Missile Sim

15-2-2025

Use Missile objects, where different types, states and properties are possible.

Use a Sim object that simulates the list of missiles.

To do:

* ~~Separate the Missile and Sim object~~
* ~~Fix the angle of attack~~
* ~~Log the missile states in a pandas.dataframe.~~
* ~~Add a PyGame visualisation~~
* ~~Fix fuel consumption~~
* ~~Keep orientation between -pi and pi~~
* ~~Make isolated code for controls. Orientation can be changed up to a max per s.~~
* ~~Trajectory seems dependent on the number of steps of the sim.~~
* **Incorporate the pygame engine into the simulator, or vice versa**
* ~~add pressure dependency in lift and drag (from altitude)~~
* ~~include iteration loop into the MissileSim~~
* ~~Fix the lift~~
* ~~Add target class (copy of missile class)~~
* ~~Tweak the coefficients~~
* Mach dependent drag
* ~~Put all missile dependent properties in the missile class~~
* ~~Keep it on screen al the time.~~
* ~~Angle of attack seems to jump with 180°~~
* ~~In measurement, the detection vector is calculated by a vector matrix product, instead of a matrix vector product, that flips the direction of rotation.~~
* ~~Steer with the tail. Aerodynamics to keep the missile straight, and to steer it.~~
* Add a detonation mechanism, in Missile and in playback
* Add all of this to Github, that is what it is for.
* ~~Control the target~~

## Isolated code for controls

Must be in the missile class

Based on ‘measurements’ of the target.

~~Make target, do multi object sim.~~

~~First of all there must be a target. Start with a target that moves under a small (random?) slope towards the origin? Target class.~~

For each missile, we can make a measurement object:

* Angle at which we see the target
* Approaching speed

Together with own speed and orientation (and maybe altitude) this should be enough for now.

Measurement(target, missile)

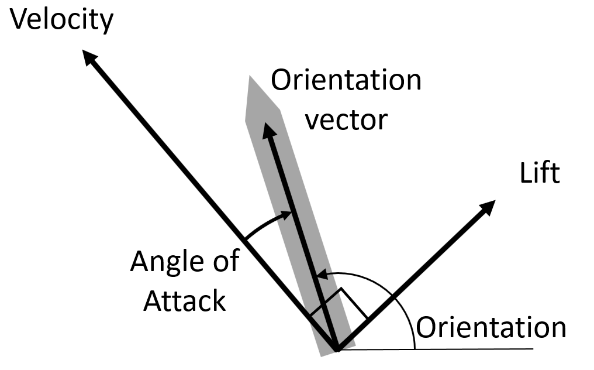
Where should this be? The missile\_sim (bad name, call it Simulator)

We can make a measurement class that in the constructor determines the allowed attributes, and those are passed to the missile class for controls.

Send the measurements to a missile.control method, that updates the orientation once.

## Fix lift

Get all signs right:



Lift is perpendicular and 90° clockwise from velocity

Angle of attack:

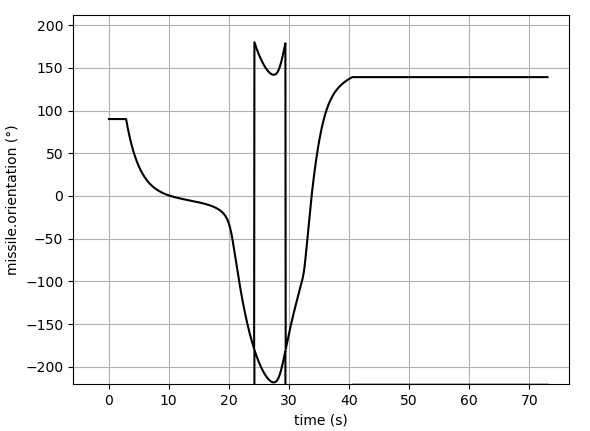
Rotate velocity with minus orientation, determine angle.

