MODELING OF UNDERWATER ROBOTICS

CONTROL BONUS OF THE SPARUS AUV - INDIVIDUAL WORK

Done by:
ELKHOLY Hassan

MIR Master program

Université de Toulon

December 2021

Contents

1 Introduction

After having a model for the Sparus AUV, the goal now is to design simple commands law to conduct some tasks. Given the time constraint while working in this project, I only did a pid controller. The task here is as the following
1- Initialise AUV position to be [0,0,0,0,0,0] or in other words, starts at the water surface
2- Perform pure heave motion and reach a depth of 5 meters
3- Move forward with constant speed of 1m/s without changing the orientation "roll, pitch, and yaw equal zero"
4- Stop moving when the Sparas reaches a position $[20m, 0m, 5m]^T$

2 PID design

Since we have 3 motors, we will design 3 PID controllers for each of them. The gains of the PID controller in the vertical motor will depend on the Δ $heave$ or in other words, the depth error. While the the PID controllers of the other two motors will depend on ΔV_{surge} or in other words the error in surge velocity. Eq. 1 to 3 illustrates the general formula for calculating the PID errors where Δt is the same as simulink which was 0.01.

$$Error_P = \Delta error \tag{1}$$

$$Error_I = \int \Delta error_{i-1} + \Delta error_i \times \Delta t \tag{2}$$

$$Error_D = \frac{\Delta \text{error}}{\Delta \text{t}} \tag{3}$$

Since the command block receives position, velocity, acceleration feedback as shown in Figure 1, we have all we need to design the PID controllers. Eq.4 to 6 show the formulas for calculating the commanded motor values. Table 2 illustrates the chosen PID gains. After having the commanded values for the motor, it is necessary to saturate them before sending them to the motor. The saturation limits were -100 % and 100 %.

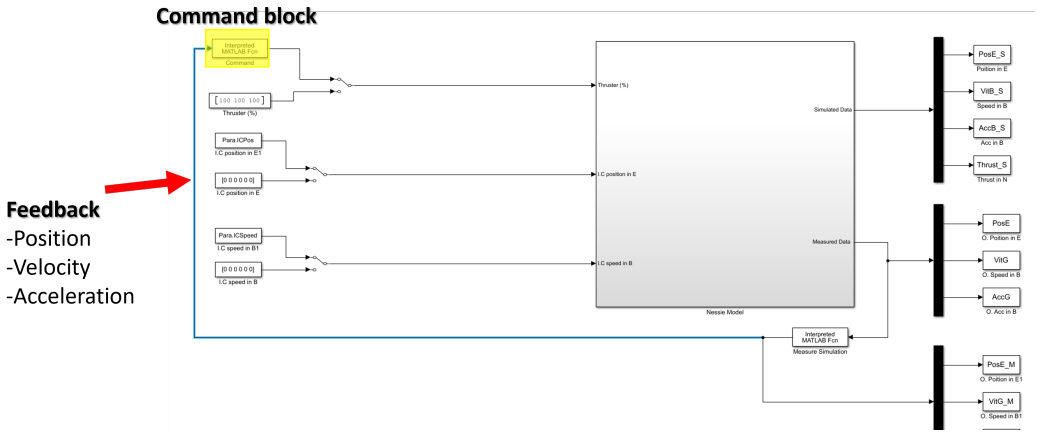

Figure 1: Simulink model

$$Output_{middle\ thruster} = K_p^m \times \Delta error_z + K_I^m \times \int \Delta error_{z\,i-1} + \Delta error_{z\,i} \times \Delta t + K_D^m \times \frac{\Delta \text{error}_\text{z}}{\Delta \text{t}} \tag{4}$$

