Master's degree in Computer Engineering for Robotics and Smart Industry

Physical Human Robot Interaction – course long project

Academic year: 2021-2022

Author: Fabio Castellini (VR464639)

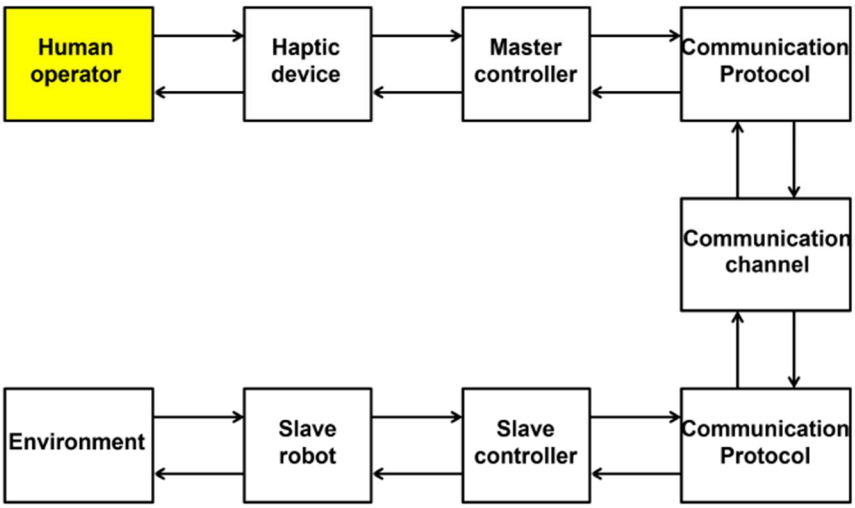
Assignment 1

- **An overview on TELEOPERATION**

The main idea is to let the operator interact with the manipulator, using an external *haptic* device like a joystick.

So, the main aspects to consider are:

- ***Slave robot***: the manipulator that really interacts with the remote environment
- ***Master robot***: device used to control the slave robot
- ***Controller***: must guarantee the overall stability while amplifying/attenuating the sensed/actuating interactions (forces, torques...)
- ***Human-robot interface***: joystick to which forces and torques are applied by the human
- ***Robot-environment interface***: interaction between slave robot and external environment
- ***Communication channel***: packet-based network onto which signals travel (delays represent a problem)



Being inside a fully simulated environment, all the above components must be modelled because they're not real.

1. First of all, the ***human intention*** has been modelled as mass-spring-damper system:

$$m_h\ddot{x}_h + b_h\dot{x}_h + k_h x_h = f_h^\star - f_h$$

Where:

x_h: operator's position
$f_h^\star$: human intentional force
f_h: human/haptic device interaction force
m_h, b_h, k_h: mass, damping, stiffness

Using the *human arm impedence* $\mathbf{Z_h}$ that maps the master robot's position (x_h=x_m during contact) into the contact force f_h, the human arm's dynamics can be rewritten as:

$$f_h = f_h^\star - Z_h \dot{x}_h$$

where f_h^* is the contact force's component imposed by the muscles (*active exogenous component*), while f_h is a *passive exogenous component*.

2. Similarly, to the human intention, the ***environment*** is modelled as such:

$$f_e = \begin{cases} m_e \ddot{x}_e + b_e \dot{x}_e + k_e(x_e - \bar{x}_e) + f_e^\star, \\ 0, \end{cases}$$

Where f_e^* is the *active exogenous component* at the environment side (for example a beating heart); $\mathbf{Z_e}$ is the environment impedance that maps the slave robot's position (x_e=x_s during contact) into the contact force (*passive feedback component*).

As seen in the course "Advanced control systems" the ***manipulator*** is modelled as such (joint space – operational space):

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + F(q,\dot{q}) + G(q) = u + J^T(q)f$$

$$\Lambda(x)\ddot{x} + \Xi(x,\dot{x})\dot{x} + \Phi(x,\dot{x}) + \gamma(x) = J_a^{-T}(q)u + f_a \qquad f_a = T_a^{-1}(\phi)f, \quad J_a(q) = T_a(\phi)J(q)$$

3. Using the linear single-axis approximation, the ***master robot*** has been modelled as:

$$m_m \ddot{x}_m + b_m \dot{x}_m + k_m x_m = f_{mc} + f_h$$

where x_m: master robot position
f_{mc}: control force due to the master controller
f_h: human/haptic device interaction force
m_m, b_m, k_m: mass, damping, stiffness

4. Using the linear single-axis approximation, the ***slave robot*** has been modelled as:

$$m_s \ddot{x}_s + b_s \dot{x}_s + k_s x_s = f_{sc} - f_e$$

where x_s: slave robot position
f_{sc}: control force due to the slave controller
f_e: environment reaction force
m_s, b_s, k_s: mass, damping, stiffness

Translating the mentioned equations in frequency domain using Laplace:

Human $$Z_h(s) = \frac{F_h^\star(s) - F_h(s)}{sX_h(s)} = \frac{m_h s^2 + b_h s + k_h}{s}$$

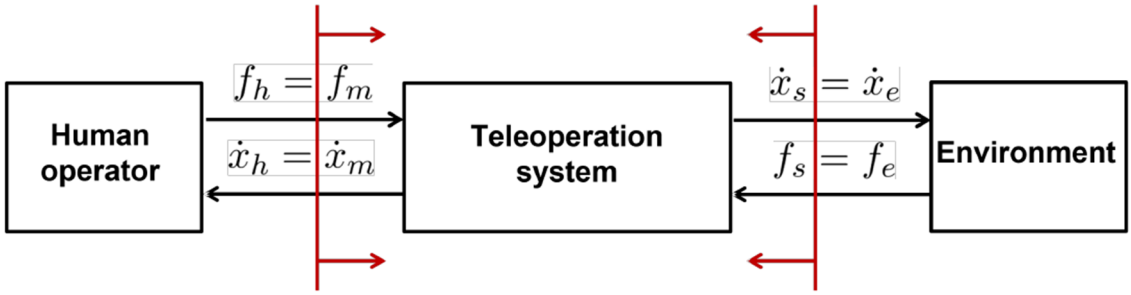
Master robot $$Z_m^{-1}(s) = \frac{sX_m(s)}{F_{mc}(s) + F_h(s)} = \frac{s}{m_m s^2 + b_m s + k_m}$$

Slave robot $$Z_s^{-1}(s) = \frac{sX_s(s)}{F_{sc}(s) - F_e(s)} = \frac{s}{m_s s^2 + b_s s + k_s}$$

Environment $$Z_e(s) = \frac{F_e(s) - F_e^\star(s)}{sX_e(s)} = \begin{cases} \frac{m_e s^2 + b_e s + k_e}{s}, & \text{contact} \\ 0, & \text{free motion} \end{cases}$$

- **Two-port representation**

The *bilateral teleoperation scheme* can be summed up as the following:



So, impedance, admittance or hybrid causalities can be extracted:

$$\begin{bmatrix} f_m \\ f_s \end{bmatrix} = \begin{bmatrix} Z_{11}(s) & Z_{12}(s) \\ Z_{21}(s) & Z_{22}(s) \end{bmatrix} \begin{bmatrix} \dot{x}_m \\ \dot{x}_s \end{bmatrix} \qquad \begin{bmatrix} f_m \\ \dot{x}_m \end{bmatrix} = \begin{bmatrix} H_{11}(s) & H_{12}(s) \\ H_{21}(s) & H_{22}(s) \end{bmatrix} \begin{bmatrix} \dot{x}_s \\ -f_s \end{bmatrix}$$

$$\begin{bmatrix} \dot{x}_m \\ \dot{x}_s \end{bmatrix} = \begin{bmatrix} Y_{11}(s) & Y_{12}(s) \\ Y_{21}(s) & Y_{22}(s) \end{bmatrix} \begin{bmatrix} f_m \\ f_s \end{bmatrix} \qquad \begin{bmatrix} f_m \\ -\dot{x}_s \end{bmatrix} = \begin{bmatrix} \bar{H}_{11}(s) & \bar{H}_{12}(s) \\ \bar{H}_{21}(s) & \bar{H}_{22}(s) \end{bmatrix} \begin{bmatrix} \dot{x}_m \\ f_s \end{bmatrix}$$

By imposing the "perfect transparency" constraints, that allow a perfect reproduction of the impedance seen at the opposite end of the teleoperator, at the master side, the hybrid matrices become: $\begin{bmatrix} H_{11}(s) & H_{12}(s) \\ H_{21}(s) & H_{22}(s) \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$ $\begin{bmatrix} \bar{H}_{11}(s) & \bar{H}_{12}(s) \\ \bar{H}_{21}(s) & \bar{H}_{22}(s) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$

As seen in theory, *perfect transparency* isn't achievable because the external forces f_h^* and f_e^* are independent on the teleoperator system behavior. Also, each element of the hybrid matrix is affected by the mechanical dynamics of the master and slave robots.

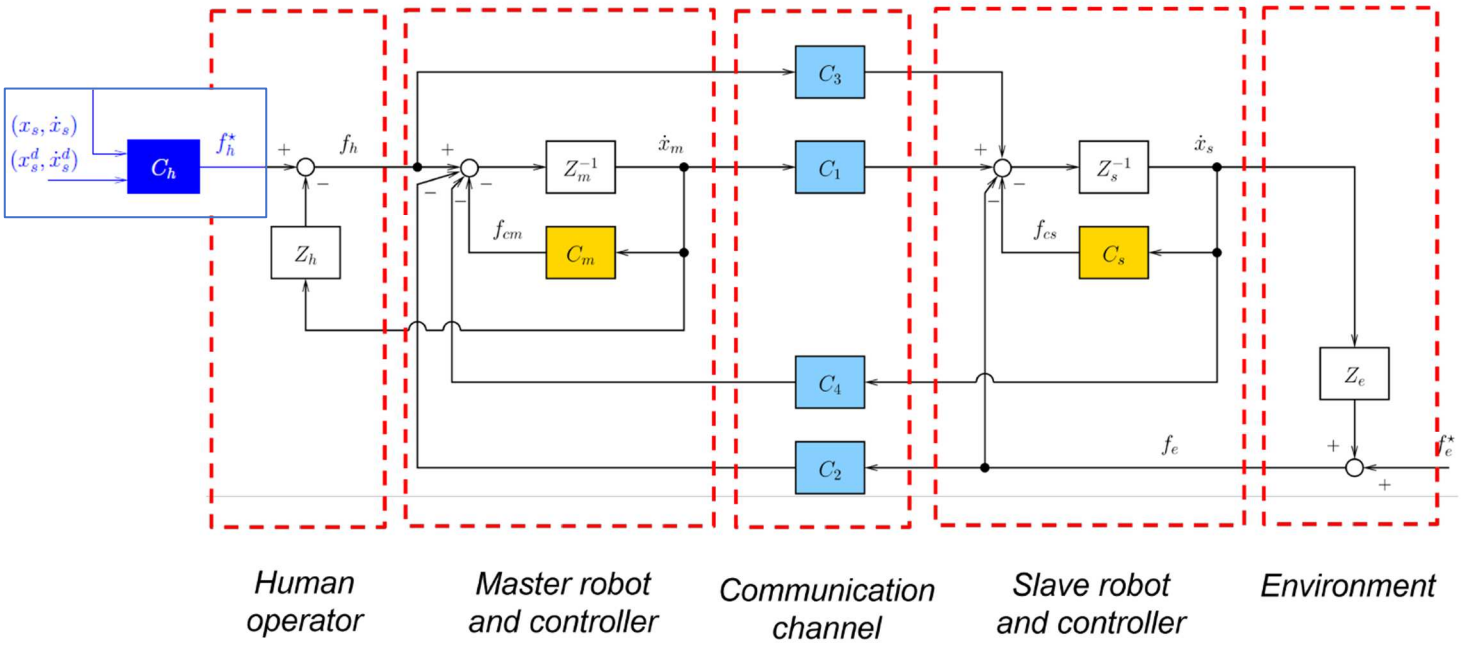
What we're looking for is <u>good transparency</u> at low frequencies. The higher the transparency bandwidth, the larger the *degree of telepresence*. A suitable control architecture to guarantee the necessary transparency has to be chosen.

- **Implement the Single-Input Single-Output Four-channel bilateral teleoperation architecture**

Four-channel means that there are two signals from master to slave and two signals from slave to master, that could be force and velocity, for instance.

Several *assumptions* were made:
- No communication delays between slave side and master side
- Perfect knowledge of the master and slave robot dynamics
- Force and velocity measurements (master/slave) available

The SISO Four-channel bilateral teleoperation scheme:



To build the Simulink model, the controllers and impedance blocks were set as such (according to the theory):

$$C_h(s) = K_h + D_h s$$

$$C_m = B_m + \frac{K_m}{s}, \qquad C_s = B_s + \frac{K_s}{s}$$

$$\begin{aligned} C_1 &= Z_s + C_s \\ C_2 &= 1 \\ C_3 &= 1 \\ C_4 &= -(Z_m + C_m) \end{aligned}$$

$$Z_e(s) = J_e s + B_e + K_e \frac{1}{s}$$

$$Z_h(s) = J_h s + B_h + K_h \frac{1}{s}$$

$$Z_m^{-1} = \frac{1}{M_m s + D_m}, \qquad Z_s^{-1} = \frac{1}{M_s s + D_s}$$

$$J_h = 0.5,\ B_h = 70,\ K_h = 2000$$

$$B_e = 100,\ K_e = 200$$

$$f_e^{\star} = 0$$

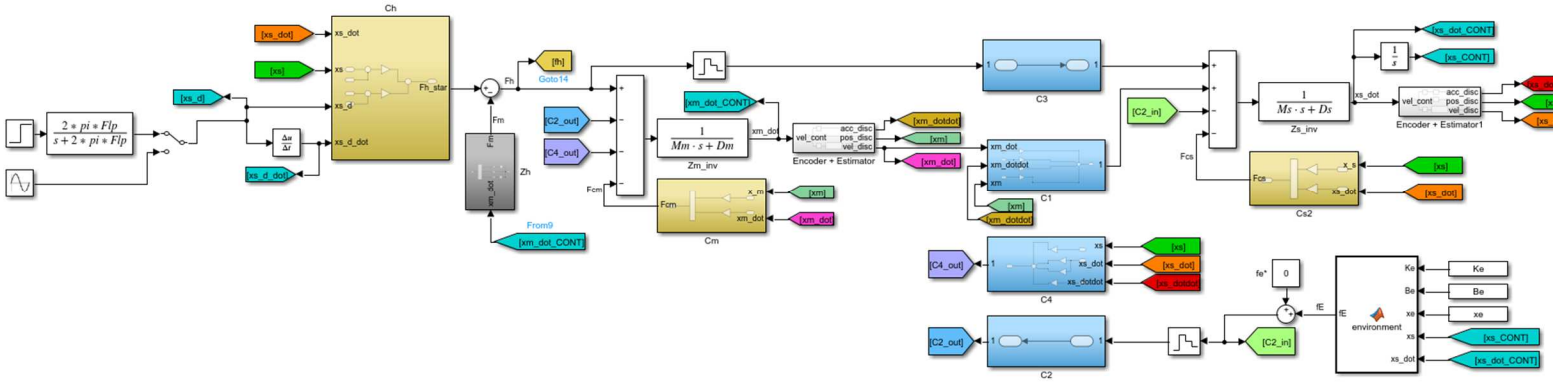
$$M_m = 0.5,\ M_s = 2$$

With reference signals:

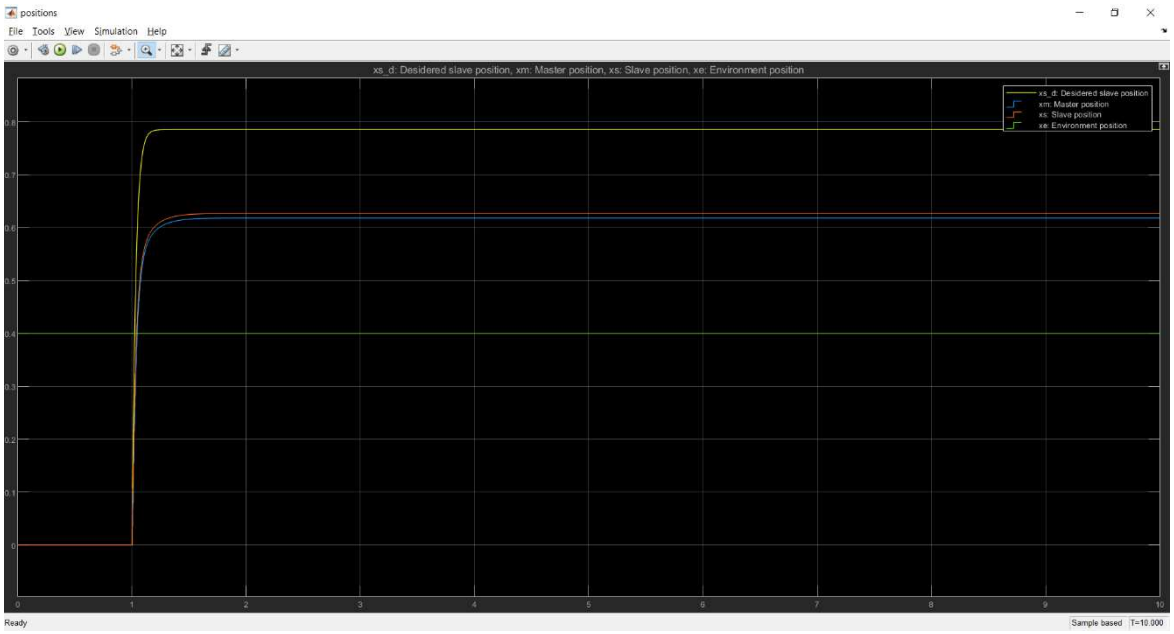
$$x^d(t) = A \sin(2\pi F_c t)$$

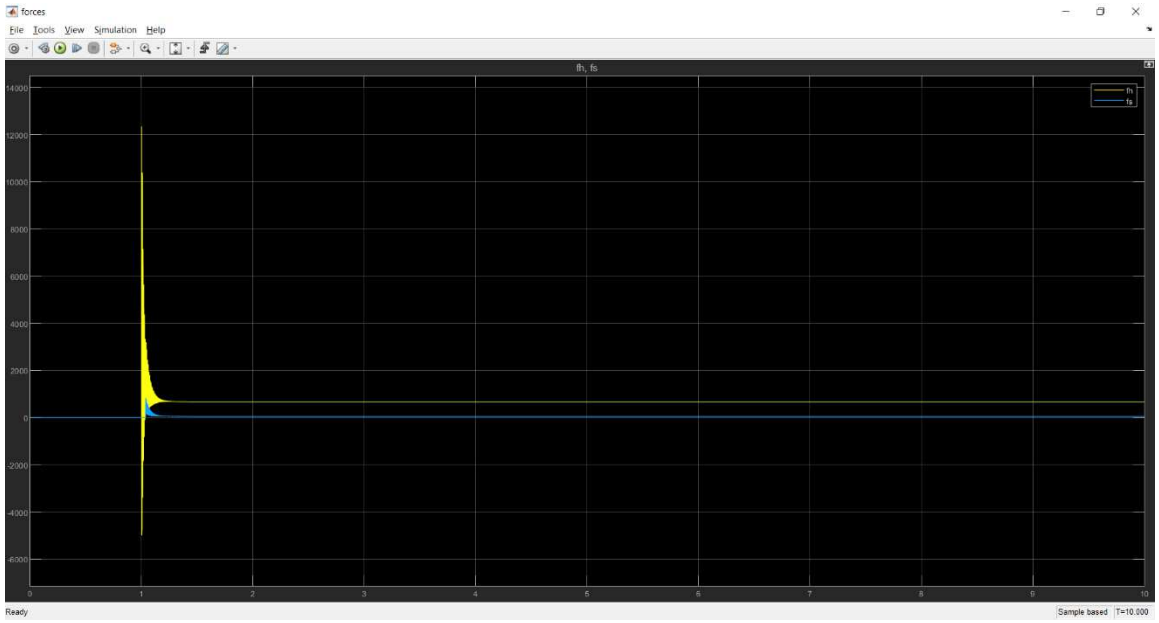
$$x^d(t) = \frac{2\pi F_{lp}}{s + 2\pi F_{lp}} A \delta_{-1}(t)$$

My Simulink model:

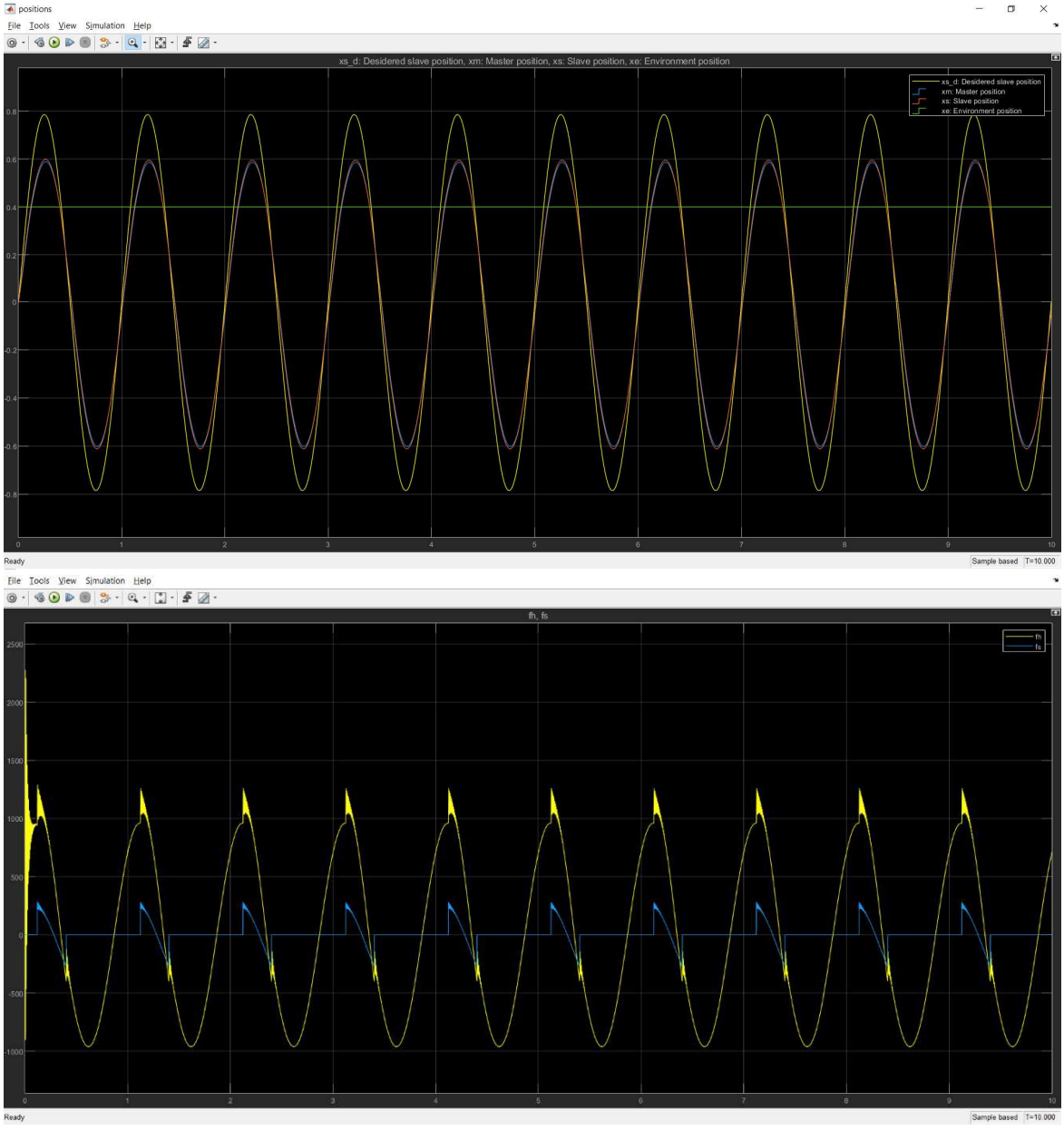


Filtered step reference signal, contact results *(positions and forces):*

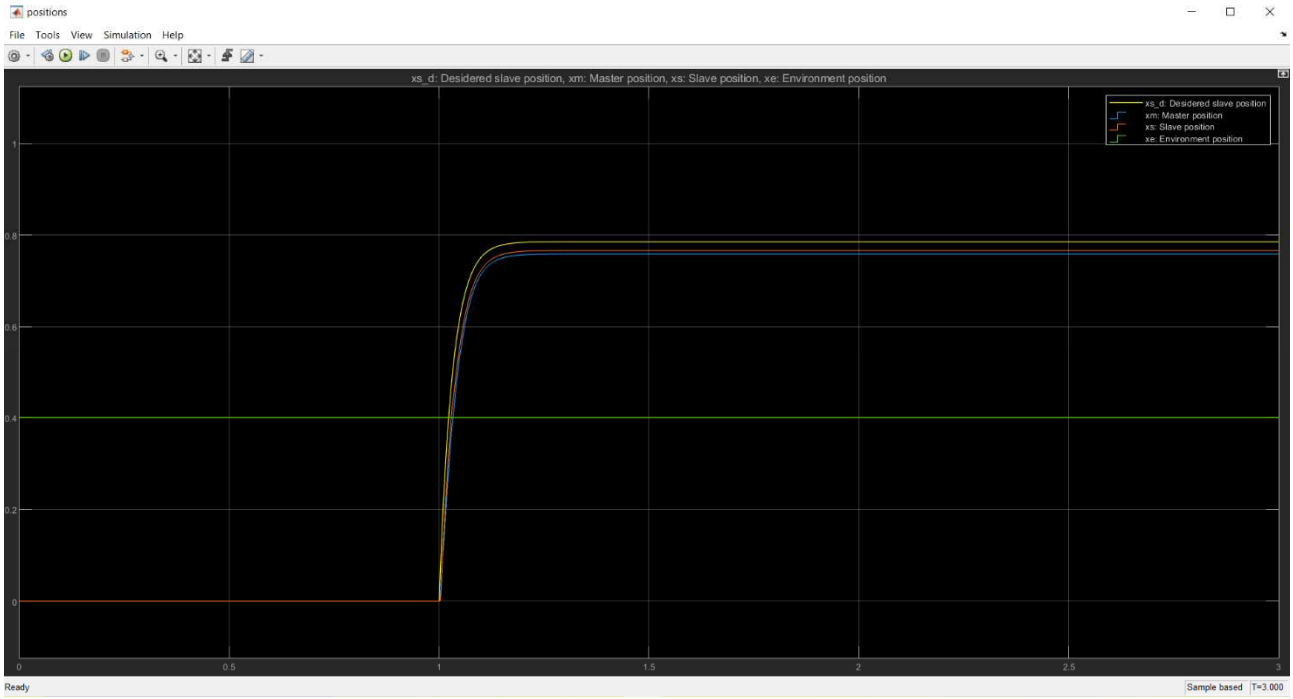




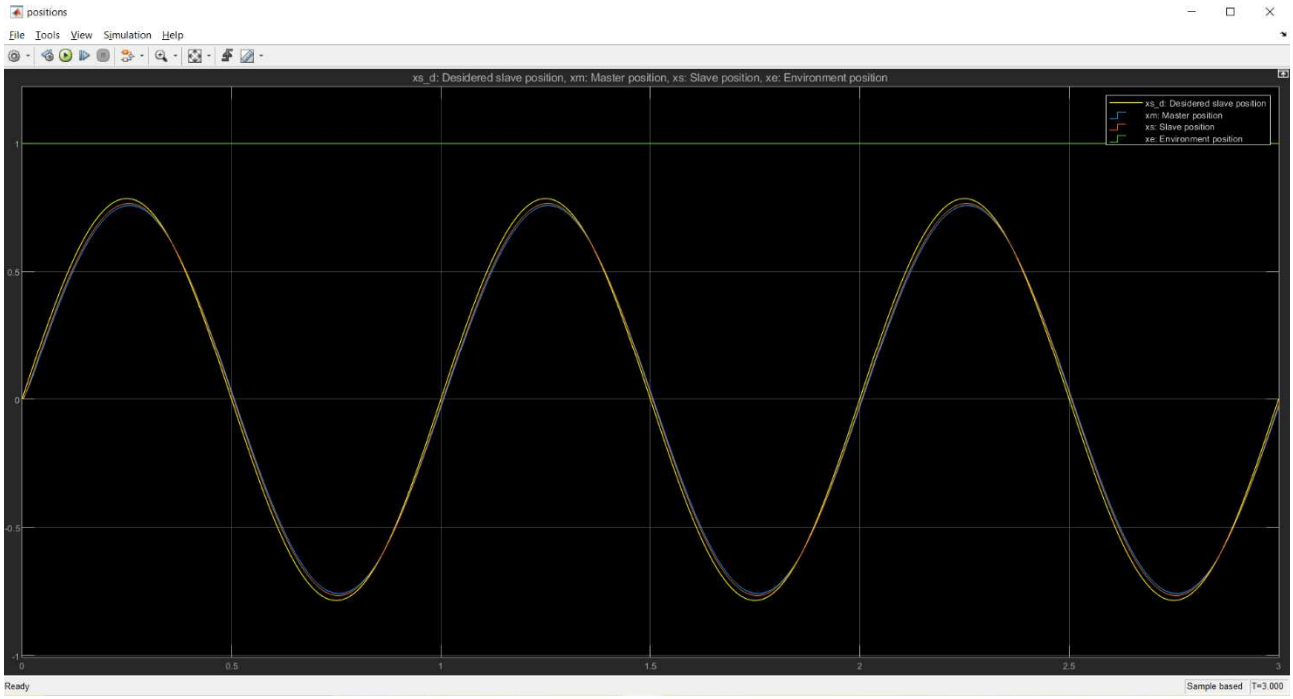
Sinusoidal reference signal results, contact *(positions and forces):*



