2022-2025 ADV Avionics System Design
Author: Andrew
Living Document - WIP

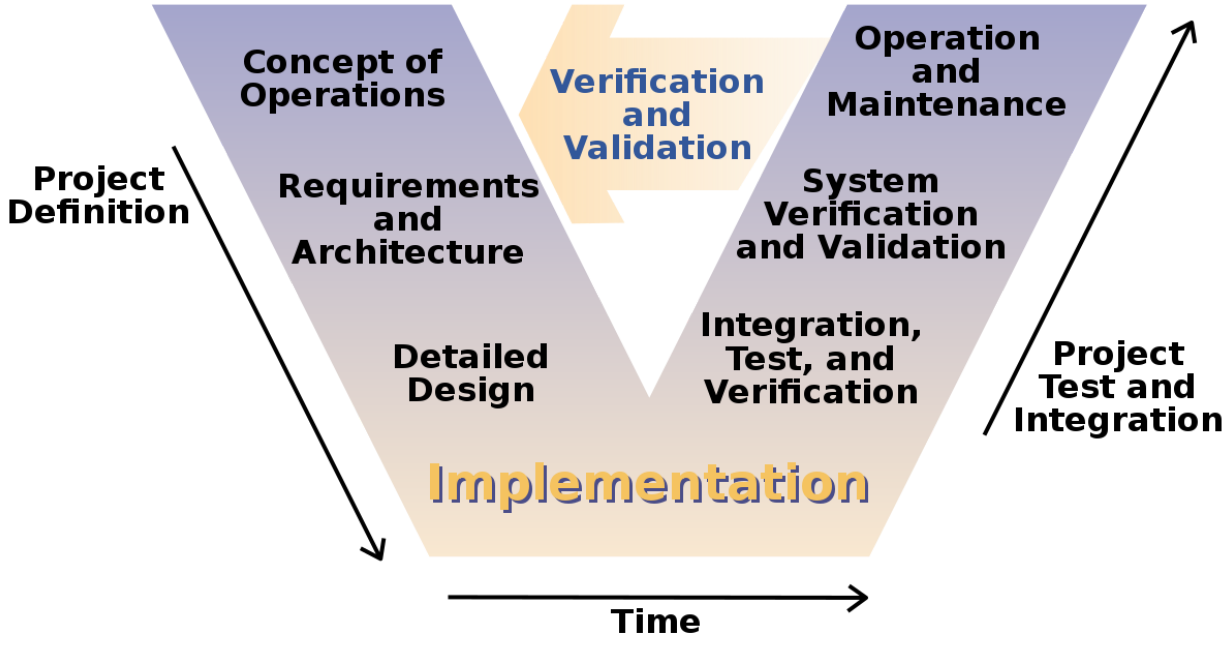
Preface

Our system is highly complect. Lots of design decisions are impacted by others at different layers of abstraction and engineering domains. Many decisions couldn't have been made until the entire system was designed. As with many real engineering challenges, it ends up being a recursive system of engineering problems, each with their own V-model and design cycles.

We strive to keep as many systems as isolated as possible, but this is ultimately not possible when things like budget, weight, time, and experience add to the entwined nature of the overall system.

Decisions for the avionics architecture are largely driven by "fucking around" and "finding out", with a lot of engineering intuition and experience mixed in to guide the whole process. Our focus will be on quantifying our system can only come *after* things are designed and built, so we must remain flexible and nimble in how we operate, and design things with modularity in mind, in order to minimize effort to solve anything that wasn't foreseen.

I will be covering the first half of the Engineering V-Model, maybe more



Some terms I will be using and not explaining

- PADA - Powered Autonomous Delivery Aircraft (The little drone we drop)
- GTV - Ground Transport Vehicle (The little rover we build)

- GPS - Global Positioning System
- RTK - Real Time Kinematic
- YoloV7 - A machine learning model

The Challenge

The challenge is to win the SAE AeroDesign competition. Aside from the report and presentation, we will do this by scoring the most points.