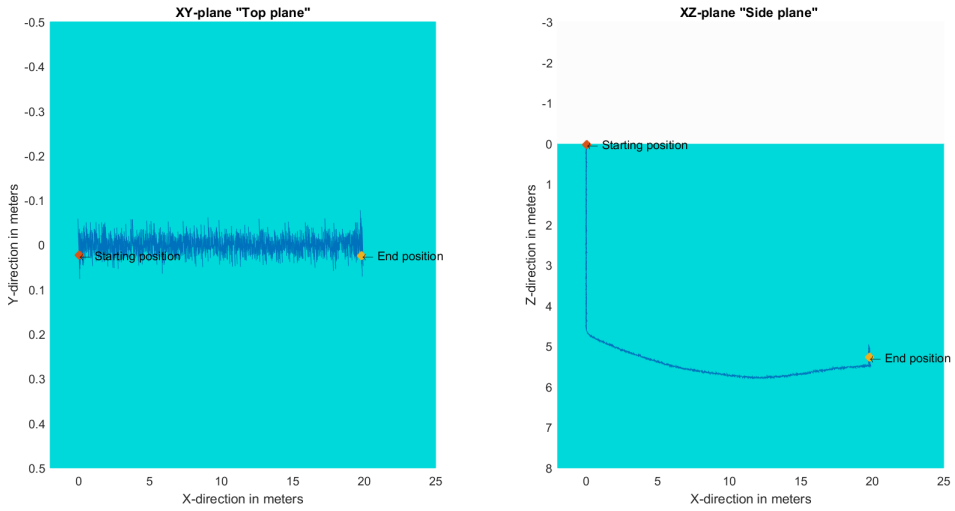

Figure 2: AUV Position

$$Output_{right\ thruster} = K_p^r \times \Delta error_{\dot{u}} + K_I^r \times \int \Delta error_{\dot{u}\ i-1} + \Delta error_{\dot{u}\ i} \times \Delta t + K_D^r \times \frac{\Delta \text{error}_{\dot{u}}}{\Delta \text{t}} \quad (5)$$

$$Output_{left\ thruster} = K_p^l \times \Delta error_{\dot{u}} + K_I^l \times \int \Delta error_{\dot{u}\ i-1} + \Delta error_{\dot{u}\ i} \times \Delta t + K_D^l \times \frac{\Delta \text{error}_{\dot{u}}}{\Delta \text{t}} \quad (6)$$

Motor	Kp	Ki	Kd
Middle	2000	25	2500
Right	1000	0.1	1
Left	1000	0.1	1

Table 1: PID gains

3 Results

Figure 2 and 3 shows the positon of the AUV during the experiment. Figure 4 show the angular position roll,pitch and yaw where it is clear that there were no noticable change. Figures 5 and 6 illustrates the linear and angular velocity profiles.Figures 7 and 8 illustrates the linear and angular acceleration profiles. It is clear the AUV did the task described in 1 with very good performance. It is not the best performance possible; it can be improved of course by better tuning the gains or even try an optimal control algorithm but the time constraints allowed only for implementing the PID. Thank you.