Step reference signal results, contact, less stiff environment:

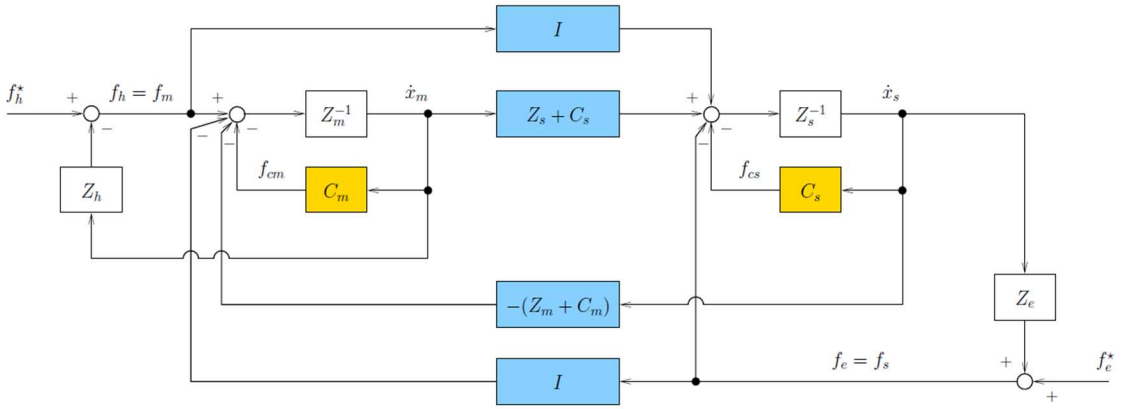


Sinusoidal reference signal results, free motion:



Assignment 2

- **Derive the expression for the hybrid matrix H (*from the scheme*)**



Assignment 2 - Derive the expression for H

Results (according to the theory):

$$\begin{bmatrix} f_m \\ \dot{x}_m \end{bmatrix} \begin{bmatrix} H_{11}(s) & H_{12}(s) \\ H_{21}(s) & H_{22}(s) \end{bmatrix} = \begin{bmatrix} \dot{x}_s \\ -f_s \end{bmatrix}$$

$$\Rightarrow \boxed{H_{11} = \left.\frac{f_m}{\dot{x}_s}\right|_{f_s=0} = \frac{(Z_m + C_m)(Z_s + C_s - C_3C_4 + C_3C_4) + C_1C_4}{C_1 + C_3(Z_m + C_m)} = \frac{(Z_m + C_m)(Z_s + C_s) + C_1C_4}{C_1 + C_3(Z_m + C_m)}}$$

Proof:

- From the scheme:

$$(1)\quad \frac{\dot{x}_s}{f_m C_3 + \dot{x}_m C_4 - f_s C_{sf} - f_s} = \frac{1}{Z_s + C_s} \qquad f_e = f_s$$

$$(2)\quad \frac{\dot{x}_m}{f_m + C_{mf} f_m - C_4 \dot{x}_s - C_2 f_s} = \frac{1}{Z_m + C_m} \Rightarrow \dot{x}_m = \frac{f_m + C_{mf} f_m - C_4 \dot{x}_s}{Z_m + C_m} \qquad -C_2 f_s$$

- Substituting (2) inside (1):

$$(3)\quad \dot{x}_s = \left(f_m C_3 - f_s(C_{sf} + 1) + \frac{C_4 f_m (C_{mf} + 1 - C_4 f_s)}{Z_m + C_m} \right) \frac{Z_m + C_m}{C_1 C_4 + (Z_m + C_m)(Z_s + C_s)}$$

$$\Rightarrow \left.\dot{x}_s\right|_{f_s=0} = f_m \left(C_3(Z_m + C_m) + C_4(C_{mf} + 1) \right) \frac{1}{C_1 C_4 + (Z_m + C_m)(Z_s + C_s)}$$

- Applying the definition of H_{11}:

$$\boxed{H_{11} = \frac{f_m}{\left.\dot{x}_s\right|_{f_s=0}} = \frac{C_1 C_4 + (Z_m + C_m)(Z_s + C_s)}{C_1 + C_3(Z_m + C_m) + C_4 C_{mf}}}$$

$$\Rightarrow \boxed{H_{12} = -\frac{f_m}{f_s}\Big|_{\dot{x}_s=0} \underset{\text{substituting (1)}}{=} -\frac{I + C_1 C_2}{C_1 + C_3(Z_m + C_m)}}$$

Proof:

- Using equation (3) previously computed and imposing $\dot{x}_s = 0$ we get:

$$0 = f_m\left(C_3 + \frac{C_1(C_{mf} + 1 - C_2 f_s)}{Z_m + C_m}\right) - f_s(C_{sf} + 1)$$

$$\downarrow$$

$$f_m\left(C_3 + \frac{C_1 C_{mf} + C_1}{Z_m + C_m}\right) = f_s\left(C_{sf} + 1 + \frac{C_1 C_2}{Z_m + C_m}\right)$$

$$\Rightarrow \boxed{H_{12} = -\frac{f_m}{f_s}\Big|_{\dot{x}_s=0} = -\frac{(Z_m + C_m)(C_{sf}+1) + C_1 C_2}{C_1 + C_3(Z_m + C_m) + C_1 C_{mf}}}$$

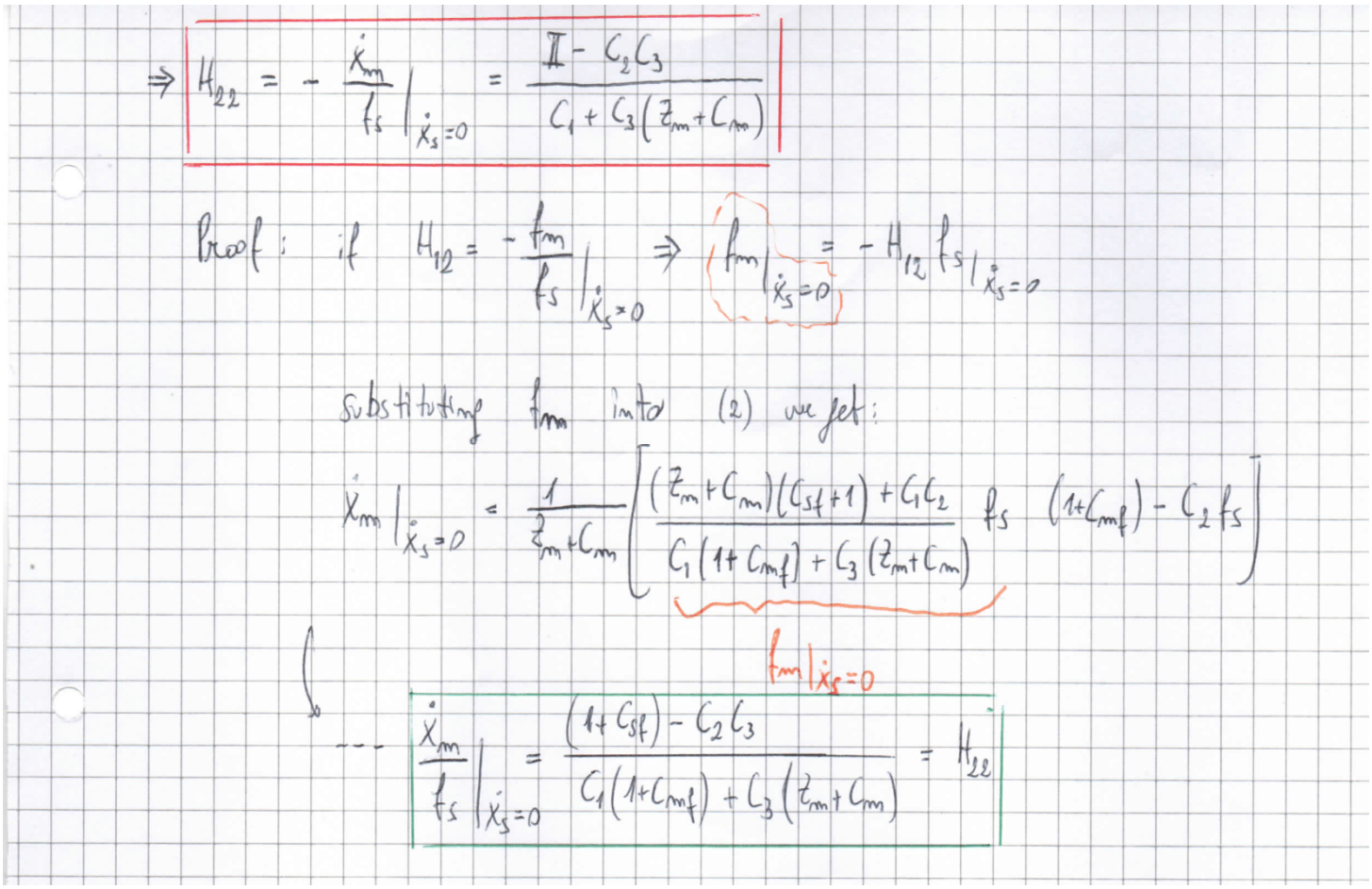
$$\Rightarrow H_{21} = \frac{\dot{x}_m}{\dot{x}_s}\Big|_{f_s=0} = \frac{Z_s + C_s - C_3 C_4}{C_1 + C_3(Z_m + C_m)}$$

Proof: if $H_{11} = \dfrac{f_m}{\dot{x}_s}\Big|_{f_s=0} \Rightarrow f_m\Big|_{f_s=0} = \dot{x}_s\Big|_{f_s=0} \cdot H_{11}$

substituting f_m into (2) we get:

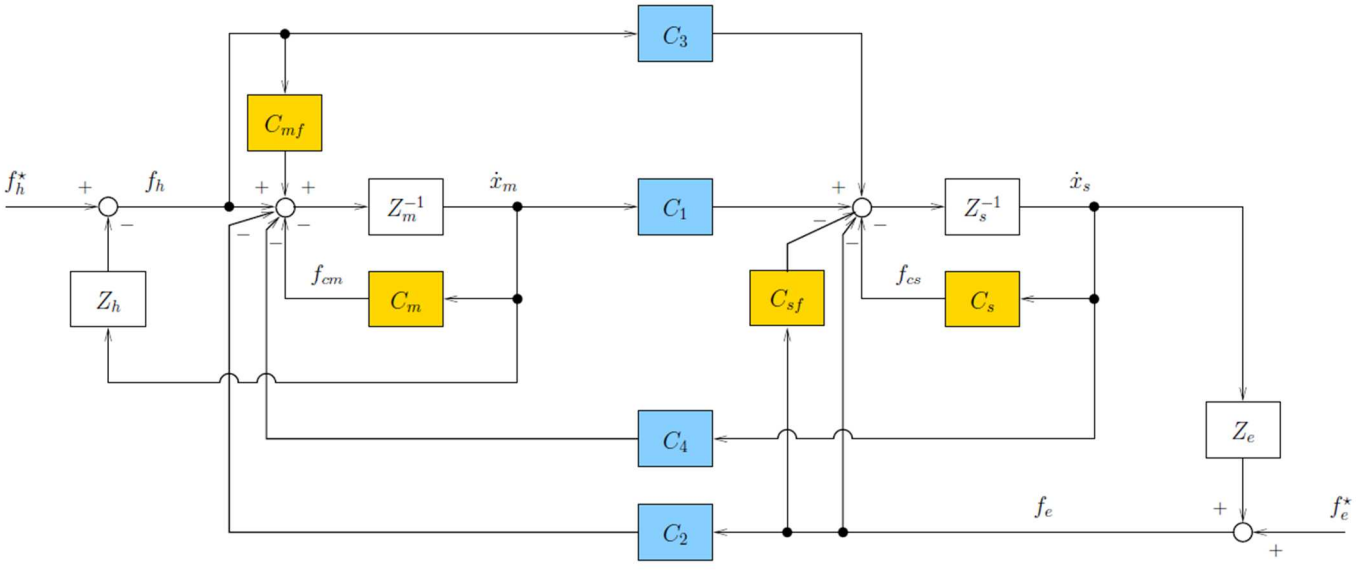
$$\dot{x}_m = \frac{\dot{x}_s}{Z_m + C_m}\left(\underbrace{\frac{C_1 C_4 + (Z_m + C_m)(Z_s + C_s)}{C_1(1 + C_{mf}) + C_3(Z_m + C_m)}}_{f_m\big|_{f_s=0}}\right)(1 + C_{mf}) - C_4\Bigg)$$

$$\Rightarrow \boxed{\frac{\dot{x}_m}{\dot{x}_s} = \cdots = \frac{(Z_m + C_m)\Big((1 + C_{mf})(Z_s + C_s) - C_3 C_4\Big)}{C_1(1 + C_{mf}) + C_3(Z_m + C_m)} = H_{21}}$$



- **Implement the Four-Channel bilateral teleoperation architecture with local force feedbacks**

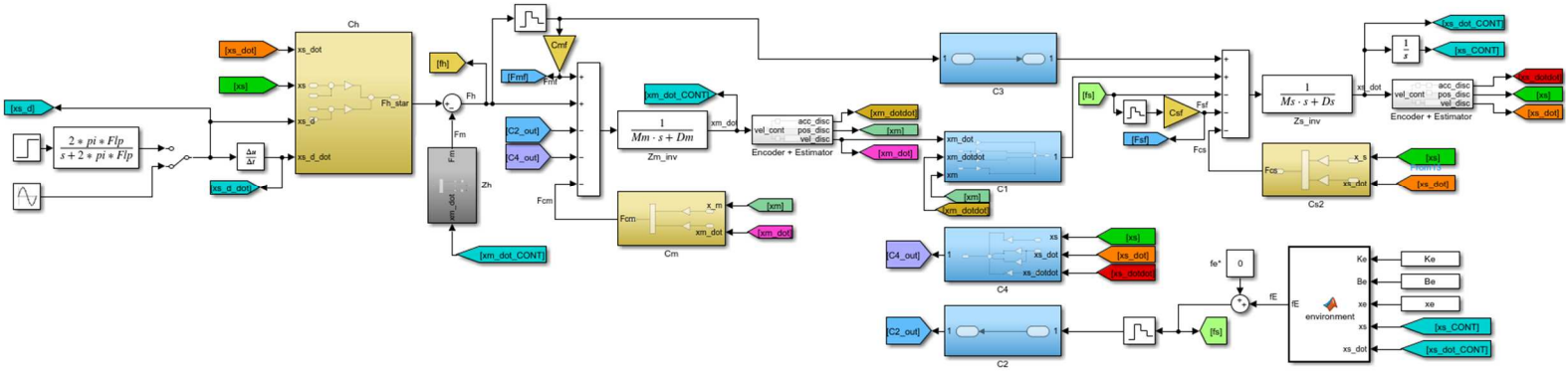
The same parameters of the previous *Assignment* were used. As suggested, C_{mf} and C_{sf} were set as constants. Moreover, C_2 and C_3 are kept as identities even though they should adapt to the environment's nature (*hard* or *soft*) in real applications.



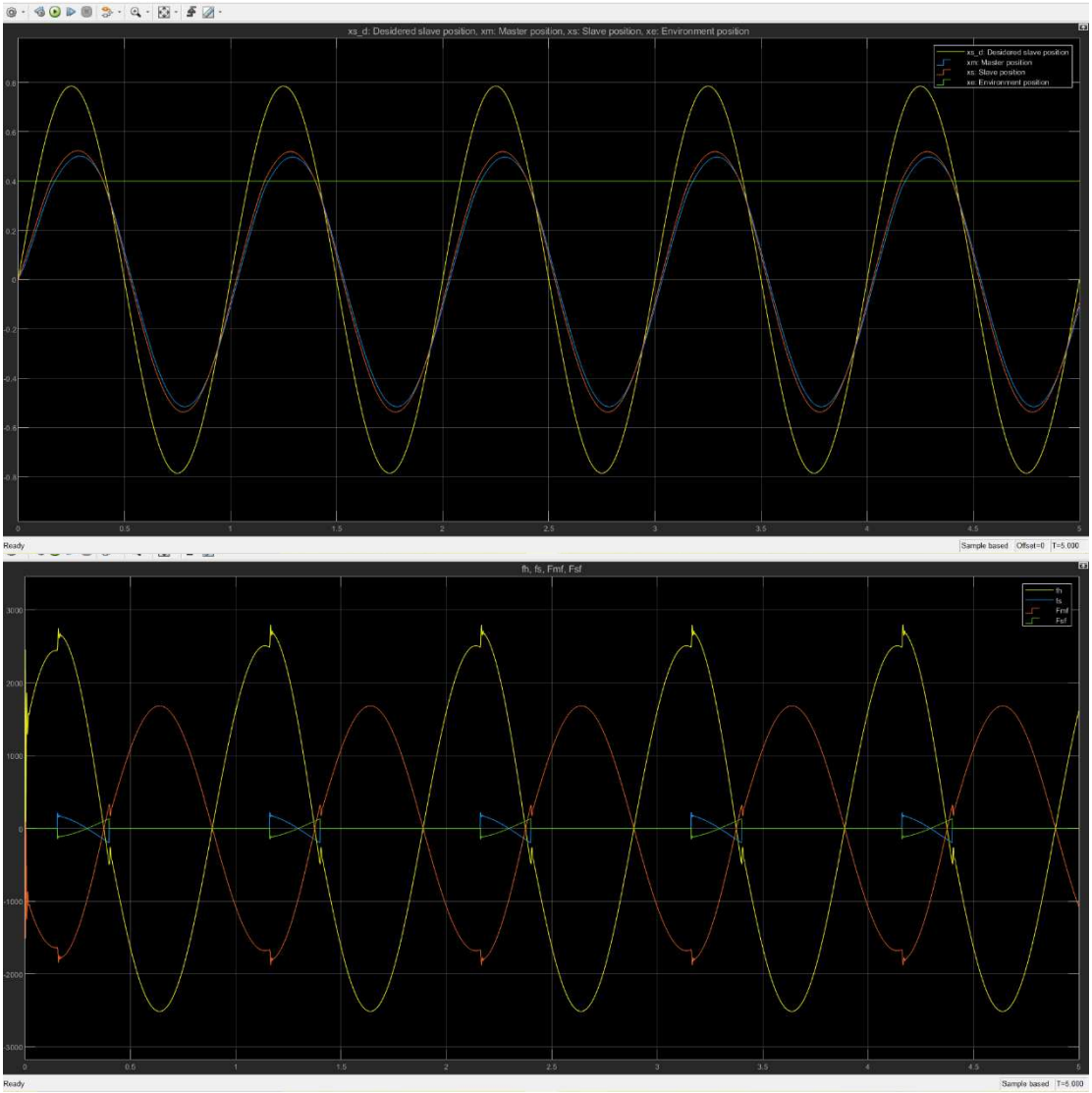
Conditions to allow *perfect transparency*:

$$
\begin{aligned}
C_1 &= Z_s + C_s = Z_{cs} \\
C_2 &= 1 + C_{mf} (\neq 0) \\
C_3 &= 1 + C_{sf} (\neq 0) \\
C_4 &= -(Z_m + C_m) = -Z_{cm}
\end{aligned}
$$

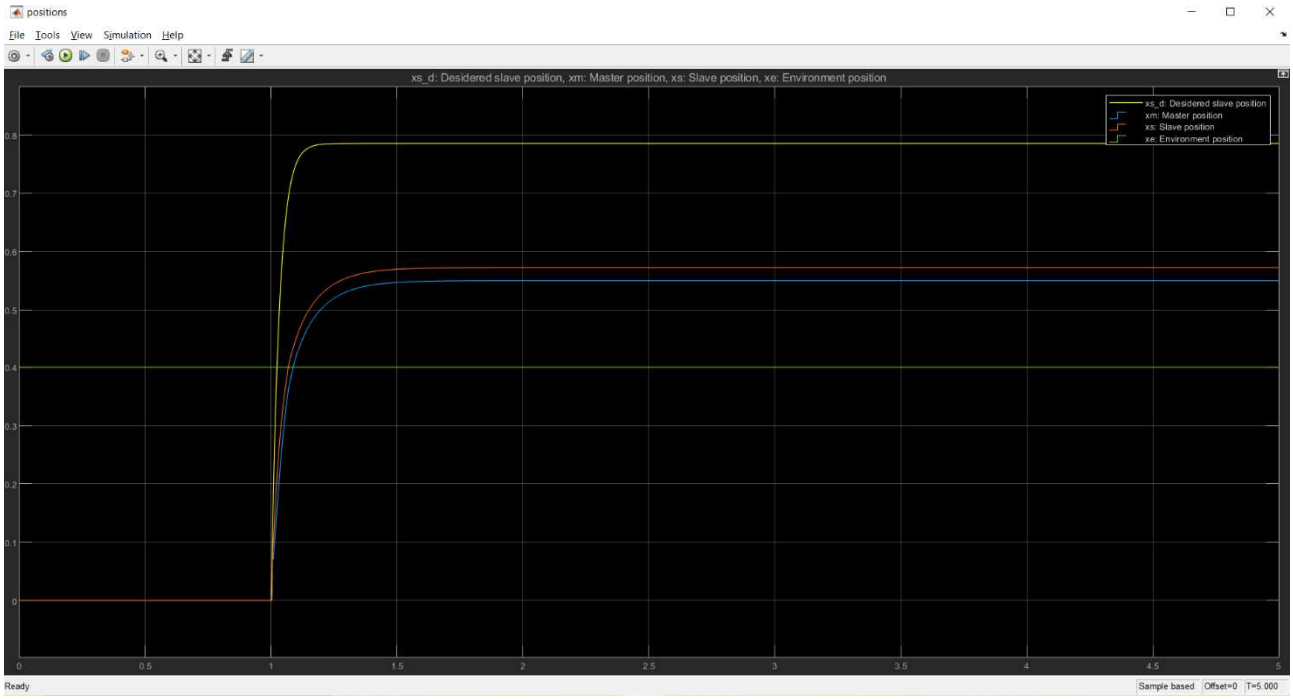
My Simulink model:

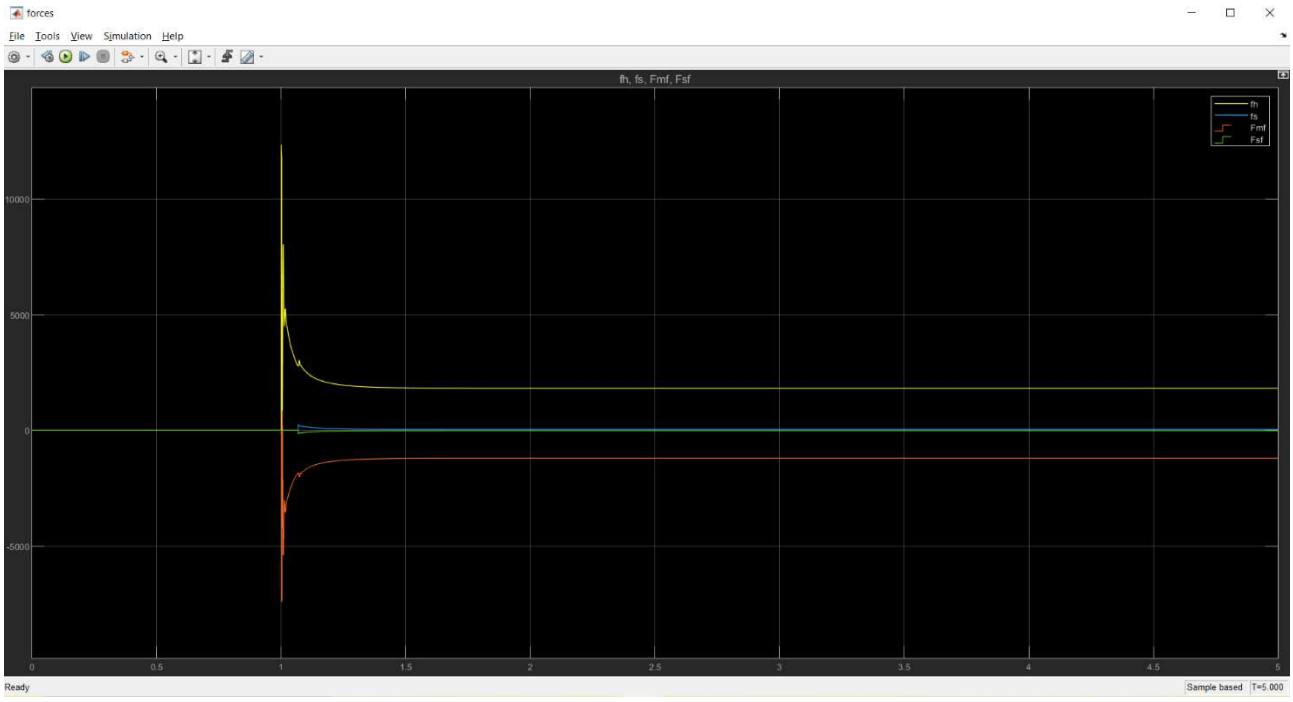


Sinusoidal reference signal results, contact *(positions and forces)*:

