Joint Space

From the first lab I understood that we needed a rectangle in workspace but then it turned out we needed it in joint/configuration space.

We can do in two ways:

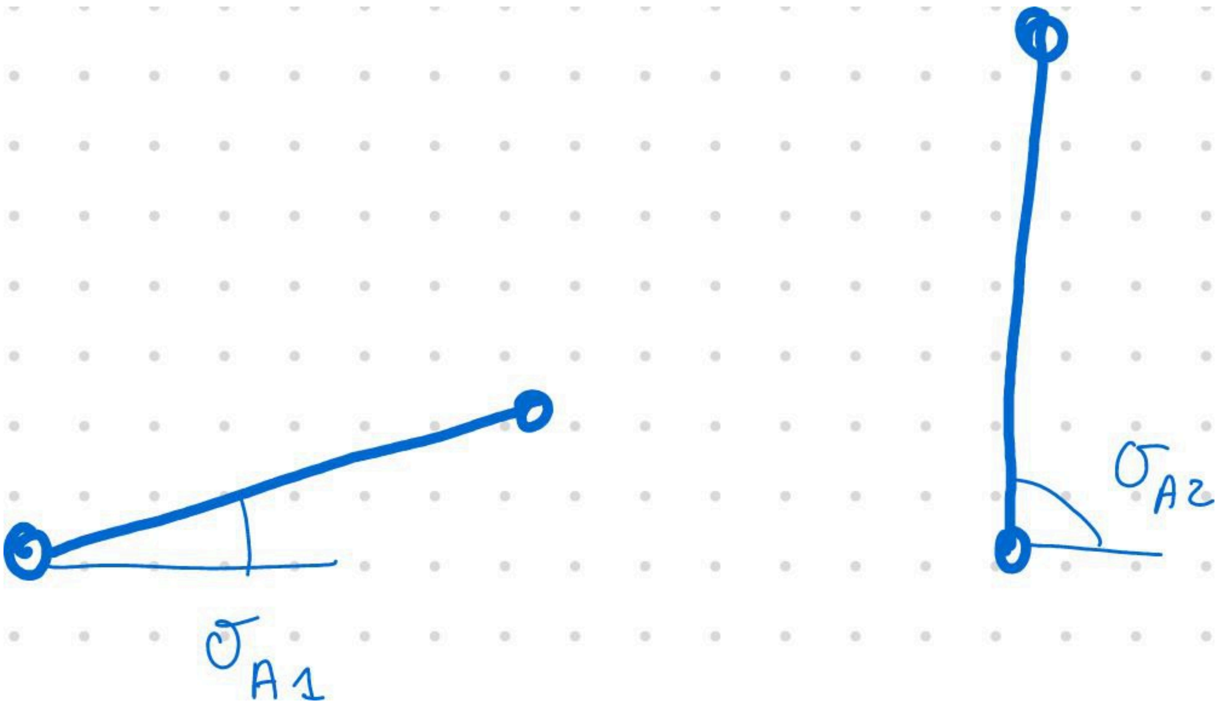
1. Sample the workspace, whenever we have an admissible configuration we save the joint configuration and plot it at the end. This solution re-uses the inverse kinematics code
2. Sample the joint space, if the direct kinematics is admissible with respect to some criteria save the pose, plot it at the end.

With 1 it's possible to save some time thanks to the code re-use but an uniform sampling in workspace doesn't translate well in joint space, leading to some artifacts that require a more fine-spaced sampling with more cumbersome calculations.

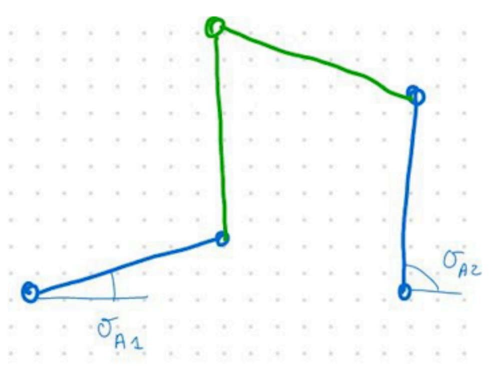
With 2 we need to write the direct kinematics function but the end result is more efficient.

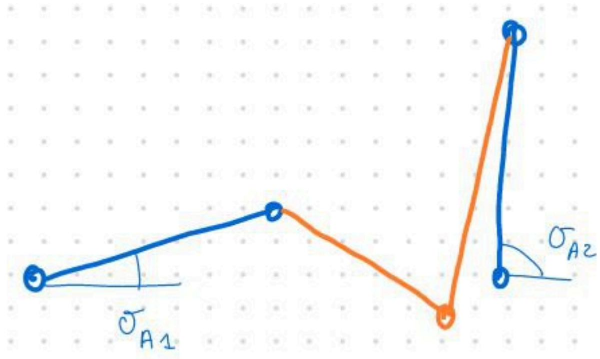
1) Direct Kinematics

Let's start with a generic configuration of our active joints:



If the configuration **admits a solution** it will have up to two possible passive joints configurations.

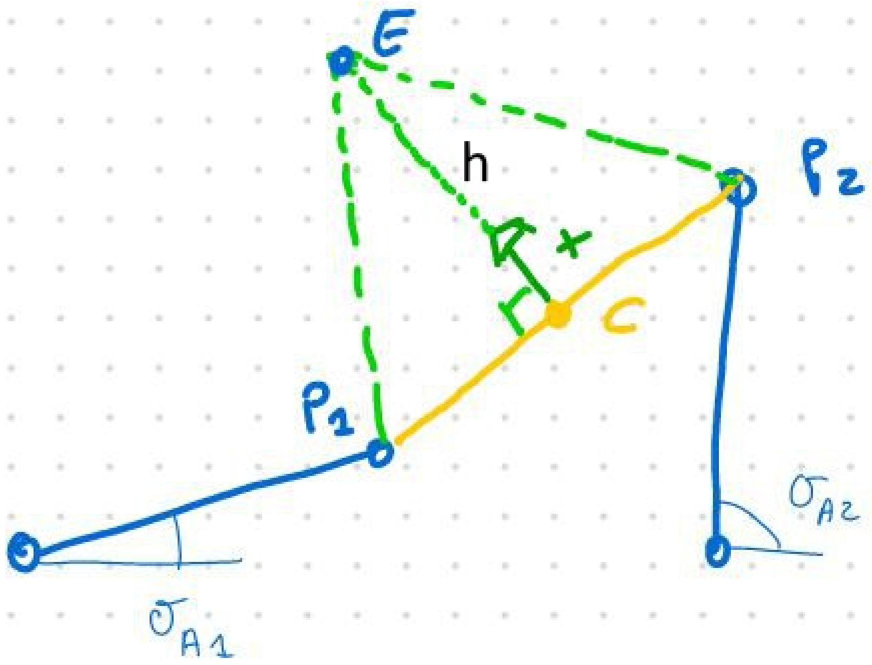




In our case we are only interested in the "green".

