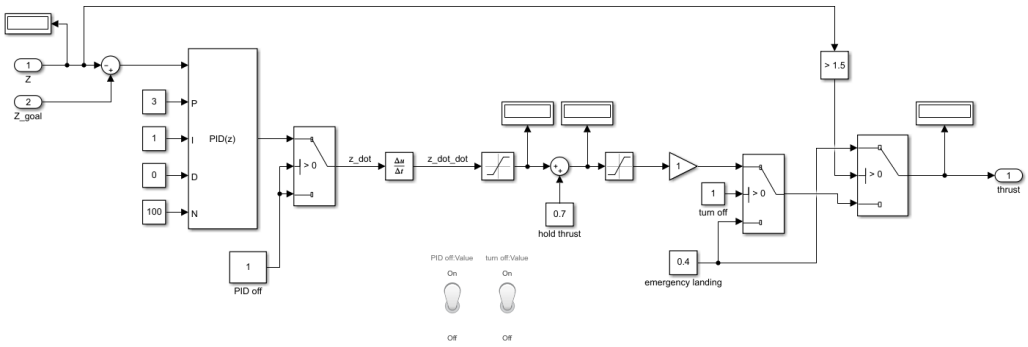
Lab 3: Aerial Robotics

Exercises

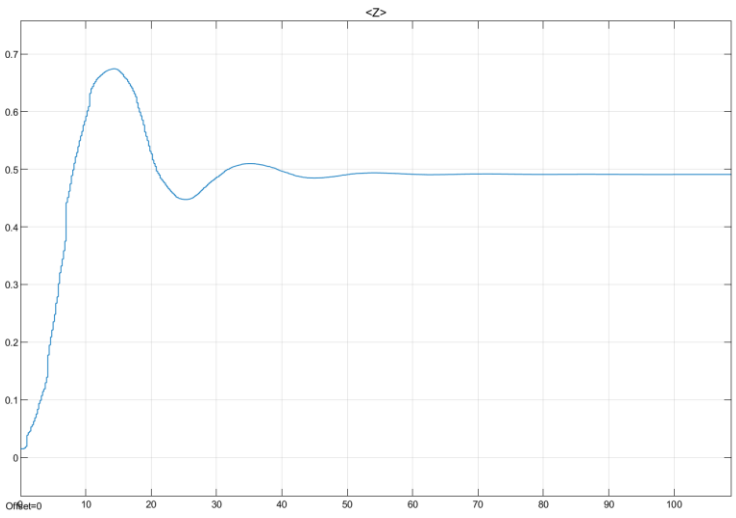
Task 1 2
Task 2 3
Task 3 5
Square constant yaw 5
Square yaw aiming next target 6
Circular motion aiming center 7
Tilted circular motion fixed yaw 8

Task 1

A PI controller is used for the hovering of the Crazy Flie. A proportional controller alone has found not to be stable enough and does not achieve the goal with the same accuracy. A saturation filter is also placed for stability. This also helps reduce overshooting. It would have been ideal to use a fine-tuned PID for this task since the derivative would solve the overshooting problem and after some tuning the system could move much faster and in a stable manner, however exploding gradient problems were found (overflowing nan derivatives) and to avoid waste of time it was decided to just set the derivative component of the controller to zero. The controller looks as shown in the picture below.



An estimated holding thrust was found kind of empirically since the value obtained by computing the robot's weight force did not work. This value is not perfectly accurate but that is not a problem since the controller takes care of the rest. Also, the controller acts on speed, since actuation on thrust was found to be very slow and unstable. This requires the computation of the derivative of a signal: without the Integral part of the controller, the system could get stuck here. Because the reset button on the simulator was not working, slow-landing and power-off switches were also implemented. The following plot shows the time-series of the robot's height during take-off.