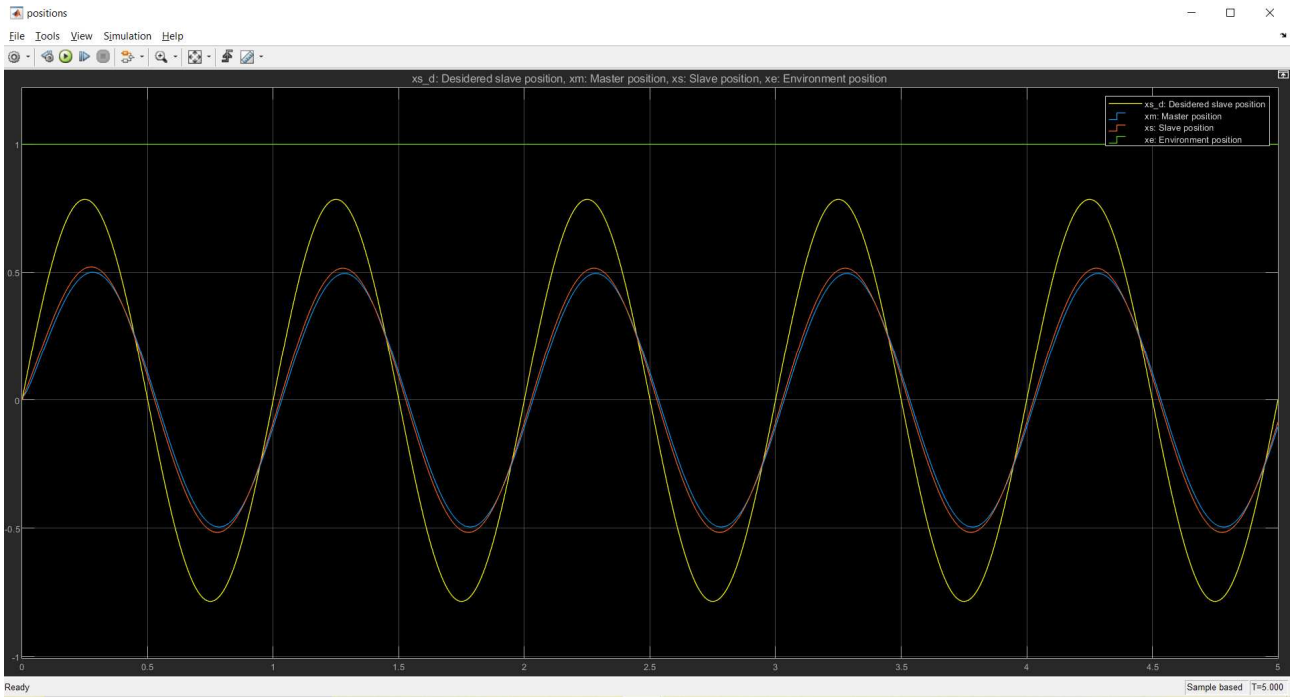


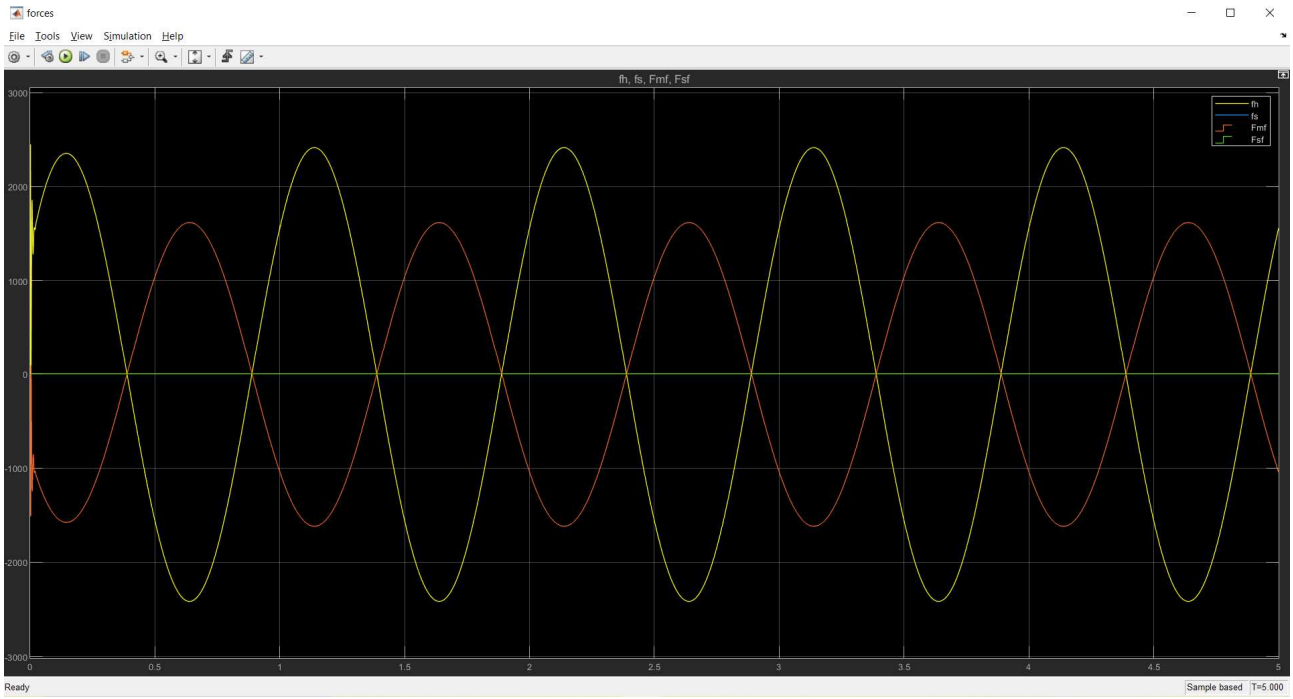
Filtered step reference signal results, contact *(positions and forces)*:





Sinusoidal reference signal results, free motion *(positions and forces)*:



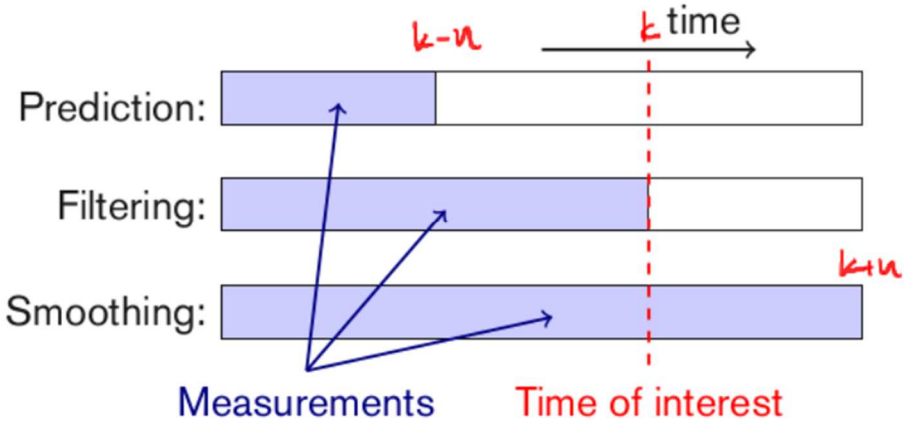


Assignment 3

- **Estimation problem**

Given noisy measurements of a signal, for instance angular position we want to retrieve a better estimation of the position at time k.

There are 3 similar approaches to solve the estimation problem:
- **Prediction**: estimate the k-*th* sample given the k-n previous samples
- **Filtering**: estimate the k-*th* sample given samples until k
- **Smoothing**: estimate the k-*th* sample given samples until $k+n$



- **Random walk of order I**

If we don't have any knowledge about the physical system which is producing the measured signal (*angular position*), we model the derivative of the signal (*angular velocity*) as ***white noise***.

A stochastic process w(t) is called white noise if its values $w(t_i)$ and $w(t_j)$ are uncorrelated $\forall i \neq j$. We also assume that w(t) is Gaussian with zero-mean and constant variance Q. Instead, v(t) is a Gaussian random variable with zero-mean and constant variance R.

$$\begin{aligned} \dot{\theta}(t) &= \omega(t) \\ \dot{\omega}(t) &= w(t) \end{aligned} \qquad\qquad y(t) = \theta(t) + v(t)$$

On the left the kinematic model to perform velocity estimation, on the right the measurement equation

$$x(t) := \begin{bmatrix} \theta(t) \\ \omega(t) \end{bmatrix} \qquad\qquad \begin{aligned} \dot{x}(t) &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} w(t) \\ y(t) &= \begin{bmatrix} 1 & 0 \end{bmatrix} x(t) + v(t) \end{aligned}$$

On the left the vector state, on the right the continuous-time state model representation

Translating the above equations in discrete time, with uniform sampling:

$$\begin{aligned} x_{k+1} &= \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} \frac{T_s^2}{2} \\ T_s \end{bmatrix} w_k \\ y_k &= \begin{bmatrix} 1 & 0 \end{bmatrix} x_k + v_k \end{aligned}$$

An important aspect is the R and Q matrices' tuning: R represents the noise variance and depends on the sensors, while Q represents the model's variance and is chosen in order to "explain" the measurements as well as possible.

- **Random walk of order II**

In this case, the acceleration is modelled as *white noise*, while the position is obtained by integrating two times. The state vector x will have as components position, velocity and acceleration.

$$\begin{aligned} \dot{x}(t) &= \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} w(t) \\ y(t) &= \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} x(t) + v(t) \end{aligned}$$

As seen before, given the (discrete) measurements y_k related to an unknown state variable x_k there are several estimation approaches:

- **Causal**: *Filtering*, *h-step ahead Prediction* and *h-step backward Smoothing*
- **Non causal (offline only)**: *Smoothing*

- **Filtering**

$$y_0, y_1, \ldots, y_k \longrightarrow \hat{x}_{k|k} \longrightarrow \hat{\omega}_{k|k}$$

- ***h*-step ahead Prediction**

$$y_0, y_1, \ldots, y_k \longrightarrow \hat{x}_{k+h|k} \longrightarrow \hat{\omega}_{k+h|k}$$

- ***h*-step backward Smoothing**

$$y_0, y_1, \ldots, y_k \longrightarrow \hat{x}_{k-h|k} \longrightarrow \hat{\omega}_{k-h|k}$$

- **Smoothing**

$$y_0, y_1, \ldots, y_N \longrightarrow \hat{x}_{k|N} \longrightarrow \hat{\omega}_{k|N}$$

- **Kalman Filter**

According to the theory, the *minimum variance estimator* theorem is used to implement the Kalman filter.

Theorem (Minimum Variance Estimator)

Let $\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^m$ be two r.v. (not necessarily Gaussian), and $g : \mathbb{R}^m \to \mathbb{R}^n$ a measurable function.

We define $\hat{\mathbf{x}}_g = g(\mathbf{y})$ as the estimator of $\mathbf{x}$ given $\mathbf{y}$ through the function g, and $\mathbf{e}_g = \mathbf{x} - g(\mathbf{y}) = \mathbf{x} - \hat{\mathbf{x}}_g$ the corresponding estimation error.

The estimator $\hat{\mathbf{x}} = \mathbb{E}[\mathbf{x}|\mathbf{y}] = \hat{g}(\mathbf{y})$ is optimal because it minimizes the error variance, i.e.

$$Var(\mathbf{e}) = \mathbb{E}[(\mathbf{x} - \hat{\mathbf{x}})(\mathbf{x} - \hat{\mathbf{x}})^T] \leq \mathbb{E}[(\mathbf{x} - \hat{\mathbf{x}}_g)(\mathbf{x} - \hat{\mathbf{x}}_g)^T] = Var(\mathbf{e}_g), \quad \forall g(\cdot)$$

where $\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}}$ is the error of the optimal estimator.

The error of the optimal estimator $\mathbf{e}$ and the estimation $\hat{g}(\mathbf{y})$ are uncorrelated

$$\mathbb{E}[\mathbf{e}\hat{g}(\mathbf{y})^T] = 0.$$

The Kalman filter (or minimum variance filter) is defined as:

$$\hat{x}_{k+1|k+1} = \mathbb{E}\left[x_{k+1}|y_0, \ldots, y_{k+1}\right] = \mathbb{E}\left[x_{k+1}|y_{k+1}, Y^k\right]$$

but has to be rewritten in order to be easily implemented, through recursion.

In particular the recursive Kalman Filter formulation is the following:

$$\hat{x}_{k+1|k+1} = A\hat{x}_{k|k} + K_{k+1}(y_{k+1} - CA\hat{x}_{k|k})$$

$$K_{k+1} = P_{k+1|k}C^T\left(CP_{k+1|k}C^T + R\right)^{-1} \qquad \textit{Kalman gain formulation}$$

$$P_{k+1|k} = AP_{k|k-1}A^T - AP_{k|k-1}C^T(CP_{k|k-1}C^T + R)^{-1}CP_{k|k-1}A^T + Q \qquad \textit{Riccati equation}$$

$$\mathbb{E}[X|Y] = \hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + \Sigma_{XY}\Sigma_{YY}^{-1}(y_{k+1} - \hat{y}_{k+1|k})$$

Optimal estimator formulation

$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + P_{k+1|k}C^T\left(CP_{k+1|k}C^T + R\right)^{-1}(y_{k+1} - C\hat{x}_{k+1|k})$$

optimal filter *optimal predictor* *innovation*

- **Kalman Predictor**

$$\hat{x}_{k+1|k} = A\hat{x}_{k|k-1} + \bar{K}_k(y_k - C\hat{x}_{k|k-1})$$

$$\bar{K}_k = AP_{k|k-1}C^T\left(CP_{k|k-1}C^T + R\right)^{-1}$$

- **Steady-state Kalman Filter/Predictor**

According to the theory, in our conditions, for k that tends to infinity we get:

Kalman filter (LTI)

$$\begin{aligned} \hat{x}_{k+1|k+1} &= A\hat{x}_{k|k} + K_\infty(y_{k+1} - CA\hat{x}_{k|k}) \\ P &= APA^T - APC^T(CPC^T + R)^{-1}CPA^T + Q \end{aligned}$$

with

$$K_\infty = PC^T\left(CPC^T + R\right)^{-1}$$

Kalman predictor (LTI)

$$\begin{aligned} \hat{x}_{k+1|k} &= A\hat{x}_{k|k-1} + \bar{K}_\infty(y_k - C\hat{x}_{k|k-1}) \\ P &= APA^T - APC^T(CPC^T + R)^{-1}CPA^T + Q \end{aligned}$$

with

$$\bar{K}_\infty = APC^T\left(CPC^T + R\right)^{-1}.$$

My MATLAB Kalman Filter code snippet with explanations:

```
function [filter_pos, filter_vel, filter_acc, P_inf] = kalmanFilter(A,B,C,M_pos,R,Q,x_0,P_0,K_inf)

    K_1 = P_0*C.' * inv(C*P_0*C.' + R);  %initial Kalman G ain: K_{0+1}
    x_1 = A*x_0 + K_1 *(M_pos(1) - C*A*x_0); %initial optimal filter: xhat_{0+1|0+1}
```

$\hat{x}_{k+1|k+1} = A\hat{x}_{k|k} + K_{k+1}(y_{k+1} - CA\hat{x}_{k|k})$ $\quad$ $K_{k+1} = P_{k+1|k}C^T\left(CP_{k+1|k}C^T + R\right)^{-1}$

```
    x_prev = x_1; %slides: x (pos+vel+acc) at time K given measurements until (K-1)
    P_prev = P_0; %slides: P at time K given measurements until (K-1)
```

$$\begin{aligned} \hat{x}_{0|-1} &= \bar{x}_0 \\ P_{0|-1} &= P_0 \end{aligned}$$

```
    filter_pos = zeros(size(M_pos));
    filter_vel = zeros(size(M_pos));

    if size(x_0,1) == 3 %acceleration estimation is required!
        filter_acc = zeros(size(M_pos));
    else
        filter_acc = [];
    end

    for k=1:(size(M_pos,1)-1)
        %Retrieve the "next" value of measured position
        y_next = M_pos(k+1); %slides: y at time (K+1)

        if nargin == 9 %steady state Kalman Filter
```

$$\begin{aligned} \hat{x}_{k+1|k+1} &= A\hat{x}_{k|k} + K_\infty(y_{k+1} - CA\hat{x}_{k|k}) \\ P &= APA^T - APC^T(CPC^T + R)^{-1}CPA^T + Q \end{aligned}$$

```
            x_next = A*x_prev + K_inf*(y_next - C*A*x_prev);
            x_prev = x_next; %update the previous estimation with the current one
        else
            %Compute the "next" value of P (slides: P at time K+1 given measurements until K)
```

$$P_{k+1|k} = AP_{k|k-1}A^T - AP_{k|k-1}C^T(CP_{k|k-1}C^T + R)^{-1}CP_{k|k-1}A^T + Q$$

```
            P_next = A*P_prev*A.' - A*P_prev*C.' * inv(C*P_prev*C.' + R) * C*P_prev*A.' + Q;
            P_prev = P_next; %update the previous P with the current one
            %Compute the "next" value of Kalman Gain (slides: K.Gain at time (K+1))
```

$$K_{k+1} = P_{k+1|k}C^T\left(CP_{k+1|k}C^T + R\right)^{-1}$$

```
            K_next = P_next * C.' * inv(C*P_next*C.' + R);

            %Estimate the "next" value of x (slides: x at time (K+1) given measurements (K+1))
```

$$\hat{x}_{k+1|k+1} = A\hat{x}_{k|k} + K_{k+1}(y_{k+1} - CA\hat{x}_{k|k})$$

```
            x_next = A*x_prev + K_next*(y_next - C*A*x_prev);
            x_prev = x_next; %update the previous estimation with the current one
        end

        %Add the estimated position, velocity and acceleration to the appropriate arrays
        filter_pos(k) = x_next(1); filter_vel(k) = x_next(2);
        if size(x_0,1) == 3 %acceleration estimation is required!
            filter_acc(k) = x_next(3);
        end
    end

    P_inf = P_prev; %return the limit for k->inf. of P(K|K-1)

end
```

My MATLAB Kalman Predictor code snippet with explanations:

```
function [predictor_pos, predictor_vel, predictor_acc, P_inf] =
                                        kalmanPredictor(A,B,C,M_pos,R,Q,x_0,P_0,Kbar_inf)
    Kbar_0 = P_0*C.' * inv(C*P_0*C.' + R);      %initial Kalman G ain: Kbar_{0}
    x_1 = A*x_0 + Kbar_0 *(M_pos(1) - C*A*x_0); %initial optimal filter: xhat_{0+1|0}
```

$$\hat{x}_{k+1|k} = A\hat{x}_{k|k-1} + \bar{K}_k(y_k - C\hat{x}_{k|k-1}) \qquad \bar{K}_k = AP_{k|k-1}C^T\left(CP_{k|k-1}C^T + R\right)^{-1}$$

```
    x_prev = x_1; %slides: x (pos+vel+acc) at time K given measurements until (K-1)
    P_prev = P_0; %slides: P at time K given measurements until (K-1)

    predictor_pos = zeros(size(M_pos));
    predictor_vel = zeros(size(M_pos));
    if size(x_0,1) == 3 %acceleration estimation is required!
        predictor_acc = zeros(size(M_pos));
    else
        predictor_acc = [];
    end

    for k=1:(size(M_pos,1)-1)
        %Retrieve the "next" value of measured position
        y_actual = M_pos(k); %slides: y at time (K+1)

        if nargin == 9 %steady state Kalman Predictor
            %Estimate the "next" value of x (slides: x at time (K+1) given measurements until K)
```

$$\begin{aligned}\hat{x}_{k+1|k} &= A\hat{x}_{k|k-1} + \bar{K}_\infty(y_k - C\hat{x}_{k|k-1})\\ P &= APA^T - APC^T(CPC^T + R)^{-1}CPA^T + Q\end{aligned}$$

```
            x_next = A*x_prev + Kbar_inf*(y_actual - C*x_prev);
            x_prev = x_next; %update the previous estimation with the current one
        else
            %Compute the "next" value of P (slides: P at time K given measurements until (K+1))
```

$$P_{k+1|k} = AP_{k|k-1}A^T - AP_{k|k-1}C^T(CP_{k|k-1}C^T + R)^{-1}CP_{k|k-1}A^T + Q$$

```
            P_next = A*P_prev*A.' - A*P_prev*C.' * inv(C*P_prev*C.' + R) * C*P_prev*A.' + Q;

            %Compute the "previous" value of Kalman Gain (slides: K.Gain at time K)
```

$$\bar{K}_k = AP_{k|k-1}C^T\left(CP_{k|k-1}C^T + R\right)^{-1}$$

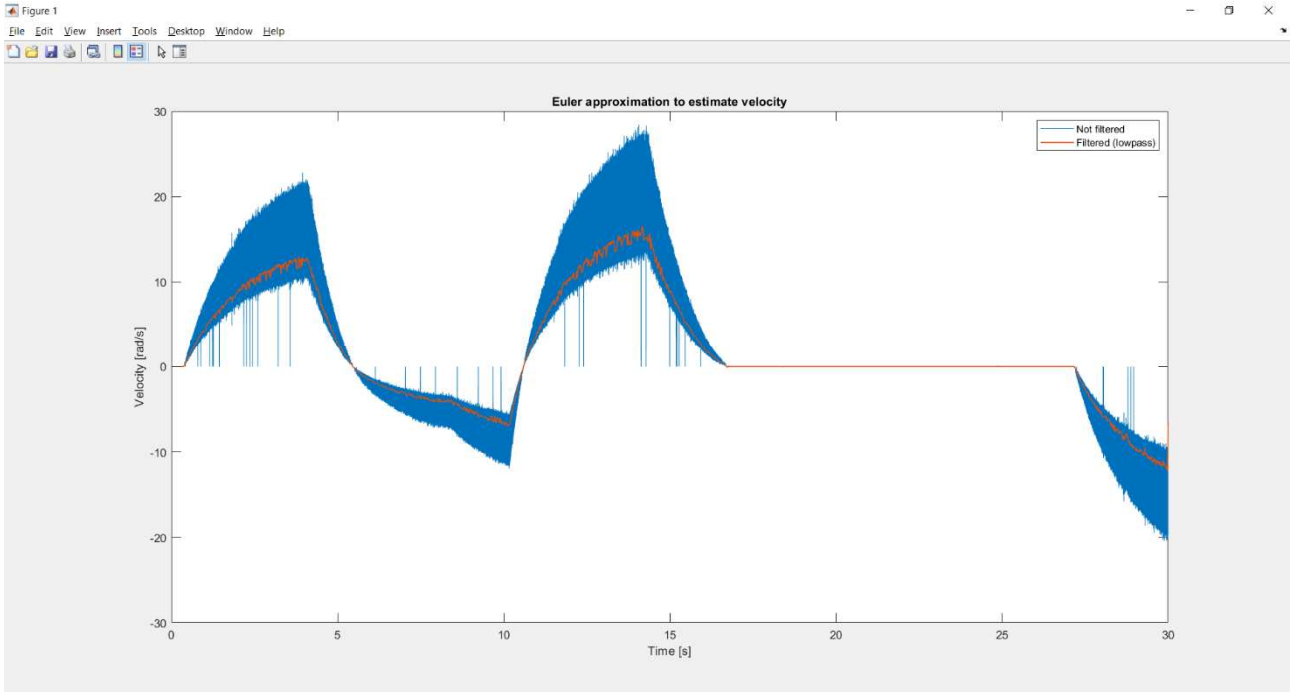
```
            Kbar_actual = A*P_prev * C.' * inv(C*P_prev*C.' + R);
            P_prev = P_next; %update the previous P with the current one

            %Estimate the "next" value of x (slides: x at time (K+1) given measurements until K)
```

$$\hat{x}_{k+1|k} = A\hat{x}_{k|k-1} + \bar{K}_k(y_k - C\hat{x}_{k|k-1})$$

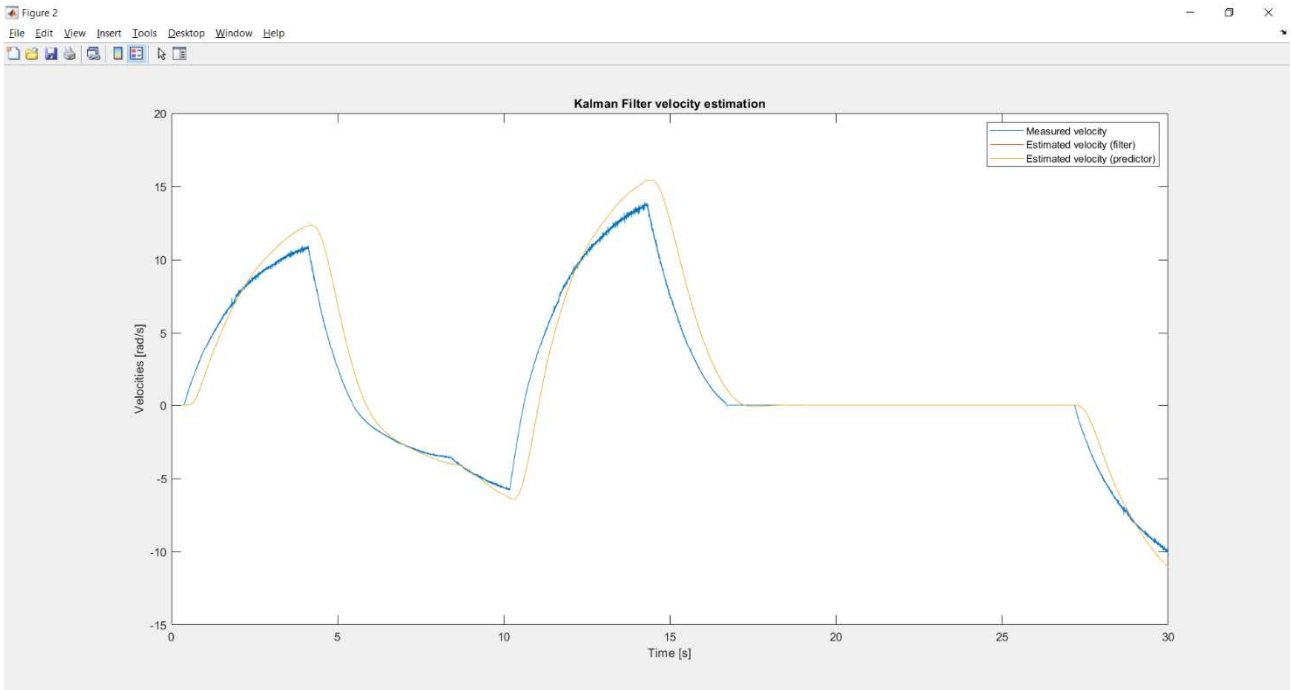
```
            x_next = A*x_prev + Kbar_actual*(y_actual - C*x_prev);
```

```
            x_prev = x_next; %update the previous estimation with the current one
        end

        %Add the estimated position, velocity and acceleration to the appropriate arrays
        predictor_pos(k) = x_next(1); predictor_vel(k) = x_next(2);
        if size(x_0,1) == 3 %acceleration estimation is required!
            predictor_acc(k) = x_next(3);
        end
    end
    P_inf = P_prev;

End
```

