3-1-2 Euler angle sequence is used, which leads to the derivation of $\mathbf{S}$ in terms of the Euler angles $\mathbf{e} = [\alpha, \beta, \gamma]^T$ as:

$$\mathbf{S}[\mathbf{e}(k)] = \frac{1}{cos(\alpha)} \begin{bmatrix} cos(\alpha)cos(\beta) & 0 & cos(\alpha)sin(\beta) \\ sin(\alpha)sin(\beta) & cos(\alpha) & -cos(\beta)sin(\alpha) \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix} \tag{6}$$

Error Ellipsoid Generation

PROBABILITY ELLIPSOID

$$(\mathbf{x}-\overline{\mathbf{x}})^T P^{-1}(\mathbf{x}-\overline{\mathbf{x}}) = \ell^2 \qquad (1)$$

$$U^T P U = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} = D\,[\lambda_1,\ \lambda_2, \ldots,\ \lambda_n] \qquad (2)$$

$$U = [\mathbf{u}_1,\ \mathbf{u}_2, \ldots,\ \mathbf{u}_n]_{n\times n} \qquad (3)$$

$$\mathbf{x}' = U^T\,\mathbf{x} \qquad (4)$$

$$\begin{aligned} P' &\equiv E[(\mathbf{x}'-\overline{\mathbf{x}}')(\mathbf{x}'-\overline{\mathbf{x}}')^T] \\ &= U^T E[(\mathbf{x}-\overline{\mathbf{x}})(\mathbf{x}-\overline{\mathbf{x}})^T]\,U \\ &= U^T P U = D\,[\lambda_1 \ldots \lambda_n]. \end{aligned} \qquad (5)$$

$$\Delta\mathbf{x} \equiv \hat{\mathbf{x}} - \mathbf{x} \equiv [\tilde{x}\ \tilde{y}\ \tilde{z}]^T \qquad (6)$$

$$P = E[\Delta\mathbf{x}\Delta\mathbf{x}^T] \qquad (7)$$

$$[\tilde{x}\ \tilde{y}\ \tilde{z}]P^{-1}\begin{bmatrix}\tilde{x}\\ \tilde{y}\\ \tilde{z}\end{bmatrix} = \ell^2 \qquad (8)$$

$$\begin{bmatrix}\tilde{x}'\\ \tilde{y}'\\ \tilde{z}'\end{bmatrix} = U^T\begin{bmatrix}\tilde{x}\\ \tilde{y}\\ \tilde{z}\end{bmatrix} \qquad (9)$$

$$P' = U^T P U \qquad (10)$$

$$[\tilde{x}'\ \tilde{y}'\ \tilde{z}']\begin{bmatrix}1/\lambda_1 & & \\ & 1/\lambda_2 & \\ & & 1/\lambda_3\end{bmatrix}\begin{bmatrix}\tilde{x}'\\ \tilde{y}'\\ \tilde{z}'\end{bmatrix} = \ell^2 \qquad (11)$$

$$\frac{\tilde{x}'^2}{\lambda_1} + \frac{\tilde{y}'^2}{\lambda_2} + \frac{\tilde{z}'^2}{\lambda_3} = \ell^2 \qquad (12)$$

Probability Ellipsoid Equation

$\ell = 3 \rightarrow 3\sigma$ probability ellipsoid with 97.1% confidence

PROBABILITY ELLIPSOID

$$\frac{\tilde{x}'^2}{\lambda_1} + \frac{\tilde{y}'^2}{\lambda_2} + \frac{\tilde{z}'^2}{\lambda_3} = \ell^2 \quad (15)$$