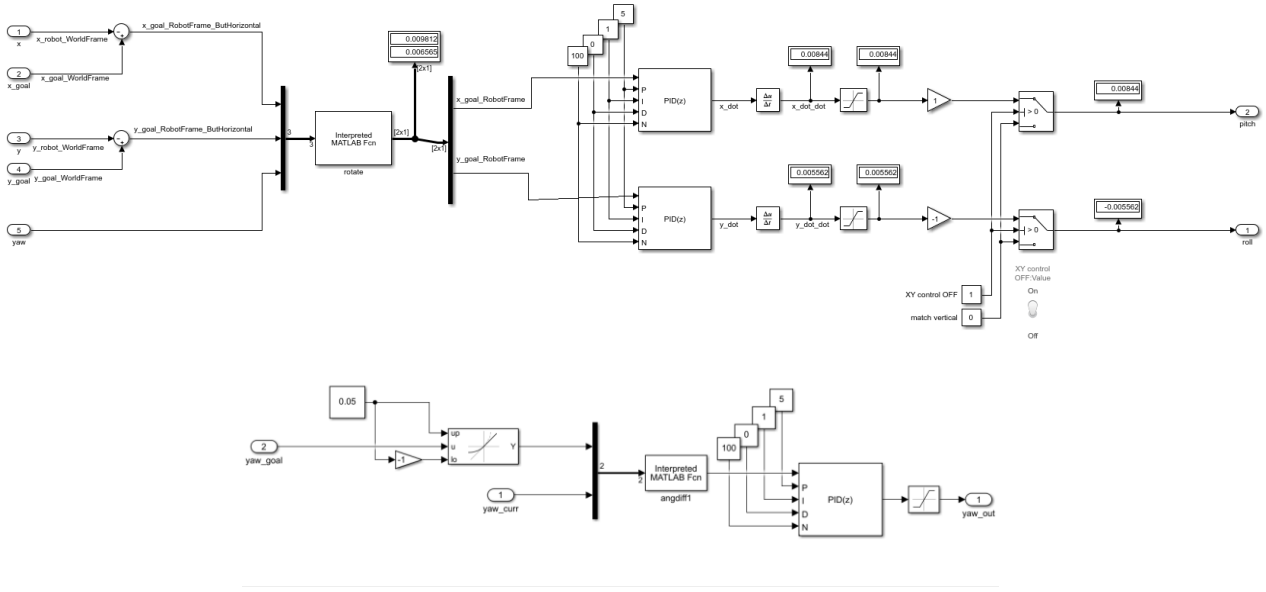


Task 2

Moving from pose A to B was achieved by applying coordinate frame transformations. If the robot's yaw is square to the axes, rolling and pitching will advance/step-back in the Y and X directions, respectively. This was the first approach for moving the robot. From there, with some modifications on the coordinate frames, the robot was programmed to run in any direction while looking in any direction. This is achieved by the following procedure:

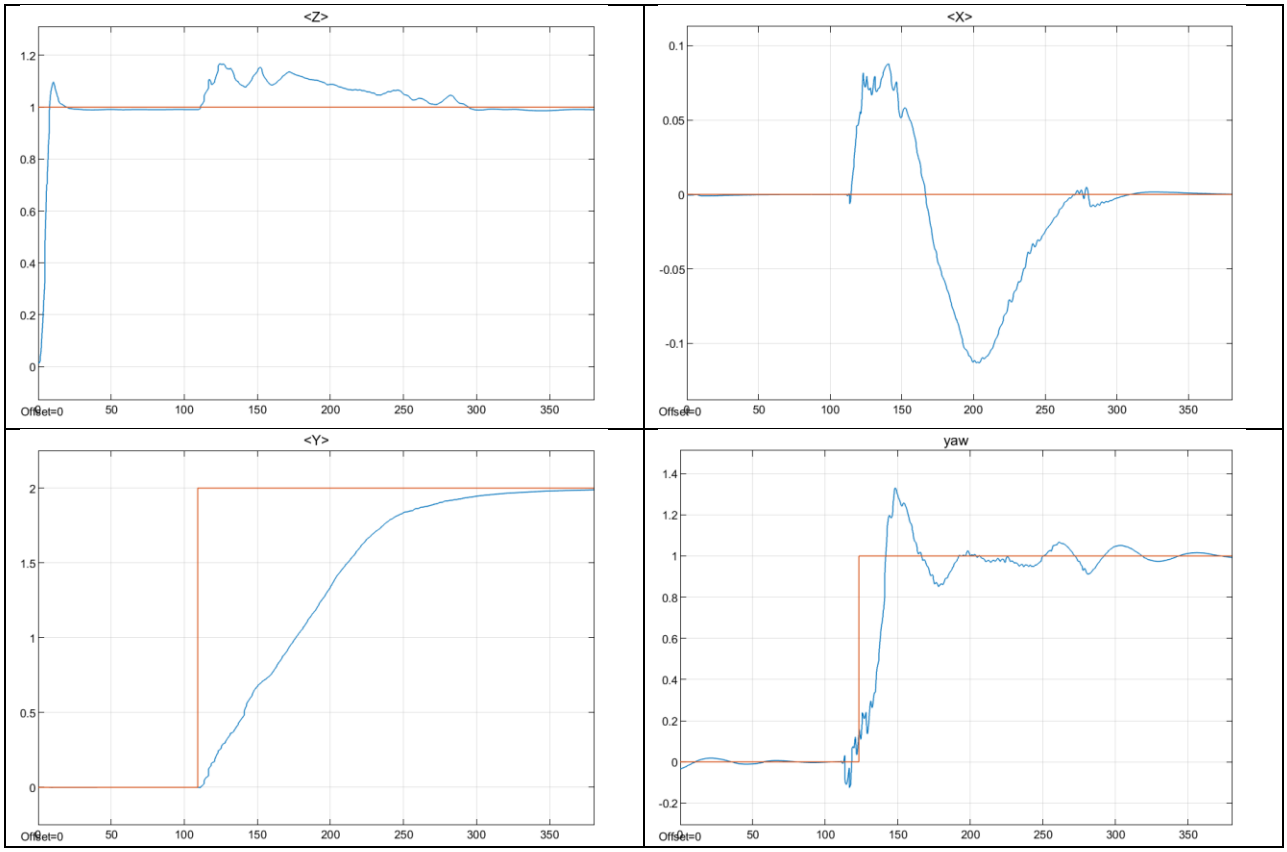
- Get the X and Y coordinates of the goal with respect to the robot, parallel to the world's coordinate frame.
- Rotate X, Y to X', Y' so that X' & Y' are parallel to the robot coordinate frame axes.
- Use rolling & pitching to move in the direction of the robot axes while applying a control loop to reduce X' and Y' to zero.

Three independent control systems were used for this task: one for the height, one for the X-Y position, and a third one for the yaw. Below, pictures show these last two, since the first is equal to the one shown in Task 1.



When the yaw reference is suddenly modified and the robot rotates, there is an extra thrust step for a little moment that elevates the robot. The height control helps damping this effect but it is not fast enough to make the position error negligible. This is why a change rate limiter was set on the yaw control. A similar effect also happens when the robot is tilted. An interaction signal was placed and slightly tuned for reducing the thrust while the robot is yawing. The same concept was tried for those times when roll and pitch are different than zero but the resulting system was not very stable.