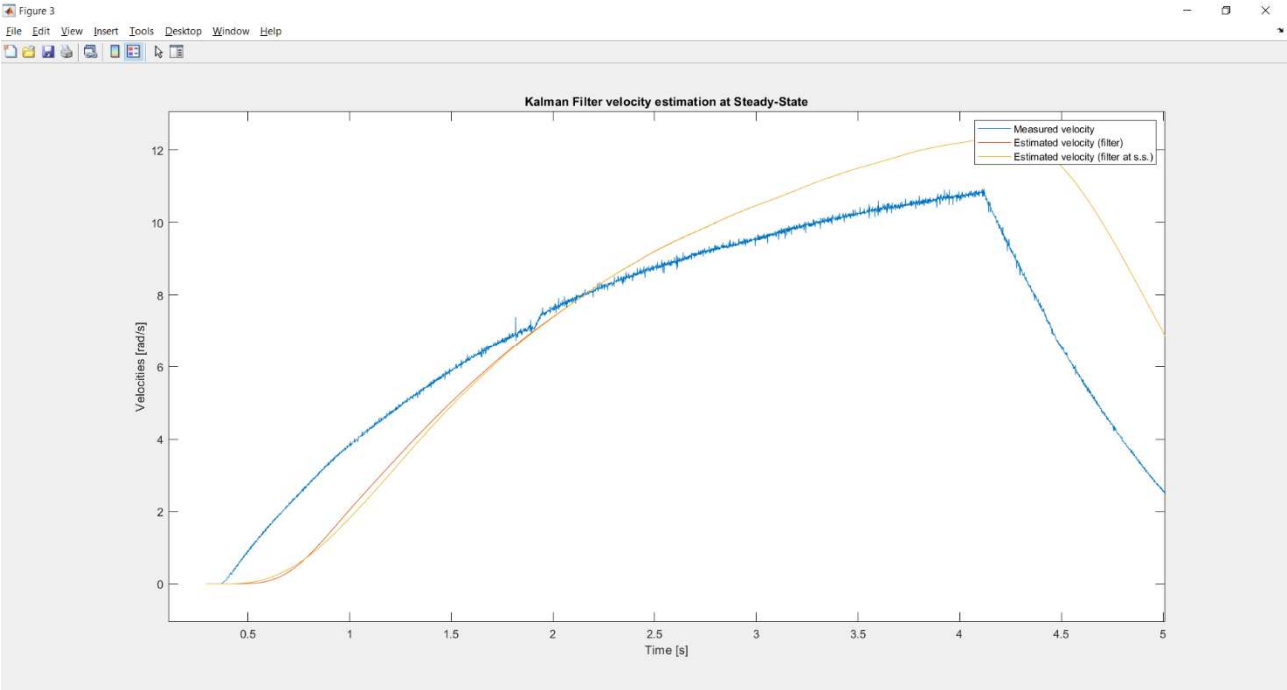
Velocity estimation using Euler method (also applying a 5Hz lowpass filter):



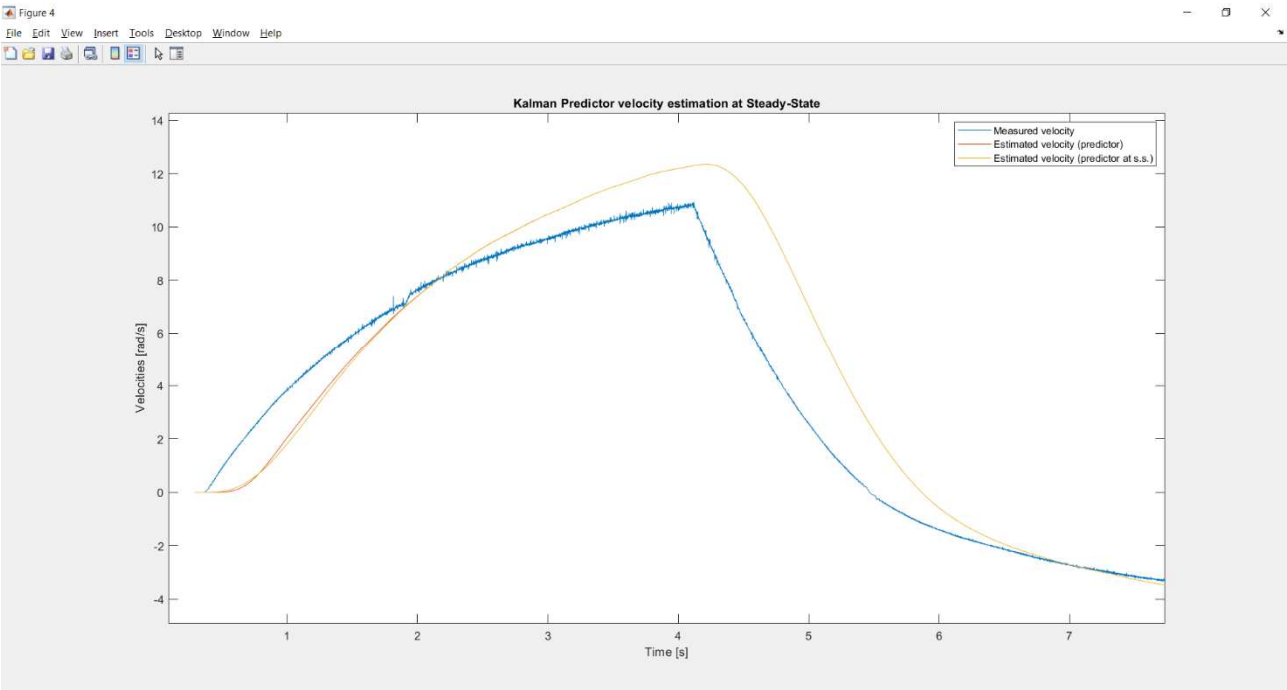
Kalman Filter vs Predictor velocity estimation:



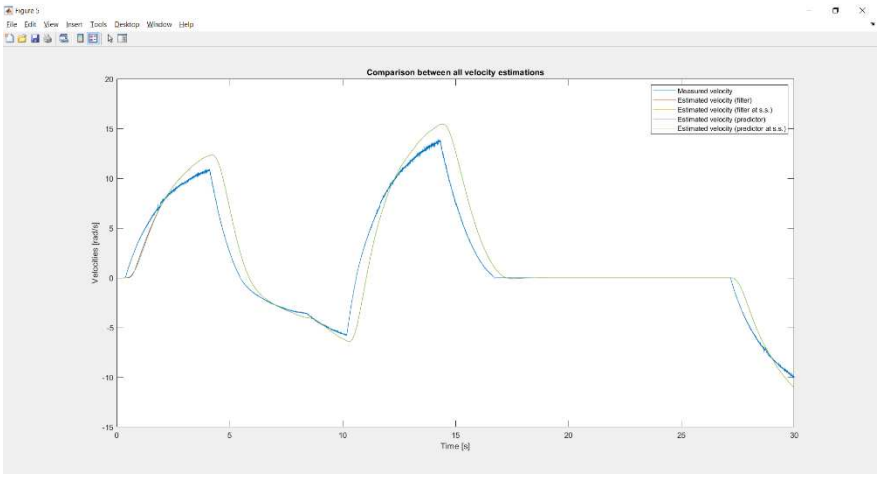
Kalman Filter vs Kalman Filter at steady-state (initial section):

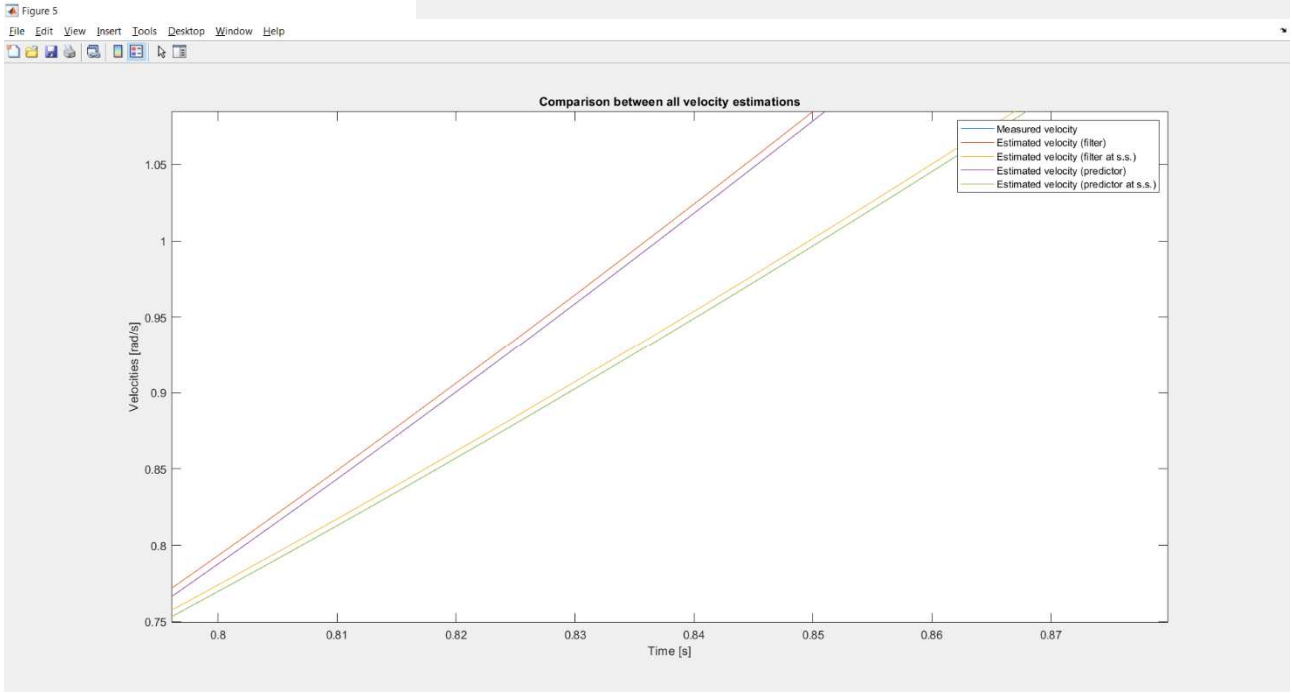


Kalman Predictor vs Kalman Predictor at steady-state:

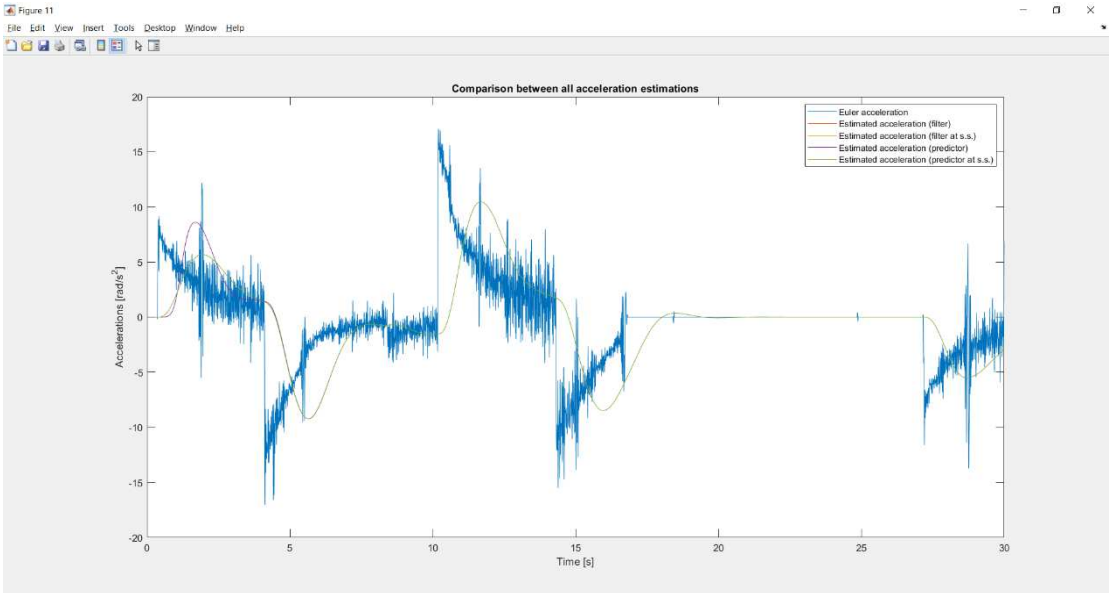


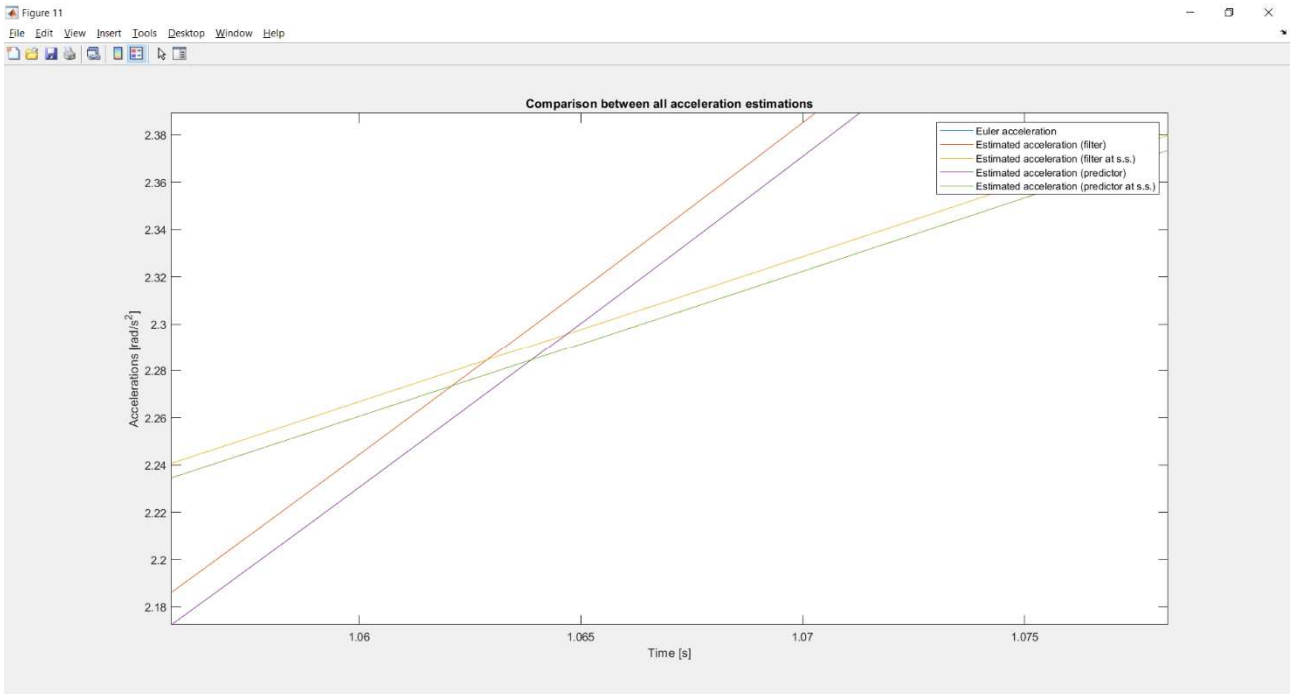
Overall comparison:



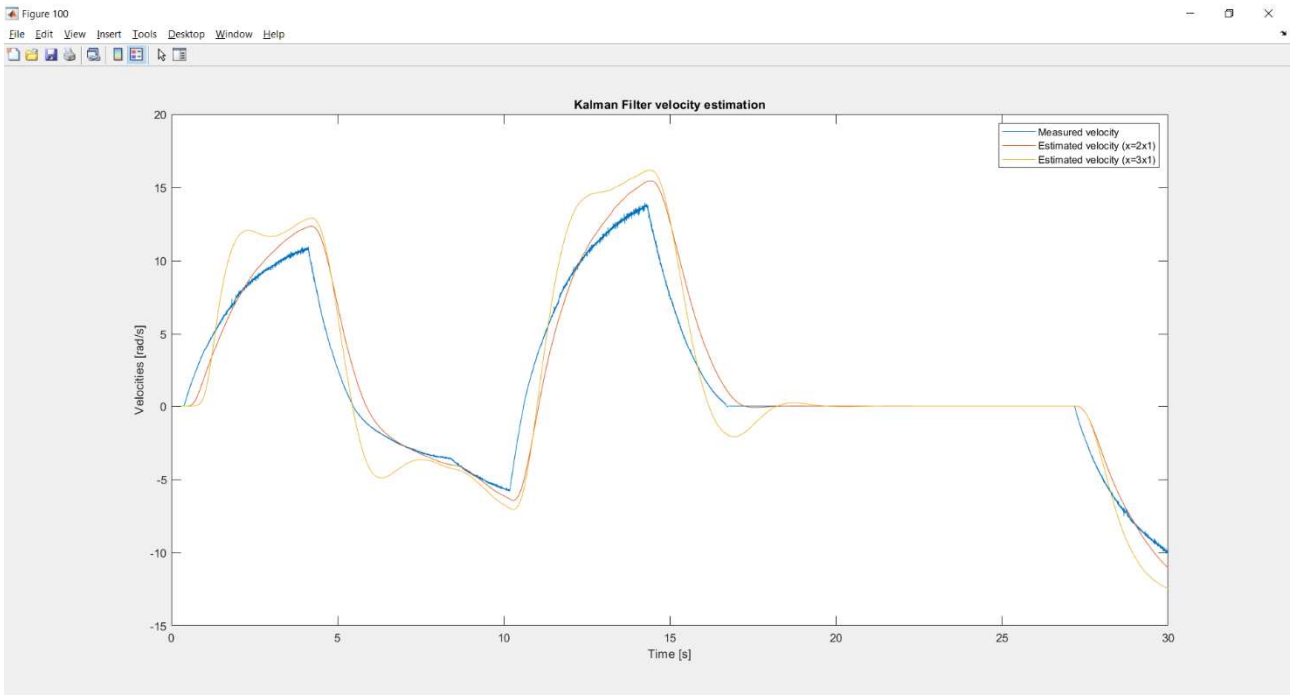


In a very similar way, but extended in terms of system's dimensions, acceleration is estimated starting from noisy position measurements. In the following image the overall comparison is reported.





Also, in this final plot, we can see a *comparison between velocity estimations*. In fact, when estimating the acceleration, velocity is also estimated but in a less precise way (because the extended state-space representation will include both of them).



Assignment 4

- **Implement the Kalman Smoother to estimate velocity and acceleration from noisy position measurements**

The Kalman Smoother algorithm is composed by two steps:

- ***Forward step***: Kalman filtering

$$\begin{aligned}
\hat{x}^f_{k+1|k+1} &= A\hat{x}^f_{k|k} + K_{k+1}(y_{k+1} - CA\hat{x}^f_{k|k}) \\
\hat{x}^f_{0|0} &= \bar{x}_0 \\
P^f_{k|k} &= \dots \\
P^f_{k+1|k} &= \dots \\
P^f_{0|0} &= P_0
\end{aligned}$$

- ***Backward step***: Smoothing

$$\hat{x}^s_{k|N} = \hat{x}^f_{k|k} + \check{K}_k \left[\hat{x}^s_{k+1|N} - \hat{x}^f_{k+1|t}\right]$$
$$\hat{x}^s_{N|N} = \hat{x}^f_{N|N}$$

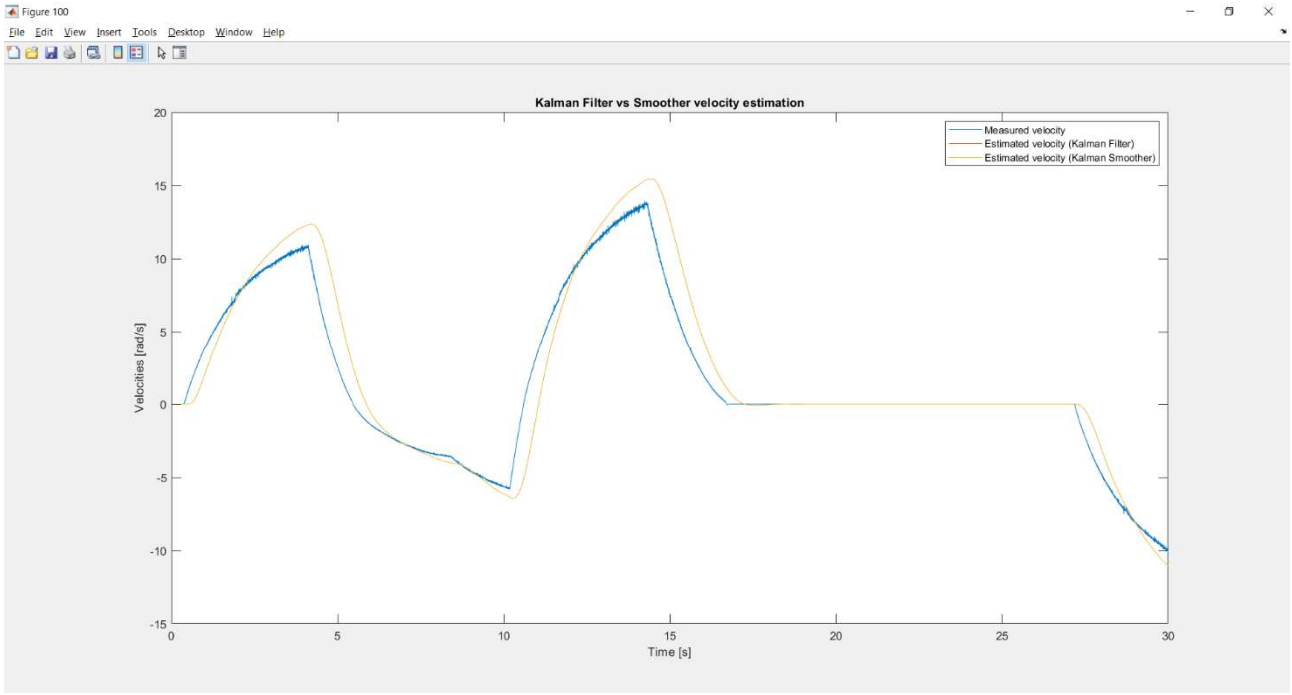
$$P_{k|N} = P^f_{k|k} + \check{K}_k \left[P_{k+1|N} - P^f_{k+1|k}\right]$$
$$P_{N|N} = P^f_{N|N}.$$

$$\check{K}_k = P^f_{k|k} A^T \left(P^f_{k+1|k}\right)^{-1}$$

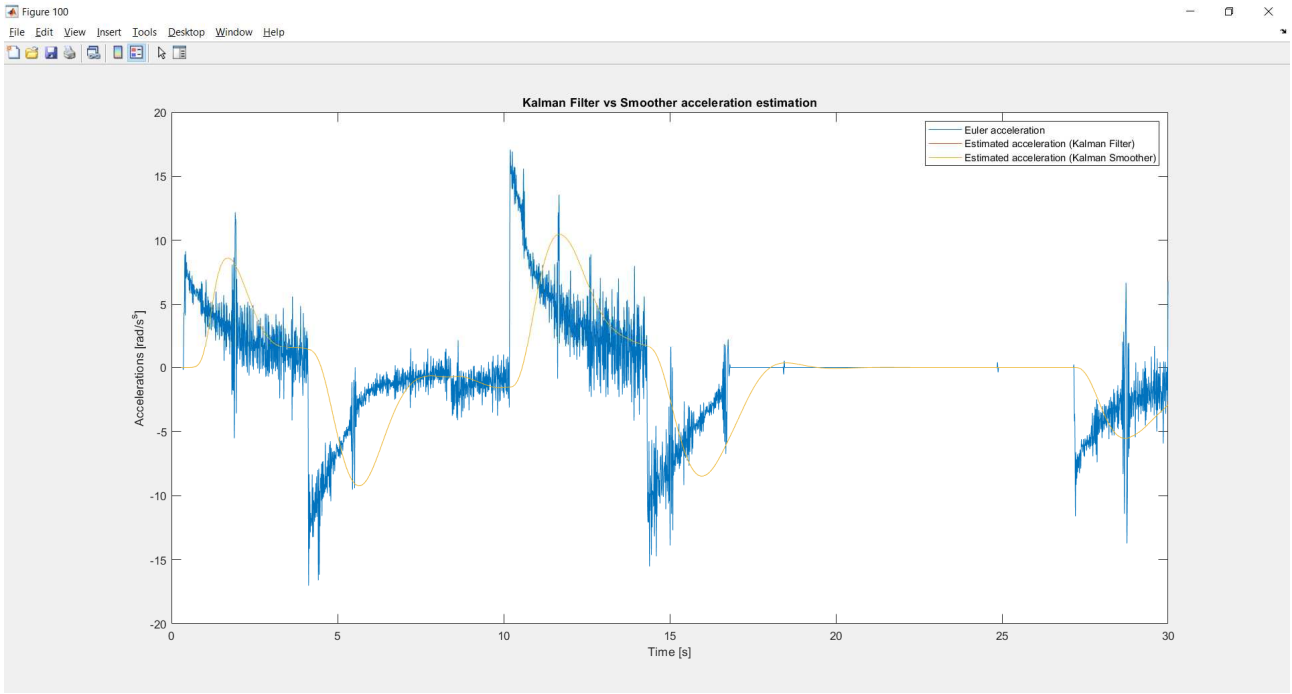
```
function [smoother_pos, smoother_vel, smoother_acc] = kalmanSmoother(A,B,C,M_pos,R,Q,x_0,P_0)

    %%- Kalman Filter (forward step) -%%
    K_1 = P_0*C.' * inv(C*P_0*C.' + R);  %initial Kalman G ain: K_{0+1}
    x_1 = A*x_0 + K_1 *(M_pos(1) - C*A*x_0); %initial optimal filter: xhat_{0+1|0+1}


    x_prev = x_1; %slides: x (pos+vel+acc) at time K given measurements until (K-1)
    P_prev = P_0; %slides: P at time K given measurements until (K-1)

    filter_pos = zeros(size(M_pos));
    filter_vel = zeros(size(M_pos));

    if size(x_0,1) == 3 %acceleration estimation is required!
        filter_acc = zeros(size(M_pos));
    else
        filter_acc = [];
    end

    for k=1:(size(M_pos,1)-1)
        %Retrieve the "next" value of measured position
        y_next = M_pos(k+1); %slides: y at time (K+1)

        %Compute P_{K|K} --> required for backward
        P_{k+1|k+1} = P_{k+1|k} - P_{k+1|k} C^T (C P_{k+1|k} C^T + R)^{-1} C P_{k+1|k}   Instead of k+1|k+1 I substituted k|k
        P_kk_all(:,:,k) = P_prev - P_prev*C.'*inv(C*P_prev*C.' + R) * C*P_prev;
```

```
        %Compute P_{K+1|K} --> required for backward
```
$$P_{k+1|k} = AP_{k|k}A^T + Q$$
```
        P_kplus1_k_all(:,:,k) = A * P_kk_all(:,:,k) * A.' + Q;

        %Compute the "next" value of P (slides: P at time K given measurements until (K+1))
```
$$P_{k+1|k} = AP_{k|k-1}A^T - AP_{k|k-1}C^T(CP_{k|k-1}C^T + R)^{-1}CP_{k|k-1}A^T + Q$$
```
        P_next = A*P_prev*A.' - A*P_prev*C.' * inv(C*P_prev*C.' + R) * C*P_prev*A.' + Q;
        P_prev = P_next; %update the previous P with the current one

        %Compute the "next" value of Kalman Gain (slides: K.Gain at time (K+1))
```
$$K_{k+1} = P_{k+1|k}C^T\left(CP_{k+1|k}C^T + R\right)^{-1}$$
```
        K_next = P_next * C.' * inv(C*P_next*C.' + R);

        %Estimate the "next" value of x (slides: x at time (K+1) given measurements (K+1))
```
$$\hat{x}_{k+1|k} = A\hat{x}_{k|k-1} + \bar{K}_k(y_k - C\hat{x}_{k|k-1})$$
```
        x_next = A*x_prev + K_next*(y_next - C*A*x_prev);
        x_prev = x_next; %update the previous estimation with the current one


        %Add the estimated position, velocity and acceleration to the appropriate arrays
        filter_pos(k) = x_next(1); filter_vel(k) = x_next(2);
        if size(x_0,1) == 3 %acceleration estimation is required!
            filter_acc(k) = x_next(3);
        end
    end


    %%- Smoothing (backward step) --%
    if size(x_0,1) == 3 %acceleration estimation is required!
        xf = [filter_pos.';
              filter_vel.';
              filter_acc.'];
        xf(:,end) = [];
    else
        xf = [filter_pos.';
              filter_vel.'];
        xf(:,end) = [];
    end

    xs = zeros(size(xf));
    xs(:,end) = xf(:,end);
```
$\hat{x}^s_{N|N} = \hat{x}^f_{N|N}$
```

    for k=(size(xf,2)-1):-1:1
```
$$\check{K}_k = P^f_{k|k}A^T\left(P^f_{k+1|k}\right)^{-1}$$
```
        Kcurve_k = P_kk_all(:,:,k) * A.' * inv(P_kplus1_k_all(:,:,k));
```
$$\hat{x}^s_{k|N} = \hat{x}^f_{k|k} + \check{K}_k\left[\hat{x}^s_{k+1|N} - \hat{x}^f_{k+1|t}\right]$$
```
        xs(:,k) = xf(:,k) + Kcurve_k*(xs(:,k+1) - xf(:,k+1));
    end

    smoother_pos = xs(1,:).';
    smoother_vel = xs(2,:).';
    if size(x_0,1) == 3 %acceleration estimation is required!
        smoother_acc = xs(3,:).';
    else
        smoother_acc = [];
    end
end
```