C\_lift = missile.Cl\_alpha \* np.sin(2 \* angle\_of\_attack)

lift\_force = lift\_direction \* C\_lift \* missile.lift\_area \* np.linalg.norm(missile.velocity) \*\* 2

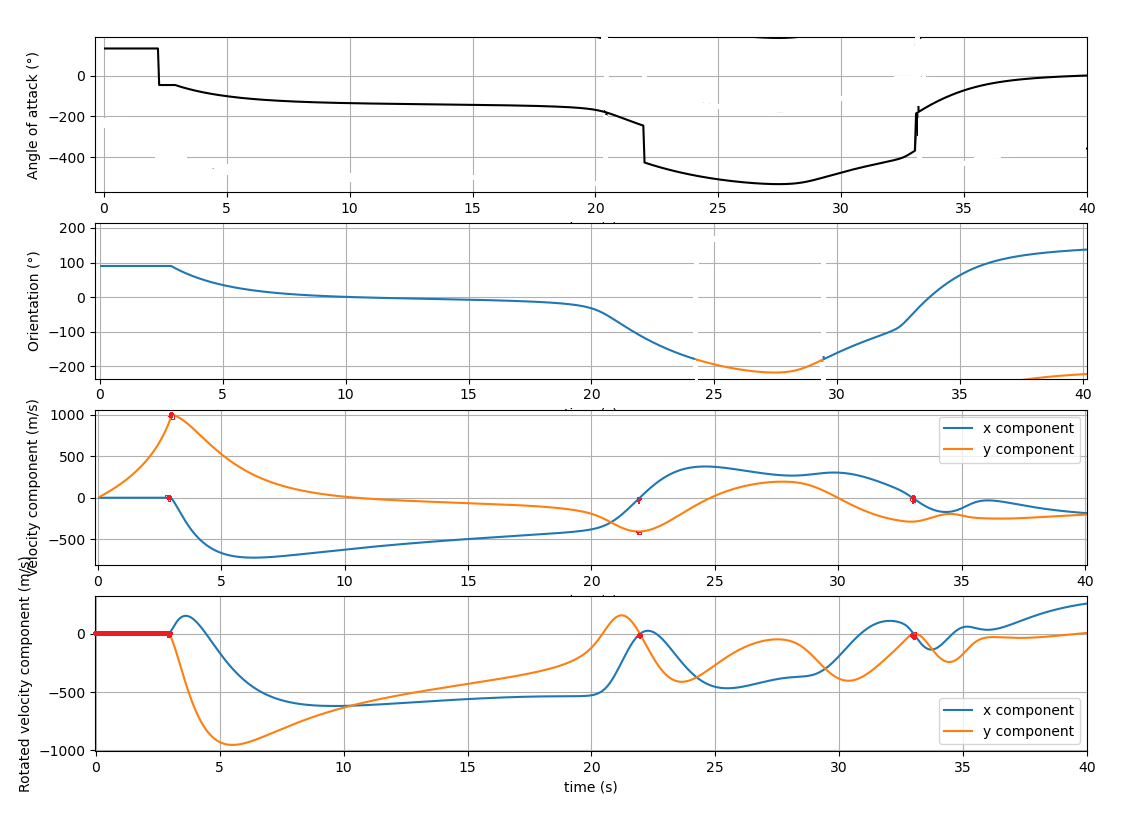
Lift is checked, everything looks good. Behaviour is as expected.