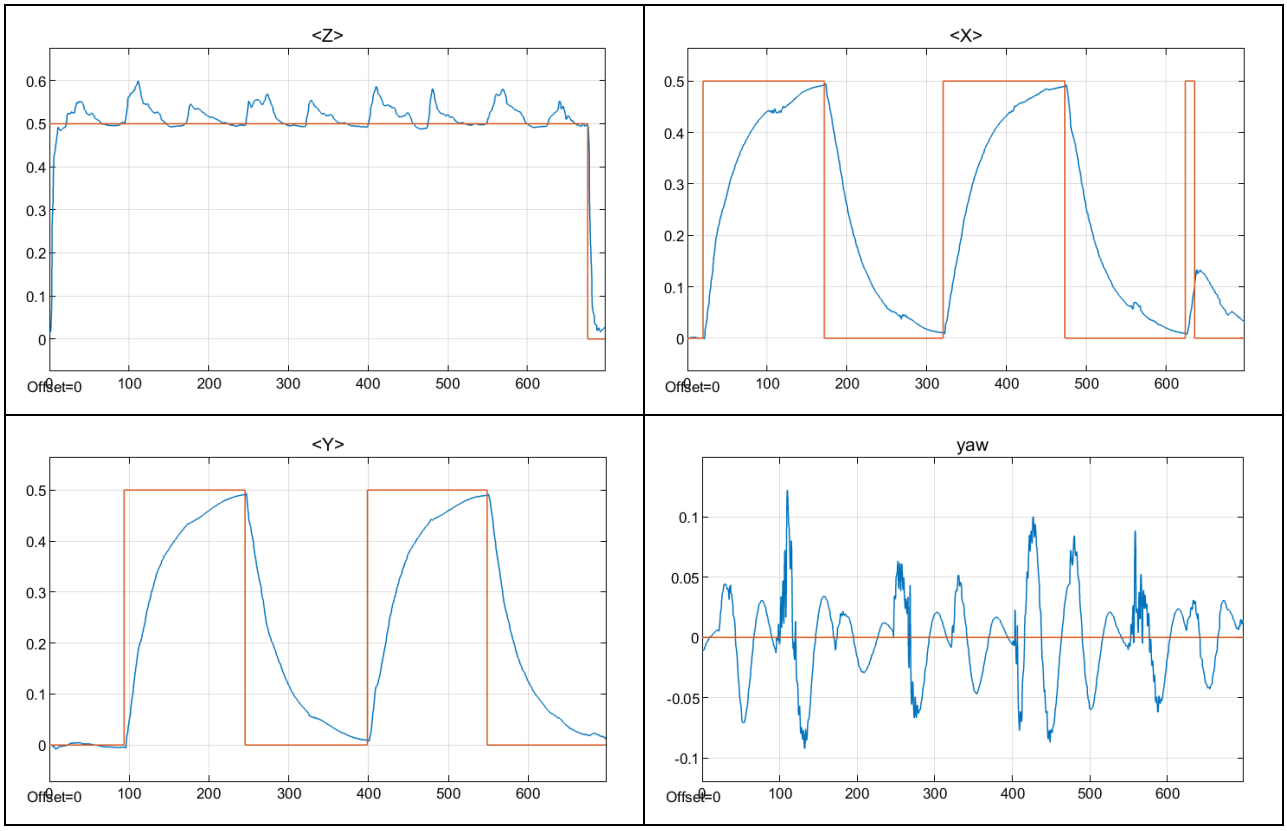
The following table shows the robot's pose for the movement (0m, 0m, 1m, 0rad) to (0m, 2m, 1m, 1rad). Notice how, after starting to move, the height rises from its reference. Despite moving only in the Y direction, there are also small perturbations on the X position of the robot. Also, the yaw is not progressively interpolated, but since the robot can move in any direction with any yaw, goes straight to its goal as fast as possible. In this implementation, the height control was retuned, so it shows a better performance. Also, when moving slowly, the accuracy of the X, Y positioning is pretty decent, despite the robot moving quite shakily in the journey there.


<Z>
<X>
<Y>
yaw
Offset=0

Task 3 – Square constant yaw

From the model presented for task 2, a simple modification on the previous simulink model consisting of progressively updating goal points was used for following the requested trajectories. These are first computed in external scripts and loaded into variables with the provided code.

