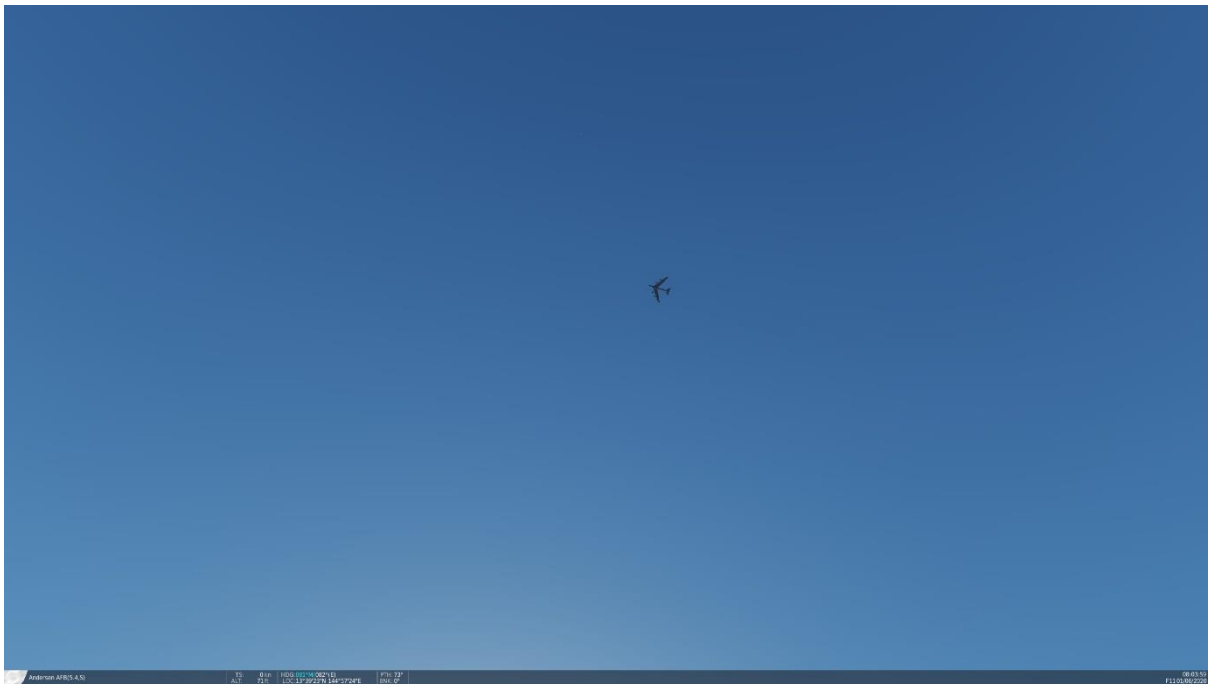
**Query. How do I automatically go from this:**



**To this, without the use of an active sensor (i.e. radar):**

{"Coordinates": [13.656864281456352, 144.96277581585085],
"Altitude": 5875.418048686289,
"Type": "B52"}

**Answer:**

Some AI, some code, and some basic mathematics.

**What problem am I trying to solve?**

Background

Some background knowledge on air combat is probably required here. I have copied the background I provided in my README file below:

Air warfare is the business of maintaining a higher situational awareness than the enemy. See: https://csbaonline.org/uploads/documents/Air-to-Air-Report-.pdf

This has been a fact since the earliest days of war in the air.

> "The enemy must be surprised and attached at a disadvantage, if possible with superior numbers so the initiative was with the patrol... the combat must continue until the enemy has admitted his inferiority, by being shot down or running away."
>
> *Oswald Boelcke in the "Dicta Boelcke", a World War 1 era flight combat manual.*

German aces of World War 2, such as Erich Hartmann (352 kills) and Gerd Barkhorn (302 kills) stressed what they referred to as "ambush tactics" in Europe. In the Pacific, American aces such as Richard Bong (40 kills) and Tommy McGuire (38 kills) developed virtually identical tactics.

A detailed analysis of 112 air combat engagements during the Vietnam War conducted by the US Air Force concluded that 80% of aircrew shot down were unaware of the impending attack.

> "Despite vast changes in aircraft, sensor, communication, and weapon capabilities over the past century, the fundamental goal of air combat has remained constant: leverage superior [Situational Awareness] SA, to sneak into firing position, destroy the opposing aircraft, and depart before other enemy aircraft can react."
>
> *Trends in Air-to-air combat - Implications for Future Air Superiority, John Stillon, CSBA report.*

The Problem

Modern sensors primarily focus on radar detection of hostile aircraft.