To differentiate the two possible solutions we draw a line between the two passive joints (yellow line). The middle point is our point C. The distance between C and E (call it h) is trivially calculated with the Pythagorean theorem.

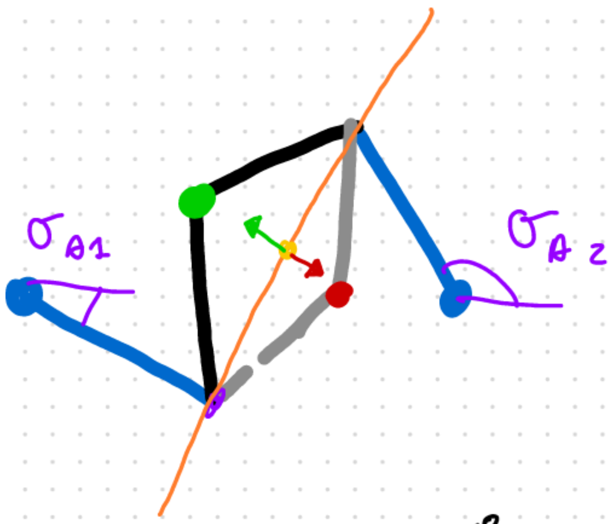
By starting from C we add a vector of size h and direction orthogonal counter-clockwise to $P_2 - P_1$

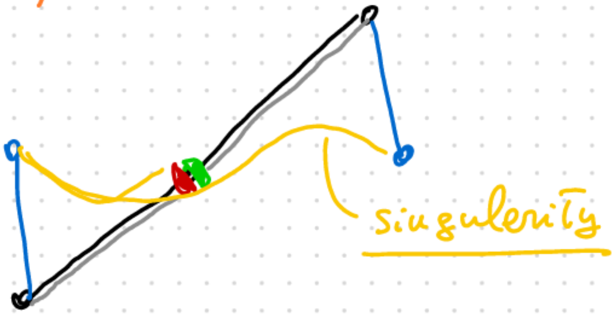


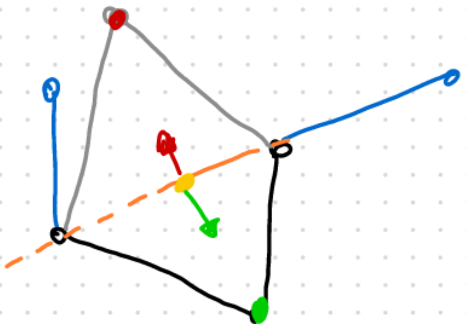
By knowing the end-effector pose we can retrieve the passive joints configuration.

2) Singularity crossings

We demonstrated how to solve the forward kinematics in the first case.



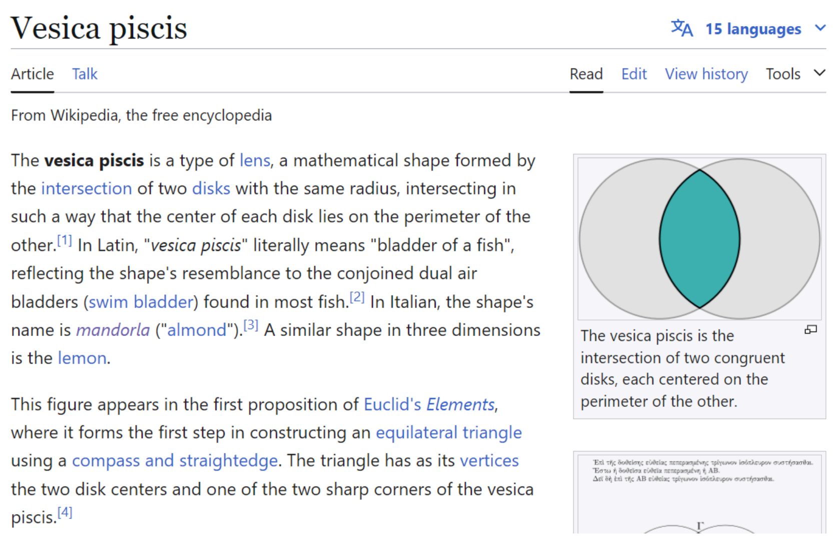




But if we apply this algorithm in the third case we'll end up with the red dot instead of the green dot.

Is it really a problem?

If you think about it, going from the green configuration in the first case to the green configuration in the third case is impossible because you have to cross a singularity. The same happens if you want to go from the green first case to the red third case. (hint: you have to align the two right links to reach the left border of the **vesica piscis** found in the workspace singularity analysis).