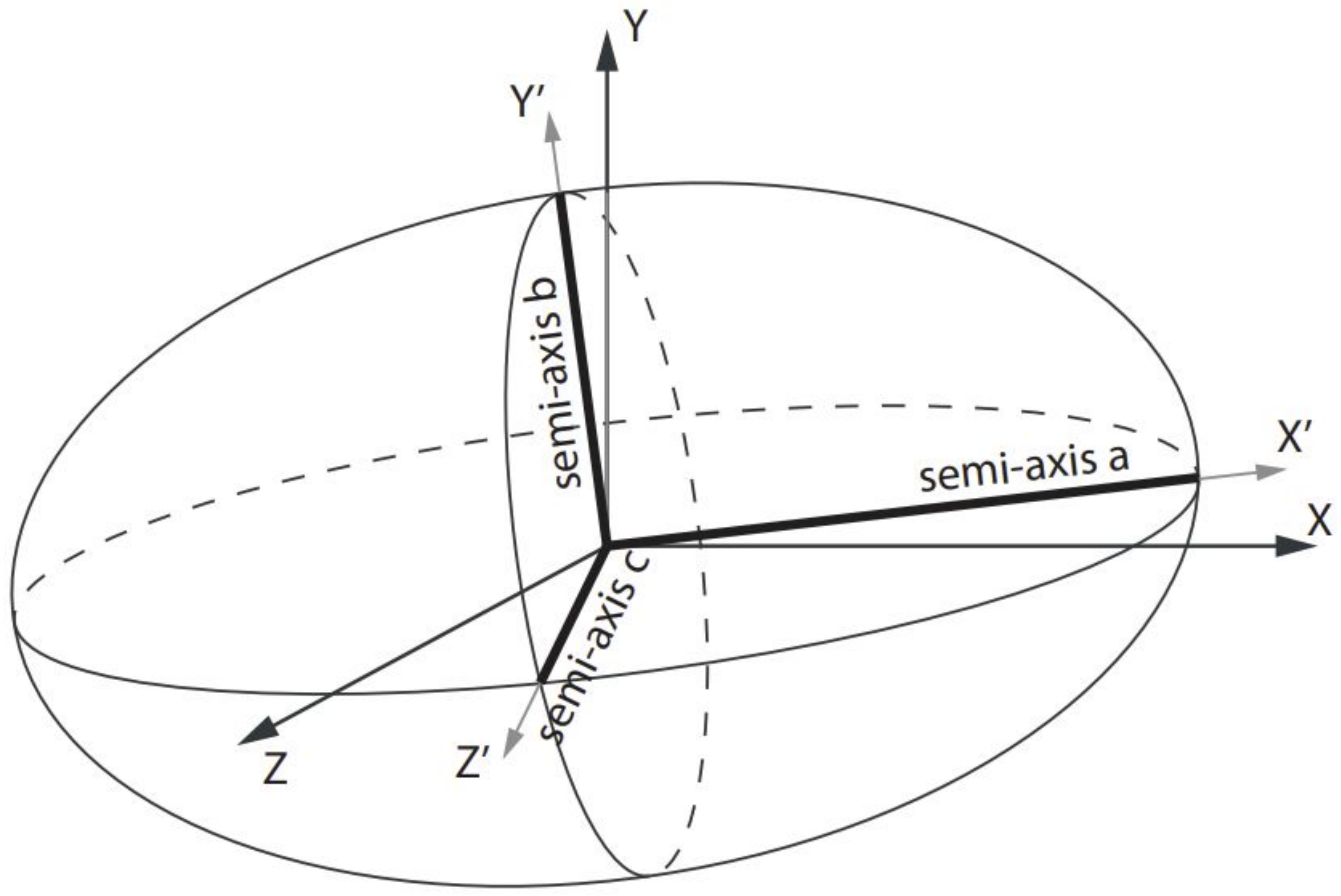
Solve for one variable,

set other two equal to 0