Kalman Filter vs Kalman Smoother (velocity):



Kalman Filter vs Kalman Smoother (acceleration):



Assignment 5

- **DC motor parameters' identification**

Given voltage measurements of a DC motor, the goal is to use the previously estimated velocity and acceleration to compute the characteristic parameters of the DC motor model. In particular, they're given by the following differential equation:

$$\frac{\tau}{k}\dot{\omega}(t) + \frac{1}{k}\omega(t) = V(t) \qquad \begin{bmatrix}\dot{\omega}(t) & \omega(t)\end{bmatrix}\begin{bmatrix}\frac{\tau}{k} \\ \frac{1}{k}\end{bmatrix} = V(t)$$

Rewriting the above equation in a more compact way, we get X vector that contains angular acceleration and velocity estimates, Y vector that contains voltage measurements and ϑ that contains intertia (J) and damping (d) parameters.

$$\begin{aligned} x &\triangleq \begin{bmatrix}\dot{\omega} & \omega\end{bmatrix} \\ y &\triangleq V \qquad\qquad y = x\theta \\ \theta &\triangleq \begin{bmatrix}\frac{\tau}{k} \\ \frac{1}{k}\end{bmatrix} \end{aligned}$$

- **Parameters' identification through Least Square method**

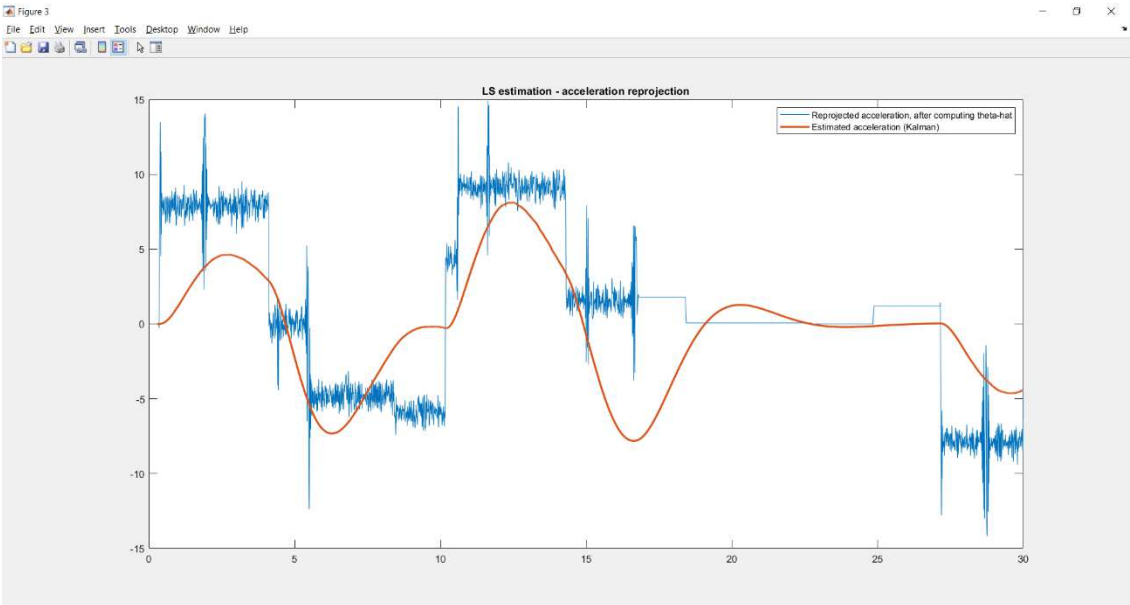
According to the theory, implementing the following equation we get a "one shot" estimation of the DC motor's parameters.

```
beta_hat = inv(X.'*X)*X.' * Y;
```

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

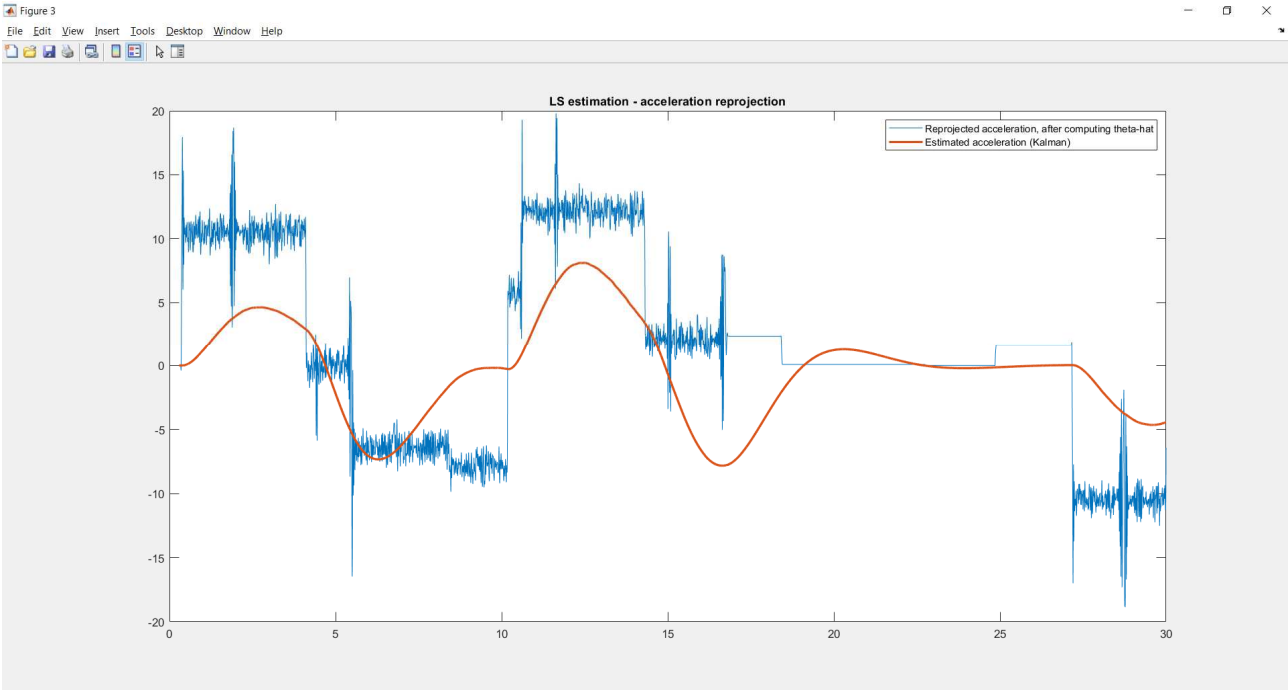
Results using the following X vector: `X = [filter3_acc, filter_vel];`

```
The estimated inertia is: 0.10511 kg*m^2
The estimated damping is: 0.06082 kg*m^2/s
```



Results using the following X vector: `X = [smoother_acc, smoother_vel];`

```
The estimated inertia is: 0.08313 kg*m^2
The estimated damping is: 0.04314 kg*m^2/s
```



- **Parameters' identification with Recursive Least Square method**

According to the theory, implementing the following equation we get a recursive expression for the estimates:

$$\begin{aligned}
\hat{\beta}(k) &= \hat{\beta}(k-1) + K(k)e(k) \\
K(k) &= P(k)x_k^T \\
e(k) &= y_k - x_k\hat{\beta}(k-1) \\
P(k) &= \frac{1}{\lambda}\left[P(k-1) - \frac{P(k-1)x_k^T x_k P(k-1)}{\lambda + x_k P(k-1)x_k^T}\right]
\end{aligned}$$

The implemented code snippet is also shown. During the implementation I had some doubts about the right expression of K(k), that according to some sources (*https://www.youtube.com/watch?v=dAyt1pGYtYA&ab_channel=IntanZaurahMatDarus*;*https://www.youtube.com/watch?v=qRBp7_7fifk&ab_channel=LasseEngboChristiansen*), should be different. Nevertheless, both expressions were implemented and there aren't big differences in the estimated values.

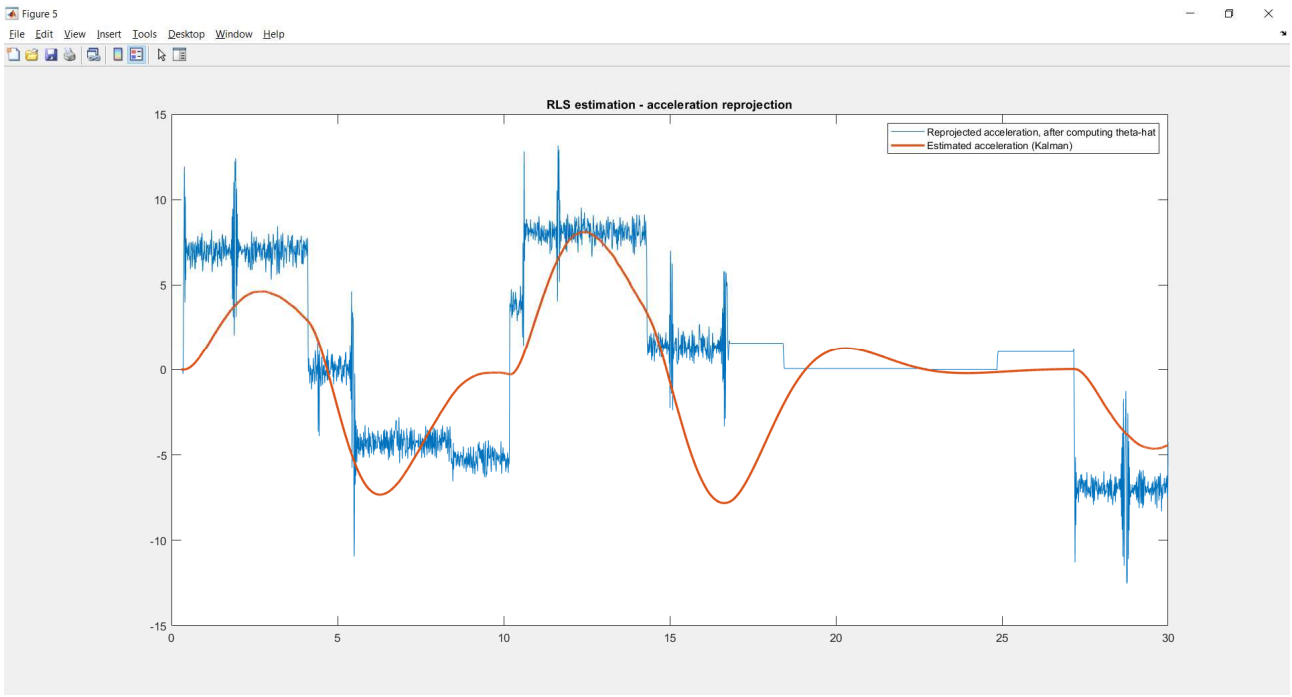
```
%Compute the prediction error
error(k) = Y(k) - Xk{k} * beta_hat{k-1};

%Compute current parameters' estimates
```

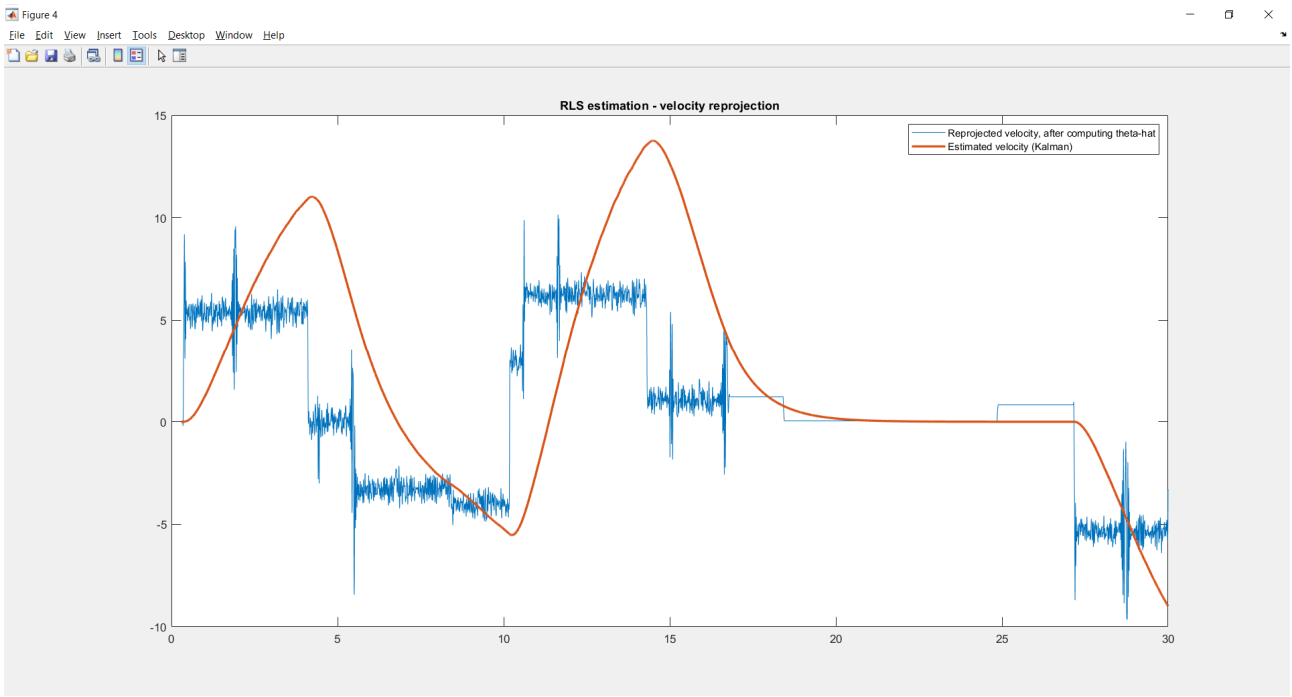
```
K = (P{k-1} * Xk{k}.') / (lambda + Xk{k}*P{k-1}*Xk{k}.'); %according to some sources
K = (P{k-1} * Xk{k}.'); %according to the slides


beta_hat{k} = beta_hat{k-1} + K*error(k,1);

P{k} = 1/lambda*( P{k-1} - (P{k-1} * Xk{k}.' * Xk{k} *P{k-1})/(lambda +
       Xk{k}*P{k-1}*Xk{k}.') );
```

To be noticed is that if the forgetting factor λ is lower than 1, wrong estimates (negative) for J are obtained.

Results using the following X vector: `X = [filter3_acc, filter_vel];`

```
The estimated inertia is: 0.10510 kg*m^2
The estimated damping is: 0.06086 kg*m^2/s
```





- **Parameters' identification with Adaptive algorithm (discrete time)**

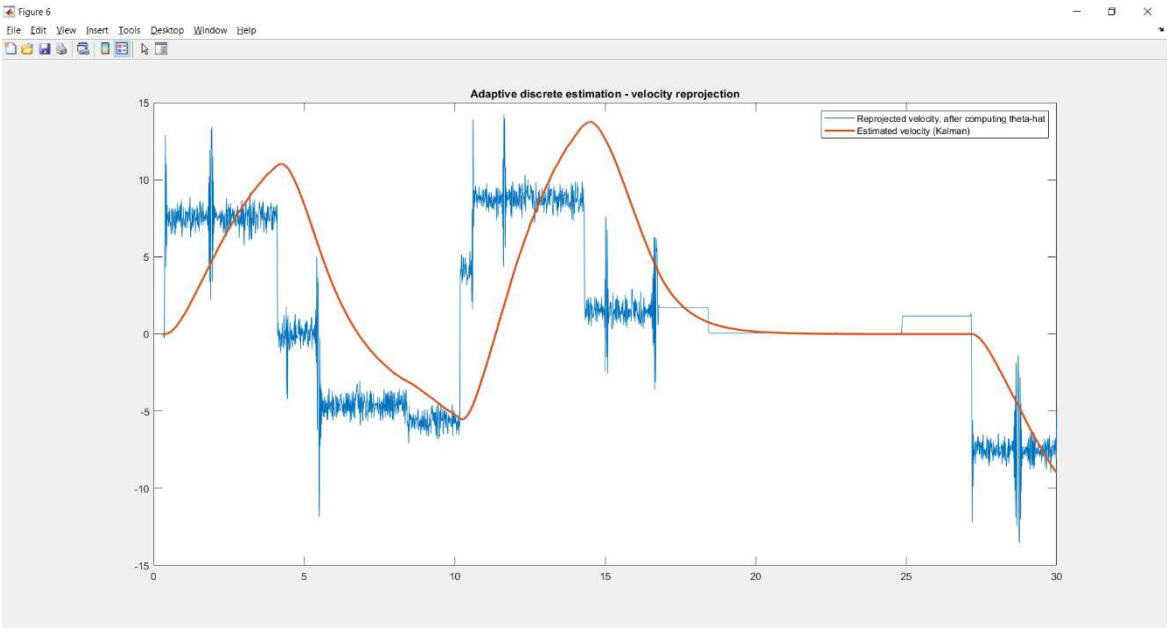
A discrete-time version of the *adaptive algorithm* was also implemented.

$$\dot{\hat{\beta}}(t) = gx^T(t)e(t)$$

$$\hat{\beta}(k) = \hat{\beta}(k-1) + T_s \cdot g \cdot x^t(k)e(\mathrm{k})$$

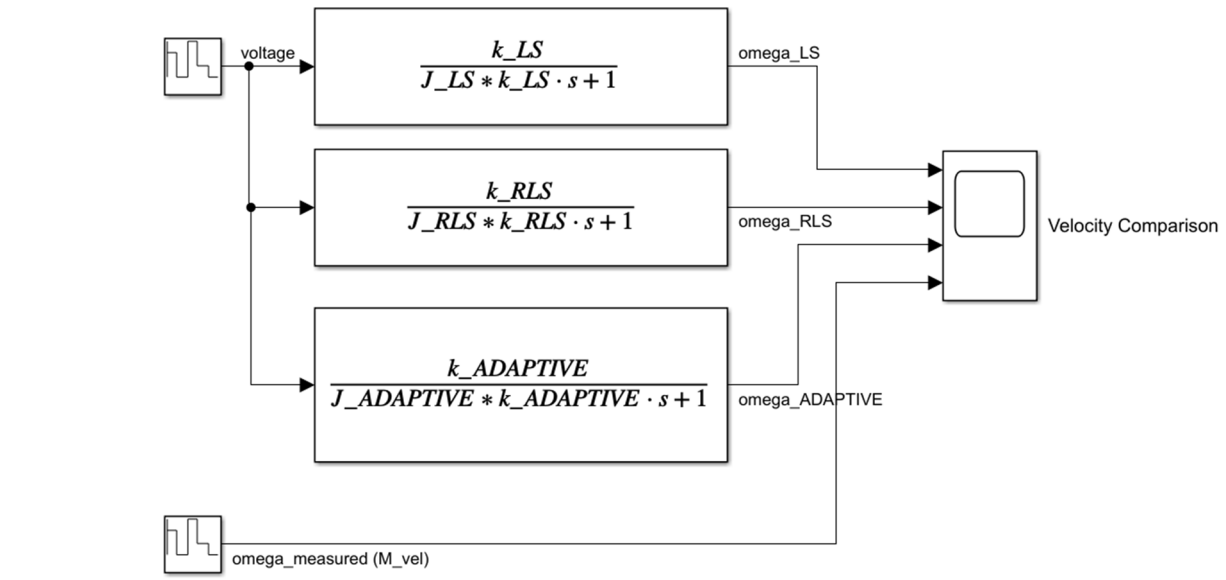
```
Ts = time(2) - time(1);
g = 0.001; %>0 but how much???

e(1) = Y(1) - Xk{1} * [0; 0];
beta_hat{1} = Ts * g * Xk{1}.' * e(1);

for k=2:size(X,1)
    error(k) = Y(k) - Xk{k} * beta_hat{k-1};

    beta_hat{k} = beta_hat{k-1} + g * Ts * Xk{k}.' * error(k);
end
```

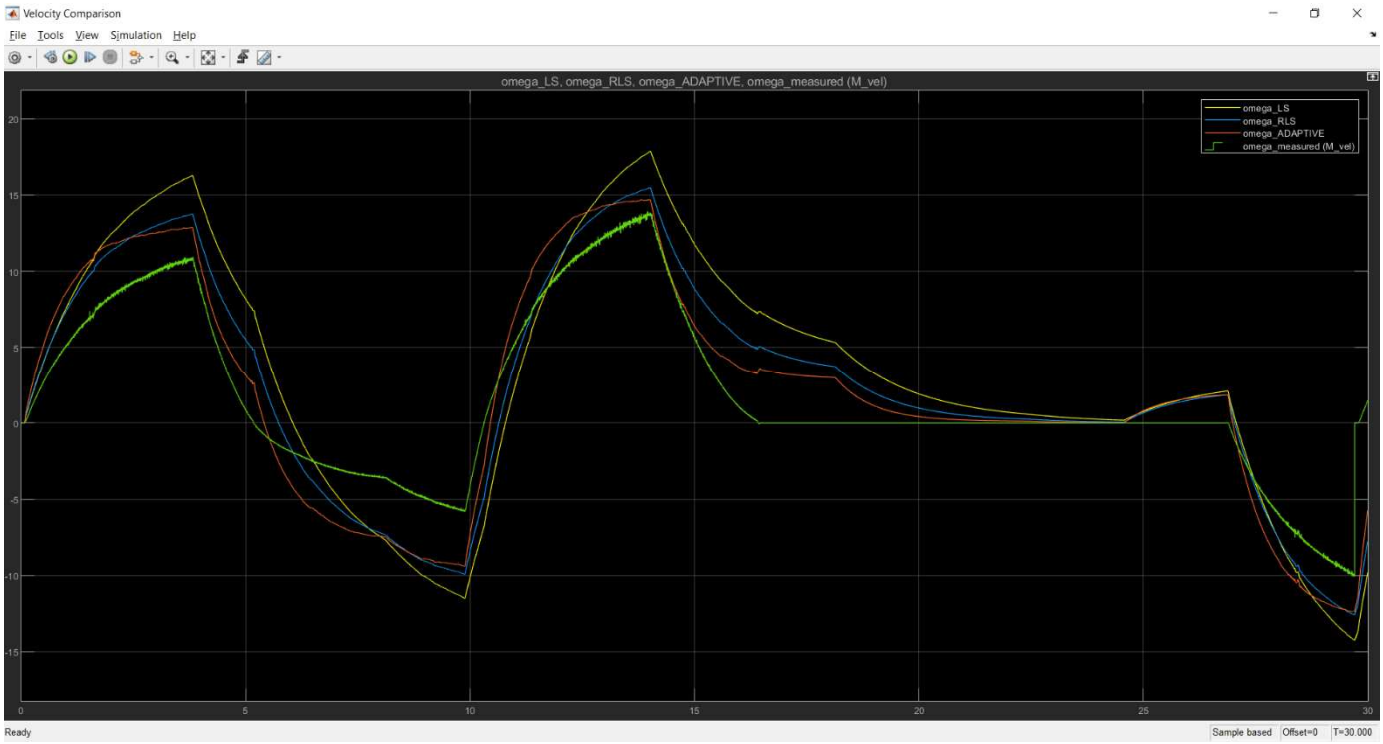
Trying several values of g, only small ones lead to a positive estimated inertia. I think that isn't so meaningful to report estimates because they strongly depend on the chosen parameter g.



- **Parameters' identification: testing**

To understand if the estimated values that compose the DC motor's transfer function are realistic, a Simulink model was developed. The idea is to provide the model, the same input voltages that were supplied to the real motor. Then, the output angular velocities are compared between the 3 model and the measured master velocity.





The above results are achieved using the following parameters:
J_LS=0.1051; k_LS=16.4413
J_RLS=0.0996; k_RLS=13.0205
J_ADAPTIVE=0.0723; k_ADAPTIVE=11.6297

- **Parameters' identification: conclusions**

To sum up, this estimation methods always depend on the input measurements and the velocity/acceleration estimations (through Kalman theory). So, there's always a strong dependence on the amount and quality of the input data. Also, RLS and Adaptive algorithms require some additional parameters that determine the final results, making the estimation *not so reliable*, in my opinion.

Assignment 6

- **Wave scattering representation**

By definition, u and y are *power variables* while w and z are *wave variables*.

$$(u, y) = W \oplus Z$$

Each pair of vectors $(u, y) \in U \times Y$ can be written *in a unique way* as a pair of vectors $(w, z) \in W \oplus Z$.

- w is the projection along Z of the combined vector $(u, y) \in U \times Y$ on W
- z is the projection along W of the combined vector $(u, y) \in U \times Y$ on Z

From a dimensional point of view, (u,y) is a power and the vector w can be seen as an *incoming wave vector*, with half times its norm being the incoming power (positive sign). The vector z is the *outgoing wave vector*, with half times its norm being the outgoing power (negative sign).

$$w = \frac{1}{\sqrt{b}\sqrt{2}}(bu + y)$$
$$z = \frac{1}{\sqrt{b}\sqrt{2}}(-bu + y)$$

SISO LTI relationship between power and wave variables; the mapping impedance is a positive constant (b)

Wave scattering separates the total power flow into the power input (*input wave*) and power output (*output wave*). It is a way to avoid direct exchange of power variables between master and slave devices that generate virtual energy in the presence of time delays inside the communication channel. The final goal is to ensure ***passivity***, despite constant time delays. The scattering operator allows writing the passivity definition integral as a function of the wave variables.

$$\int_0^T y(s)u(s)ds \geq 0 \qquad \textit{Passivity condition using POWER variables}$$

$$\int_0^T w^2(s)ds \geq \int_0^T z^2(s)ds \qquad \textit{Passivity condition using WAVE variables}$$

- **Implement the Scattering-based bilateral teleoperation architecture for the Force-Position case**

Following the "*Stable Adaptive Teleoperation*" paper the goal is to implement a bilateral teleoperation scheme robust to transmission time-delays while retaining force feedbacks. In fact, communication delays affect the stability of

force-reflecting telerobotic systems. Using the *wave variables* formalism passivity is ensured and therefore stability of the overall system.