Vesica piscis

15 languages

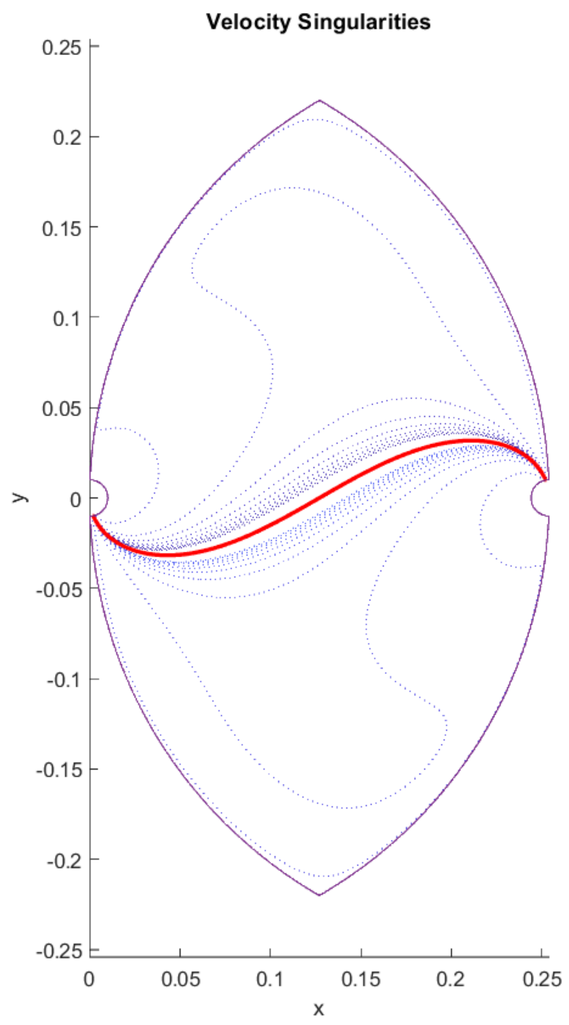
Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

The **vesica piscis** is a type of lens, a mathematical shape formed by the intersection of two disks with the same radius, intersecting in such a way that the center of each disk lies on the perimeter of the other.[1] In Latin, "*vesica piscis*" literally means "bladder of a fish", reflecting the shape's resemblance to the conjoined dual air bladders (swim bladder) found in most fish.[2] In Italian, the shape's name is *mandorla* ("almond").[3] A similar shape in three dimensions is the lemon.

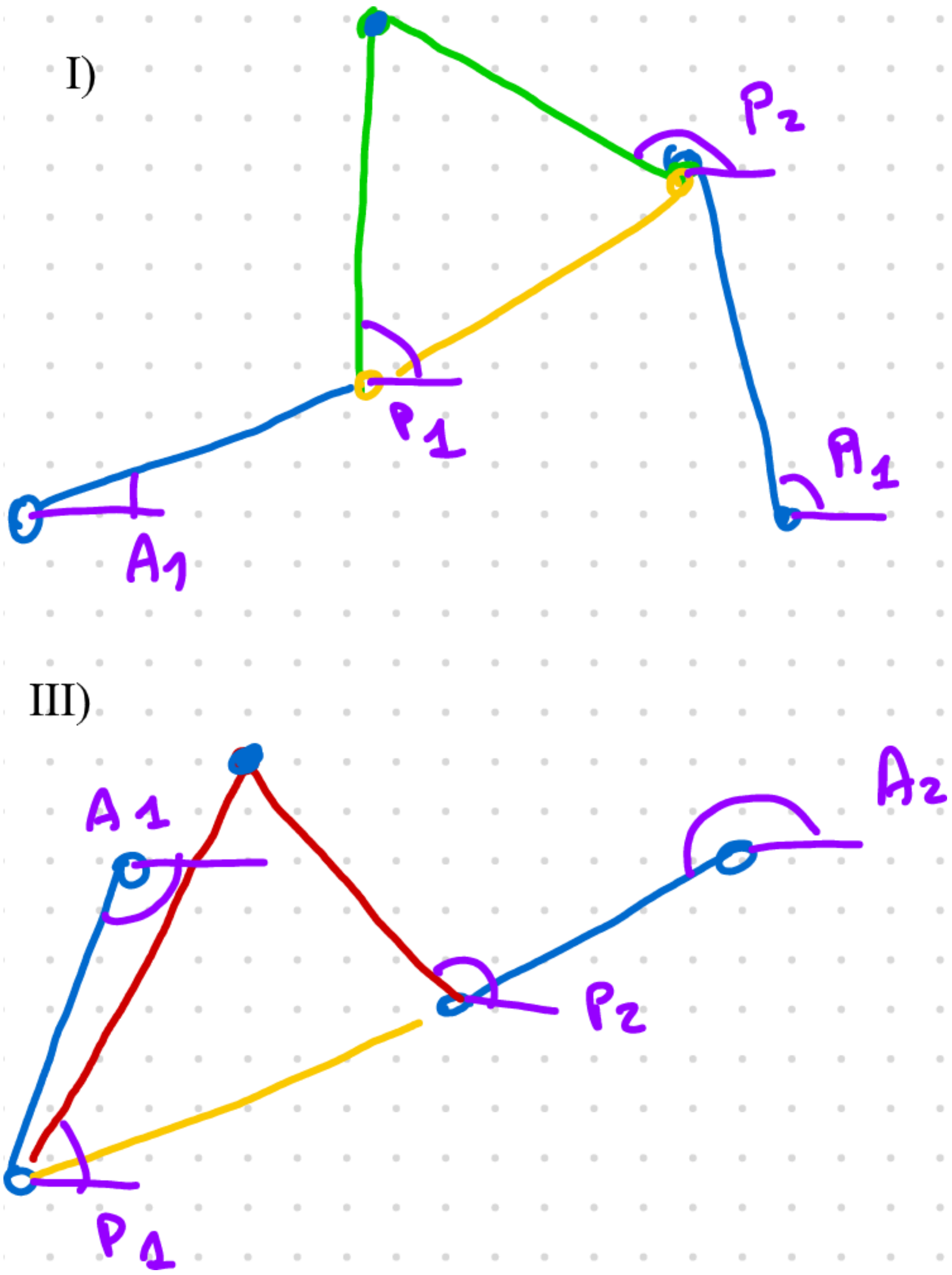
This figure appears in the first proposition of Euclid's *Elements*, where it forms the first step in constructing an equilateral triangle using a compass and straightedge. The triangle has as its vertices the two disk centers and one of the two sharp corners of the vesica piscis.[4]

The vesica piscis is the intersection of two congruent disks, each centered on the perimeter of the other.



Our focus now should be to detect the third case and categorize it as an inadmissible configuration.

2.1) Invariants



In the third case the direct kinematics algorithm finds the red configuration instead of the green configuration.

What's the difference between them?

First case

θ_{p1} is counter-clockwise to θ_{a1}

θ_{p2} is counter-clockwise to θ_{a2}

Second case

θ_{p1} is counter-clockwise to θ_{a1}

θ_{p2} is <u>clockwise</u> to θ_{a2}

2.1.a) Why does it happen?

