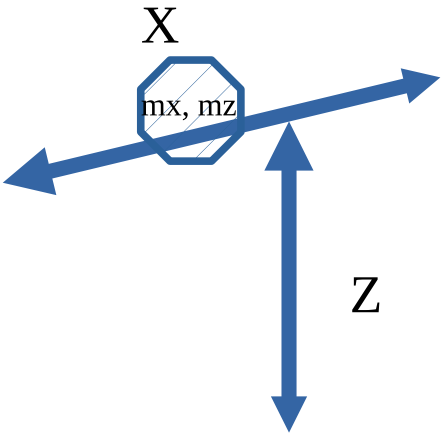
ARP 2021 - FIRST ASSIGNMENT

The code to design, develop, test and deploy is an interactive simulator of hoist with 2 d.o.f, in which two different consoles allow the user to activate the hoist.

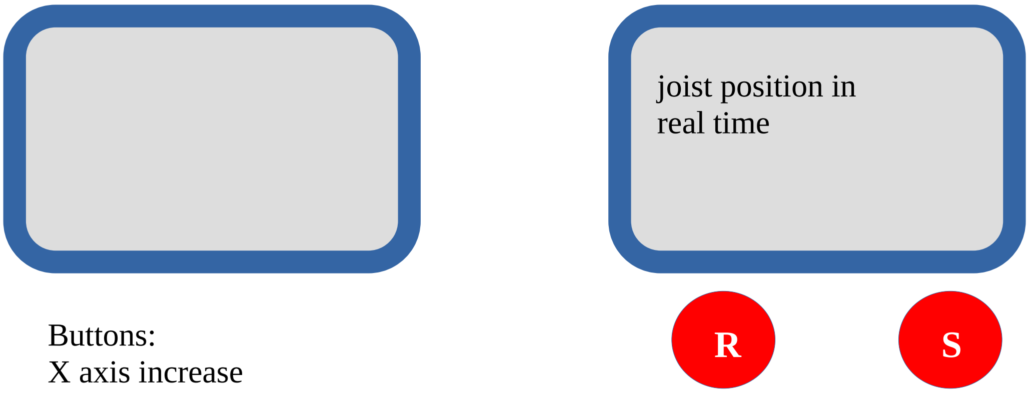


In the octagonal box there are two motors mx and mz, which displace the hoist along the two respective axes. Motions along axes have their bounds, say ***0 - max_x*** and ***0 - max_z.***

From the user side there are two consoles (shell windows) and keys with different aims, that simulate a real system.

command console

inspection console



Buttons:
X axis increase
X axis decrease
X axis stop
Z axis increase
Z axis decrease
Z axis stop

commands specify a constant speed motion

S: emergency stop, the joist stops immediately until a command from the first console arrives

R: reset, the joint stops, both axes go to a zero position and wait for commands.

The simulator requires (at least) the following 5 processes:

command console, reading the 6 commands, for example using keyboard keys or inputs (free choice)

inspection console, receiving from motors the hoist positions while it moves, and reporting on the screen somehow (free choice); the inspection console manages the S ad R buttons as well (simulated in a free way using the keyboard)

motor x, simulating the motion along x axis, receiving command and sending back the real time position (see below)

motor z, similar to motor x (see below)

watchdog: it checks the previous 4 processes periodically, and sends a reset (like the R button) in case all processes *did nothing* (no computation, no motion, no input/output) for a certain time, say, 60 seconds.

motor x and y specs:

The motor process simulates the constant speed motion, including a simple error in position measurement. It receives a command (move, stop, go to zero position), acts, sends back a position estimate, stops when the end displacement is reached. here is a rough sketch of the motor loop:

```
last command = stop
loop [
        update/receive command
        if ( command == reset ) <reset procedure>
        if ( command != stop || ~(position == displacement_end) ) [
                position += step
                estimated position = position + error
                <send estimate position>
        ]
        sleep (dt)
]
```

Remarks:
error can be simulated using some random generator and a bias (use reasonable figures)
dt simulates the real motion; for example, 1 sec for a speed of 0.1 meter/s
displacement end can be small for testing, e.g. 1 meter.

The *stop* button is an emergency button, therefore it does not send a stop command: it must use *signal* to interrupt motion.

Several syscalls can be used. Try to figure a real implementation on a real hoist like the one in the image, and choose the proper ones.

A shell script *hoist.sh* for *compilation* and *execution* must be written, so that the project is deployed with sources and the hoist.sh files.