Scoring Equation:

$$Final\ Flight\ Score = FS_1 + FS_2 + FS_3 + GTV$$

Where:

$$FS = Flight\ Score = W_{Payload} + 8 * (Z_{PADA} + B_{PADA})$$

$$GTV = Ground\ Score = \frac{A_{GTV} * W_{delivered}}{4}$$

$$B_{PADA} = PADA\ Landing\ Bonus^{*} = 5 * \left(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{d^2}{2\sigma^2}}\right)$$

**Note, this is the normal probability density function with mean of 0*

$W_{payload} = Water\ (lbs)\ Successfully\ Flown$

$W_{delivered} = Total\ Water\ (lbs)\ Delivered\ by\ GTV\ During\ Demonstration$

$A_{GTV} = GTV\ Autonomy\ Multiplier: 2\ if\ autonomous, 1\ if\ manual$

$Z_{PADD} = PADA\ Zone\ Multiplier: 2\ for\ random, 1\ for\ static$

$d = Distance\ of\ PADA\ to\ center\ of\ landing\ zone, rounded\ down\ to\ nearest\ ft$

$\sigma = Team\ supplied\ Standard\ Deviation\ from\ TDS\ (ft)$

I will not bother to show the math, just understand that landing pada = big points = win. The GTV can net a lot of points too, but only in an ideal world, and ultimately engineering is done in the realm of man. Our approach will be to optimize the avionics and PADA systems as much as possible, while only building a GTV in order to appease the judges.

1. Concept of Operations

Our concept of operations is synonymous with our data pipeline. Note that this is after iteration, not off our initial design. I will explain the choices that lead to this throughout the report.

1. Plug in arming plugs

2. Flight controller boots up and gets GPS lock
3. Raspberry Pi boots up and runs preflight self-checks
4. Raspberry Pi sends initial time-averaged GPS and altitude to the ground station to use as reference point, and as a way to do a communication check
5. The tracking beacon uses this ping to position itself better
6. Plane takes off
7. Raspberry Pi detects altitude change and begins image/data collection

From this point forward we track the data's path.

8. The data and images are stamped with a timestamp and added to queues.
9. Data that doesn't match the image sampling rate will be interpolated
10. Images that don't contain a disk are filtered out via an Edge AI algorithm
11. Images and data are paired by closest timestamp, to a threshold, and then binarized
12. The binary data is transmitted via ZMQ sockets to the tracking beacon
13. The data is unpacked on the laptop
14. The GPS and altitude are ingested by Mercury for distribution

From this point forward we go asynchronous.

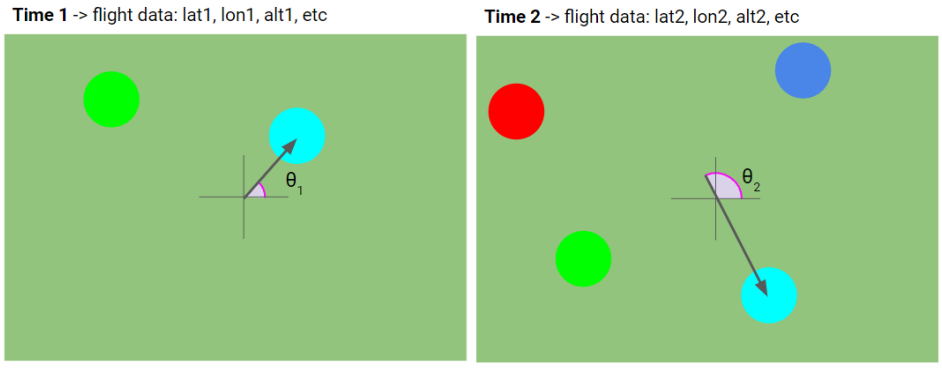
Tracking beacon:

1. The GPS and altitude are read
2. A new pitch and yaw are calculated in order to position the antenna better

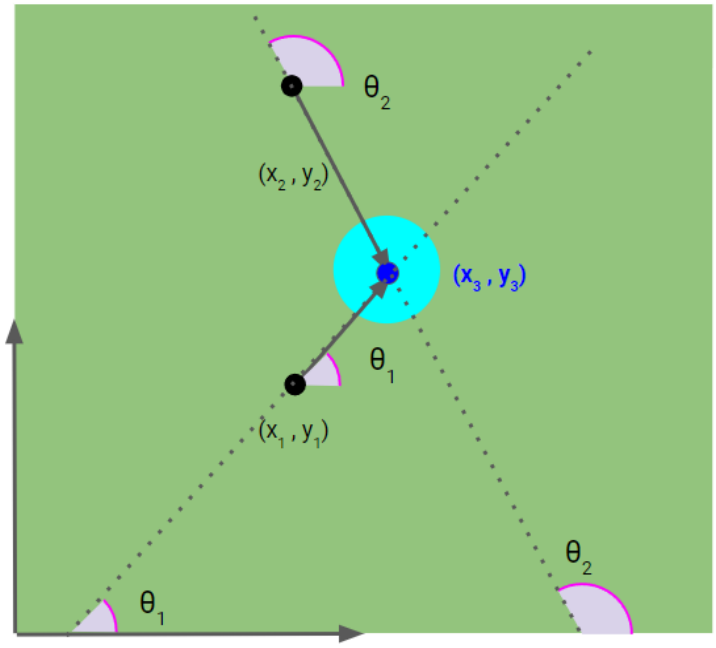
Vision Engine:

1. The images and GPS coordinates are read
2. The images are run through our YoloV7 model to locate the disk in the image.
3. This yields the bounding box, which we can easily use to grab the middle of the disk (centroid)

4. Using the colour and plane location, we can assign a unique disk ID to each captured image, which will allow us to perform further calculations only on ones with matching unique disk IDs
5. We can then take the centroid, and the middle of our image (middle of our camera) and calculate the angle between them. This is what we call the X-algorithm



6. Doing this on two images with the same unique ID, taken at different times, gives us enough information to calculate the position of the disk. We can do this through cartesian math or bearing. This entire process is moved into a Rust submodule, as it can be computationally expensive after many images are collected.



7. The results for each unique disk are published back to Mercury

Gamer GUI:

1. The images are read
2. The ground station specialist chooses two images to compare
3. The Manager of Naming Knowledge for Efficiency and Yield (MONKEY) will click on the disks they see, which then are run through simpler computer vision algorithms in order to find the centroid.
4. From this point on the flow is the same as the Vision Engine

Atlas:

1. The flight data and the results from Gamer GUI and Vision Engine are read
2. The GPS coordinates and predictions are displayed on a map
3. The altitude is displayed live

Now we skip forward until we have finished flying over the field once.

1. Any remaining data in the queues are sent and processed
2. The MONKEY chooses a GPS coordinate to drop to from the following options:
3. The Vision Engine predictions
4. The Gamer GUI predictions
5. The Static Disk
6. Once chosen, the ground station sends this data back to the plane
7. The plane changes states and arms the PADA by generating a flight path for it
8. Once armed, the plane changes states to prepare for drop

9. Once in an ideal location for dropping, the RPi activates the drop servos, releasing the PADA
10. The PADA should detect this release and begin to fly toward the target location
11. The plane will loiter until the PADA lands
12. The plane lands

2. Requirements and Architecture

2.1 Processing

Our highest level architecture choice, that was already hinted at, is choosing between doing live image processing on the PADA, doing it all on the plane, or doing it on the ground, or a combination.

1. Doing it on the PADA would allow us to "heat seek" the targets but this comes with a few key drawbacks.
    - The target going out of view of the PADA would instantly break this.
    - The PADA is already weight constrained, adding the computer and camera necessary for this is not desirable
    - No one on the team has experience with real time embedded image processing
    - No one on the team has experiencing designing a real time control system that can accomplish this

2. Doing a combination of PADA + Plane to track both the PADA and target location would just be an insanely complex endeavor, and the Plane would likely fly past the target much earlier.

3. Doing all the processing we would do on the ground, but on the plane, could become difficult due to the processing power required. Another downside of this is that we would have no insight into how well the algorithm is doing, so we would want to send images down to the ground anyway. If we do this we might as well run things on a faster laptop.