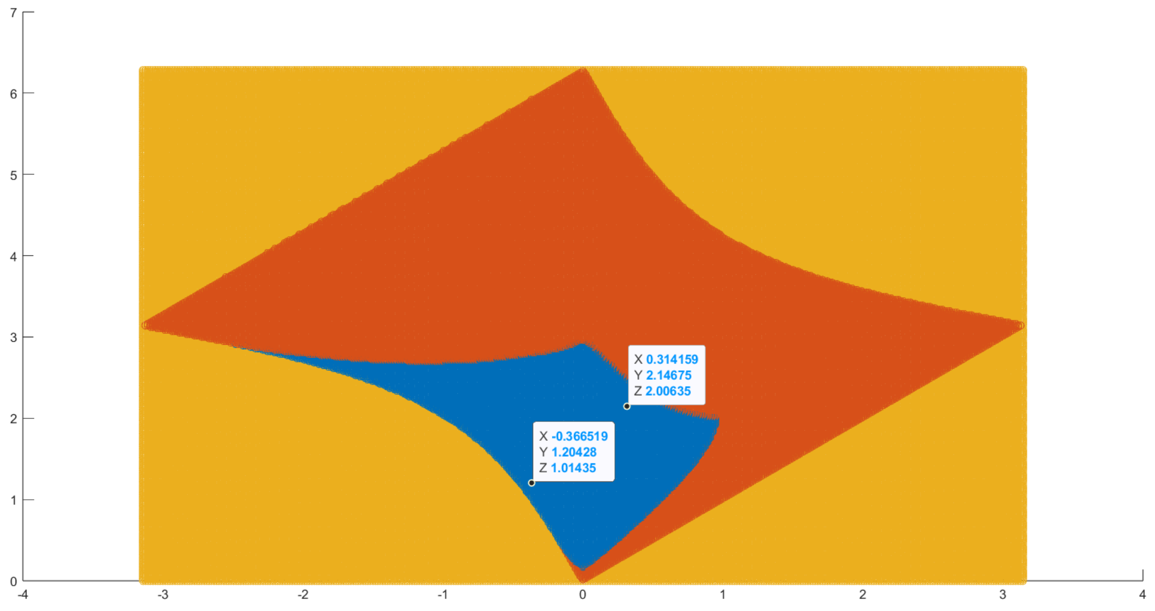
To pass between those two configurations you need to fully extend the links of one side to reach the "vesica piscis" border. In this configuration on one side (either left or right) you have the active and passive joints perfectly aligned. From there you "invert" the the relative rotation and you can reach the red configuration.

You can see the external borders of the workspace like some sort of "portal" that allows you to switch between the two possible direct kinematics.

3) Algorithm

```
for every sampled ($\theta_{a1}, \theta_{a2}$) in joint space
    calculate position of $P_1$ and $P_2$
    if distance($P_1$,$P_2$) < 2L then
        $x, y, \theta_{pi} \leftarrow$ directKinematics($\theta_{a1}, \theta_{a2}$)
        if counter-clockwise_check() is true then
            return Pose = $(x, y, \theta_{pi})$
        else
            return RedConfigurationError
        end
    else
        return DistanceError
    end
end
```

4) Result



X-axis = θ_{a1}, Y-axis = θ_{a2} in radians

Blue = admissible

Red = you have to cross a singularity to reach it.

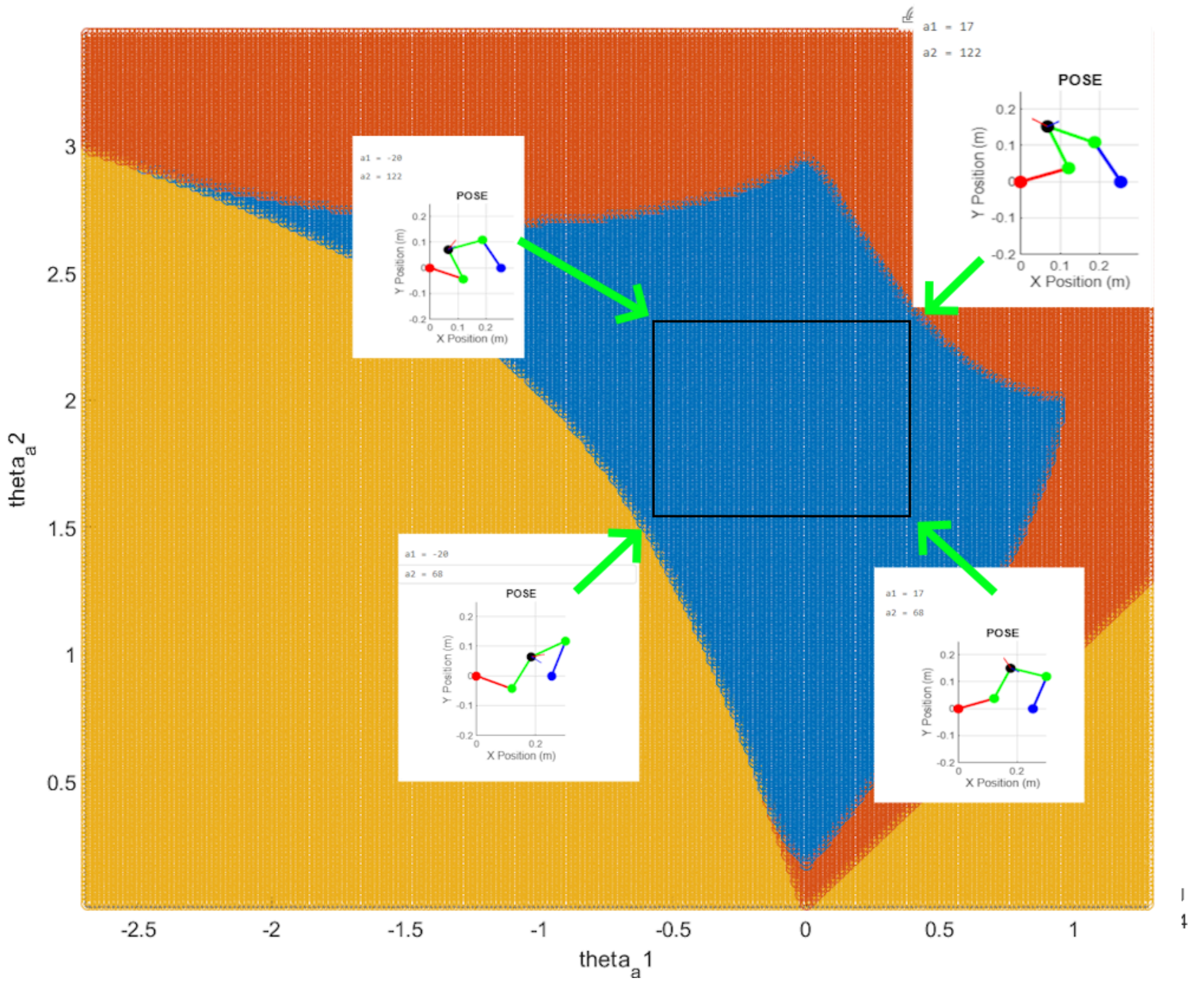
Orange = you have to break the links to reach it.