There are several problems intended to be solved by this project:

1. In order to avoid being surprised by radar guided munitions, modern military aircraft usually come equipped with a radar warning receiver. This set of antennae and processing equipment passively "listen" for radio waves used by radar and provide the pilot with warnings on (i) the presence of a radar, (ii) when the radar is tracking the aircraft, and (iii) when the radar is guiding a missile onto aircraft.

   Radars cannot be kept on indefinitely. Imagine a dark room. Turning the radar on is the equivalent of turning on a torch in the said dark room. Anti-radiation missiles can home in on the radar emissions. Certain aircraft are also specially equipped to triangulate the position of an emitting radar to provide coordinates to other precision guided munitions (see: AN/ASQ-213 HARM targeting system).

   Combining radar detection with other, passive sensors, permit greater SA than radar systems alone.

2. Missiles rocket motors do not burn the entire time of flight. They usually only have a few seconds of fuel before the missile relies on momentum to coast to the target (exceptions such as the MBDA Meteor exist, but that is an exception). This means that missiles can be kinematically defeated by simply turning in the other direction. The further away the target is when launch occurs, the more reaction time the pilot has.
   A passive sensor would permit the radars to remain off until the target enters the no escape zone (where the missile will still catch the target if it turns and runs and there is accordingly a high Pk (kill probability)).

3. No sensor net is perfect, additional passive sensors that can be acquired cheaply (cameras and consumer GPUs) offers a capability boost with a small expenditure of resources.

**Theoretical example of how this system should work:**

Imagine for a moment that your country is at war with an adversary that has an actual air force possessing capable SEAD (suppression of enemy air defence) assets. Turning your radars on is basically an invitation for your radar to be shot at, and you can only keep them on for a short time before needing to be moved. Or maybe the adversary has highly trained pilots who are skilled at low-level navigation and terrain masking, popping up into view of your radars when they are taking that 30-degree cutaway when climbing for their strike 3 miles from their target.

This system when paired with a network of optical sensors, will tell you that there are some aircraft in the air, it will tell you what type of aircraft is on the way, and it will tell you where they are, giving you coordinates and altitude.

Now imagine that a group of red air strikers has crossed the border. They were picked up by the optical system and are displayed on a map where data from various sensors are fused. You take a look at the situational awareness (SA) picture and see that they're going to directly overfly a NASAMs (ground based AMRAAMs, with onboard radar) site. Because you haven't looked at them with a radar, they think they have not been spotted. Well, big mistake on their part. You contact the NASAMs site and tell them "hey, you've got some bad guys going to overfly you soon, be ready to fire".

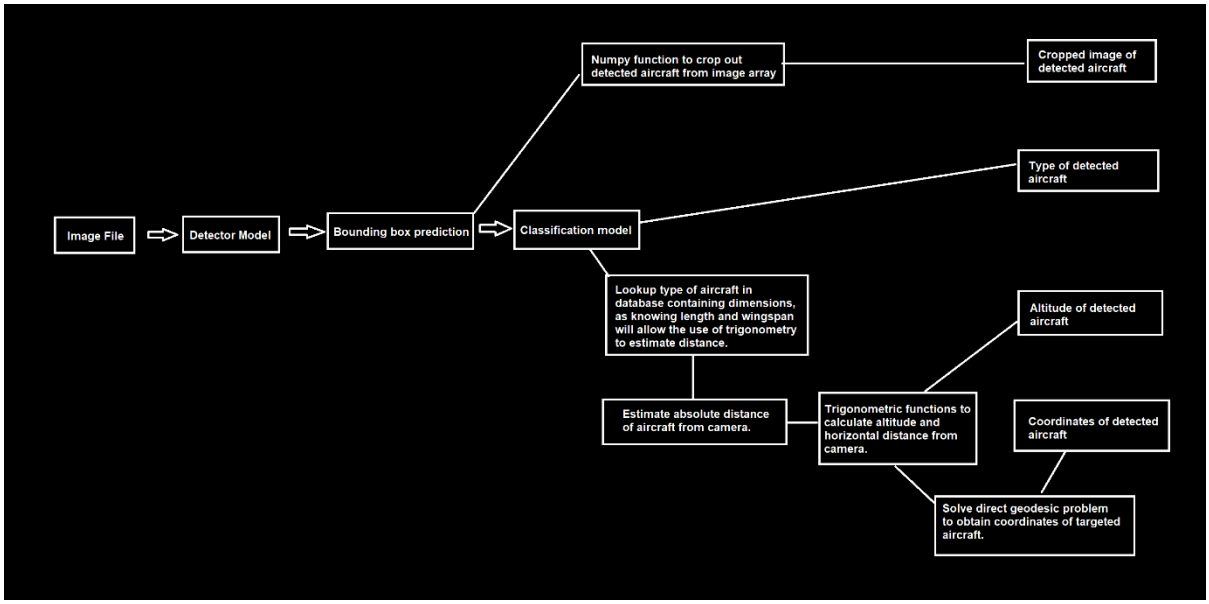
Since your sensor-fused integrated air defence network knows where the hostile jets are, the sensors used by the NASAMs site immediately cue onto the aircraft when turned on. The missiles are launched, at close range and the hostile aircraft have 3 seconds to react. It's over for them, and whatever it was they were trying to bomb is safe.