Wave transformations can be implemented as long as there's a power relationship between incoming and outcoming signals. In particular, for the ***force-position*** case we have master velocity and slave force as inputs while master force and slave velocity as outputs.

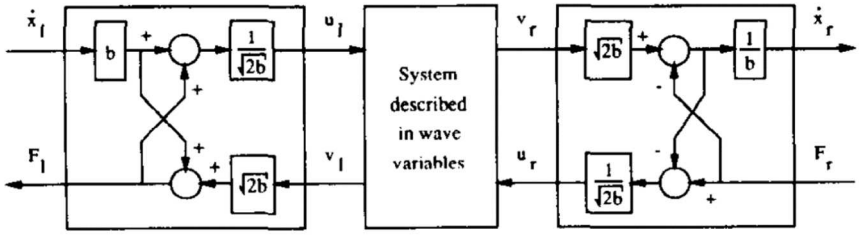
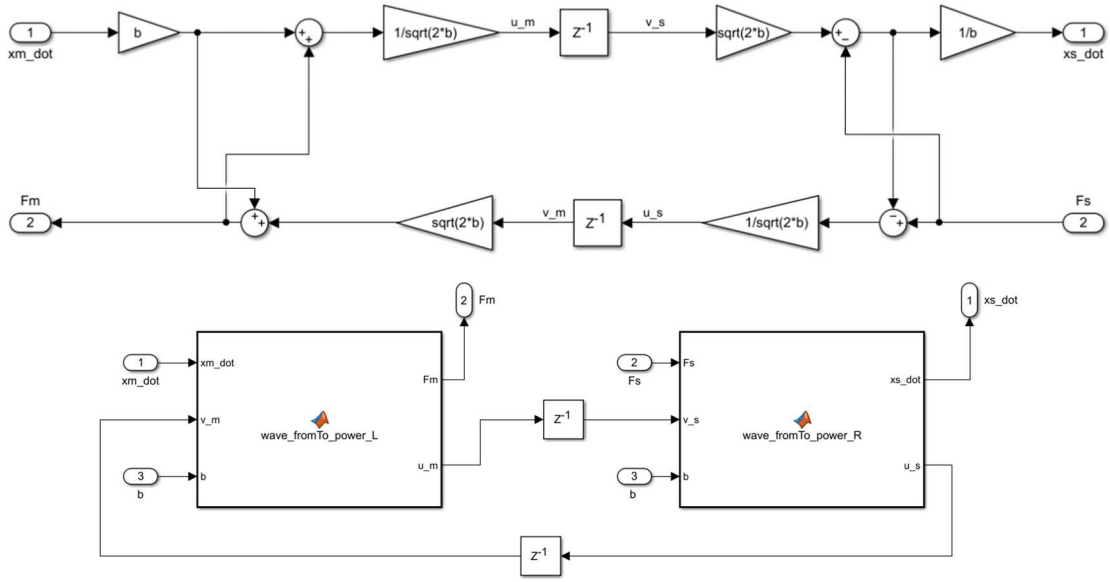


Fig. 4. Wave transformation using $\dot{x}_l$ and F_r as inputs.

My Simulink implementation (with blocks and with MATLAB functions):



Also, according to the paper, as in natural systems, ***wave reflections*** can be avoided if the impedance of the attaching subsystems is matched to the characteristic impedance b of the wave transmission.

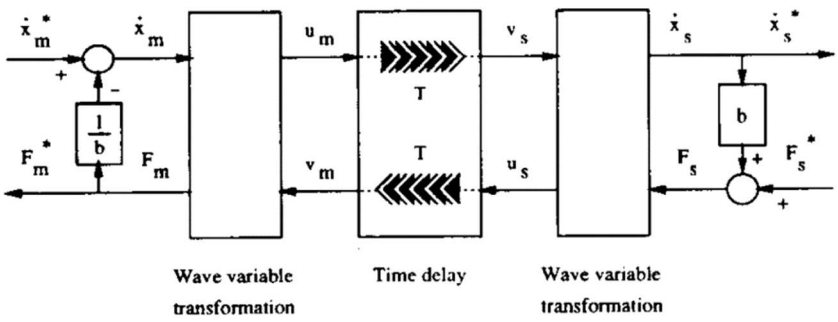
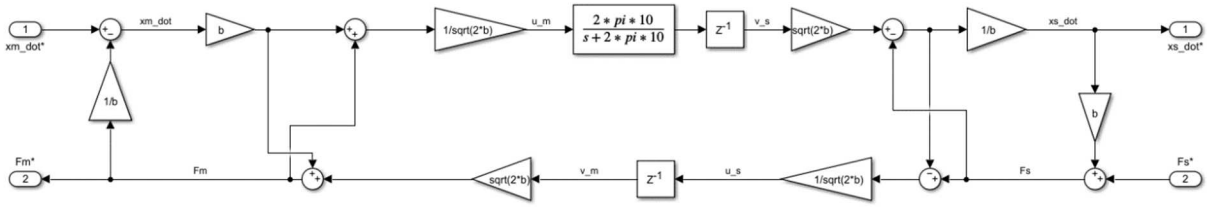
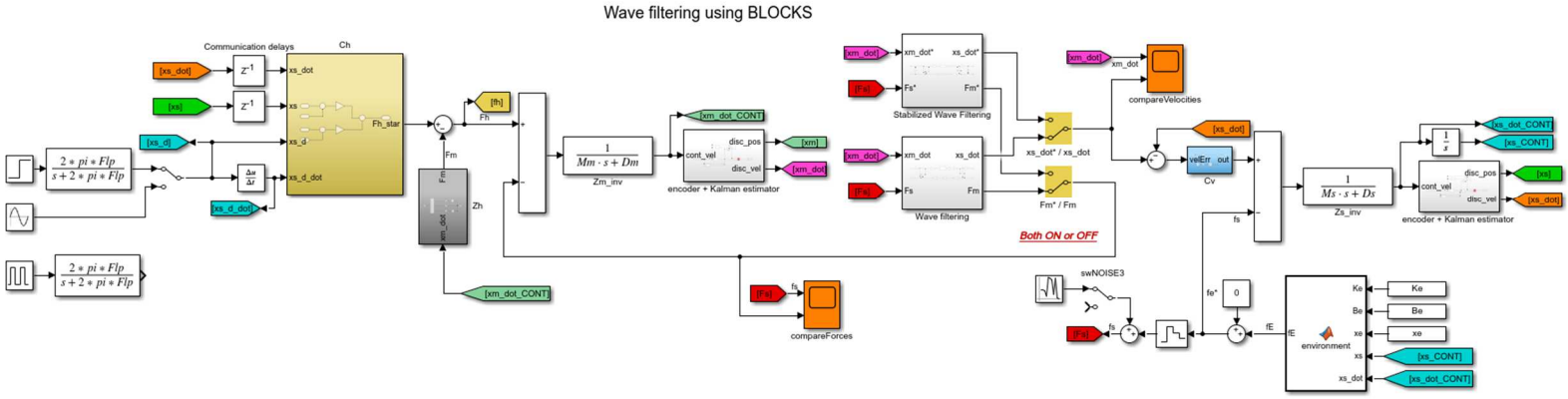


Fig. 8. Wave transmission with matched terminations.

My Simulink scheme of the wave transmission with impedance matching:



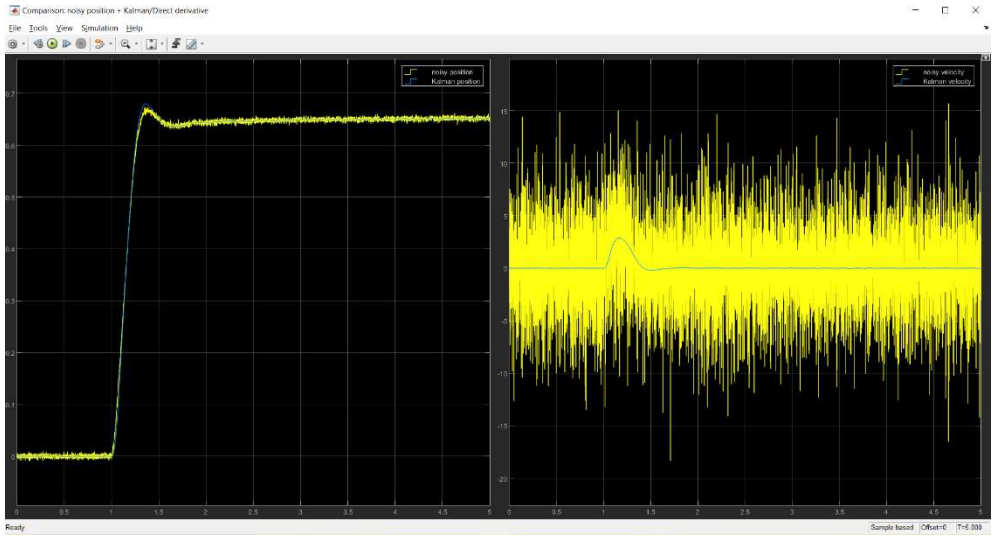
My Simulink Force-Position scheme (discrete-time):



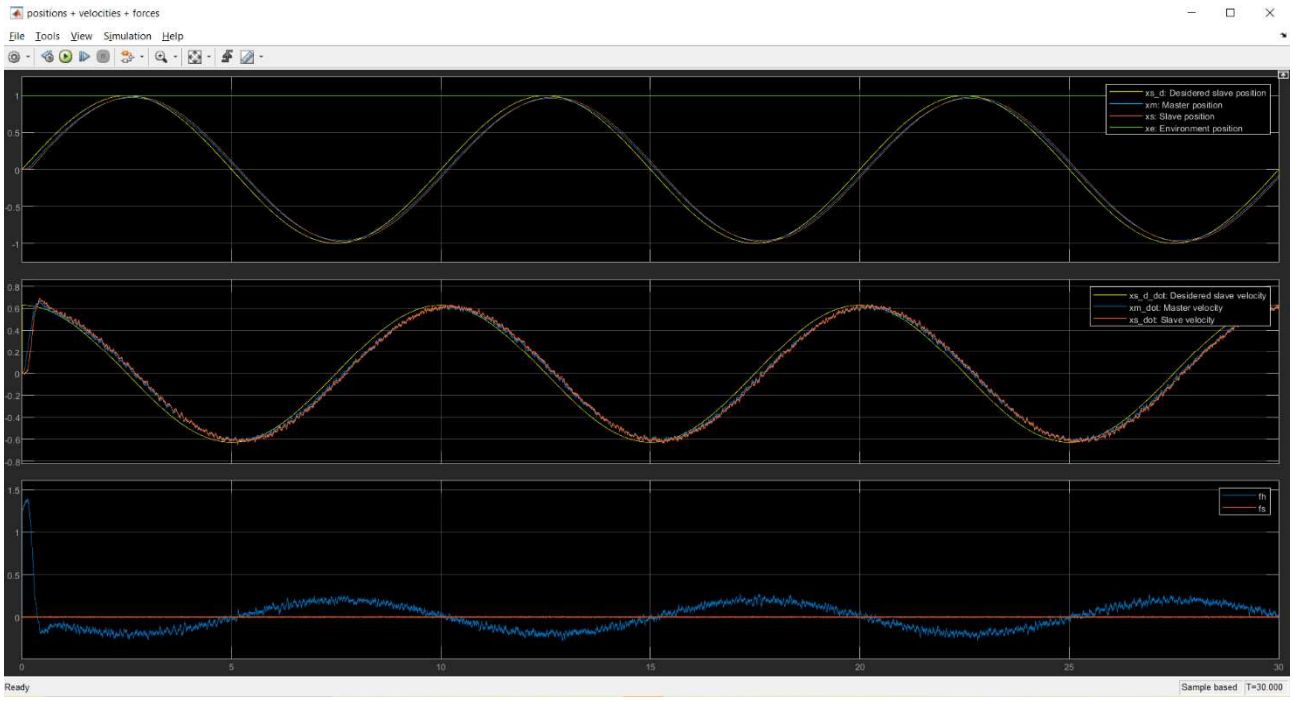
As can be seen from the above screenshot, my Simulink scheme has several possible working configurations, that can be chosen from the main MATLAB script. In particular, it always includes discretization where sensors should be placed, and it can be used:

- *without noise* on the measured positions and forces, in this case normal *discrete derivatives* are used to compute velocity from noise free position measurements;
- *with noise* on the measured positions and forces, in this case velocity is estimated using *Kalman filter* from noisy position measurements;
- there's also the ability to choose between standard wave filtering and stabilized wave filtering, but it would require a different controllers tuning;

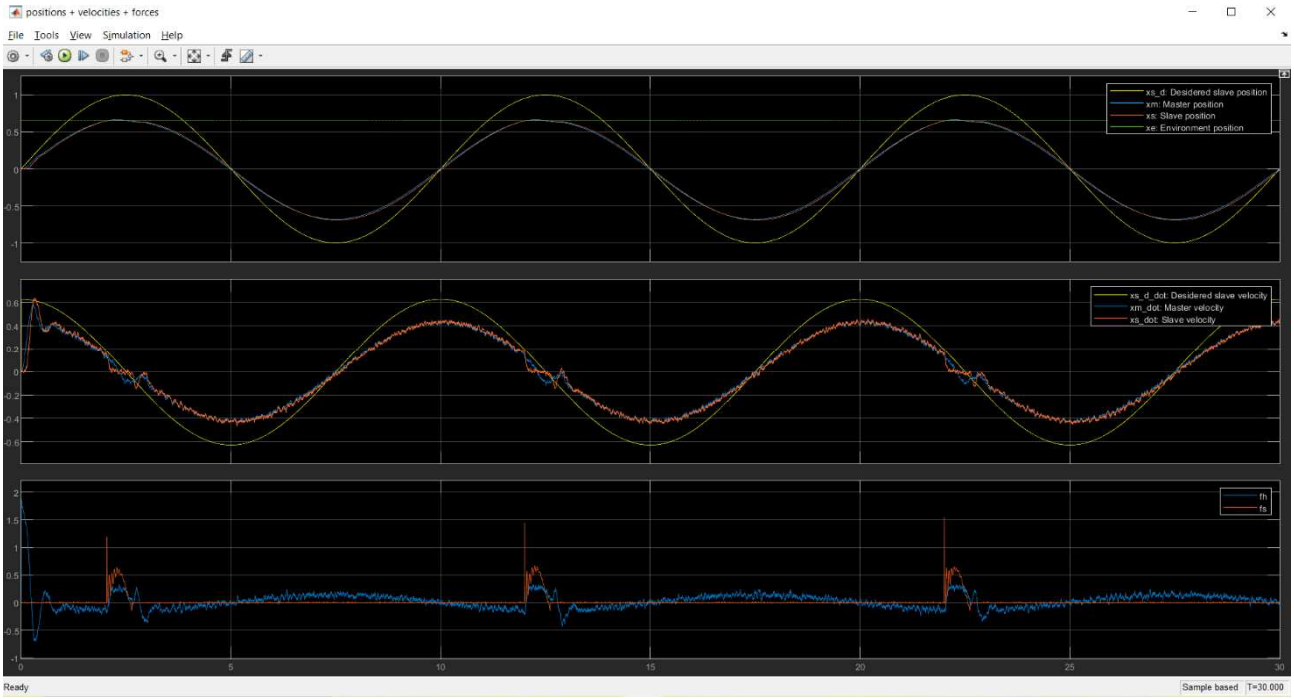
Position and velocity estimation from noisy position measurements:



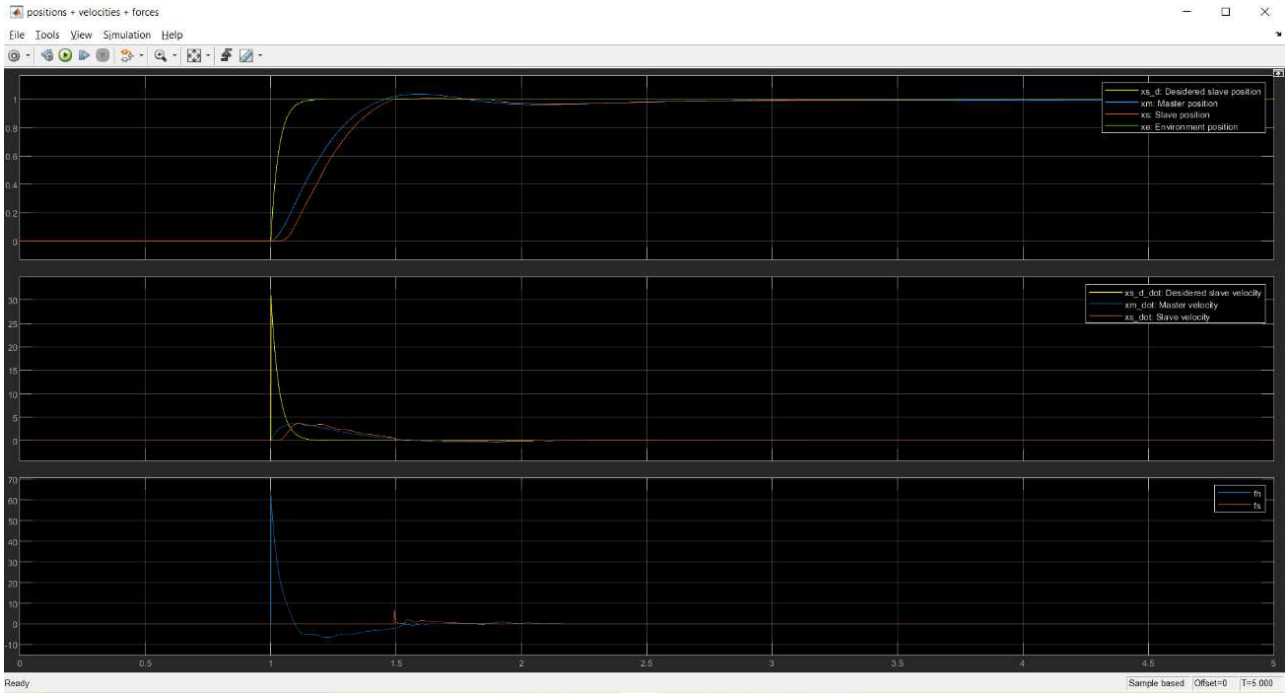
Sinusoidal reference signal results, delay=20ms, noise, free motion:



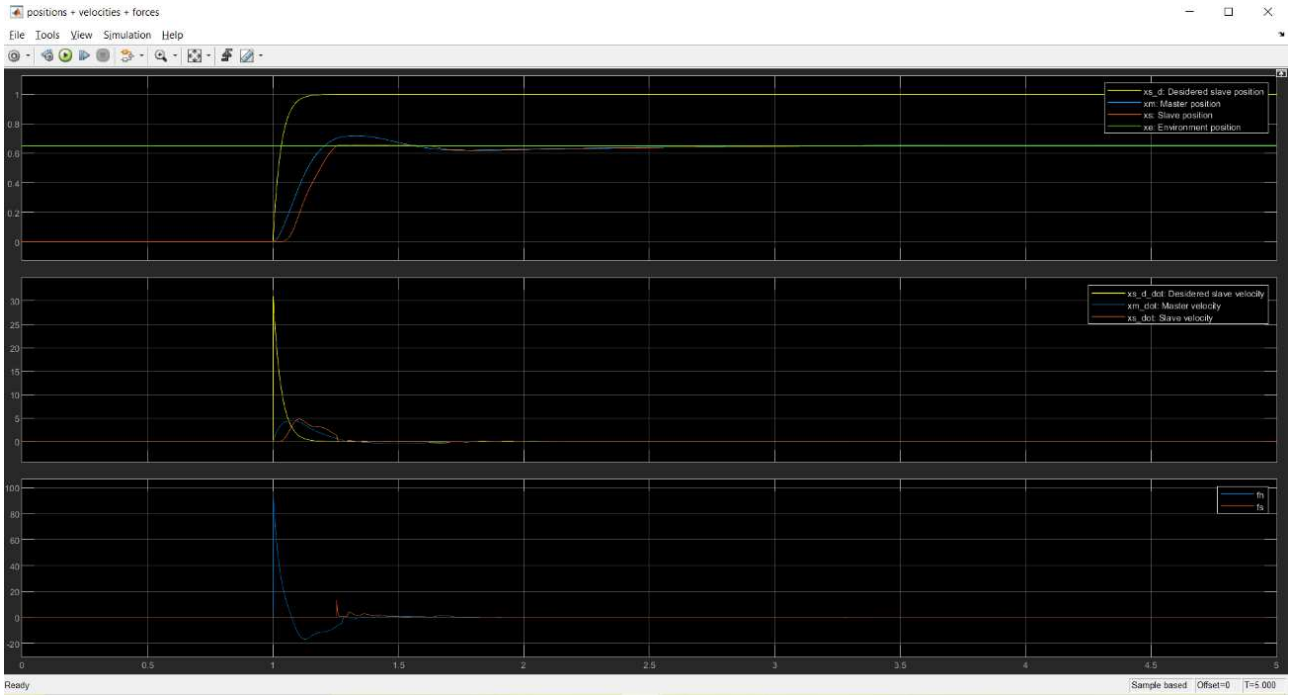
Sinusoidal reference signal results, delay=20ms, noise, contact:



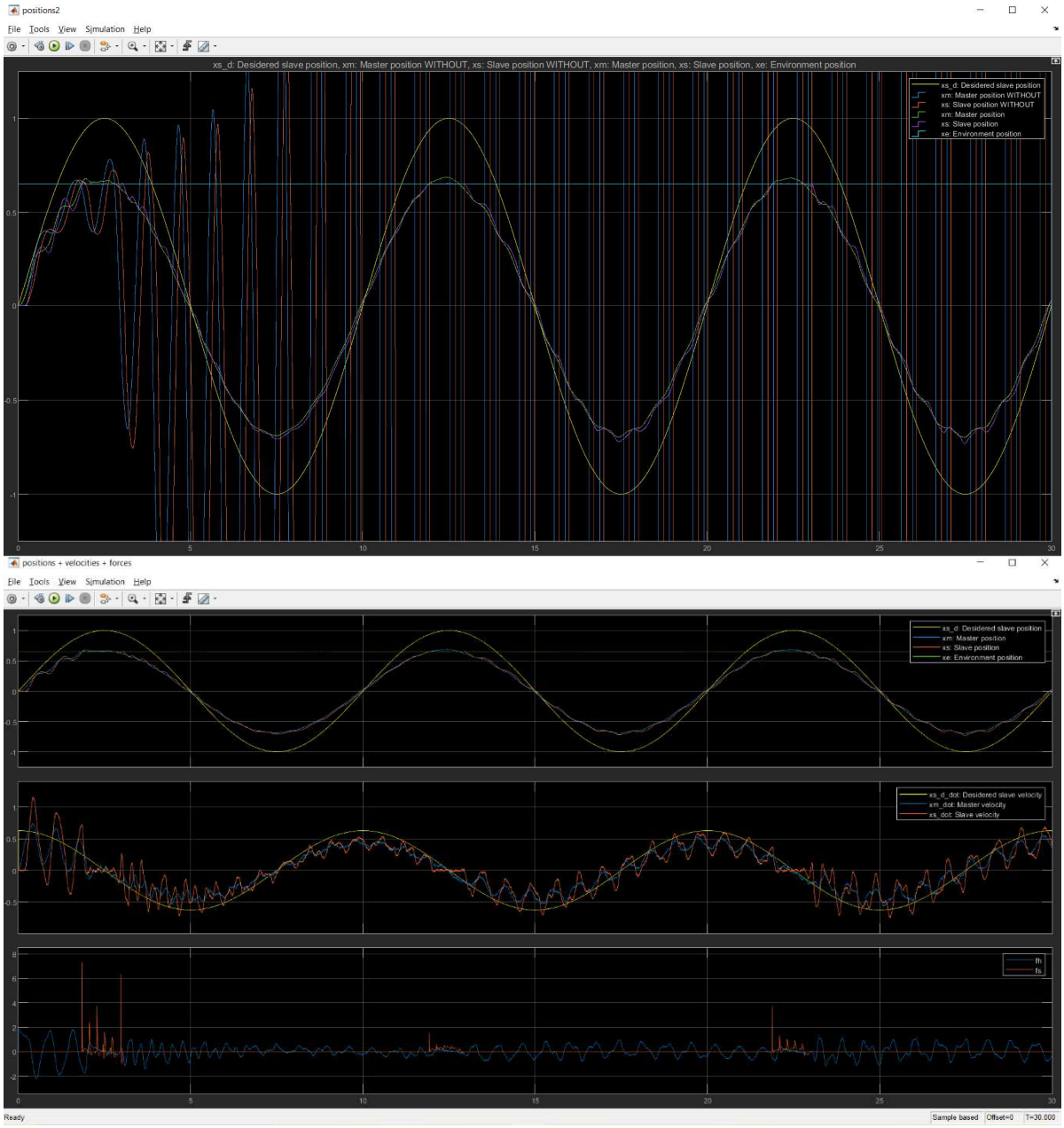
Step reference signal results, delay=20ms, without noise, free motion:



Step reference signal results, delay=20ms, without noise, contact:



Sinusoidal reference signal results, delay=200ms, noise, contact (with/without wave filtering comparison):

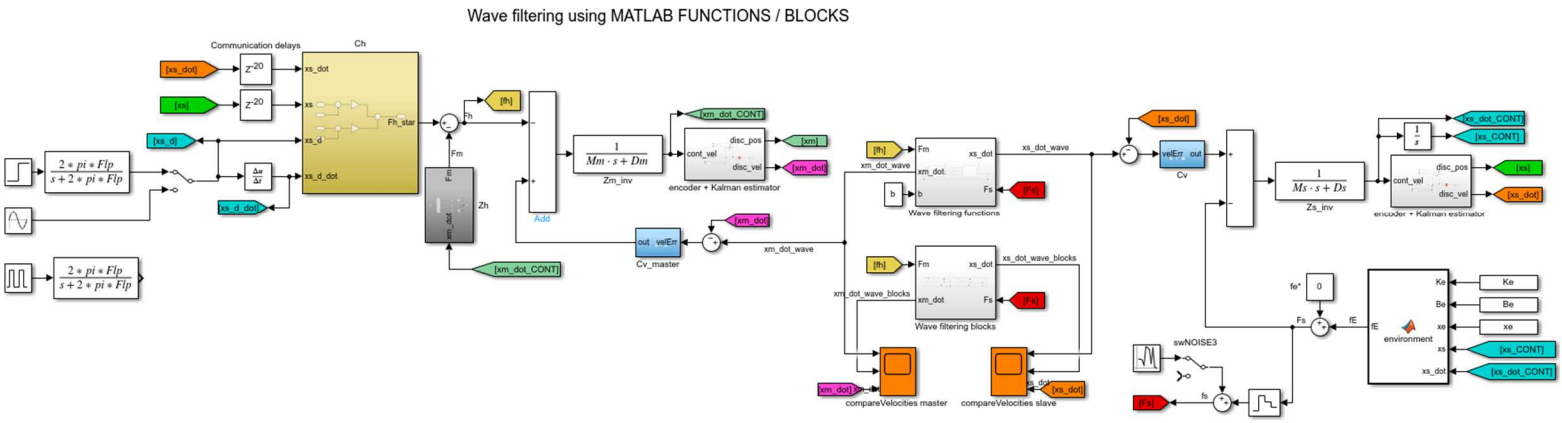


According to the paper, wave filtering ensures passivity even in presence of uniform delays. In fact, with a constant delay of 200ms the scheme with wave filtering gradually degrades its performance, while the "unfiltered" scheme diverges.