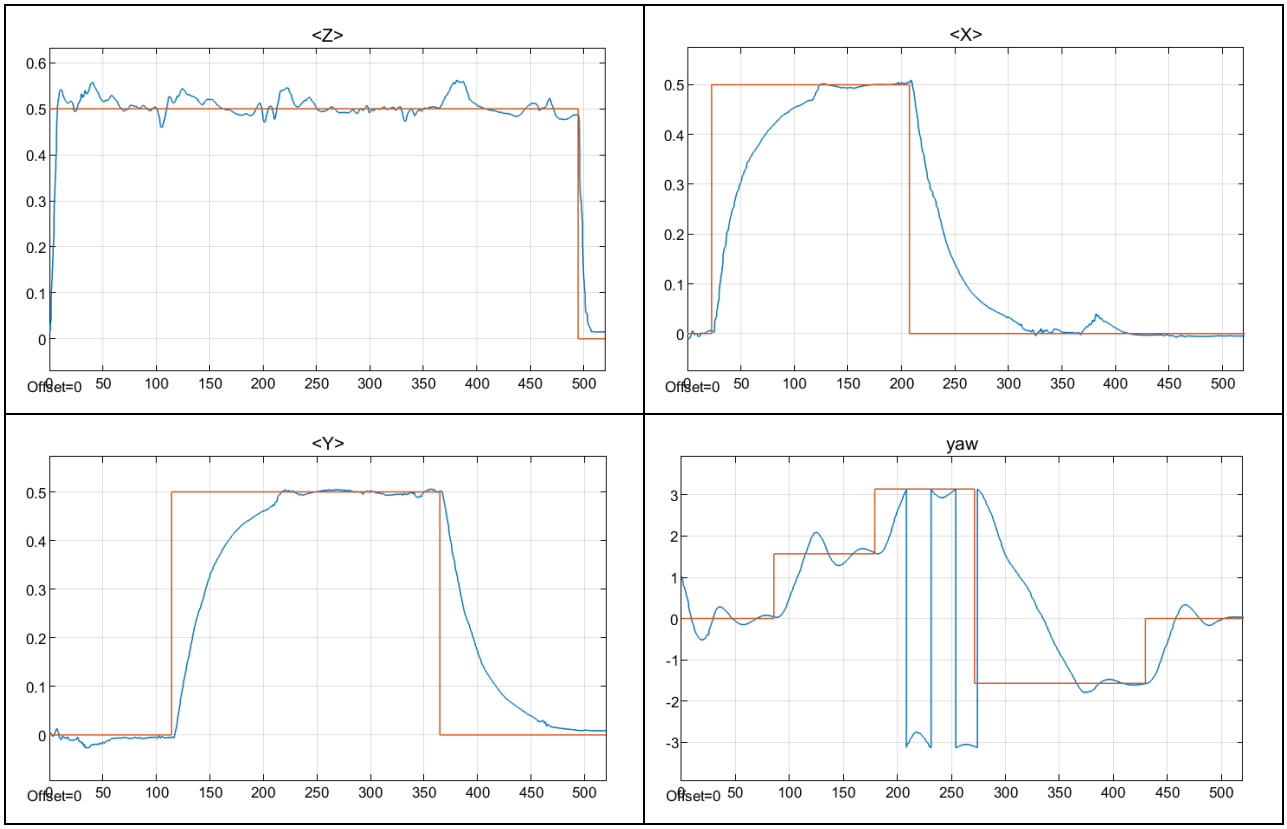
Plots for the square trajectory holding a constant yaw are shown below. Note how every time there is a big change in X or Y the thrust overshoots and a little Z increment takes place. The plots are for completing the course twice. In the end the robot was landed by manual control, so please ignore the incongruences at the end of the plots (especially that of X).




```
%% horizontal square fixed dir

via = [...
    0       0       0.5     0;...
    0.5     0       0.5     0;...
    0.5     0.5     0.5     0;...
    0       0.5     0.5     0;...
    ];

xx = via(:,1)
yy = via(:,2)
zz = via(:,3)
yawyaw = via(:,4)
```