## Missile seems to fly backwards now



This missile starts upwards and turns right on the image. In the

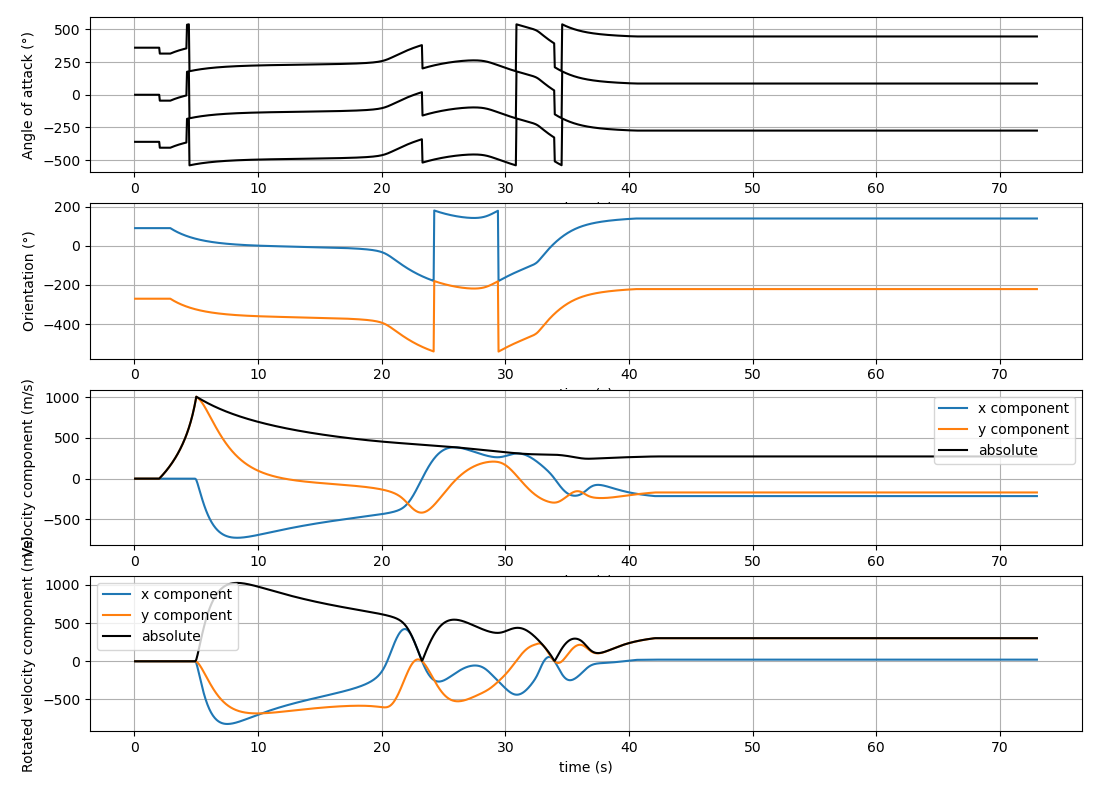
## Angle of attack seems to jump with 180°



To obtain the angle of attack, I rotate the velocity vector with minus the orientation. Then I determine the angle of the resulting rotated vector. That is the AoA.

The red marks indicate places where the rotated velocity is zero, where the original is not. That is impossible.

In the following example also the absolute velocity is drawn.



That’s not good.

Is it correct what I’m doing? The missile has a velocity vector, and an orientation, that can change independently.

I rotate the velocity vector with minus the orientation angle, which should conserve its length.

Then I determine the angle of the rotated velocity, which is (minus) the angle of attack.

|  |  |
| --- | --- |
| An Aircraft's Angle Of Attack: Everything You Need To Know | An Aircraft's Angle Of Attack: Everything You Need To Know |

## Missiles seem to fly backwards now. Orientation points tailwards

Orientation is correctly 90° at launch. But decreases to go left in the visualization. That is confusing.

But, the target starts at a negative x value, which is left, so the image is not flipped?

|  |  |
| --- | --- |
|  | (0,0,255), (100,100), 10) blue  (0,255,255), (100,700), 10) cyan  (255,0,255), (700,100), 10) purple  (255,255,255), (700,700), 10) white |

This is good to have, but if the target shows up left, we are not mirrored, so if the orientation is still pointing left at an angle of zero, the x axis on the angle is flipped, or the way the orientation is used is flipped horizontally.

I probably have to unflip it twice to fix and have consistent behavior.