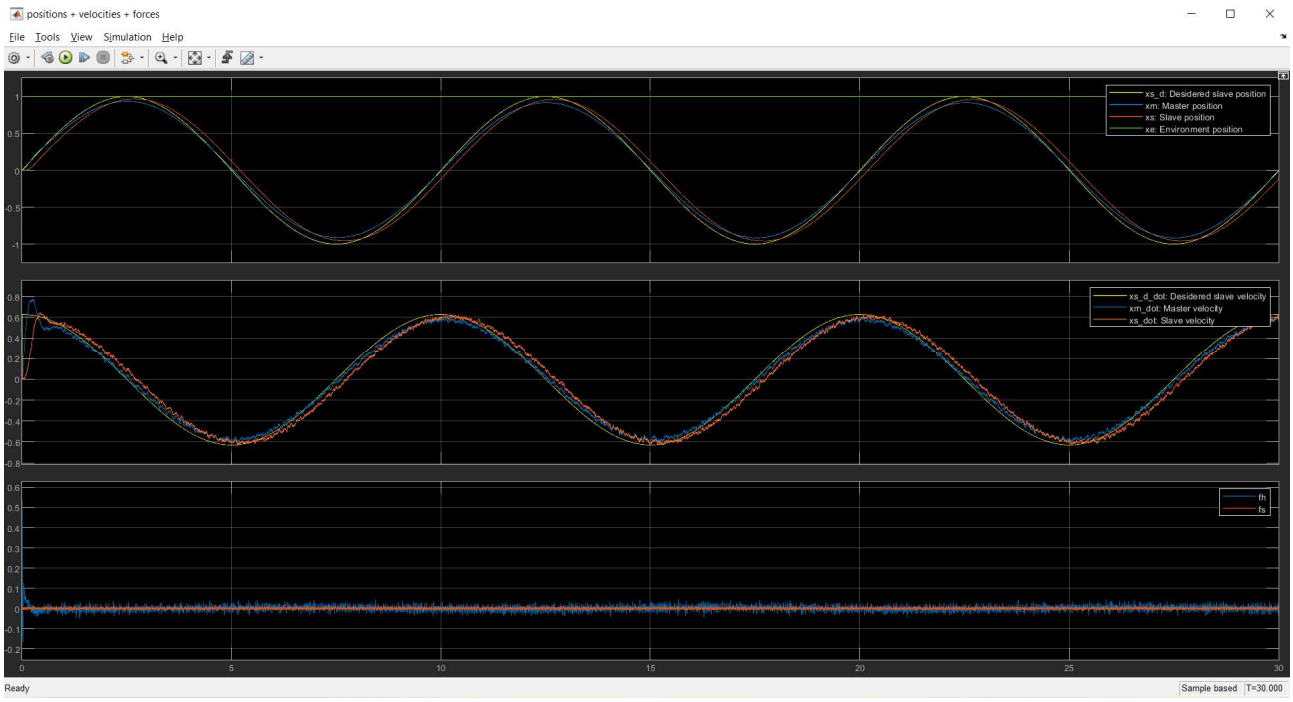
- **Implement the Scattering-based bilateral teleoperation architecture for the Position-Position case**

In a similar way, the Position-Position teleoperation scheme was developed, sticking to continuous time.

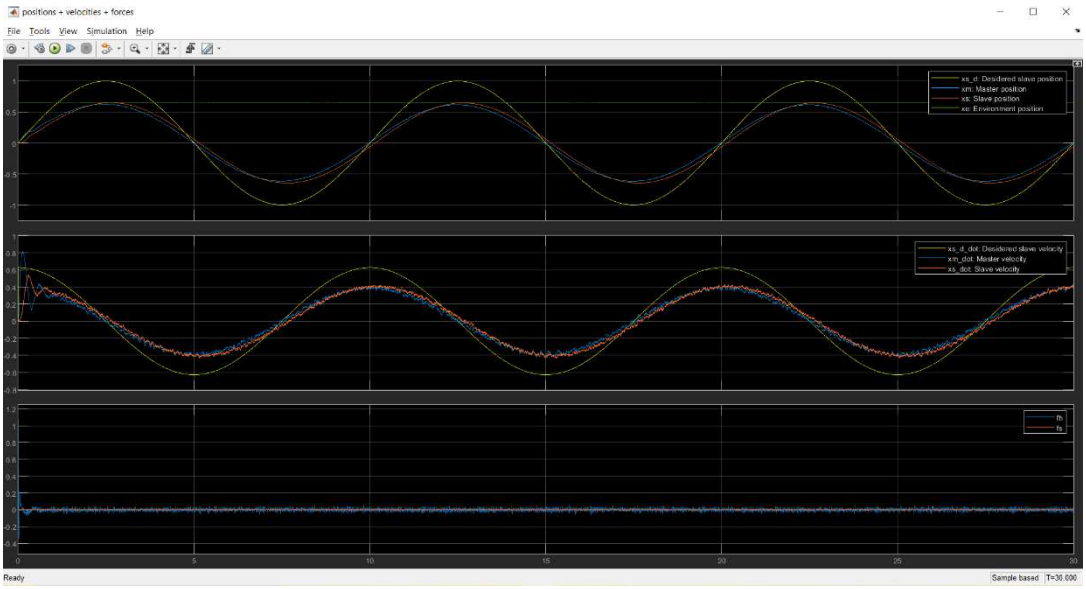
My Simulink Position-Position scheme (discrete-time):



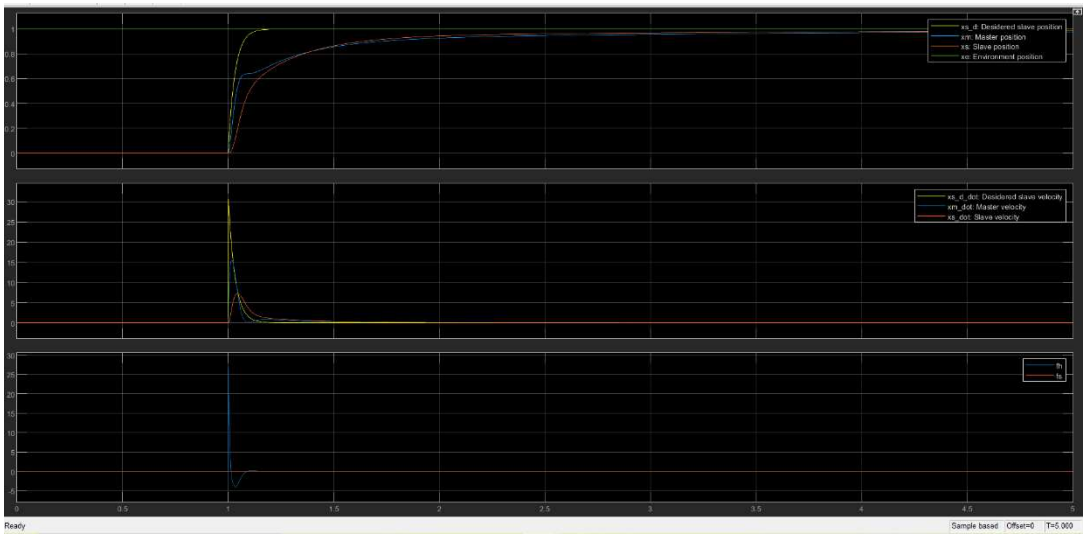
Sinusoidal reference signal results, delay=20ms, noise, free motion:



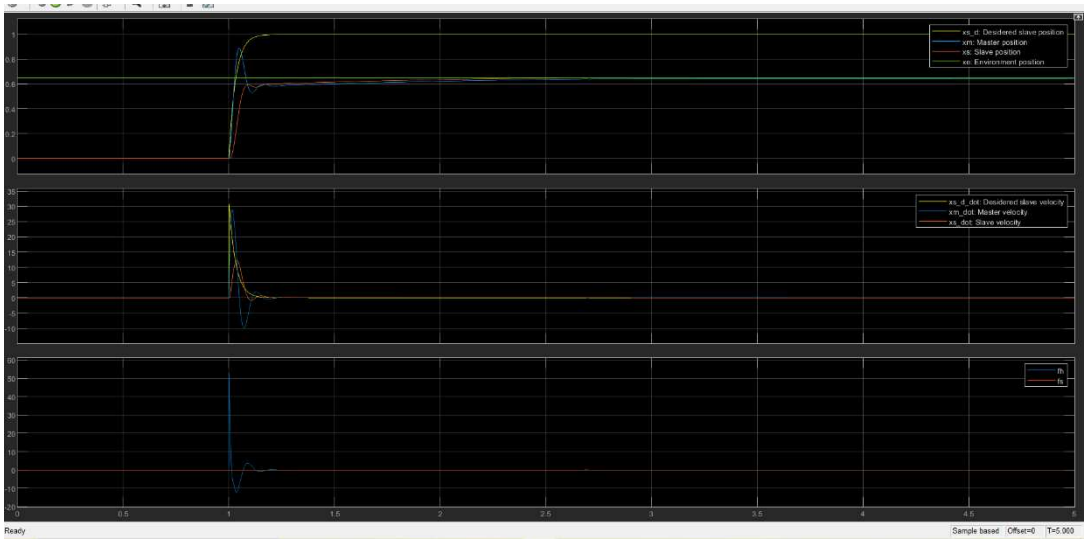
Sinusoidal reference signal results, delay=20ms, noise, contact:



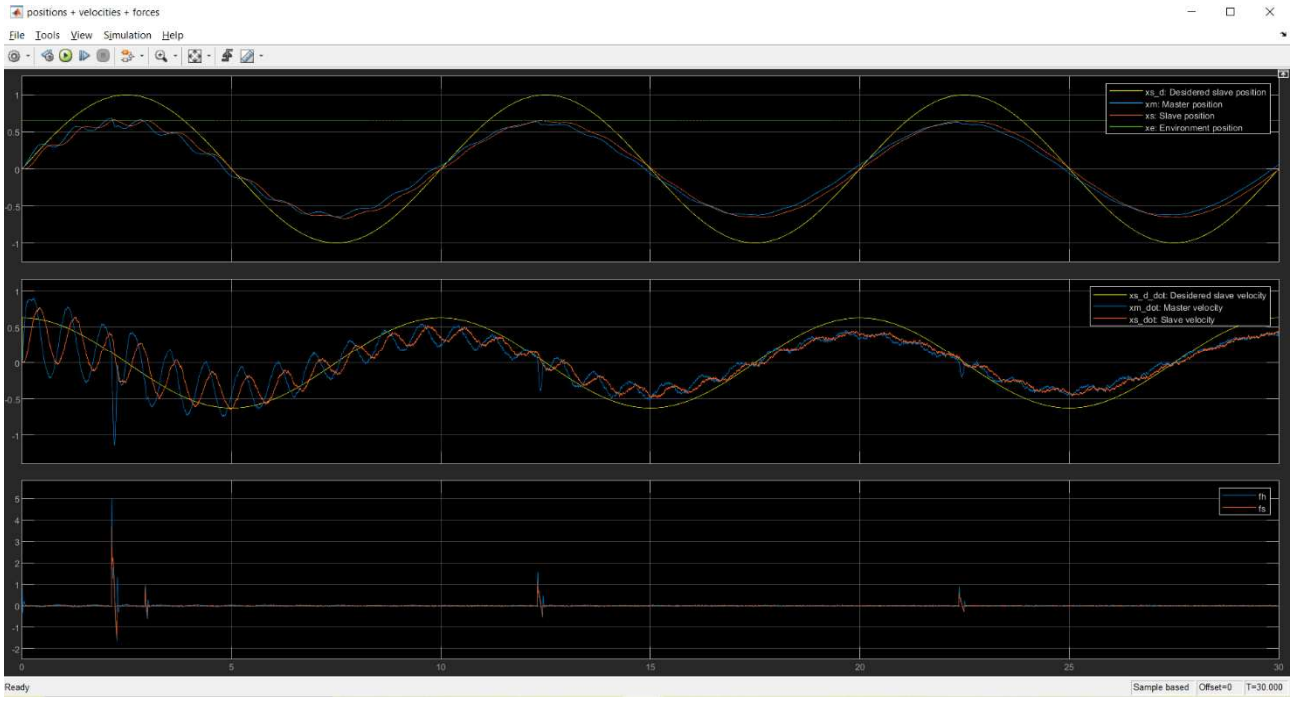
Step reference signal results, delay=20ms, without noise, free motion:



Step reference signal results, delay=20ms, without noise, contact:



Sinusoidal reference signal results, delay=200ms, noise, contact:



Assignment 7

- **Bilateral telemanipulation with time delays: a two-layer approach combining passivity and transparency**

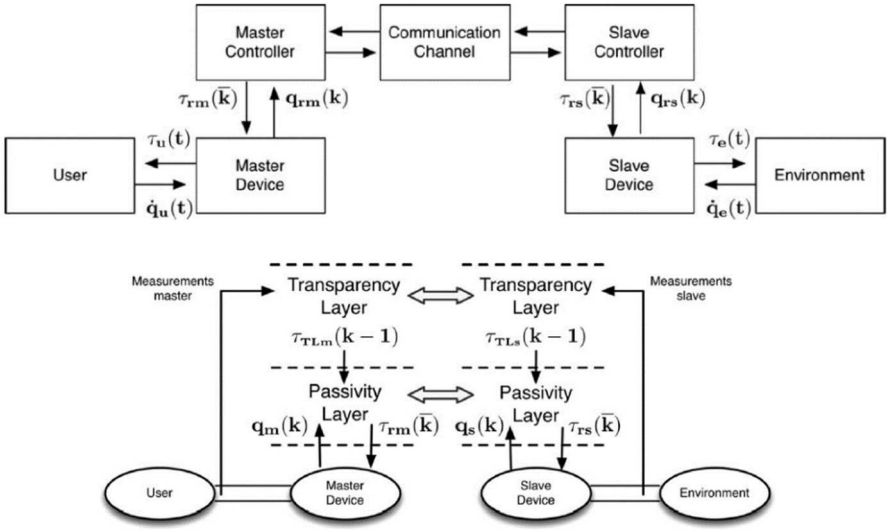
In this last assignment, another paper was analyzed and implemented. The authors present it as an evolution of the previous work (*assignment 6*). In fact, the proposed architecture would be *robust to any kind of communication delay and several time varying destabilizing factors.*

The main idea is to separate the ***transparency layer***, aimed at improving the user perception of direct interaction with the environment, from the ***passivity layer***, needed to ensure stability of the overall teleoperation system, despite the mentioned problems.

According to the paper, passivity-based approaches such as *wave scattering* ensure stability even in presence of significant delays but the level of transparency that can be obtained is *criticized.*

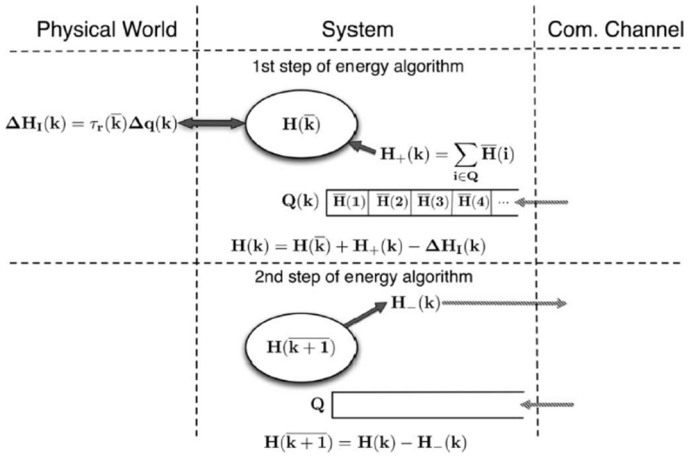
The proposed framework can be summed up in two steps:

- Inside the *transparency layer*, torques are computed in order to maximize transparency and reach the best possible performance. In particular, a slave velocity controller for the **F-P** architecture and the master + slave velocity controllers for the **P-P** architecture.
- Then, those commands are given to the *passivity layer* that decides if they're feasible or not, based on virtual energy that is stored inside "**energy tanks**". So, the master and slave robots' motion are powered by energy contained in the respective *storage tanks.*

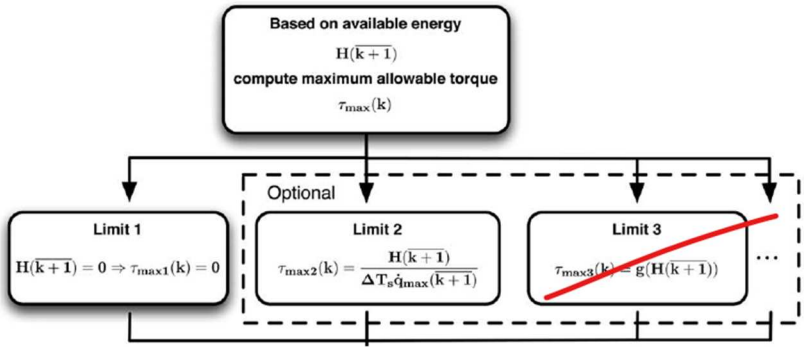


Following the proposed pipeline:

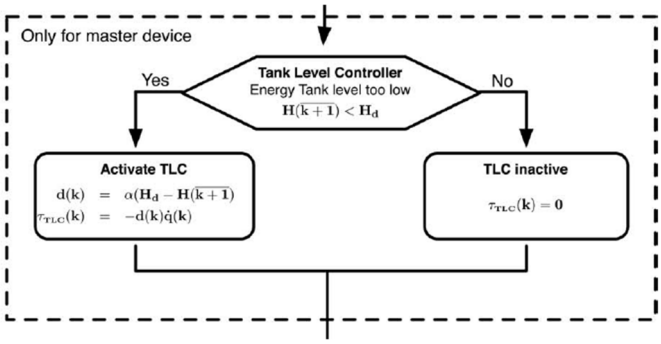
- the *energy exchange between the discrete-time controller and physical world* is computed (ΔH_I),
- then the *energy stored in the receiving queue in the form of packets* is computed at each time instant k (H_+).
- Knowing this information, the *energy tank levels* are computed (H(k)).



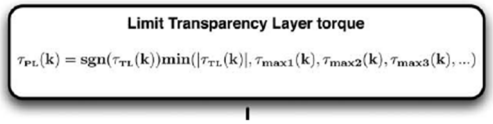
- Based on the available energy inside the tanks, the maximum allowable torque/force is computed (2 saturation techniques).



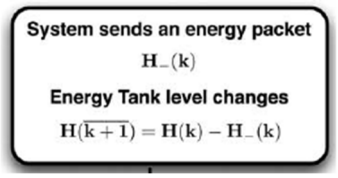
- Only on the <u>master</u> side, *Tank Level Controller* algorithm can be activated if the energy level inside the tank is too low. The idea is to apply a small opposing torque τ_{TLC} (*modulated viscous damper*) to the user's movement and extract energy from the user into the energy tank.



- At this point the *maximum torque* for the next sample period is set according to the following equation:

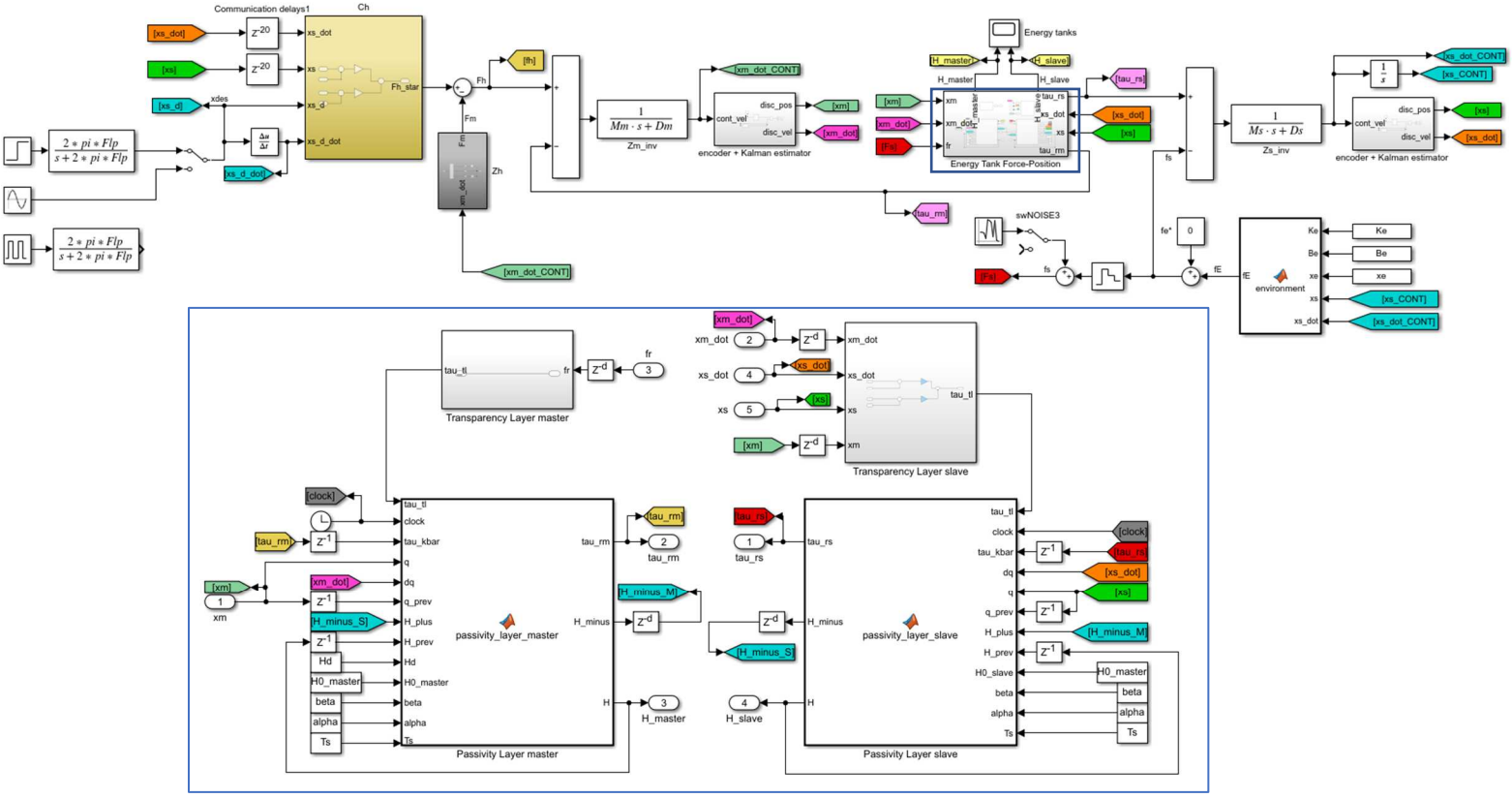


- According to the *energy transfer protocol* (SETP: Simple Energy Transfer Protocol, in my case), energy packets are sent to the other side of the teleoperation system.

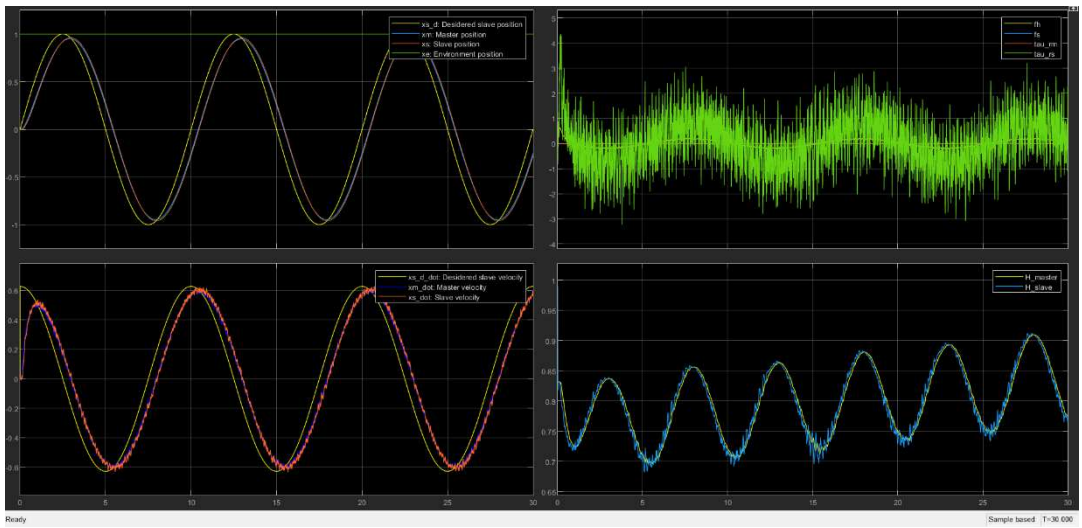


- ***Implement the Tank-based bilateral teleoperation architecture for the Force-Position case***

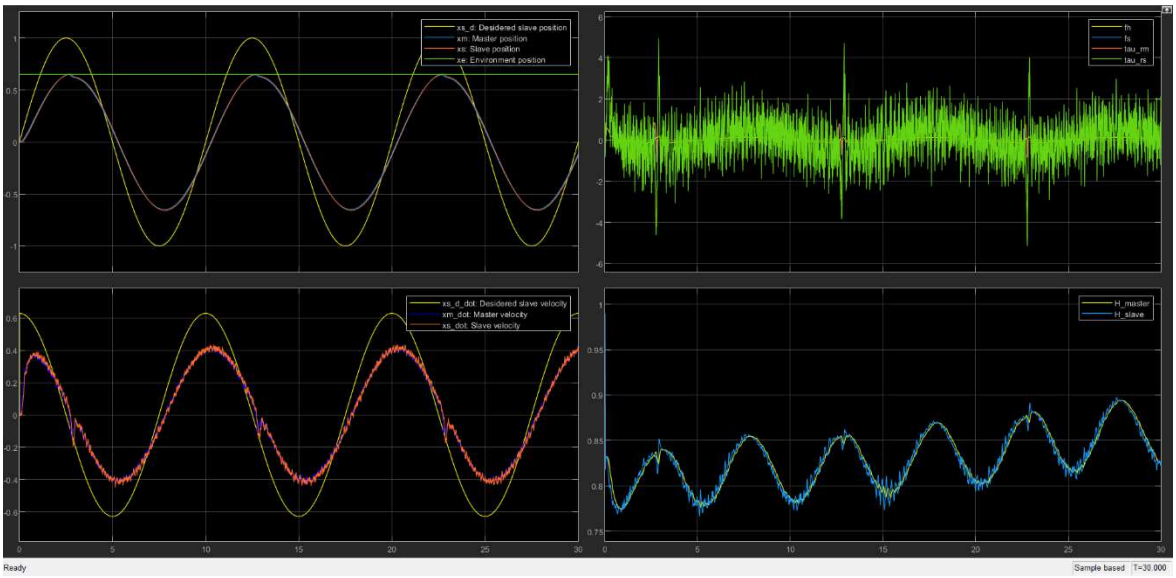
My Simulink Force-Position scheme (discrete-time):