The optical system is there to catch the occasional leak from the existing sensor net.

This was the idea I had in mind when developing this program.

**System development plan:**

When dealing with a project with a string of requirements, I like having a flowchart:




Based on that flowchart, here's what I figured I had to develop:

1. Object detection model for detecting aircraft.
2. Classification model for identifying the type of aircraft.
3. Function to extract the detected aircraft from an image to display to a human operator (the man-in-the-loop, though if a weapons-grade track were acquired, this human could be replaced with an IFF interrogator, but that's outside the scope of this project).
4. Database of aircraft and their dimensions.
5. Set of functions to compute:
   (a) absolute distance of the aircraft from the sensor;
   (b) altitude and horizontal distance from the sensor; and
   (c) coordinates of the aircraft in decimal degrees.

**Dataset**

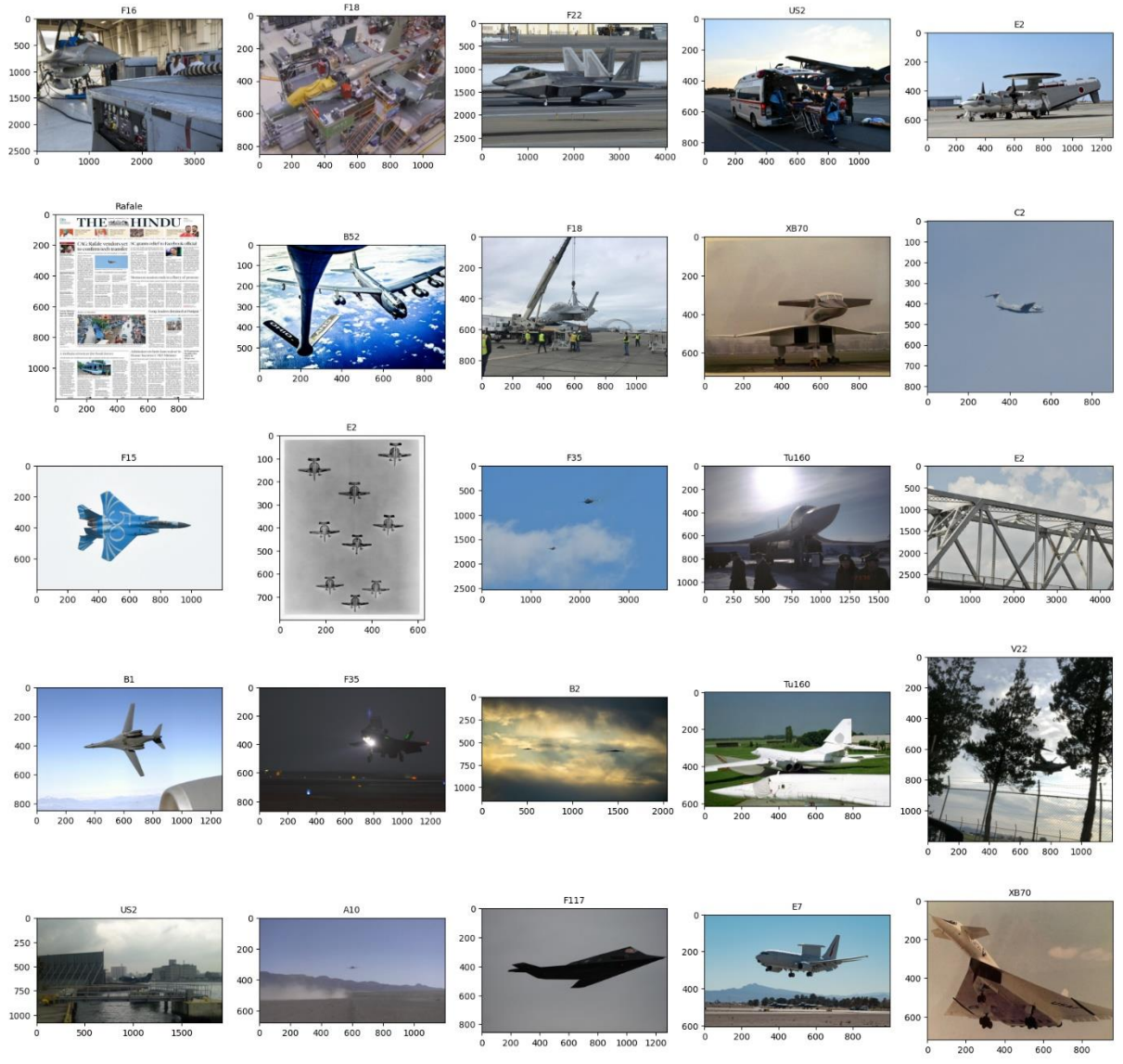
For an actual, real-life system, a custom dataset would be ideal, consisting of both visible light and infrared images (at least 1,500 each per type of aircraft). However, I did not have the time to grab airshow footage (everybody shows off their toys, which makes getting this data surprisingly straightforward), extract every frame from the videos, and annotating hundreds of thousands of images.