Let’s run through the code so find out where the problem is:

**Missile:**

Launches at 90 deg

**Target:**

Launches at 0.2 rad

**Measurement:**

It is used wrong, added to the list, no flipping of other things. So maybe it detects at a shifted angle, but this is compensated in the controls

**Simulator:**

**Simulator.update:**

Wrapping is correct:

obj.orientation = (obj.orientation + np.pi) % (2 \* np.pi) - np.pi

**Simulator.get\_acceleration:**

Orientation vector is a row vector, looks good.

orientation\_vector = np.array([np.cos(missile.orientation), np.sin(missile.orientation)])

Thrust is in direction of orientation:

thrust\_force = missile.max\_thrust \* orientation\_vector

Looks good

Simulator.angle\_of\_attack():

Needs a way to rotate a vector. Write a helper function to reliably do that, regardless of a row or column vector.

I switched it for the rotate\_vector function of the toolbox.

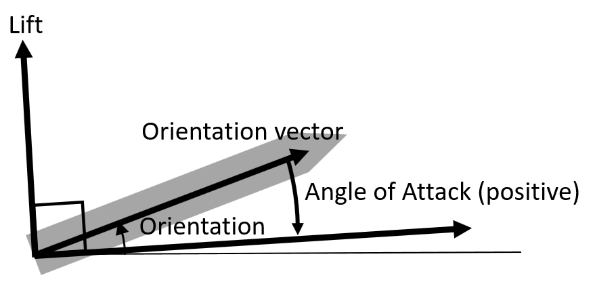
Proper angle of attack breaks the sim.

Check the usage if angle\_of\_attack

**Simulator.get\_acceleration():**

|  |
| --- |
| #Lift  lift\_direction = np.array([missile.velocity[1], -missile.velocity[0]]) / np.linalg.norm(missile.velocity) # Velocity direction rotated 90° clockwise  C\_lift = missile.Cl\_alpha \* np.sin(2 \* angle\_of\_attack)  lift\_force = lift\_direction \* C\_lift \* missile.lift\_area \* np.linalg.norm(missile.velocity) \*\* 2 |

Lift direction,



With a positive angle of attack, the lift must be in the direction of velocity rotated counter clockwise

|  |
| --- |
| lift\_direction = rotate\_vector(missile.velocity, np.pi/2) / np.linalg.norm(missile.velocity)  C\_lift = missile.Cl\_alpha \* np.sin(2 \* angle\_of\_attack)  lift\_force = lift\_direction \* C\_lift \* missile.lift\_area \* np.linalg.norm(missile.velocity) \*\* 2 |

At an angle of attack of 0, lift is zero

At a small positive angle of attack, the lift is positive

At a angle of attack of 90° the lift is zero again. Looks good.

Behavior:

|  |  |
| --- | --- |
| At launch the angle of attack is zero  At a small change in orientation, the lift is huge and pushes the missile sideways in the wrong direction. As the angle of attack goes to 90° the lift is zero, with an orientation of ~190° lift points it left again. |  |

Behavior is bad. Lift is in wrong direction.

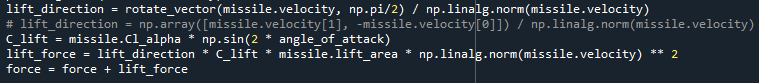
* I changed the rotate\_vector function, first check that, but now it only does the same thing with negative lift.
* Make all visible. In a matplotlip plot I guess per 0.5 second
* Check the signs of the measurement class

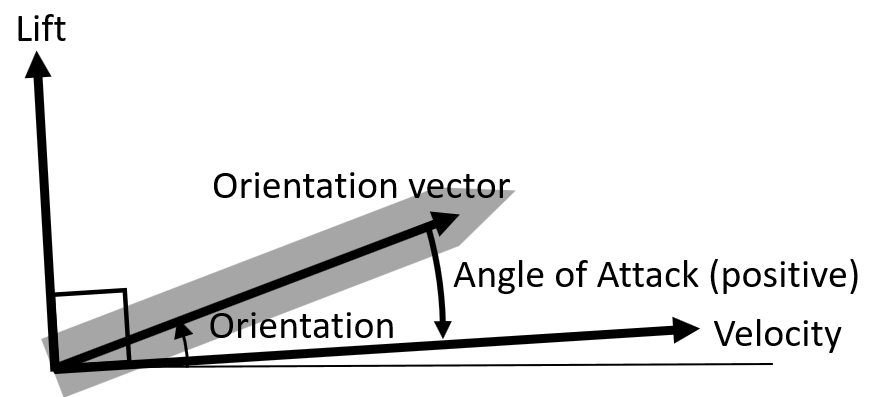
|  |  |
| --- | --- |
| Lift is in wrong direction. | When I make the lift negative, the behavior makes more sense.    Exept the orientation changes very quickly to almost perpendicular to the speed, so limited lift is expected. When the target is “behind” the missile, it just points backwards, making that its lift can push it away from the target, giving a weird trajectory. |