Task 3 – Square yaw aiming next target

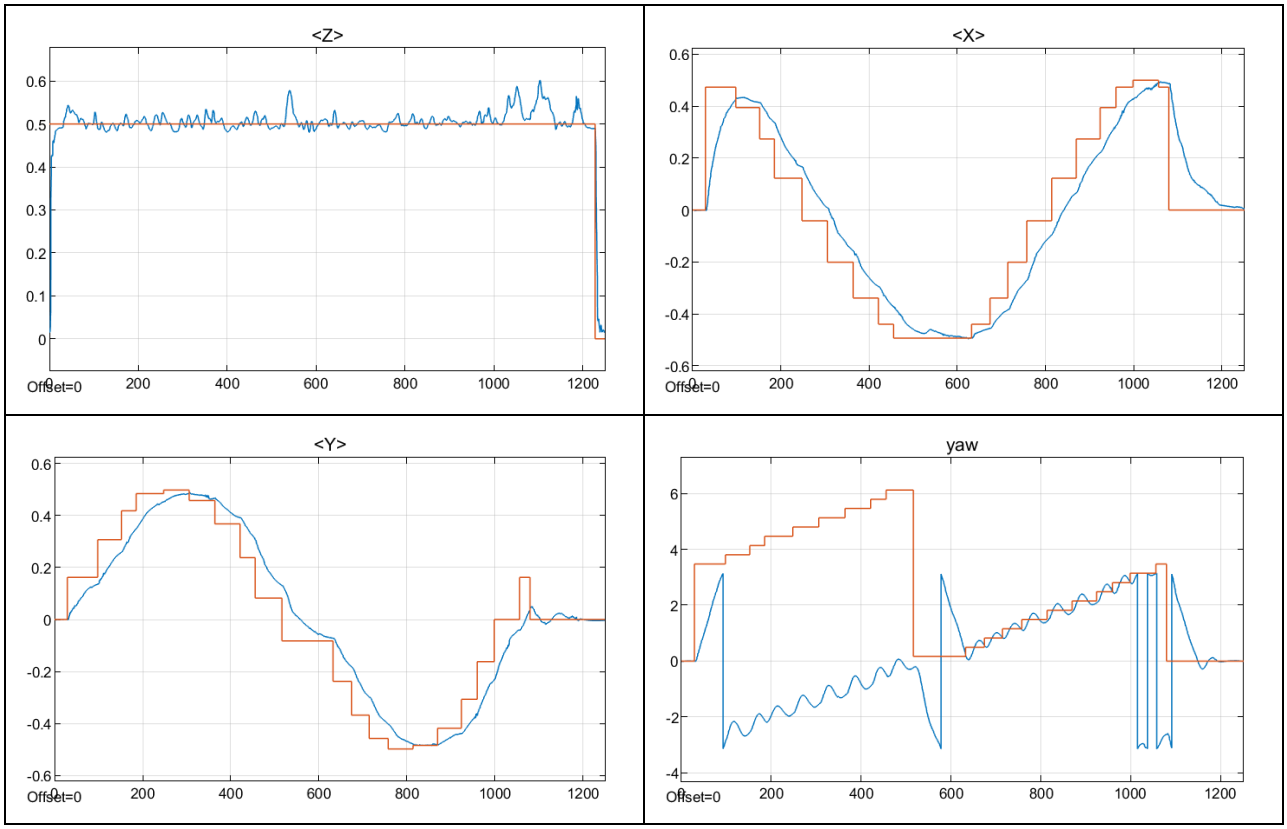
Next, plots for the square trajectory while the yaw aims to the next target are shown. In this simulation, when the yaw overshoots the 2pi range, it came back doing the long turn instead of the short one.




```
%% horizontal square looking next waypoint

% Strat: (loop) arrive, stop, adjust orientation, start...
via = [...
    0       0       0.5     0;...
    0.5     0       0.5     0;...
    0.5     0       0.5     pi/2;...
    0.5     0.5     0.5     pi/2;...
    0.5     0.5     0.5     pi;...
    0       0.5     0.5     pi;...
    0       0.5     0.5     -pi/2;...
    0       0       0.5     -pi/2;...
    ];

xx = via(:,1)
yy = via(:,2)
zz = via(:,3)
yawyaw = via(:,4)
```