I did what most aspiring data scientists do, and grabbed a 12GB dataset from Kaggle:

https://www.kaggle.com/datasets/a2015003713/militaryaircraftdetectiondataset

This actually consists of two datasets.

The Detection Model Dataset




This is what the training images look like. Each image is accompanied by a .csv file containing the following information:

| filename | width | height | class | xmin | ymin | xmax | ymax |
|---|---|---|---|---|---|---|---|
| 0a1b628512650c43246c152c409f9c41 | 1200 | 800 | US2 | 773 | 307 | 929 | 354 |

Sorting these for model training was simple enough. The following steps were taken:

1. Iterate through every single .csv file, saving the filename (which is shared by each image and the corresponding .csv file) and the corresponding “class” tag into a dataframe.

2. Split the dataframe into train, test, and validation sets, stratified by class so each subset has the same proportions of each class.

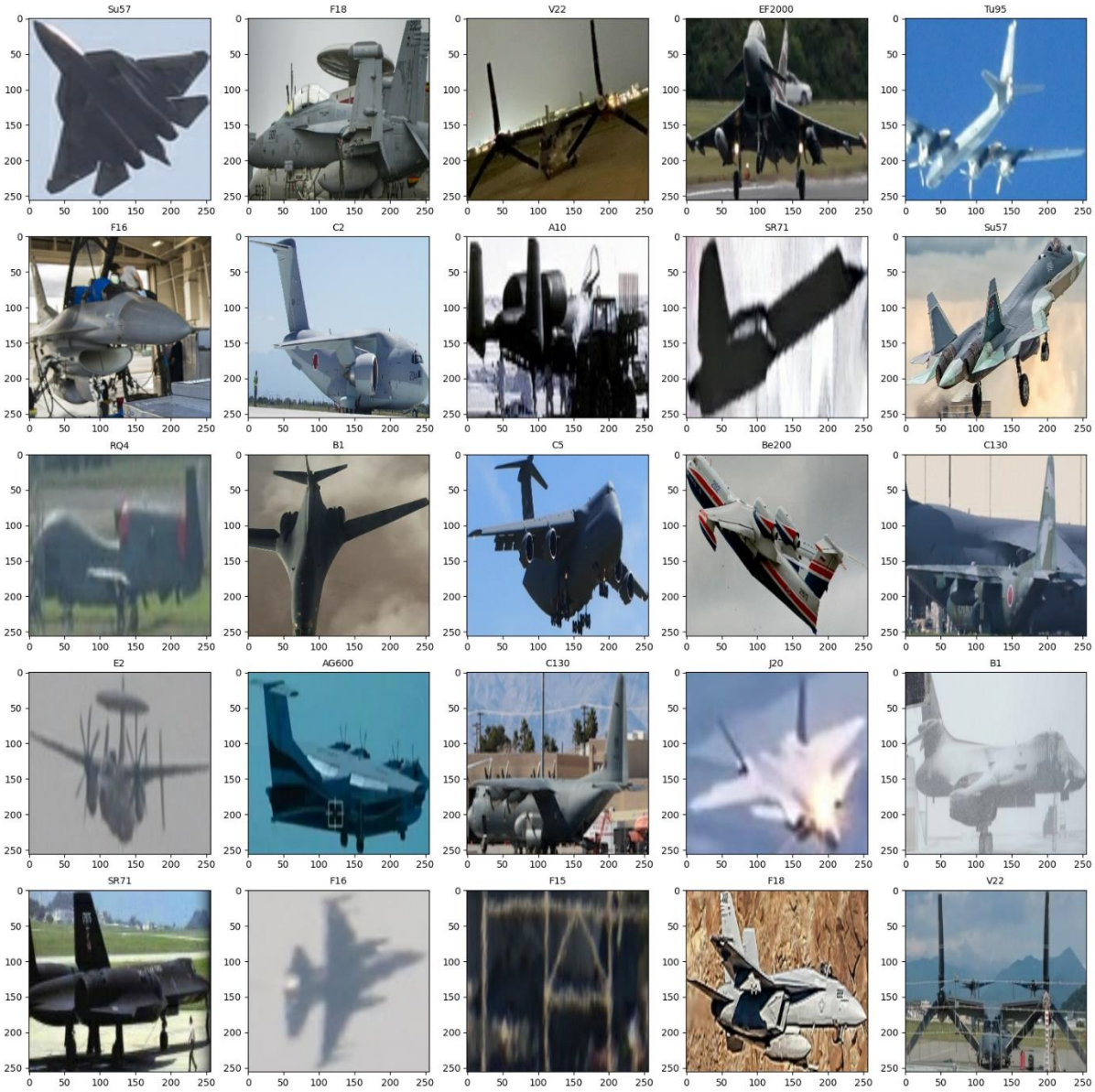
3. Use that dataframe and some code to make copies of each image and corresponding .csv file in train, test, and val directories.

4. Generate .yaml files to tell the model what to look for.

<u>The Classification Model Dataset</u>

This was fortunately, more straightforward. It consisted of thousands of images, but sorted into labelled directories.

When loading the images into a tensorflow dataset, I resized them into squares for the model to understand. The resized images looked like this:




Fortunately for me, the keras_core data loader automatically splits the data into train and val.

## Modelling

This was challenging mainly because of the sheer amount of time it took. I have a single RTX4090 which took quite a while to train each model (around 12 hours per detection model).

The results of the models are below:

**Detector**

| Model | Epochs | Precision | Recall | mAP50 | Remarks |
|---|---|---|---|---|---|
| Yolov8 | 300 | 0.229 | 0.171 | 0.122 | Wholly unacceptable |
| RT-DETR | 100 | 0.762 | 0.756 | 0.753 | Optimiser - SGD |
| RT-DETR | 200 | 0.751 | 0.742 | 0.708 | Optimiser - SGD |
| RT-DETR | 300 | 0.754 | 0.727 | 0.745 | Optimiser - SGD |
| RT-DETR | 100 | 0.767 | 0.7 | 0.728 | Cos LR Scheduler |
| RT-DETR | 200 | 0.754 | 0.692 | 0.719 | Cos LR Scheduler |
| RT-DETR | 100 | 0.468 | 0.401 | 0.363 | Cos LR Scheduler + extra mixup augmentation |
| RT-DETR | 200 | 0.649 | 0.614 | 0.517 | Cos LR Scheduler + extra mixup augmentation |
| RT-DETR | 300 | 0.712 | 0.652 | 0.662 | Cos LR Scheduler + extra mixup augmentation |
| RT-DETR | 400 | 0.713 | 0.683 | 0.687 | Cos LR Scheduler + extra mixup augmentation |
| RT-DETR | 100 | 0 | 0 | 0 | Optimiser - Adamax |
| RT-DETR | 100 | nan | nan | nan | Optimiser - AdamW |
| RT-DETR | 200 | 0.758 | 0.759 | 0.759 | Optimiser - SGD + extra augmentation |
| RT-DETR | 300 | 0.751 | 0.737 | 0.715 | Optimiser - SGD + extra augmentation |