Zooming in we can pick the joint bounds

I settled on:

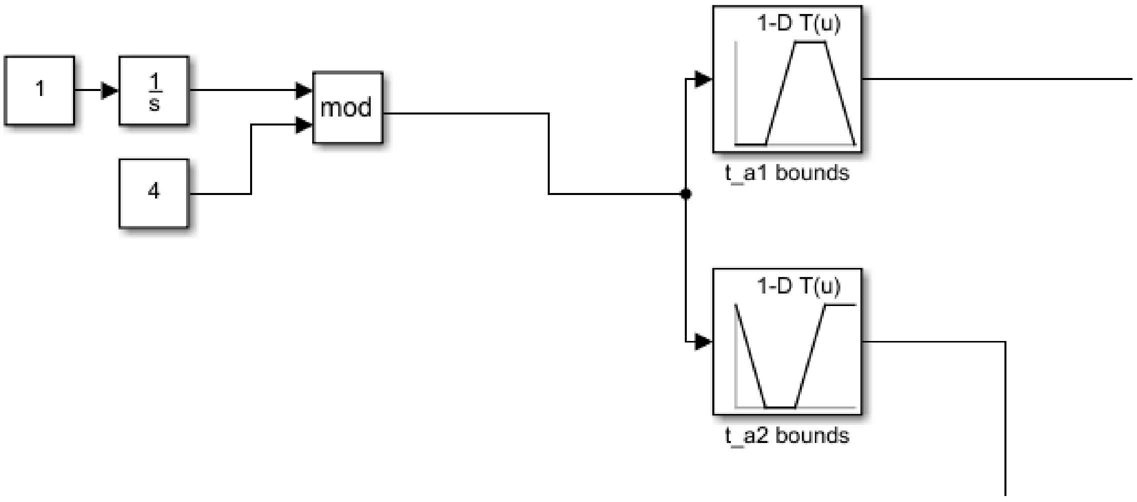
$\theta_{a1} = [-20° \ , \ 17 \ °] = [-0.36\text{rad} \ , \ 0.314\text{rad}]$

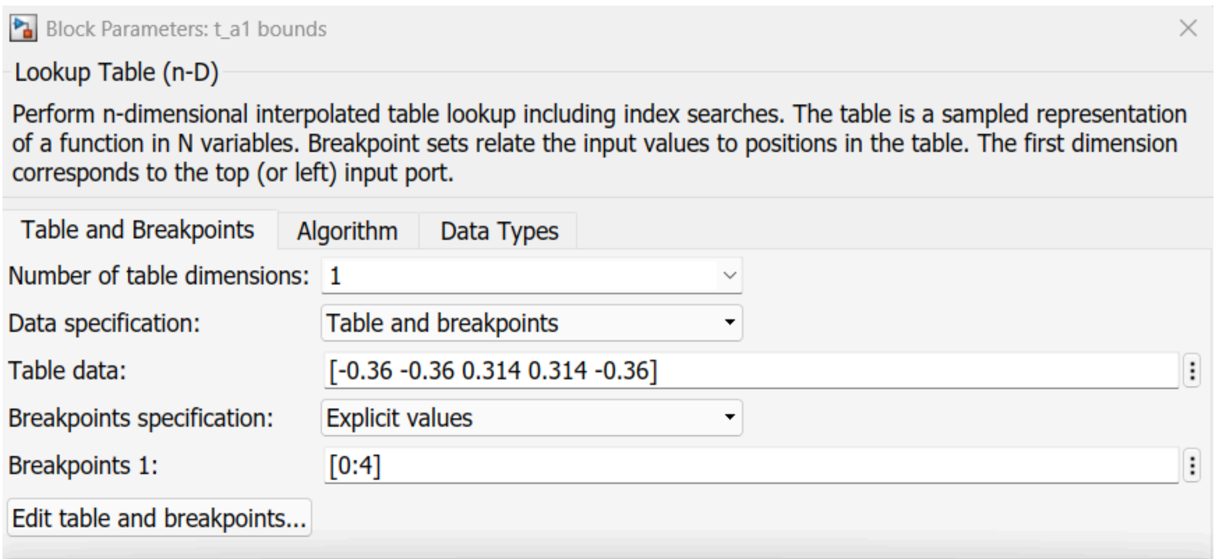
$\theta_{a2} = [\ 68° \ , \ 122 \ °] = [1.204\text{rad} \ , \ 2.14\text{rad}]$



5) Simulink Implementation

Test of trajectory around the safety bounds.