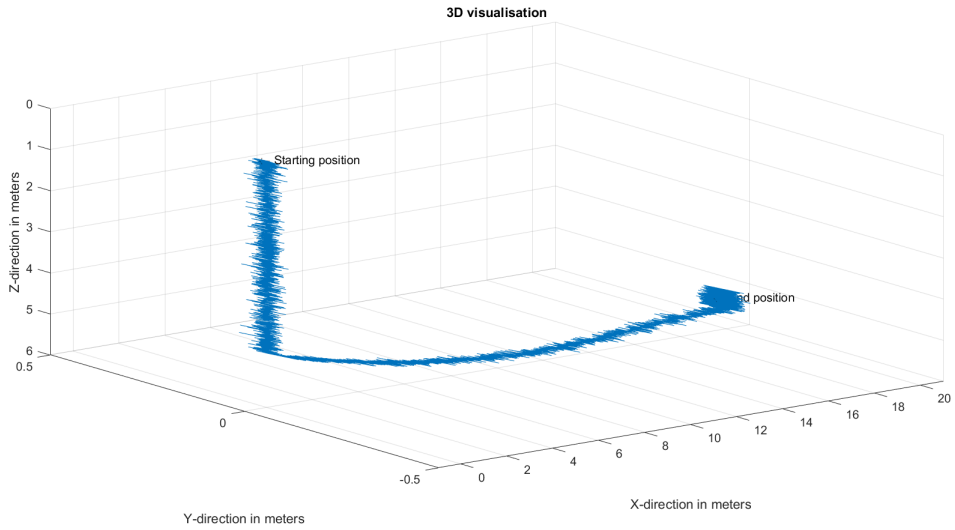

Figure 3: AUV Position - 3D visualisation

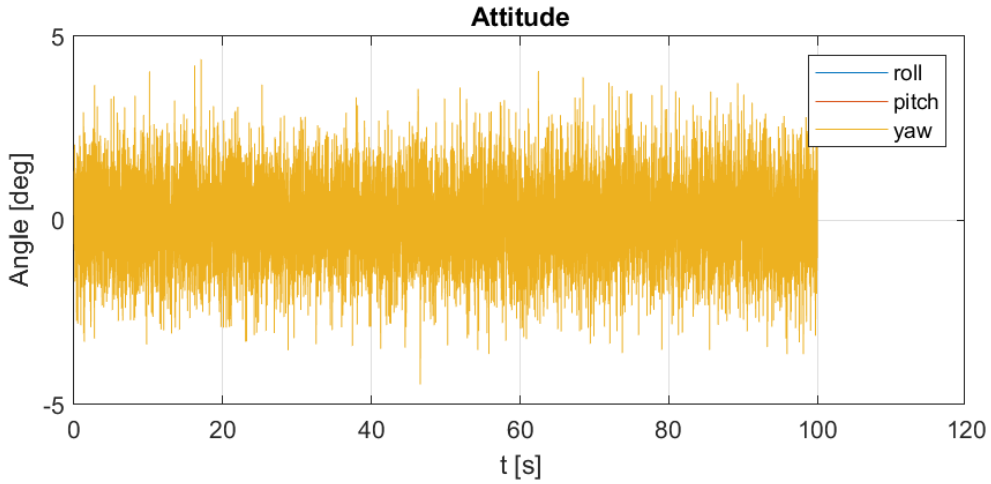

Figure 4: AUV Angular position

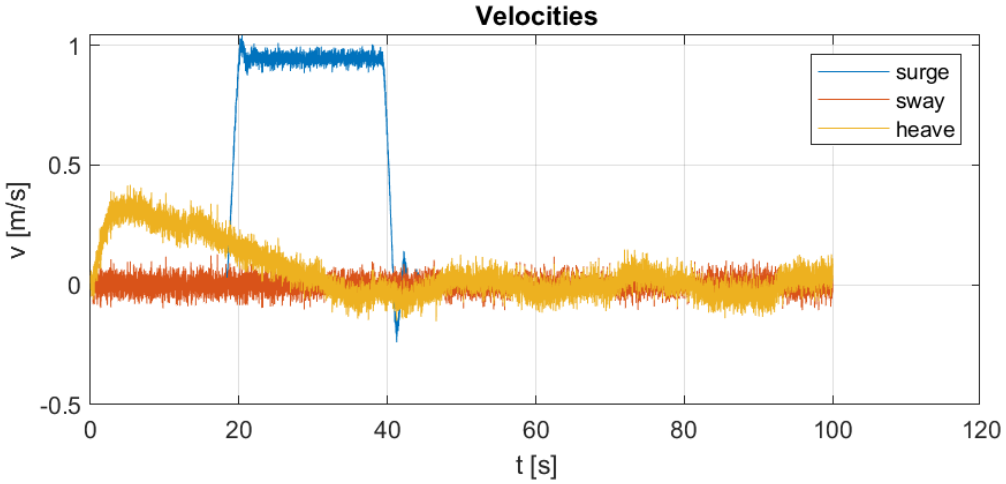

Figure 5: AUV linear velocity

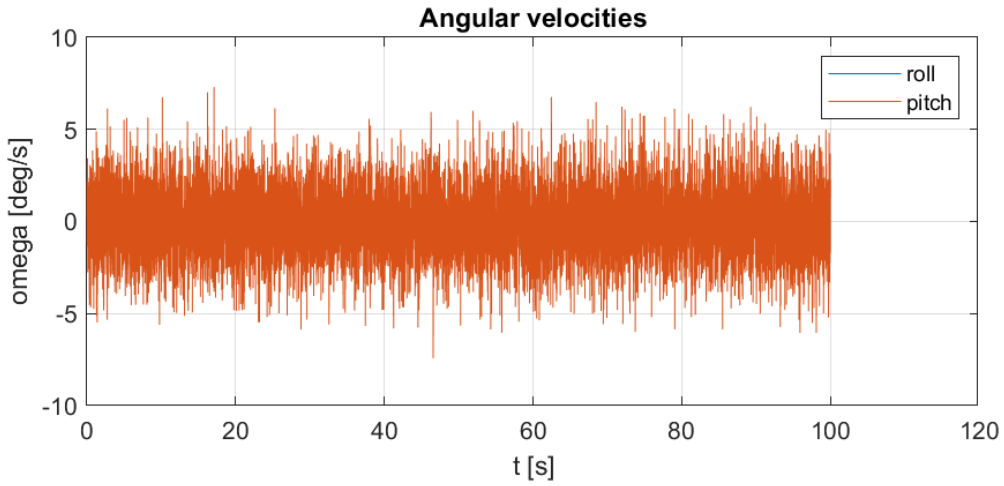

Figure 6: AUV Angular velocity

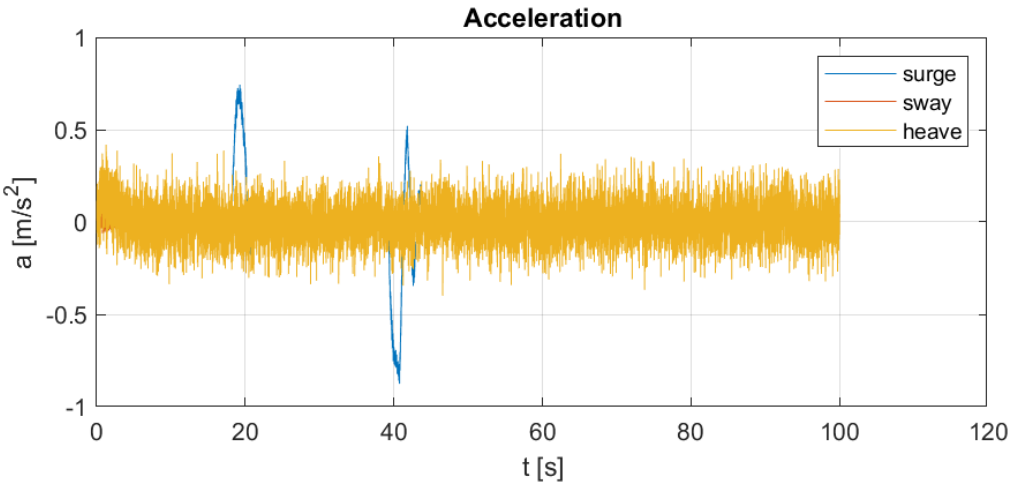

Figure 7: AUV linear acceleration

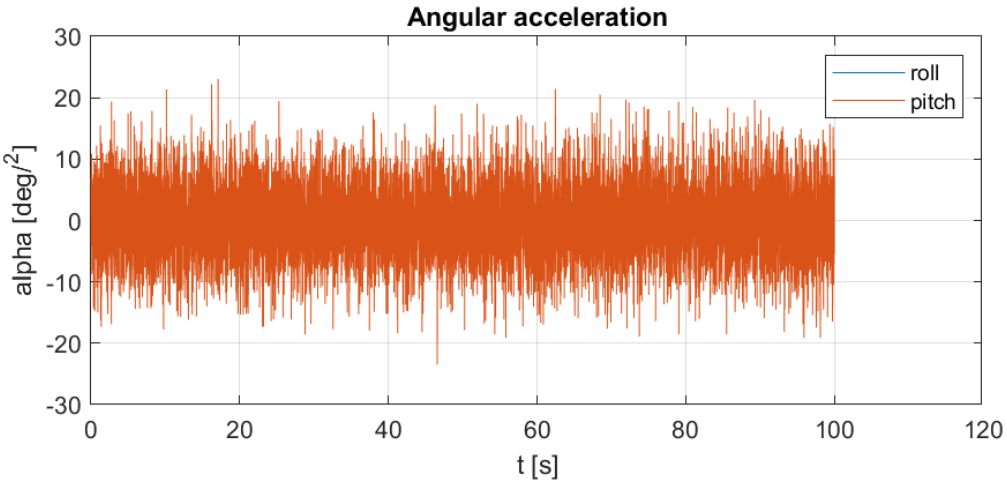

Figure 8: AUV angular acceleration

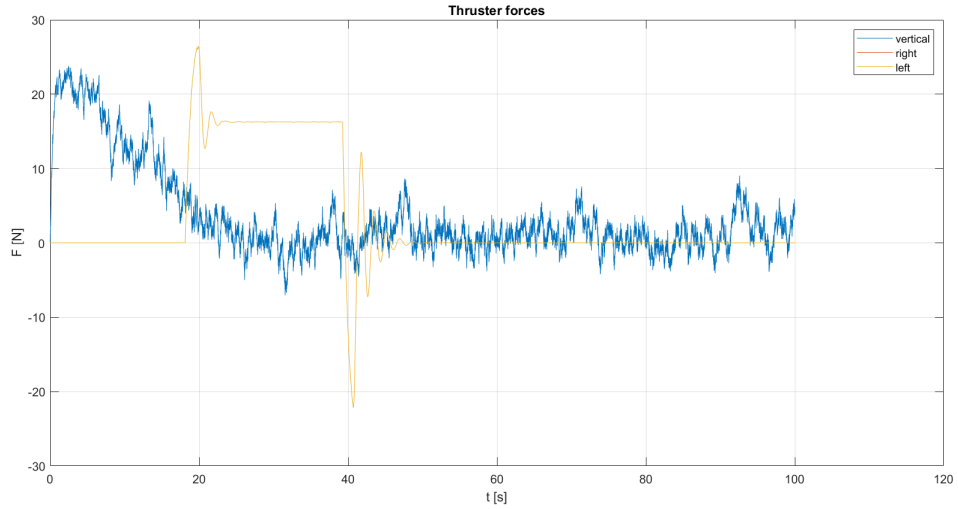

Figure 9: AUV thrusters

4 Code used in command block

```
function [Out] = Command(PosE,VitB,AccB)

% desired_z=5;
% error_z=PosE(6)-desired_z
%
% commande nulle
global delta_z0 int_delta_z0 delta_Vsurge0 int_delta_Vsurge0 step1 step2 step3 c
Kp_m=2000;      %middle motor
Kd_m=2500;
Ki_m=25;

Kp_r=1000;          %right motot
Kd_r=1;