4. Our final choice is to do a little bit of processing on the plane when convenient (having good data as input to our ground systems), and doing the bulk of the processing and running larger ML models on the ground, while using some form of wireless communication to send data bidirectionally from plane and ground.

Qualitatively speaking, I think option 4 is the lowest in terms of complexity, and most achievable by us. How would you quantify this? No idea.

Moving forward, architecture option 4 will be used.

2.2 Algorithm

Another key architecture choice, that drives decisions at both the sensor level and the programming level, is the algorithm we actually use to figure out where to send the PADA.

Two main options emerge from the choice of using GPS. Without GPS, we would have to solve the problem of simultaneous localization and mapping (SLaM). This is hard, especially at our distances and speeds, computationally expensive, and just not feasible for us.

Our problem is simplified by the use of an RTK GPS, which solves our issue of localization instantly. The challenge then becomes figuring out where the landing zone markers (LZMs) are relative to the plane.

1. We could point our camera forward to maximize the amount of time we can see each disk. To figure out location, we would need to work with the camera geometry and pixel measurements.
2. We could point our camera perpendicular to the ground, removing the need for perspective warp corrections. We could then use computer vision to measure the radius of each disk, and use it to figure out the distance between the disk and the center of the camera, which we know the position of
    - This is heavily affected by lens warping
    - This requires an extremely high resolution camera to execute with high precision
3. We could wait until we pass really close or right over a disk, potentially even direction the pilot, and just use the closest result to the center of the camera as the drop location
    - I ran the simulations for this, expected value for points is lower than dropping to static
4. We could always drop to the static disk, if we're a bunch of fucking losers.
5. We could use the previously mentioned X-algorithm!
    - As lens distortions are almost always radially symmetric, and the X-algorithm is agnostic to radial distortion (our basis is radius and angle, and only radius is affected by the distortion)
    - We are agnostic to altitude as we only care about the angles from the center, not the actual size of the disks.

All these algorithms suffer the same common pitfalls:

- Motion blur may ruin our detection methods. This should be mitigatable depending on if we can control the shutter speed of our camera.
- Tilting of the camera may skew our predicted positions as the relative positions in the camera don't match up to the true distance between disks and the camera.
- Other objects end up getting caught up with the disks and throw off our data, although good outlier detection should prevent this.

Someone should figure out how to design these out, **otherwise we will be moving forward using the X-algorithm.**

2.3 Communication Method

The last key architecture choice in my opinion, based on the fact that we want to transmit images to the ground no matter what, is *how* we actually send these images (and data). The challenge for this is threefold. We need something that is long enough to stay connected to the

plane, high enough bandwidth to send images without buffering, and flexible enough to allow us to manipulate the data how we want. All of this must be achieved without using the 2.4 GHz band. We identified three main concepts.

1. **OpenHD -** [Introduction - Open.HD (gitbook.io)] would let us combine data read from the flight controller and the camera and send both to the ground using MAVLINK protocol. It is a nice solution as it is a well supported open source project for drone development. OpenHD leverages standard wifi chipsets by modulating the signals in a way that allows for better SNR at the cost of other features normally present in standard WiFi communication, meaning the range achievable was much greater than what would have been possible with the same hardware transferring normal WiFi packets (IEEE 802 type beat).
2. **Cellular Data** would allow us to achieve infinite range, but it is less reliable than direct connection based solutions. One extra step introduced by using cellular data is that a relay server must be set up that handles the messages from the plane and sends them to the ground. It is also heavily dependent on cellular reception, which we've found to be alright in Texas, but cannot guarantee.
3. **Point-to-Point WiFi.** This was our original design choice last year. The goal was to use a 2.4Ghz Yagi Directional Antenna and build a tracker to keep it pointed at the plane at all times to maintain a strong connection. This year we'll need it to be a 5Ghz antenna if we want to choose this. Point-to-point wifi will have largely the same software architecture as the cellular system, but without the handler server, meaning it reaps all the same benefits except range while also being lower latency.

