algorithm design

GOAL

we want to be able to predict where an object will be at an arbitrary time and intercept that object

the accuracy needs to be high enough for defence level stakes

therefore the target MUST get hit. 1 shot is enough to neutralise the target.

- that means we can do 1 shot really well by directly aiming at the target(Target-As-Target) -

-or-

- we can aim at a cell containing the target and spray multiple shots(Cell-As-Target), all this is to maximise the interception area and subsequently, the probability of hitting the shot.

WARNING: **it is crucial to note that the differences in the algorithms taken, have hardware design consequences. in the event of such consequences, favour superior algorithm and fit hardware design onto the superior algorithm. (superior algorithm > hardware design)**

Philosophical Background

in all our operations, we need to minimise number of processes/compute as our mission here requires that we do a lot of those in a constrained time frame.

We therefore need to think of efficiency and efficiently.

starting with this simple example, suppose at some point in our operations we are going to work with mean averages. There is a more efficient method than the traditional way of

- summing up n elements and dividing by n.

see it below:

moving average formula

remember, when calculating a mean average, it is computationally cheaper to use the recursive expression than the batch expression

$$\bar{x}_k = \bar{x}_{k-1} + \frac{x_k - \bar{x}_{k-1}}{k}$$

this operation has a big(O) gain over the everyday common average formula

to get an appreciation please research this as it is an important precursor for the algorithm we intent to use to predict intercept point (KALMAN FILTERS) which is the real project if we are being honest.

But First, A little House Keeping…

Environment Design

Before diving into the algorithm, it is important to define the field of view formally

This is important since a coordinate system allows for precise aim

Take this example idea: *if we breakdown the field of view into a cartesian plane, we can get a 2D view which we can slice up according to our gun's degrees of freedom (perhaps this is better in spherical coordinates)*

Therefore, i suspect that the gun's degrees of freedom will determine our mapping of field of view into coordinates

Lets consider an example where our gun has 180 degrees of freedom in the x & 90 degrees in the y axis

This means it can aim from level to straight up and from left to right and everythhing enclosed to the degree(if we have precise gear control)

We can therefore, for each coordinate, keep a mapping of the commands to send to our control system which then aims the gun for that point when the said command is carried out.

It would be best if the system did relative aiming, meaning to say if we get an array of points [A, B, C] to aim and fire, it doesnt have to return to origin every time it aims and fires on a point [A]

for example, instead it must move relative from where it was [A] to the next point [B] (this may even give us room to implement optimum graph traversal algorithms - shortest path from point A to B)

only when it has been idle for sometime do we return to rest

Coordinate choice (according to chatGPT)

- Use spherical / pan-tilt angles as the *control* coordinates because your actuators are rotational.
- Use Cartesian (camera or world) for sensor fusion and projectile physics. Convert between them as needed.

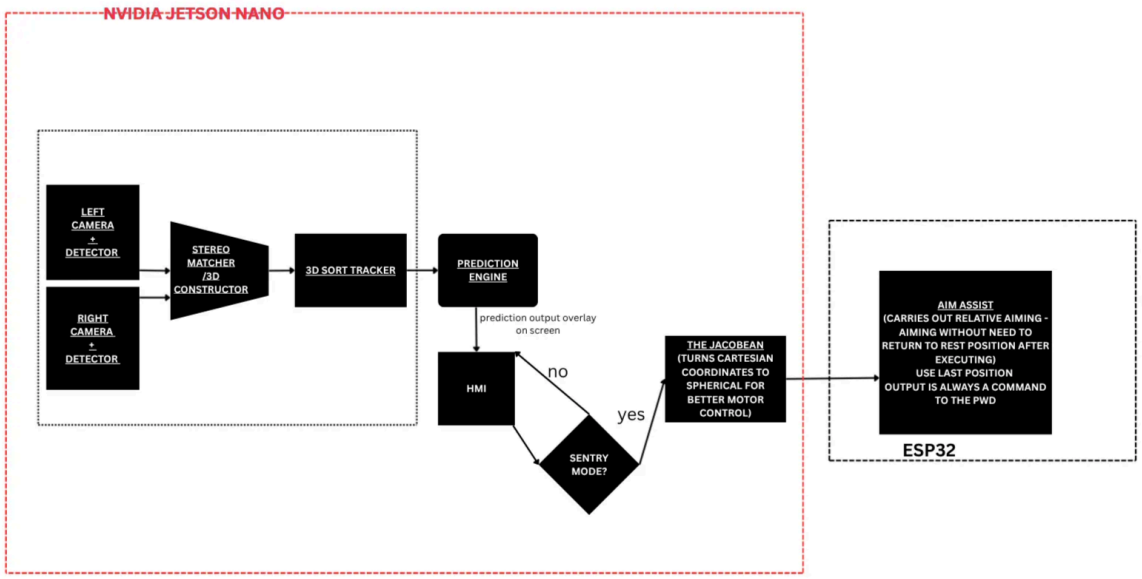
…we shall be working with 3 dimensions. Here's how

with 2 POVs whose focal length and distance apart you know, you can calculate depth.

Software Question:

do 2 cameras mean 2 inference pipelines with 2 SORTs or 2 inference pipelines with 1 SORT?

Answer: *Check out the architecture below. (2 synchronised inference pipelines with 1 SORT)*



architectural overview concerning coordinates

Principle:

1 bounding box (x1, y1, x2, y2) is given in 2 perspectives and the 3rd dimension is calculated depending on camera set up (ideally to turn a bounding box into a bounding cube***** interesting)

NB: the same bounding box at the very same instance, no phase difference in time at all between the 2 boxes

Mathematical Calculation of 3rd dimension

1. Known Quantities

- Left camera focal length f (in pixels or meters).
- Baseline B = distance between the two camera centers (meters).
- Image coordinates of the same object's centre in both cameras:

  $(x_L, y_L), (x_R, y_R)$

  obtained from the bounding boxes.
- The optical axes are rectified (horizontally aligned, epipolar lines are horizontal).

2. Disparity

$d = x_L - x_R$

Units: pixels.

The **smaller** the disparity, the farther the object and vice versa

3. Depth (Z)

$Z = \frac{f\,B}{d}$

4. Lateral coordinates (X, Y)

Measured relative to the left camera’s optical axis:

$X = \frac{(x_L - cx)Z}{f}, Y = \frac{(y_L - cy)\,Z}{f}$

where cx,cycx,cy are the principal point coordinates (image centre).

5. Result

$P_3D = (X,\ Y,\ Z)$

This point represents the 3D position of the target centre in the left camera’s coordinate frame.

6. Example (numeric)

Baseline B=0.2m, focal f=1200px

$x_L = 640, x_R = 620, c_x = 640, c_y = 360$

$d = 20$

$Z = \frac{(1200 * 0.2)}{20} = 12m$

$X = \frac{(640 - 640) * 12}{1200} = 0$

$Y = \frac{(360 - 360) * 12}{1200} = 0$

→ object 12 m straight ahead.

Enter, the Jacobean

PROCESS OUTLINE:

we are going to convert coordinates from the prediction system (which come out in cartesian) to spherical coordinates in scale with gun’s field of view

research suggests that it is much more *efficient* to convert these into spherical coordinates and send to esp32 (which will only care about theta and phi angles in spherical coordinates for servo motors pwm control)

since we are going to need to change coordinates from one form to another, we are basically perfoming the function of a jacobean (which makes the job easier ~ oh so we think)