Ki_r=0.1;

Kp_l=1000;          %left motor
Kd_l=1;
Ki_l=0.1;

dt=0.01; %like in simulink
if step1==1 && step2==0
    desired_z = 5; %5 meter depth       % desired output, or reference point
    desired_Vsurge=0;
elseif step1==0 && step2==1 && c>=20
    desired_z = 5;
    desired_Vsurge=1;

end
if step1==1 && step2==0 && PosE(3) <= 5*1.1 && PosE(3)>= 5*0.9
    step1=0;
    step2=1;
```

```
    desired_z = 5; %5 meter depth       % desired output, or reference point
    desired_Vsurge=0;

end
if step1==0 && step2==1
    desired_z = 5; %5 meter depth       % desired output, or reference point
    desired_Vsurge=0;
    c=c+1;
    if c>=20
        desired_z = 5;
        desired_Vsurge=1;
    end
end
if PosE(1) <= 20*1.1 && PosE(1)>= 19
step3=1;

end
if step3==1

desired_z=5;
desired_Vsurge=0;
end
% desired_z
% desired_Vsurge
delta_z= desired_z-PosE(3);
diff_delta_z=(delta_z-delta_z0)/dt;
int_delta_z=int_delta_z0+delta_z*dt;

delta_Vsurge= desired_Vsurge-VitB(1);
diff_delta_Vsurge=(delta_Vsurge-delta_Vsurge0)/dt;
int_delta_Vsurge=int_delta_Vsurge0+delta_Vsurge*dt;


thrust_middle= Kp_m*delta_z+Ki_m*int_delta_z+Kd_m* diff_delta_z;
thrust_right= Kp_r*delta_Vsurge+Ki_r*int_delta_Vsurge+Kd_r*diff_delta_Vsurge;
thrust_left= Kp_l*delta_Vsurge+Ki_l*int_delta_Vsurge+Kd_l*diff_delta_Vsurge;
%
thrust_middle=saturate_thruster(thrust_middle);
thrust_right=saturate_thruster(thrust_right);
thrust_left=saturate_thruster(thrust_left);

Thrust=[thrust_middle thrust_right thrust_left];
delta_z0=delta_z;
int_delta_z0=int_delta_z; %initialized as zero in parameters
delta_Vsurge0=delta_Vsurge;
int_delta_Vsurge0=int_delta_Vsurge;%initialized as zero in parameters

Out=Thrust;
```