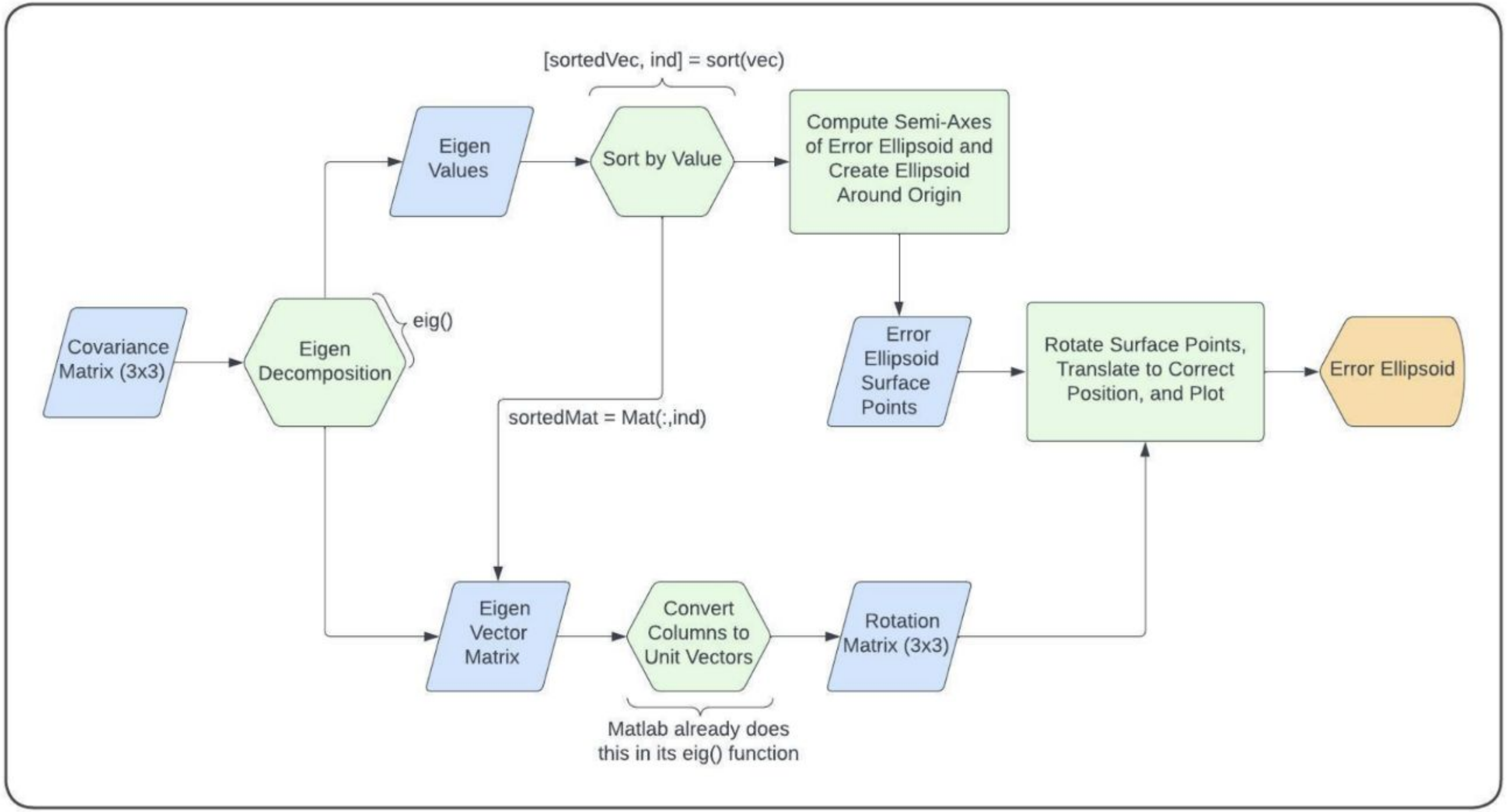
$$a = \tilde{x}' = \sqrt{\lambda_1 * \ell^2}$$