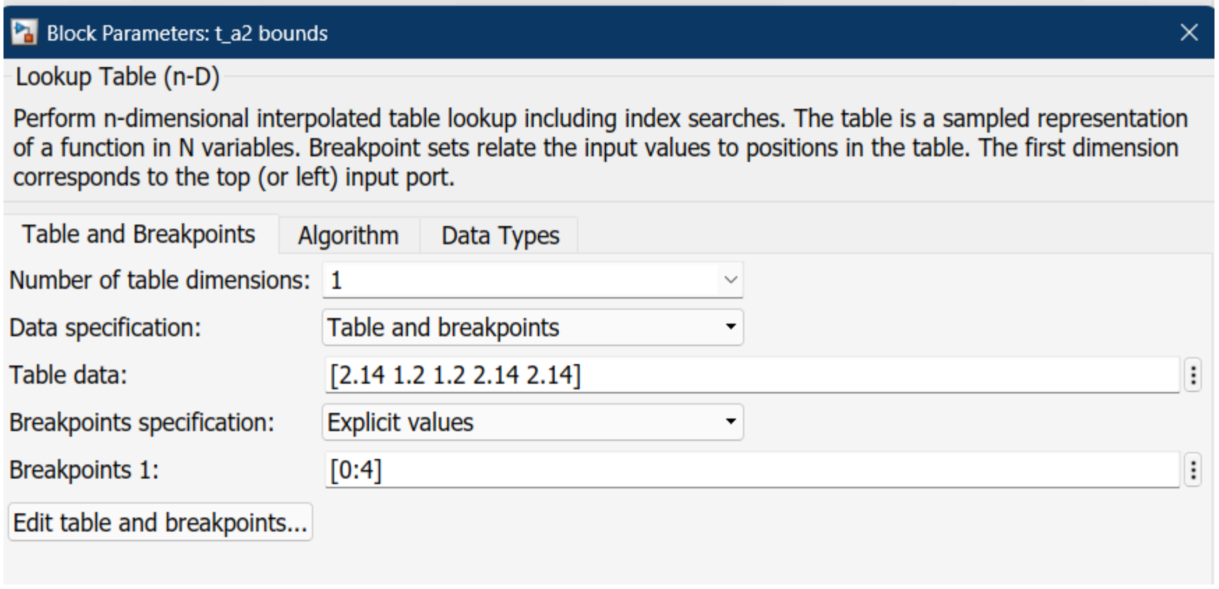



Block Parameters: t_a1 bounds

Lookup Table (n-D)

Perform n-dimensional interpolated table lookup including index searches. The table is a sampled representation of a function in N variables. Breakpoint sets relate the input values to positions in the table. The first dimension corresponds to the top (or left) input port.

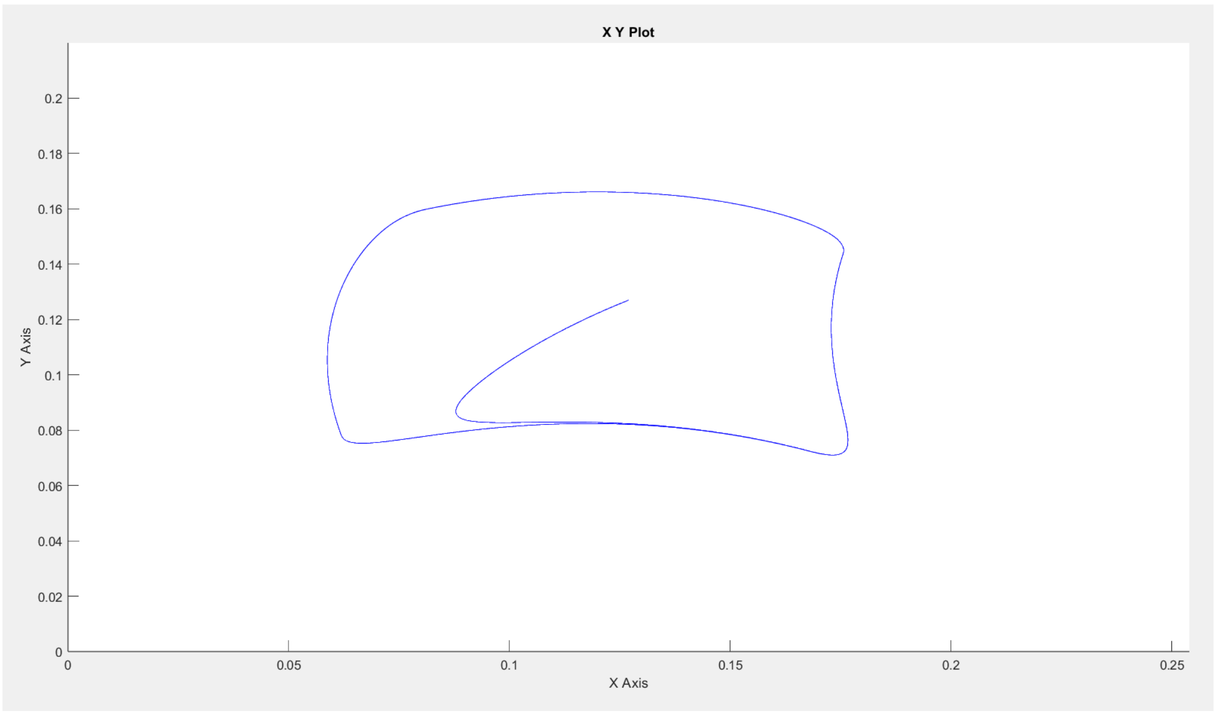
Number of table dimensions:	1
Data specification:	Table and breakpoints
Table data:	[-0.36 -0.36 0.314 0.314 -0.36]
Breakpoints specification:	Explicit values
Breakpoints 1:	[0:4]


Block Parameters: t_a2 bounds

Lookup Table (n-D)

Perform n-dimensional interpolated table lookup including index searches. The table is a sampled representation of a function in N variables. Breakpoint sets relate the input values to positions in the table. The first dimension corresponds to the top (or left) input port.