**Classifier**

| Model | Optimiser | Train Acc | Train Loss | Val Acc | Val Loss | Fit time |
|---|---|---|---|---|---|---|
| ConvNext Base | AdamW | 0.9811 | 0.0647 | 0.7831 | 1.1933 | 58m 29s |
| ConvNext Base | RMSProp | 0.9760 | 0.0702 | 0.7766 | 1.3348 | 51m 11s |
| ConvNext Base | Adam | 0.9761 | 0.0743 | 0.7724 | 1.1510 | 50m 41s |
| ConvNext Base | Adamax | 0.9981 | 0.0330 | 0.8031 | 0.8546 | 1hr 12m 39s |
| ConvNext Base | SGD | 0.9620 | 0.2507 | 0.7802 | 0.8217 | 12hr 17m 2s |
| ConvNext Base | Adadelta | 0.9978 | 0.0091 | 0.9100 | 0.5438 | 76m 13s |
| ConvNext Base | Adagrad | 0.0362 | 15.5349 | 0.0384 | 15.5020 | 52m 52s |
| ConvNext Base | Nadam | 0.5144 | 1.6397 | 0.1886 | 4.7987 | 1hr 59m 7s |
| ConvNext Base | Ftrl | 0.0619 | 3.6134 | 0.0553 | 3.6123 | 2hr 47m 16s |

Comments on the detector

This would probably benefit from a new dataset consisting only of aircraft in flight (and many more instances per type). The dataset includes aircraft missing wings on the ground. This is unrealistic for the projected use case.

While the best set of metrics were around 0.76 for both precision and recall, the model actually performed quite well in detected aircraft in flight:


I thus decided to try using this model to feed cropped images into the classification model.

Comments on the classification model

The classification model reported high levels of accuracy in determining the type of aircraft:

| | | | | | | |
|---|---|---|---|---|---|---|
| ConvNext Base | Adadelta | 0.9978 | 0.0091 | 0.9100 | 0.5438 | 76m 13s |

99.78% accuracy on training data and 91% accuracy on validation data.

Overall remarks on modelling

If further development were to be undertaken, I believe this system would benefit from a far larger dataset (and a GPU with more VRAM). This dataset can be obtained by extracting frames from airshow footage and annotating each frame for model training.

It might also be worth intentionally overfitting the models as there are only so many angles which you can look at an aircraft from.

**Locating the aircraft**

This was an exercise in basic mathematics – trigonometry.

The dimensions of the aircraft are known. If you have the degrees per pixel of your camera, you can quite easily calculate the distance using trigonometric ratios.

The problem was getting images containing:

1. degrees per pixel of image;
2. geographic coordinates of camera;
3. azimuth which the camera is pointing at;
4. elevation angle of the camera; and
5. altitude of the camera.

Conveniently, there's a combat flight simulation video game – DCS, which tells you all of that.

To test, this, I built a UI to feed all that information into the models and functions. You can see it working here:

https://www.youtube.com/watch?v=MkIymKNswYQ

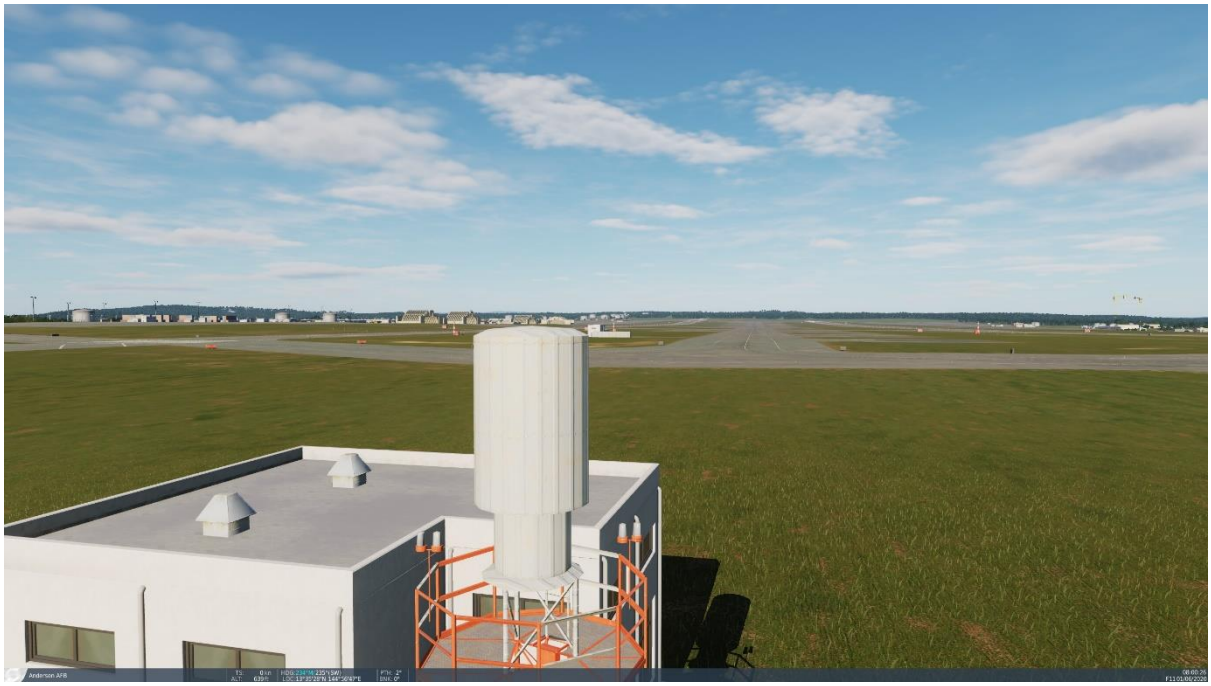
**What just happened?**

The model took in the image, information on where the camera is and where it is looking, giving us coordinates, altitude, and type!

I should also highlight that the program takes into account where the object is on screen, and adds or subtracts the appropriate angle from where the camera is pointing. Let's walk through the results of the video demonstration.

Here is the image it was working with:

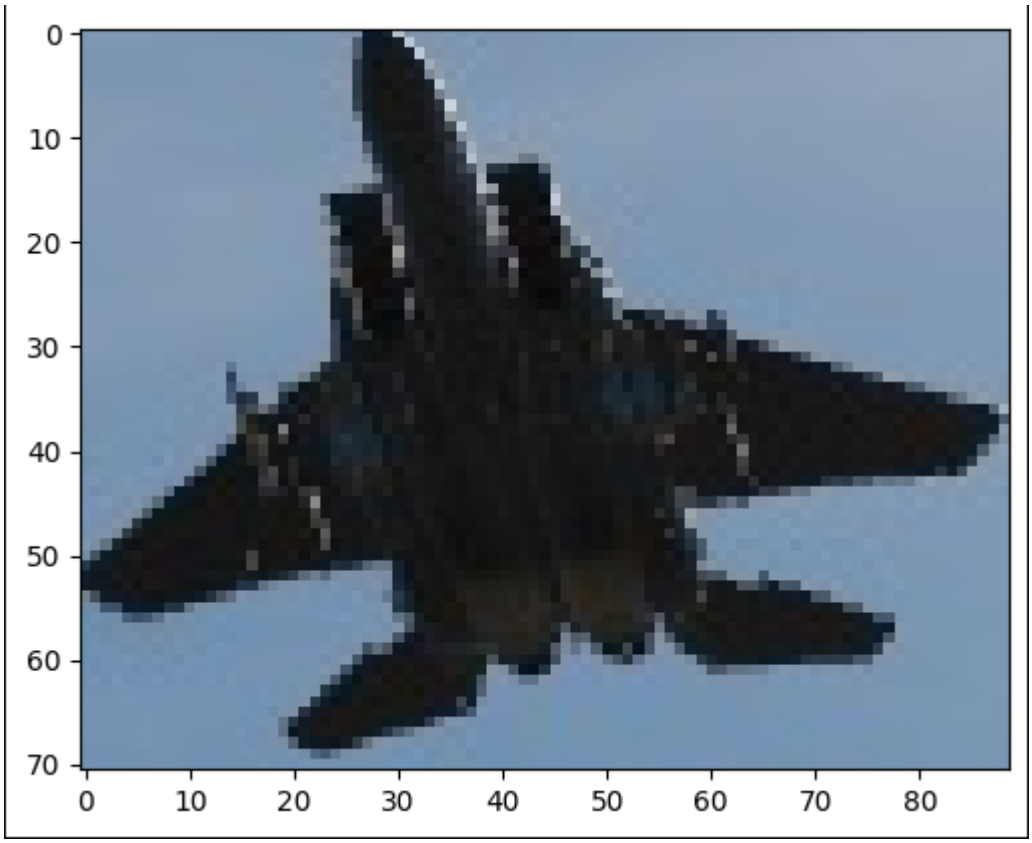
The camera was positioned at the little hit between the start of runway 24L and 24R and Andersen AFB in Guam:



Let’s take the coordinates from the images and put them into google maps to verify that it is giving us the correct information:


The hut is indeed there in the real world.