OpenHD was not chosen for the sole reason that it restricts our flexibility too much by locking us into specific hardware / operating systems that are much more difficult to develop with than a stock Raspberry Pi, which the other two systems would run. As our design team needs to operate on such short timelines with such a small amount of man-power, we must remain as flexible as possible. Hardware is not flexible, software is. By relying on socket connections written in Python, we are able to easily swap *how* our link is created instead of having to completely rewrite all our code to support something like OpenHD.

Point-to-Point WiFi was chosen over cellular data as cellular data has more points of failure, more "moving" parts. Both the ground and plane need a strong cellular data connection, the server itself must handle the data bidirectionally without crashing. Point-to-Point WiFi's main disadvantage is requiring a way to keep an antenna pointed at the plane. Because we will have to use 5GHz wifi chips and antennas, we need to supplement the loss of range by using a directional antenna. This directional antenna poses a new design challenge of how to keep it locked to the plane, which we will discuss later.

Moving forward, the system will be assumed to use point-to-point WiFi, but the cellular data option will be kept as a Plan B.

2.4 System Architecture

Should I be translating things into requirements? Yes. But someone else can do that tbh.

1.
2. The propulsions and control servos
3. The servos for the PADA drop

We can first break our system into a few fragments that are largely decoupled, ignoring the PADA and GTV entirely.

In the sky:

4. The sensors
5. The power supplies for the plane
6. The plane software for data transmission

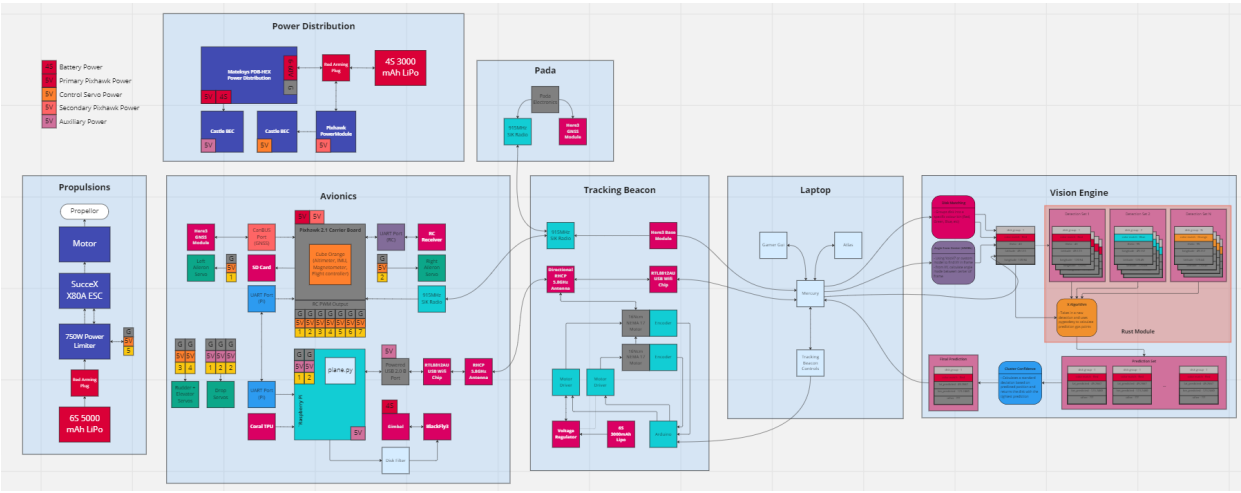
Inbetween:

7. The hardware for data transmission on the ground and plane

On the ground:

8. The tracking beacon
9. The data distribution to the ground system
10. The algorithm
11. The GUIs

Let's take a peak forward to where we're going:



Sensors

The basic sensors we need to accomplish

3.X Optimizations

Now that we understand the core system, we can begin thinking about ways we can optimize this. So far, two key design decisions were made.

1. The use of a human to do the job of YoloV7.
2. Filtering out disks

4.X Contingency

A few systems remain untested. etc etc.

- Cellular data instead of tracking beacon