Number of table dimensions:	1
Data specification:	Table and breakpoints
Table data:	[2.14 1.2 1.2 2.14 2.14]
Breakpoints specification:	Explicit values
Breakpoints 1:	[0:4]



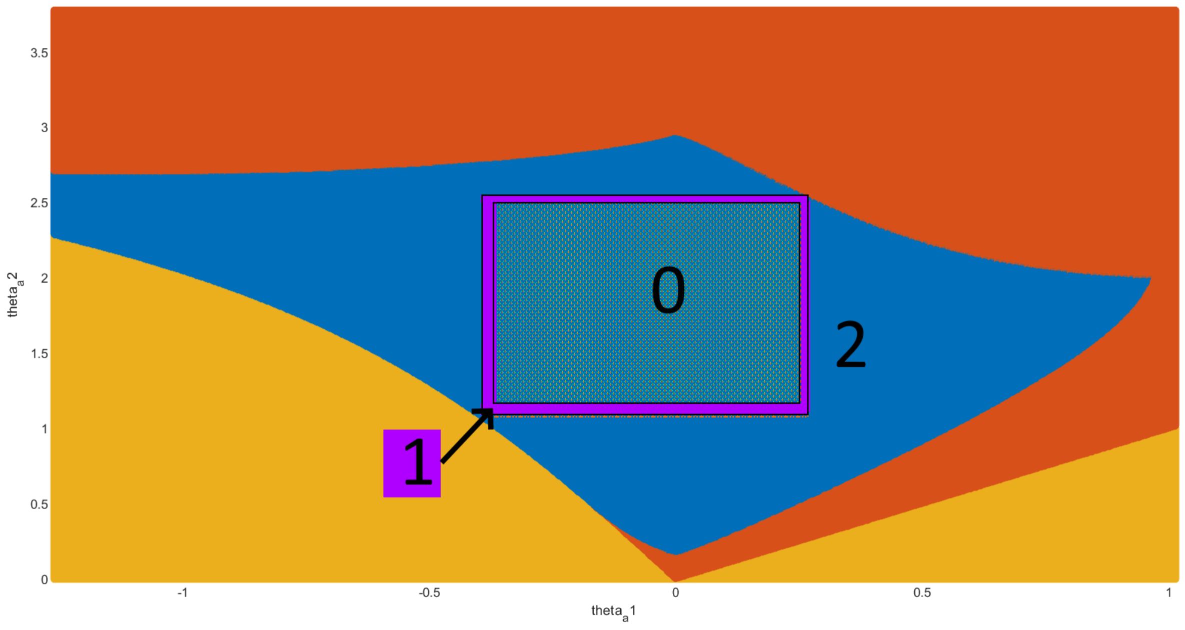
(Plot of trajectory in workspace)

Once we set-up a safety bound we can decide what happens when it's crossed:

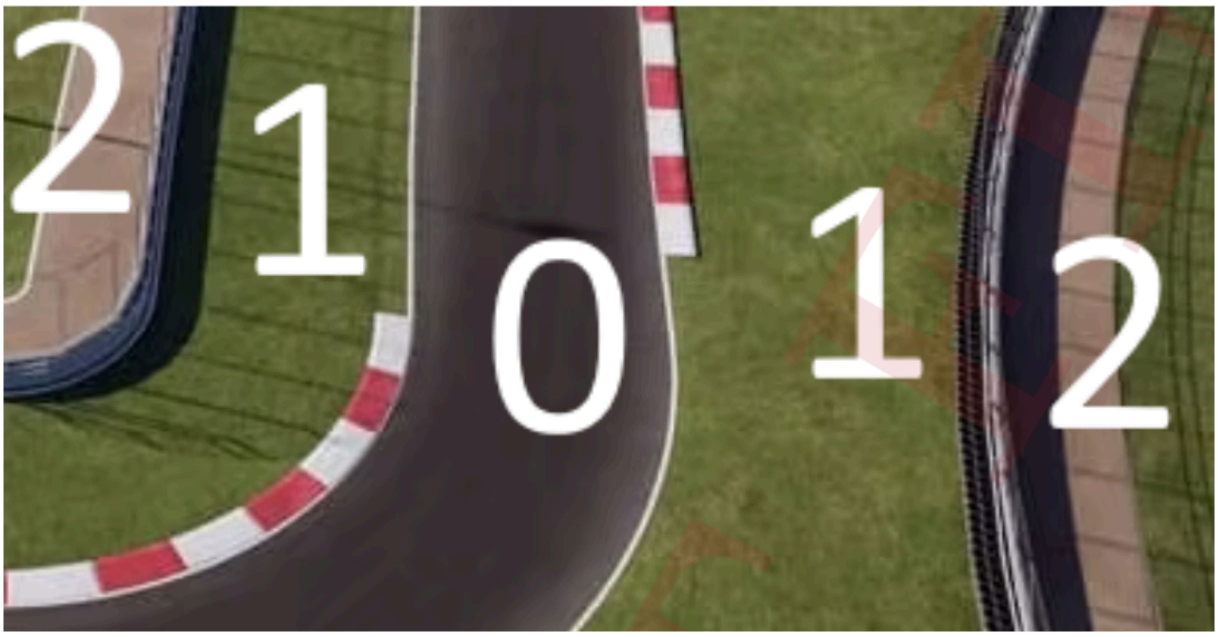
We can virtually saturate the control voltage to slow it down and let us intervene or we can directly stop the machine.

My proposal is to have a "tiered" safety system:

- Inside the safe bounds we use the nominal voltage limits (10V amplitude) (0th zone)
- In a small area around it (10° ?) we reduce the voltage limits to slow it down. (1V or 2V amplitude ?) (1st zone)
- Outside this we just give 0V and have a safe shutdown for 1 (?) second to get some debug traces. (2nd zone)



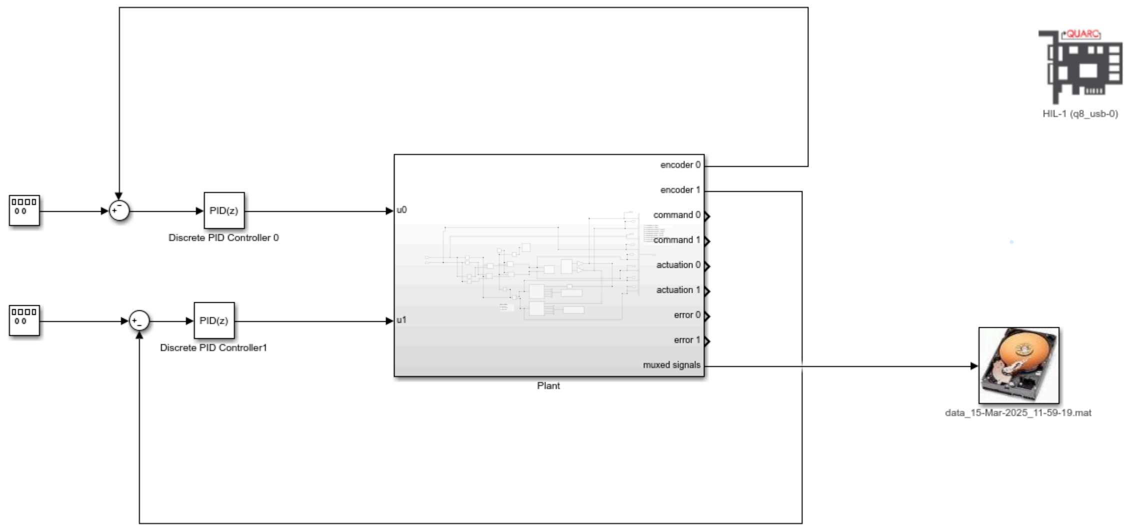
(safety bounds not for scale, we have to decide)