So lift if negative. We must find the cause.

A next point is:

Steer with the tail. Aerodynamics to keep the missile straight, and to steer it.





Lift direction is velocity rotated 90° to the left. (function works as expected)

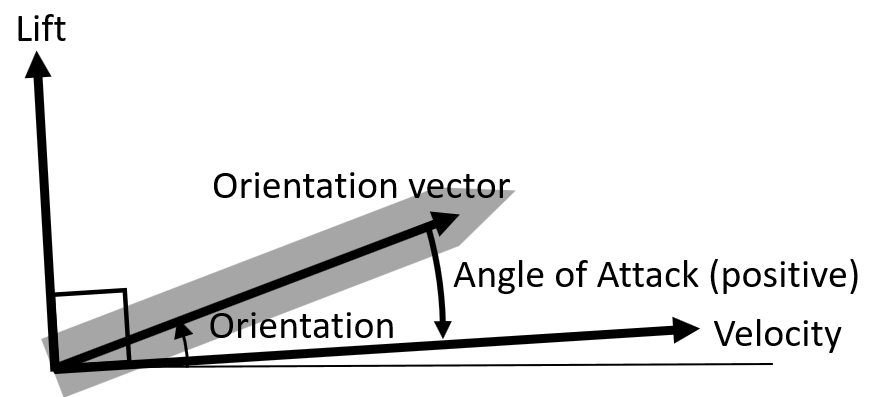
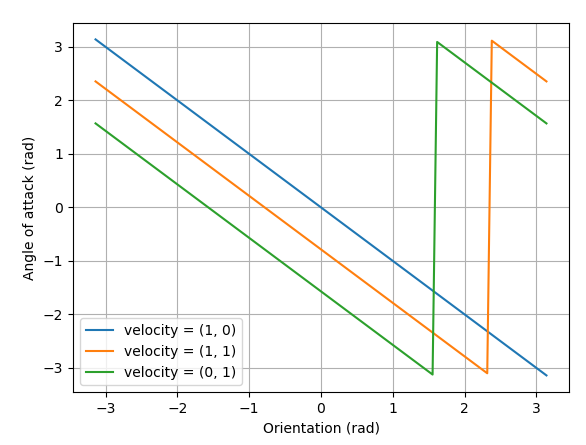
When a (small) angle of attack is positive (check

Lift should be positive.

Look at all the ingredients:

|  |  |
| --- | --- |
| C\_lift | Is positive |
| Cl\_alpha | Is positive |
| Angle\_of\_attack | Check |
| Lift\_area | Is positive |
| Lift direction | Correct direction |
|  |  |

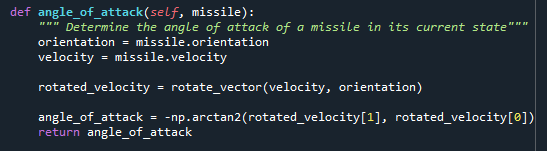
Check the angle of attack:



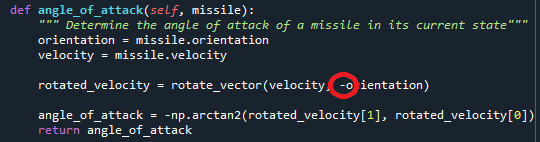
In the blue graph, velocity is horizontal to the right. An increasing positive orientation should give an increasing angle of attack, but shows here as negative.