$$b = \tilde{y}' = \sqrt{\lambda_2 * \ell^2} \quad (16)$$

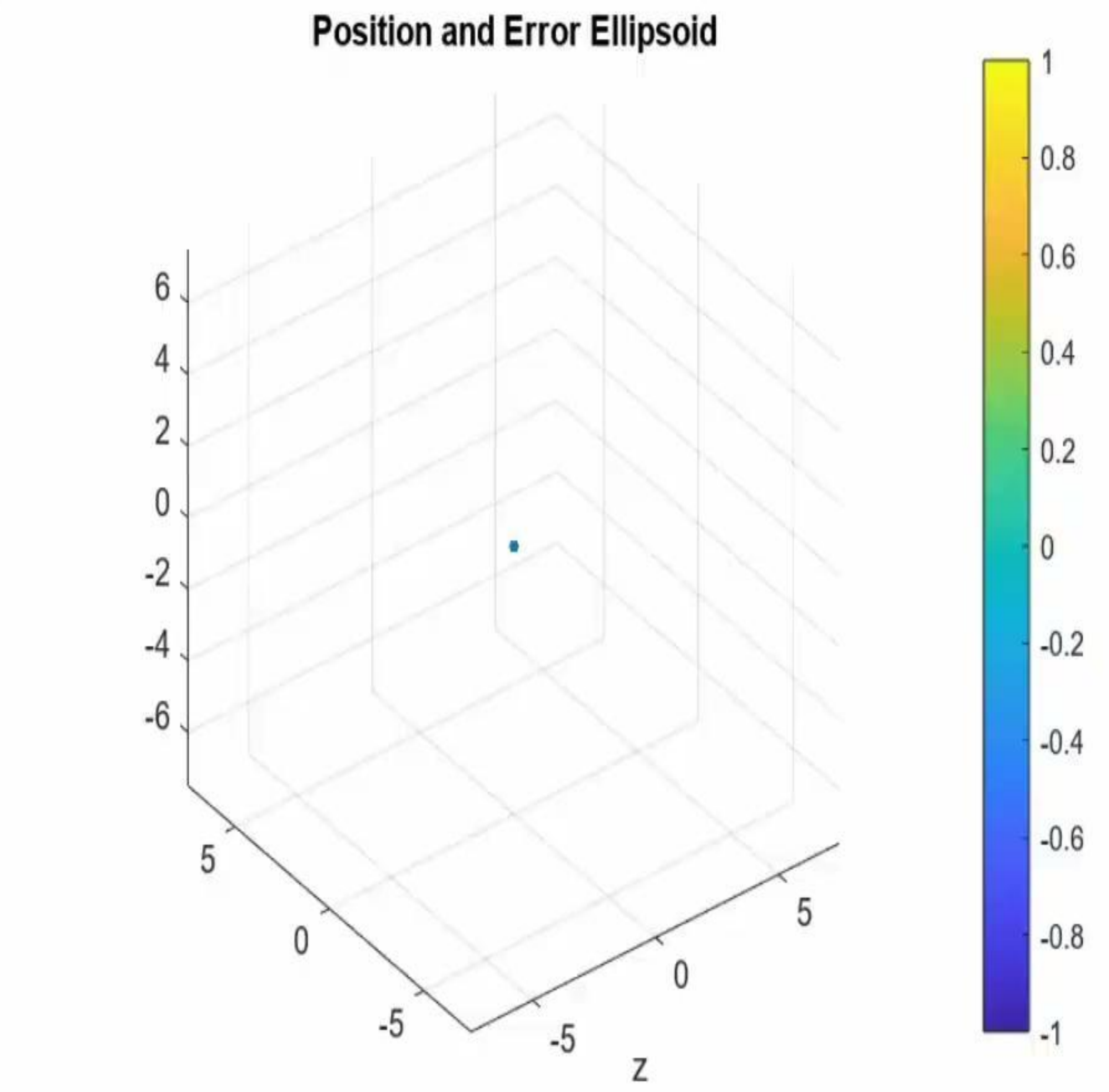
$$c = \tilde{z}' = \sqrt{\lambda_3 * \ell^2}$$