Now, here is what the program gave to us:

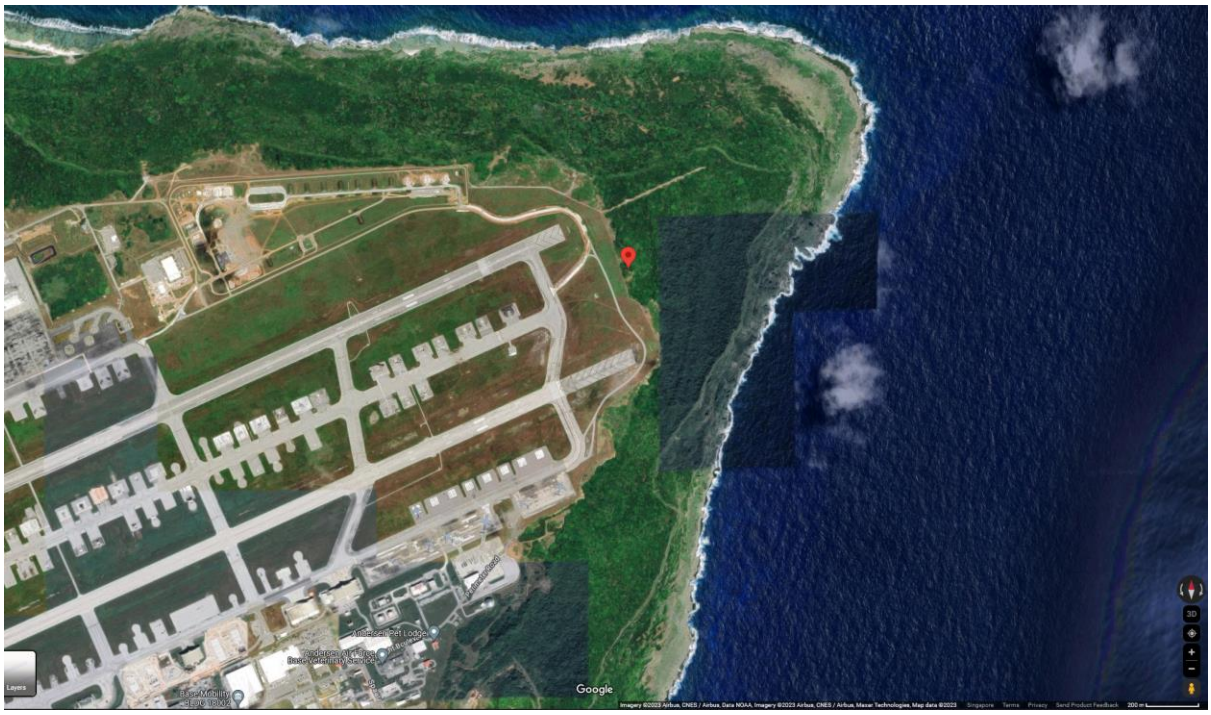



{"Coordinates": [13.592156214424612, 144.94771157856493],
"Altitude": 1103.6660373560921,
"Type": "F15"}

**Is that correct?**

Well, that is an F-15E Strike Eagle. The type is correct. It was set to between the runways of Andersen AFB, and at the time the image was captured, it was over the wooded area to the east of the airbase.

Pasting the coordinates into google maps gives us this:




That is indeed where the aircraft was at the time.

What about altitude?

The aircraft was set to fly at 1,000 feet (Negative Ghostrider, the pattern is full! [Heard and ignored]) {Top Gun reference, in case you haven't seen that film}.

The program determined that the aircraft was at 1,103 feet. 103 feet is not a lot in aviation, and it is precise enough to cue another sensor on target.

**Conclusions**

The system works.

Further development can be undertaken to:

1. Pipe in a video feed, and use the models to infer on one frame every second (Additional software functions required to extract the frames);
2. Build hardware to automatically pipe in camera location information and data on where the camera is looking (probably from a gimbal); and
3. Changing the file output from JSON to a format which can be accepted and understood by the central system where sensor fusion takes place.