Green, a velocity straight up, should give an angle of attack of zero at an orientation of pi/2, but actually it is pi there. AoA is zero at an orientation of -pi/2.

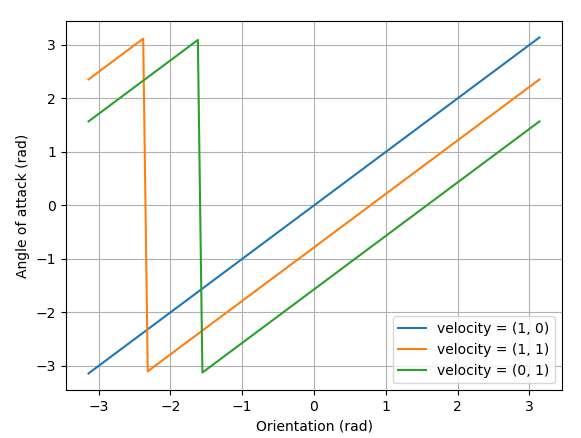
So there are two problems: The AoA scales negatively with orientation, also it scales negatively with velocity direction. The code is in Simulator.Angle\_of\_attack():



One change, completely in agreement with theory.



Results in:



Blue: Zero at the correct point, and growing in the right direction

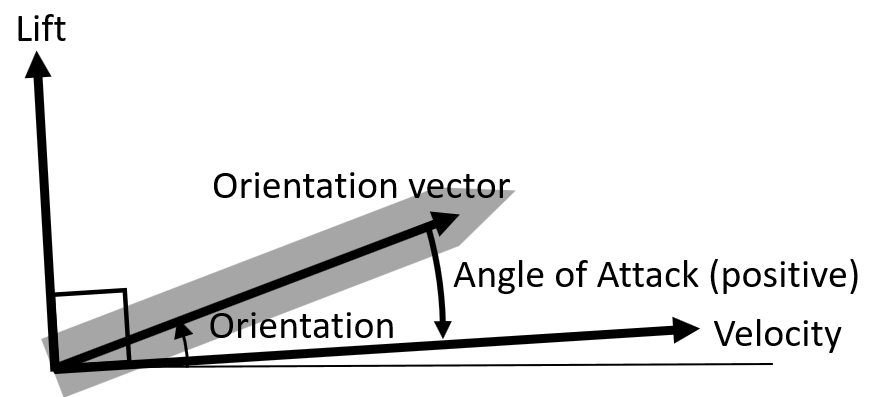
Green: Zero at the right point.

This is solved.

Behavior is still weird, because the rocket can turn its orientation very quickly, resulting in unphysical situations where it is powersliding sideways.

# Steer with the tail. Aerodynamics to keep the missile straight, and to steer it

Torque goes with sin(angle of attack). Lets make the speed proportional to the torque, to avoid osscilations.