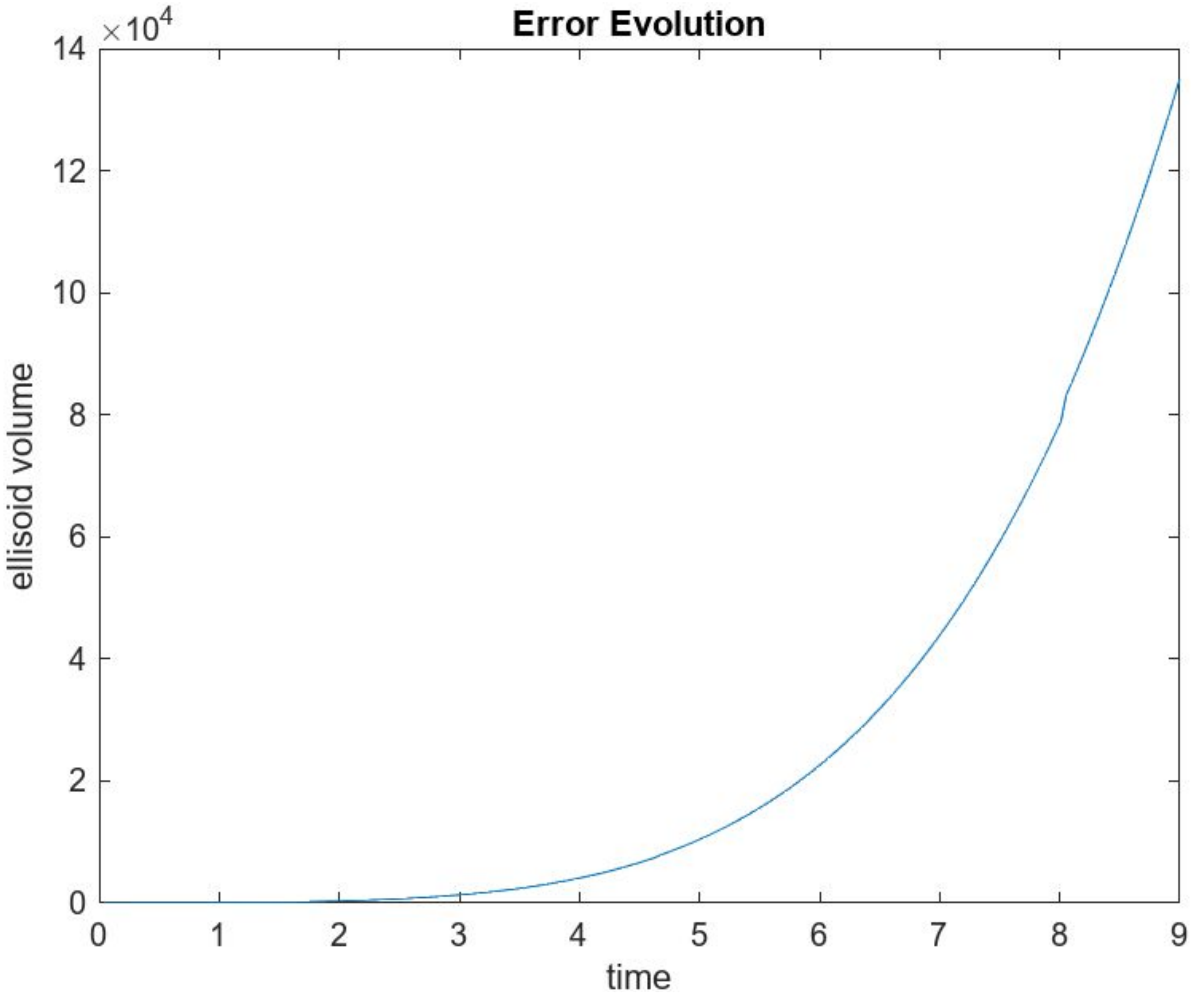
From Covariance Matrix to Error Ellipsoid



Visualization on Quad Data (early stages)


Position and Error Ellipsoid
6
4
2
0
-2
-4
-6
5
0
-5
-5
0
5
z
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1

Error Ellipsoid Evolution without Measurement Updates



Next Steps - Mahi

Next Steps

1.

- Quantify IMU uncertainty -> model with bias
- Work on kalman filter to incorporate sensor data with defined timestep
- Visualize drone in real-time during flight
- Incorporate Kalman filter with python+ros setup to visualise error ellipsoid around drone in real-time
- Build new drones with upgraded hardware

THANK YOU.

The University of Texas at Austin
Aerospace Engineering and Engineering Mechanics
Cockrell School of Engineering