Imagine this as a racetrack where in the 0th zone you can drive freely, in the 1st zone you are slowed down for safety but you can still make it back, and the 2nd zone is a no-go.

5.1) Low level safety systems.

Going forward we are going to complicate the Simulink scheme a lot so my proposal is to implement all the safety checks in one block that we are going to reuse every time we want to interface with the target plant.

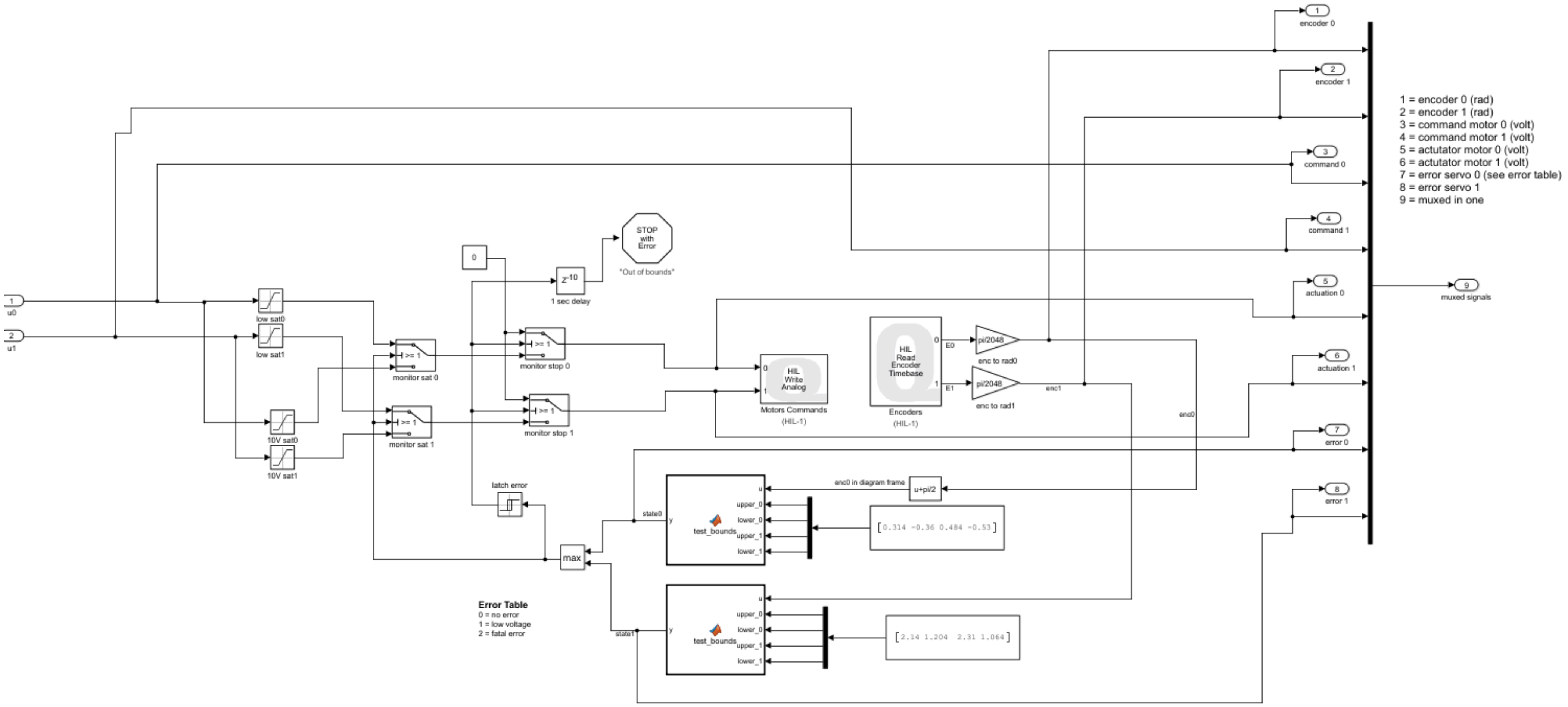


Example of a generic control scheme: the intrinsic safety measures don't clutter the high-level design.

All the interesting signals of the plant are muxed to be able to connect to the logger with a single cable. We can always add other signals from the controler with a muxer afterwards.

As a quick reference here is the index of all the signals.

muxer id	.mat row	**Name**	Unit	Description
1	2	ENC0 / Y0	rad	angular distance from rest position
2	3	ENC1 / Y1	rad	angular distance from rest position
3	4	U0	V	voltage requested by control law (copy of input signal)
4	5	U1	V	voltage requested by control law (copy of input signal)
5	6	U0_SAT	V	actual voltage provided to actuator after saturation (for anti-windup)
6	7	U1_SAT	V	actual voltage provided to actuator after saturation (for anti-windup)
7	8	EC0	/	Error code for joint 0
8	9	EC1	/	Error code for joint 1



Schematics of the safety system.

Copy of the code in the function block:

```
function y = test_bounds(u, upper_0, lower_0, upper_1, lower_1)
%  0 = everything ok , 1 = low voltage ,  2 = shutting down
if u<upper_0 && u>lower_0
    y = 0;
elseif u<upper_1 && u>lower_1
    y = 1;
else
    y = 2;
end
```