ELEC-E8101 Group project: Lab B report Group 11

HOUILLE Florent, MONTE SASTRE Enrique, SOULARD Yoann, WANG Tong

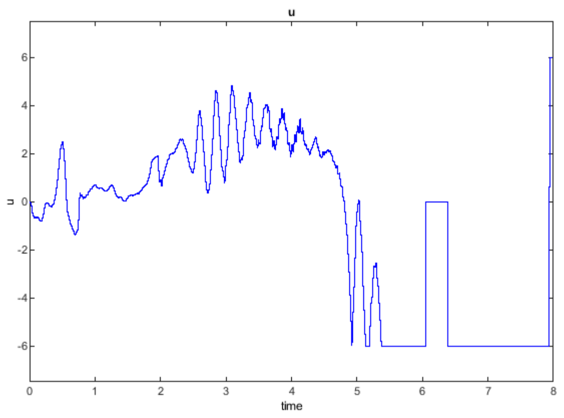
November 19, 2019

Reporting of Task 5.1

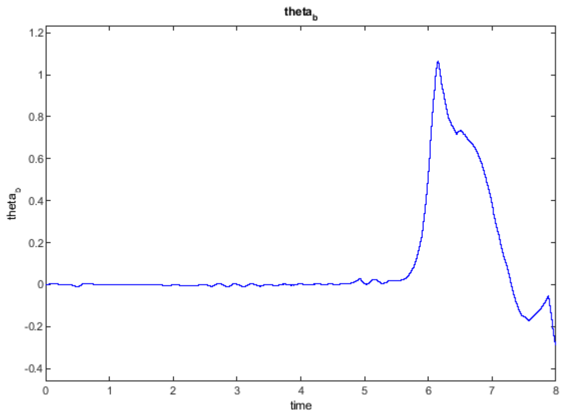
We implemented our PID with the values we obtained in lab A. Unfortunately, the sampling time (42ms) was not working at all. The robot's wheels were just stuck, making a nearly imperciptible back and forth movement. If the sampling time would have been too high, the system would have been unstable and the command would have maxed the limit out right at the beginning. So we didn't find a good reason to explain this behaviour. We tried to bend the robot to force the wheels in one particular direction, but still they are alterning quickly between forward and backward movement. So instead we used the minimum sampling time (5 ms) and tuned the PID's parameter to finally obtain :

$$K_p = -250; K_i = -500; K_d = -0.6$$

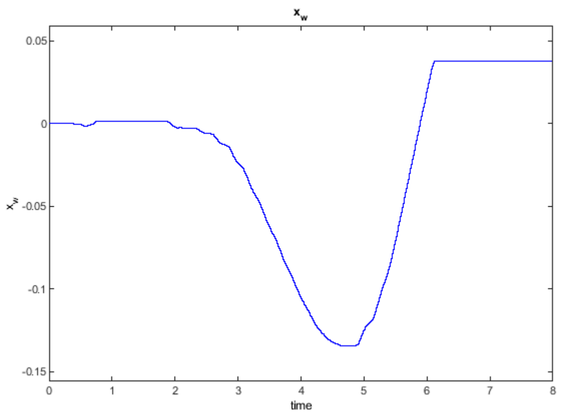
Actually these parameters were difficult to tune because the robot's behaviour is highly depend of its initial angle. For instance sometimes it could stay up for more than 5 seconds and other times it fell in only 1 second with the exact same parameters set. We decreased K_p to get a smoother behaviour around the null angle (i.e. improve stability, prevent jitters). We decreased integral term to prevent overshoot (once the robot reached the null angle, it was falling the other side). We increased derivative term so that the robot is better "aware" that it is going to fall down and can act before it's too late. Here is a result we recorded with Simulink :



(a) $u(t)$



(b) $\theta_b(t)$



(c) $x_w(t)$

Figure 1: Simulation results (the robot falls after 5.5 seconds)

The command goes at its maximum value at $t = 5s$. It corresponds to the time when the robot stops moving (i.e. x_w is stabilized), before going the other way. We think this brutal command is the reason why the robot fell down. One can also notice that the command is oscillating. This might be due to high frequency noises coming from the sensors and then transmitted to the PID's derivative action (we didn't use any filter).

Reporting of Task 5.2

5.2.1 Here are the controllability $\mathcal{C}$ and observability $\mathcal{O}$ matrices in parametric and numeric form (according to lab A state-space):

$$\mathcal{C} = \begin{bmatrix} B & AB & AAB \end{bmatrix} = 1.10^5 \begin{bmatrix} 0 & 0 & 4.10^{-4} & \approx 0 \\ 4.10^{-4} & \approx 0 & -0.3087 & -1.44.10^{-2} \\ 0 & 0 & -1.6.10^{-3} & \approx 0 \\ -1.6.10^{-3} & \approx 0 & 1.3216 & 6.15.10^{-2} \end{bmatrix}$$

$$\mathcal{O} = \begin{bmatrix} C \\ AC \\ AAC \\ AAAC \end{bmatrix} = 1.10^6 \begin{bmatrix} 0 & 0 & \approx 0 & 0 \\ 0 & 0 & 0 & \approx 0 \\ 0 & 3.3.10^{-3} & 1.10^{-4} & -1.10^{-4} \\ 0 & -2.7943 & -0.0262 & 0.0587 \end{bmatrix}$$

$$\Delta\mathcal{C} = -1.10^{-5} \quad \Delta\mathcal{O} = 0$$

$\mathcal{C}$ matrice is full-rank (4), therefore the system is controllable. $\mathcal{O}$ matrice is not full-rank, therefore the system is non-observable.

5.2.2 a) Our system is non-observable. The system is unobservable if there is at least one state we can't observe. When we represented the system state space form, C matrix had 3 null columns (out of 4). That's why our system is unobservable. This can be due to some missing physical sensors.

5.2.2 b) We should design observer based on state-space model to obtain specific closed-loop characteristics of the estimation error. We should build state estimator to obtain our estimation state vector so we can compute estimation error (and therefore observe all states).