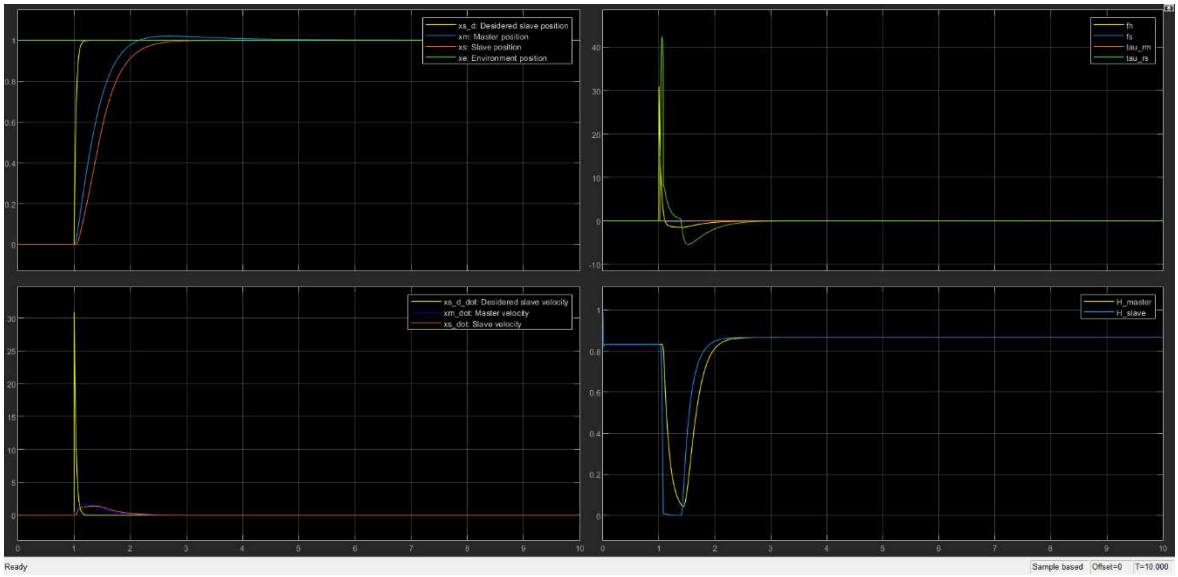
Sinusoidal reference signal results, delay=20ms, noise, free motion:



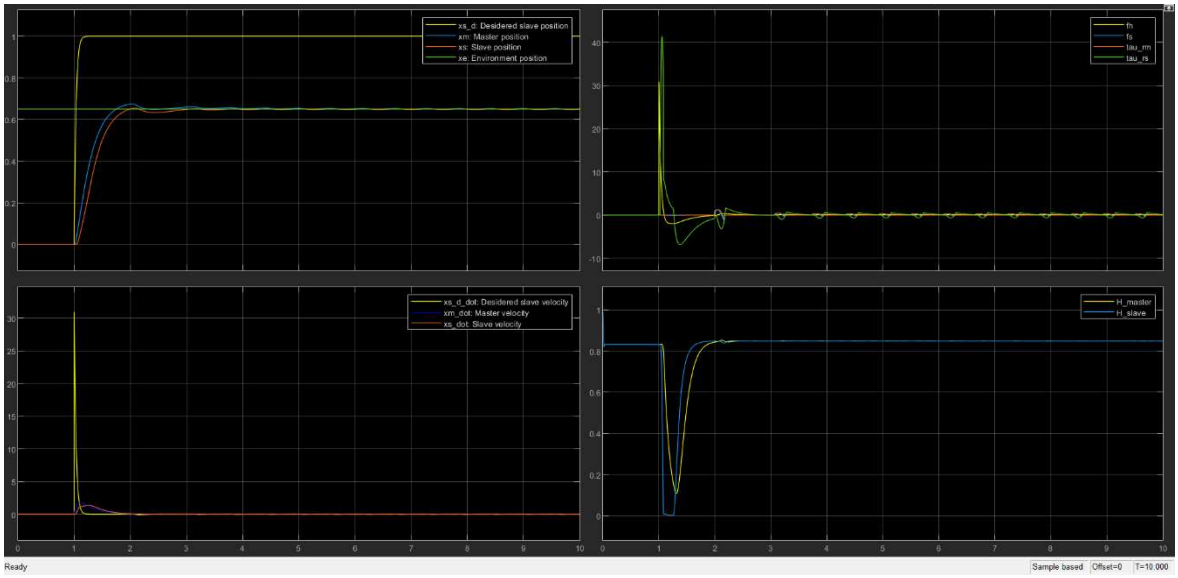
Sinusoidal reference signal results, delay=20ms, noise, contact:



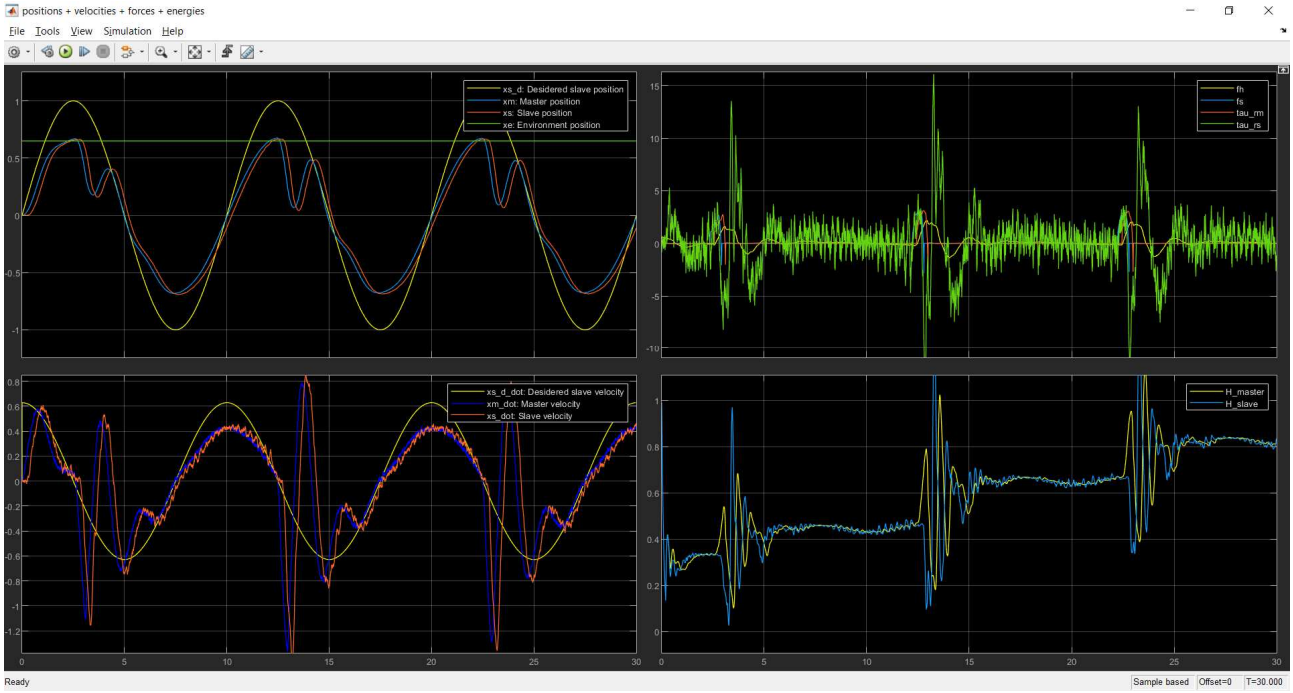
Step reference signal results, delay=20ms, without noise, free motion:



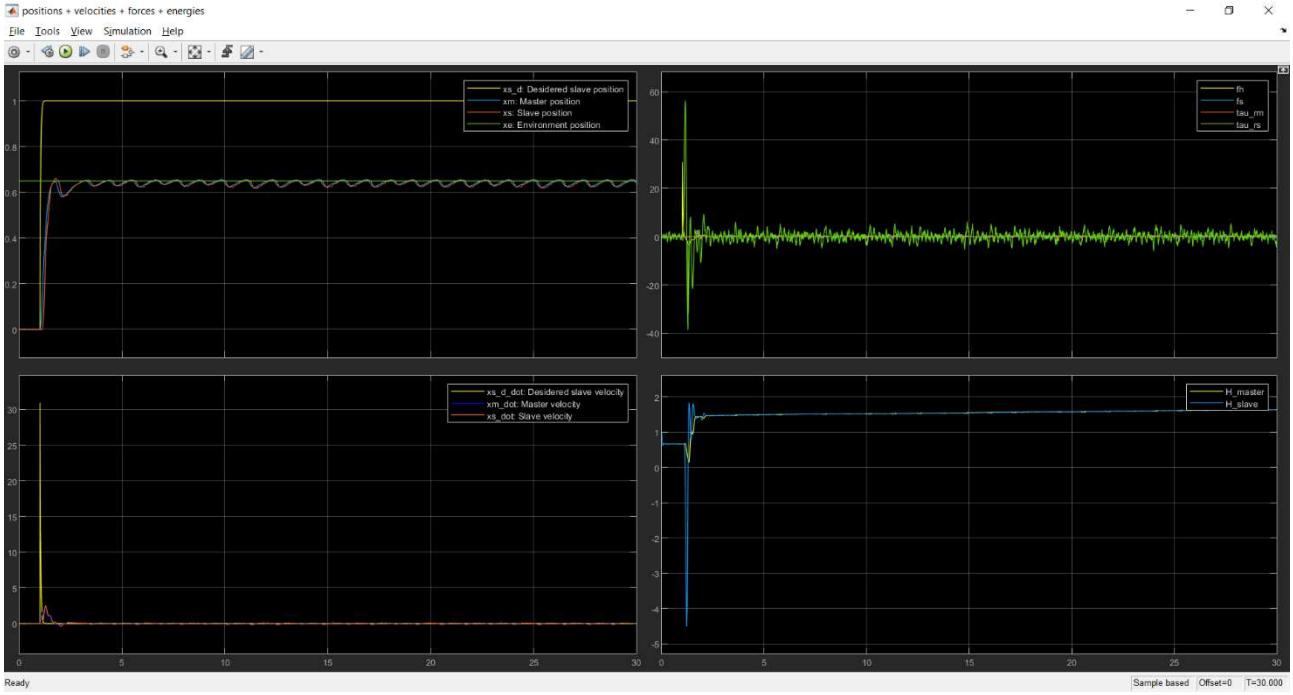
Step reference signal results, delay=20ms, without noise, contact:



Sinusoidal reference signal results, delay=200ms, noise, contact:

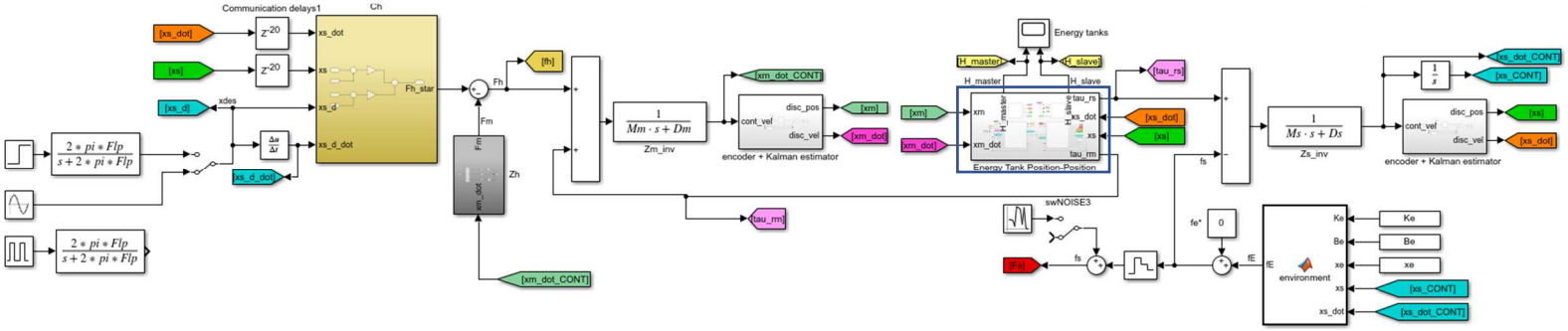


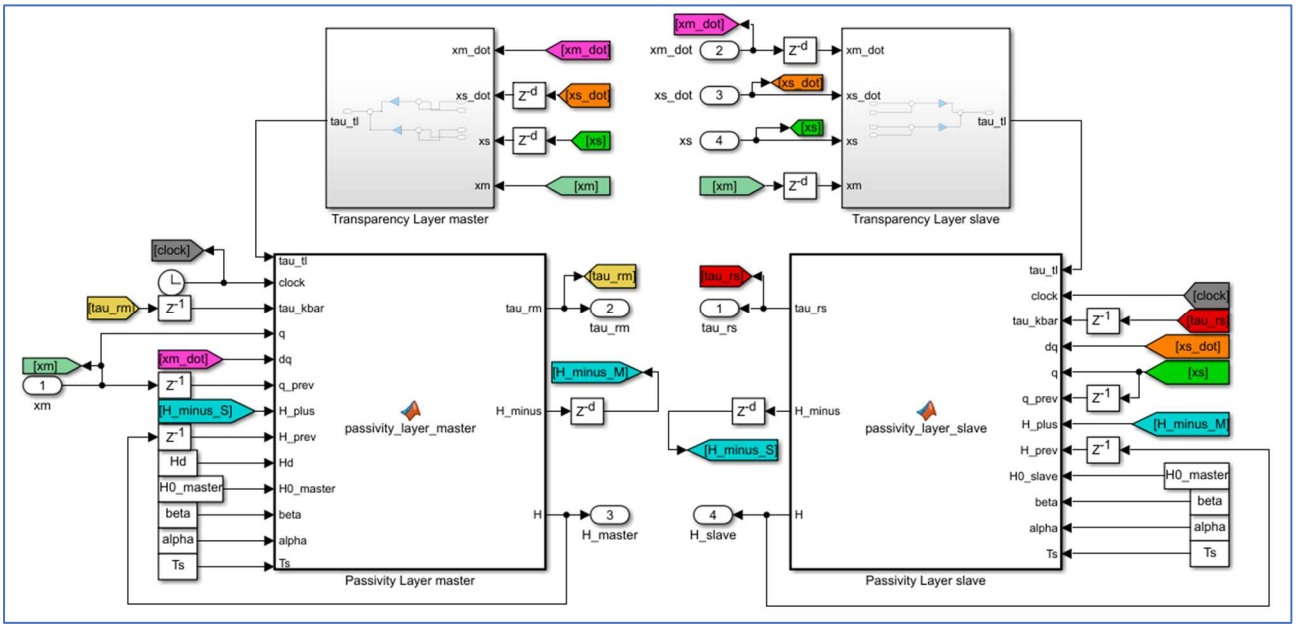
Step reference signal results, delay=100ms, noise, contact



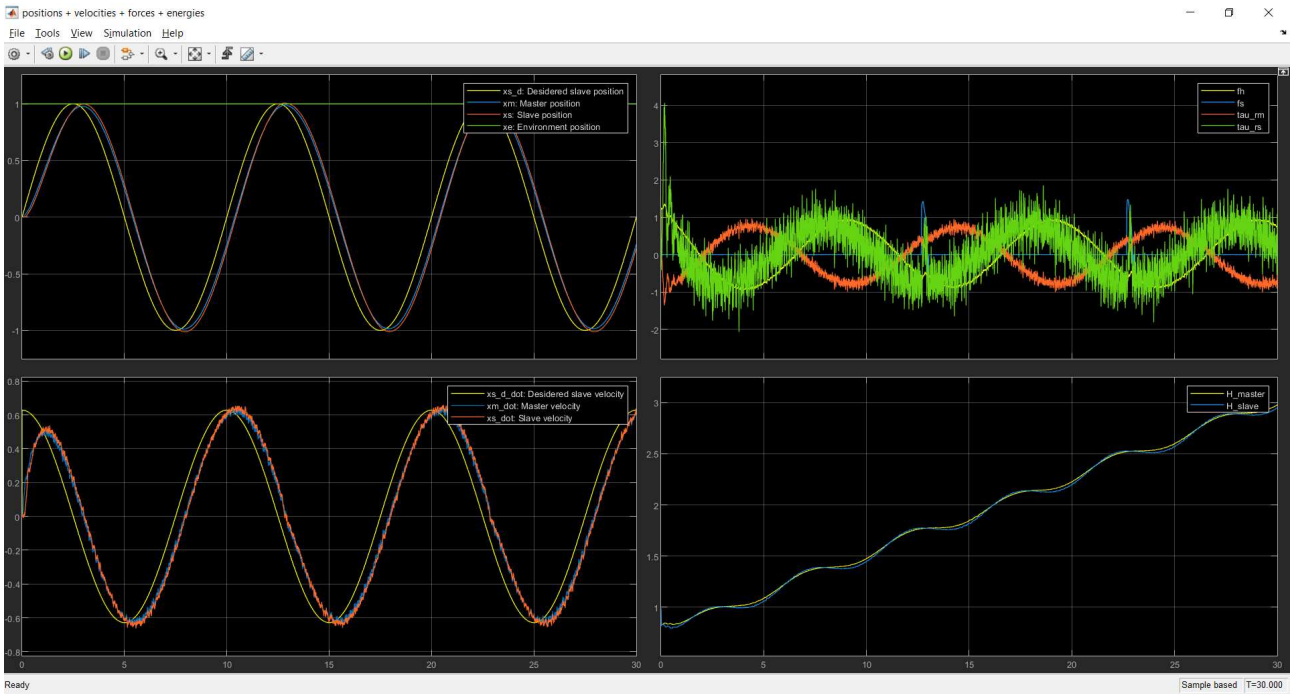
- ***Implement the Tank-based bilateral teleoperation architecture for the Position-Position case***

My Simulink Position-Position scheme (discrete-time):

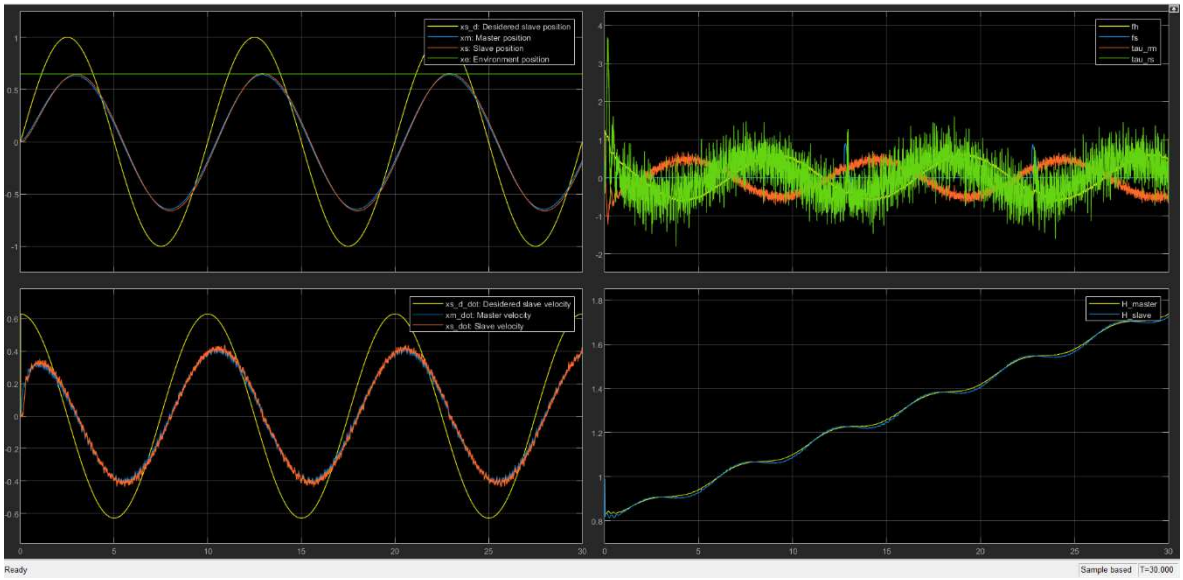




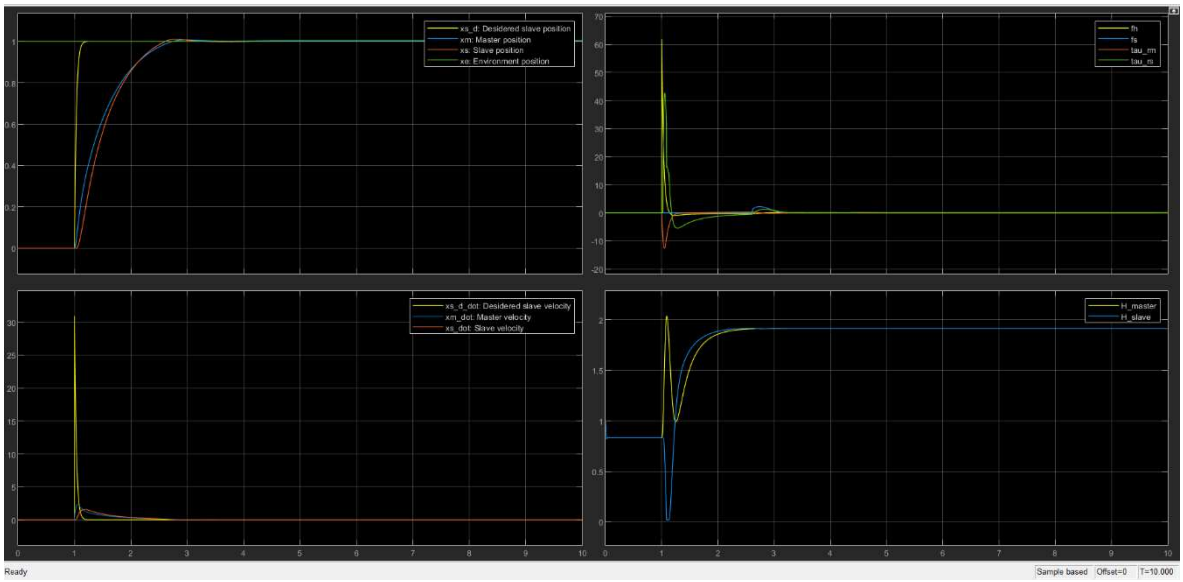
Sinusoidal reference signal results, delay=20ms, noise, free motion:



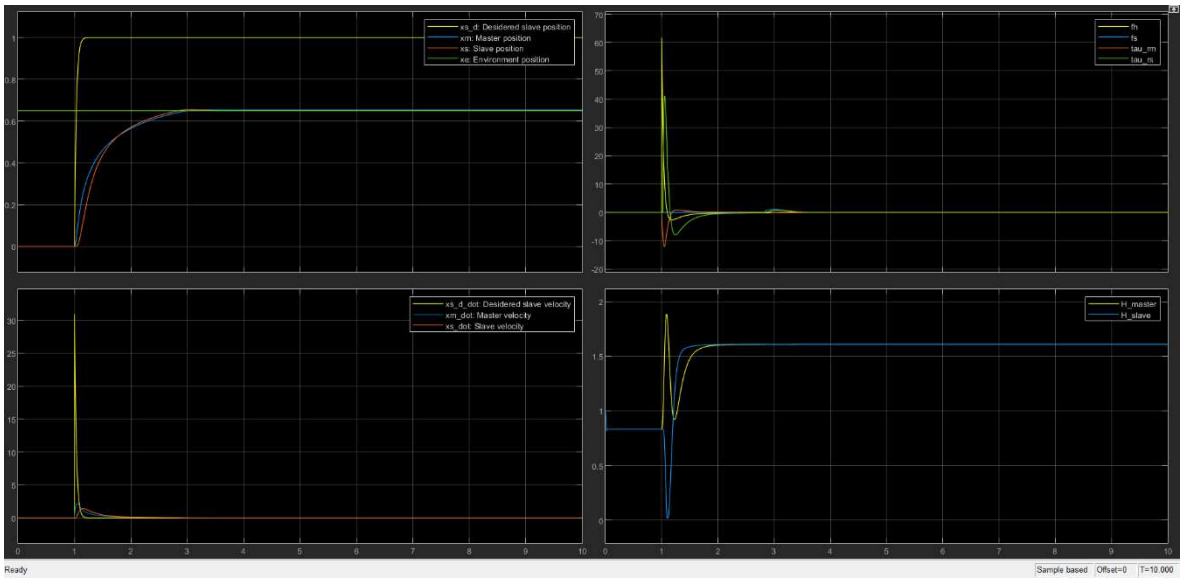
Sinusoidal reference signal results, delay=20ms, noise, contact:



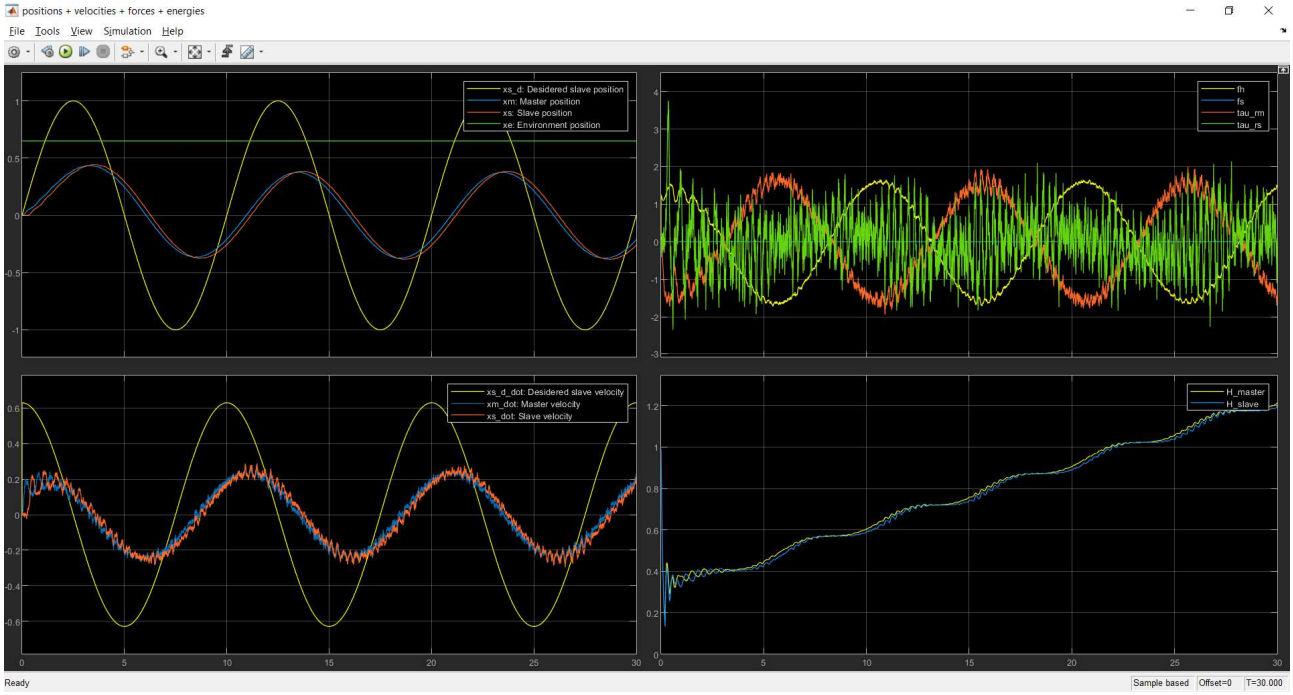
Step reference signal results, delay=20ms, without noise, free motion:



Step reference signal results, delay=20ms, without noise, contact:



Sinusoidal reference signal results, delay=200ms, noise, contact:



Step reference signal results, delay=200ms, noise, contact