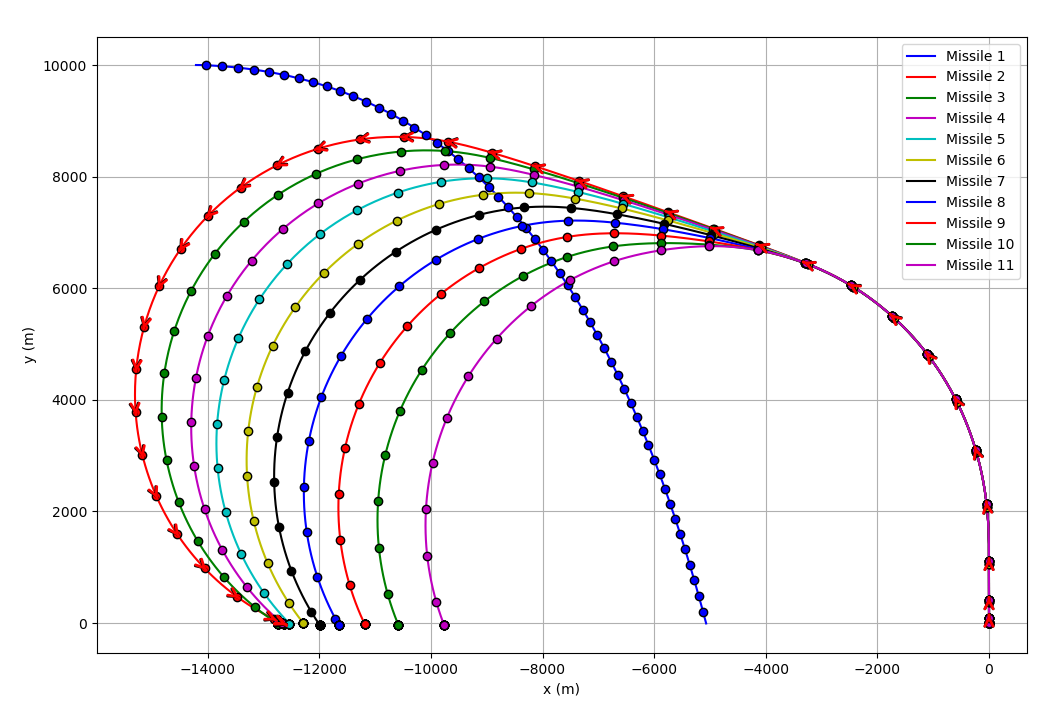


When angle of attack is positive, like in this picture, the rotational velocity that must cause, is negative. So it goes with -sin(AoA)

The missile gets a control property, from -1 to 1.

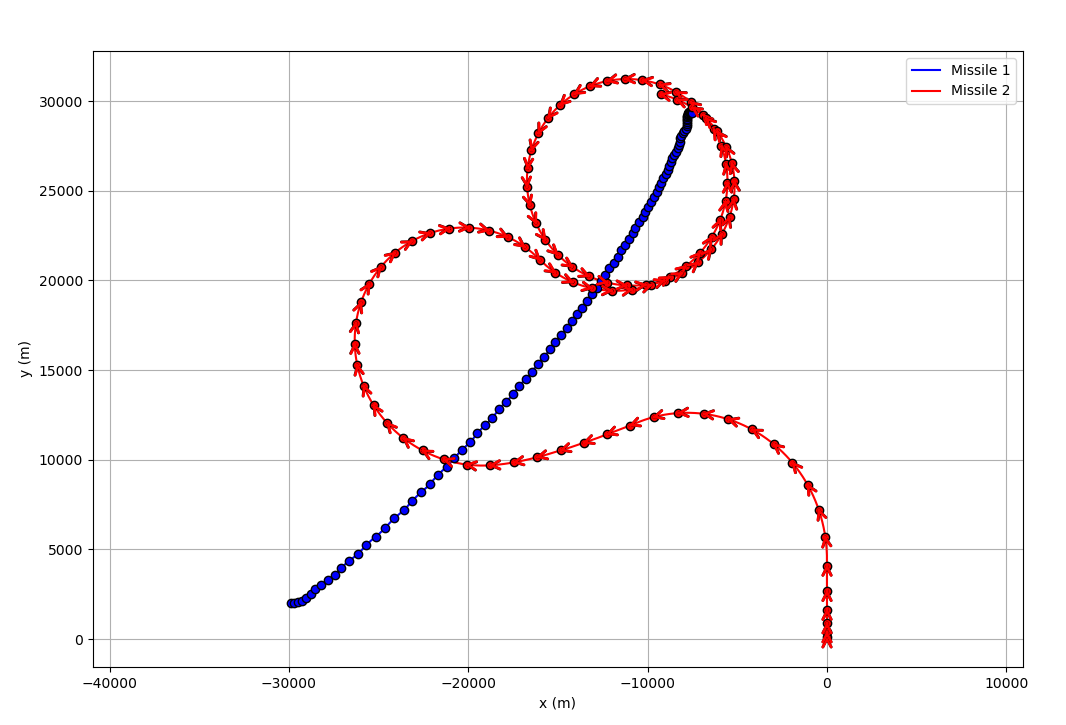
This steers the rocket by shifting the tail behavior.