Task 3 – Circular motion aiming center

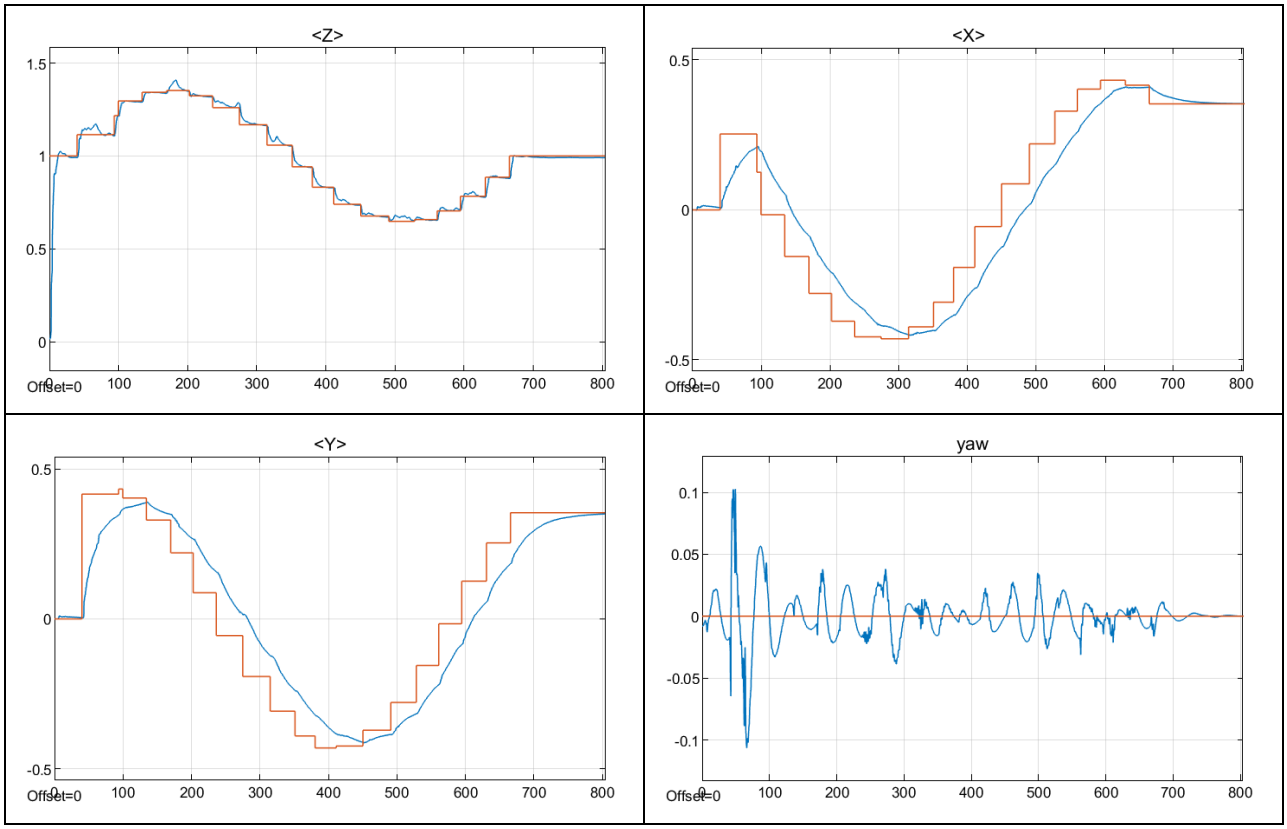
Again, after overshooting the 2pi range, the robot did the whole turn instead of turning the smallest path.




```
%% circular reference 1m diameter yaw pointing to center

via = hor_circular(1, 0.5, 20)

xx = via(:,1)
yy = via(:,2)
zz = via(:,3)
yawyaw = pi + atan2(yy, xx)      % by drafting in a paper

function coords = hor_circular(diameter, height, npoints)
 %%% Draws a horizontal 3d circle centered at the origin
 theta = linspace(0, 2*pi, npoints)
 coords = []
 coords(:,1) = diameter/2 * cos(theta)
 coords(:,2) = diameter/2 * sin(theta)
 coords(:,3) = height .* ones(length(theta),1)
end
```

Task 3 – Tilted circular motion fixed yaw

This trajectory was computed by drawing a circle and turning it about the Z and X axis. Then it was implemented the same way as the others, as a series of waypoints.




```
%% circular reference 1m diameter tilted

% draw the trajectory, tilt it and then get only the points
via = hor_circular(1, 0, 20)
via = SE3.Rz(deg2rad(45)) * SE3.Rx(deg2rad(45)) * SE3(via)
via = via.transl

% extract each coordinate and fix orientation
xx = via(:,1)
yy = via(:,2)
zz = via(:,3) + 1
yawyaw =  0 .* via(:,1)
% plot3(xx, yy, zz)             % check if shape is correct

function coords = hor_circular(diameter, height, npoints)
 %%% Draws a horizontal 3d circle centered at the origin
 theta = linspace(0, 2*pi, npoints)
 coords = []
 coords(:,1) = diameter/2 * cos(theta)
 coords(:,2) = diameter/2 * sin(theta)
 coords(:,3) = height .* ones(length(theta),1)
end
```