When the measurement.direction is positive, we want a positive control. So we must shift the tail behavior accordingly.



That works now (1-3-2025)

Control target

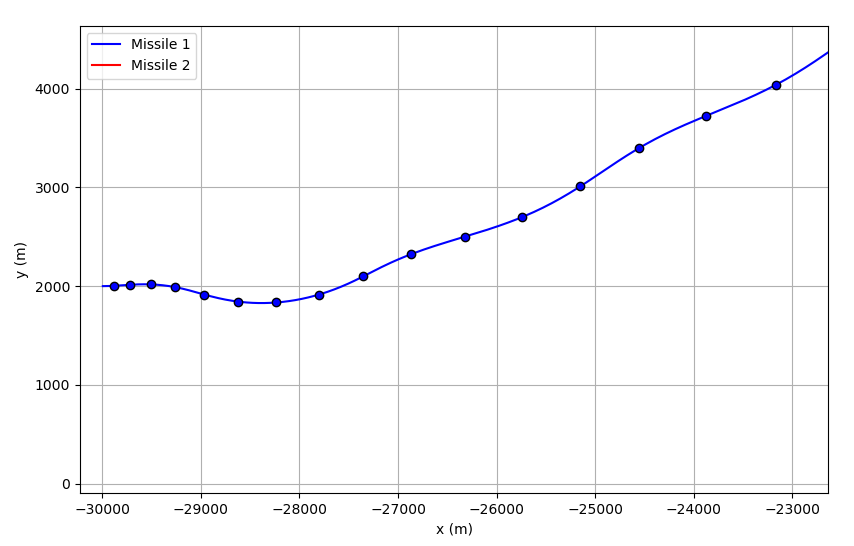


|  |
| --- |
| def control(self, measurement, timestep):  self.control\_surface = self.control\_surface - 0.1 \* timestep \* (self.position[1] - 2000) |

Thet height term is positive, making control surface negative, stuck on -1. That seems to make him pull up.

self.control\_surface = self.control\_surface + timestep \* (0.01 \* np.clip((self.position[1] - 2000), -100,100)

+ 50.0 \* self.orientation)

Making the minus a plus:  


There seems to be some feedback, it turns at 2k, but keeps climbing up to 27k, because it runs out of energy there.

The control surface is continuously 1 during the climb.

|  |  |  |
| --- | --- | --- |
| 0.01 | 0.1 | unstable |
| 0.01 | 1 | Unstable -6k |
| 0.01 | 10.0 | Small osscilations |
| 0.01 | 50.0 | Stable |

self.control\_surface = self.control\_surface + timestep \* (0.01 \* np.clip((self.position[1] - 2000), -100,100) + 50.0 \* self.orientation)

Now I want to control the angle that the missile steers on to hit the target.

|  |
| --- |
| def control(self, measurement, timestep):  """Use the measurements of the target detection to control the missile"""  self.control\_surface = self.control\_surface + 2.0 \* timestep \* (measurement.direction + **0.1**) |

|  |  |
| --- | --- |
| **Value** | **Minimum proximity** |
| 0 | 481 |
| 0.1 | 492 |
| -0.1 | 477 |
| -0.3 | 473 |
| -1 | 440 |
| -2 | 401 |
| -2.3 | 381 |
| -2.5 | 361 |
| -2.8 | 215 |

Optimum found.

# Incorporate the simulator into the pygame engine

Two modes:

Simulate while watching and rewatch and earlier simulation. (IF you can simulate in real time, it is maybe not needed to rewatch an earlier simulation. There is no benefit.) So first focus on real time simulation while watching. Keep the rest open.

We add an simulation step and visualize the last state in the logs of the objects.

There is an issue with the scaling. Before the simulation is done, we don’t know how to scale the field. For now: use a fixed size that fits the expected behaviour.