NATIONAL UNIVERSITY OF SINGAPORE

EE4307: Control System Design and Simulation

Project

This project constitutes the second part (40%) of the final module score. Students are to work on the project independently, although parts of this manual are designed for their guidance. Laboratory assistants will also be available to provide limited guidance. The objectives of this project are:-

1. to provide hands-on experience in the acquisition and processing of real data from real physical systems and the use of computer-based data acquisition devices;
2. to familiarize with key non-parametric approaches to system identification for control design purposes;
3. to familiarize with some common and important functions collected within the *System Identification ToolBox* of MATLAB
4. to provide learning experience over the entire cycle of parametric system identification;
5. to provide the experience with computer simulation of dynamical systems;
6. to simulate and implement real-time control systems which are designed based on the models obtained earlier.

1. Introduction

Controller designs for dynamical systems are centred around a system model, which can be in a non-parametric or parametric form. There are systematic steps which should be undertaken before a control system can be designed and finally implemented.

First, control variables and objectives are defined, based on which the relevant information of the system should be collected in the form of raw data. Modern computer-based data acquisition devices can be used efficiently for this data logging process. The raw data collected may need to be further conditioned, in view of noise and other disturbances inevitably present, before it can be used for the next phase of system modelling.

System modelling can be carried out with a physical approach, using fundamental first principles to relate the variables of interests. However, this physical approach is typically constrained to small and relatively simple systems. Even then, there are unknown physical parameters to be determined. Another approach is based on system identification which uses only the logged data of the system and a pre- selected model structure to yield a model. In a way, this can be considered a “black- box” approach as the system, to be identified, is treated as an unknown black box to be fitted to the model structure based on observations of the system only. In this project, the two approaches of system identification mentioned above will be explored.

System identification approaches can be broadly classified under a non-parametric or parametric approach. A non-parametric approach will yield a non-parametric mode of the system, unconstrained to any parametric structure. For simple control design, such information, which can be in the time or frequency domain, is sufficient. A parametric approach, on the other hand, will yield a full parametric model and it involves an iterative loop to arrive at the final model. This process involves decision making on the part of the person in search of models, as well as fairly demanding computations to furnish bases for these decisions. A user will typically go through several iterations in the process of arriving at a final model, where at each step, earlier decisions are revised. Model-based control designs can be carried out when the parametric model is obtained.

For system identification, interactive software is a natural tool for approaching system identification. It is also a convenient way to package the rather extensive theory, making it available to the user. MATLAB is an excellent environment for such interactive calculations, with its workspace concept, graphing facilities, and easy data import and export. The System Identification Toolbox is a collection of M-files that implement the most common and useful parametric and non-parametric techniques available. It is specifically aimed at giving the user help, not only at computing, but also at evaluating models. Through this project, students will realize how easily a system identification task may be accomplished through the use of such modern interactive software.

Once a model becomes available and the controller designed, a simulation study on the adequateness of the model and the control system can be done prior to actual implementation on the real system. Simulation is usually done for prediction, control or even training purposes. The simulation study will allow fine adjustment to the model and/or controller until a satisfactory level of performance is achieved. This step is useful and necessary in many case, as it is not always possible to do an experiment directly on the actual system over a prolonged period for time, due to cost and safety concerns, or due to the fact that the actual system simply have not yet being developed.

The project will bring the students through all the phases of control design and simulation. In Part 2, an introduction on the three assigned systems is given. In Part 3,

acquisition device and control platform is elaborated. The main project comprises of 5 parts which includes Parts 4 to 8 of the project manual, corresponding to each key phase of the process. The lectures pertaining to Parts 4, 5 and 6 will be covered by Prof. Sam Ge, while the lectures for the remaining parts will be covered by Prof. Ge on behalf of Prof.AA Mamun. In Part 9, requirements on the project report is given

2. System Identification: Parametric Approach

In this part of the project, we will first learn the essential steps of a parameter estimation cycle. After that, you will work on an assigned plant to derive a parametric model.

2.1 A Guided Example

This example is designed to familiarize you with the software quickly and to guide you through the steps of system identification covered in the lectures. You must understand the rationale for each of the step.

Data has been collected from a laboratory scale process. The process operates very much like a common hand-held hair dryer. Air is blown through a tube after being heated at the inlet to the tube. The input to the process is the voltage applied to a mesh of resistor wires that constitutes the heating device. The output of the process is the air temperature at the outlet measured in volts by a thermocouple sensor.

One thousand input-output data points were collected from the process as the input was changed in a random fashion between two levels. The sampling interval used is

80 ms. The data were loaded into MATLAB in ASCII form and are now stored as the vector y (output) and u (input) in the file ***ee4307a.mat***.

First load the data

```
load ee4307a
```

(At any stage, you may use >*help cmd* to know more about *cmd*, e.g., >*help load*)
This example selects the first 300 data points for building a model. For convenience,

the input-output vectors are merged into a matrix, z :

```
z = [y(1:300)  u(1:300)];
```

You may take a look at the data using:

```
iddata(z)
```

Note the sampling interval used in the graph by default is one time unit, and all data points are plotted.

You can select the values between sample numbers 200 and 300 for a close-up view, and at the same time, obtain correct time scales with the sampling interval of

80ms instead, using:

```
iddata(z(200:300),0.08)
```

We will also select and reserve a portion of the data to be used to cross-validate the model:

```
zr=[y(700:900)  u(700:900)];
```

<u>Data treatment</u>

The plot of the data points provides an opportunity to check for any deficiencies which may be treated

In this experiment, we are not keen in modelling the DC equilibrium conditions. We want to model the changes in temperature corresponding to the changes in input voltage. We will remove the constant levels and make the data zero mean with

```
z = detrend(z);
```

```
zr = detrend(zr);
```

Other actions typically taken by user at this stage include removal of outliers, data pre-filtering, lowering of sampling rate, etc. Some relevant commands available for use are *idfilt* and *decimate*.

Ready-made model

We shall work with models of the following kind (ARX):

$$y(t) + a_1 y(t-1) + \cdots + a_{na} y(t-na) = b_1 u(t-nk) + \cdots + b_{nb} u(t-nb-nk+1) \quad (2.1)$$

Correlation analysis

When a ready-made model is to be selected, the user has to specify the delay and model order. A gross estimate of the delay may be obtained from the impulse response of the system.

Obtain the impulse response of the system with a correlation analysis:

```
ir=cra(z);
```

The function pre-whitens the input so that it approximates the behavior of white noise.

Briefly explain why this pre-whitening process is necessary.

The impulse response of system is shown in the 4th plot.

Estimate the time delay of the system.

Time-delay estimate

A more accurate value may also be estimated in the following way. The order of the model is fixed to second-order (*na*=*nb*=2), we then choose likely values of the delay between 1 and 10. The loss function, which show how good is the fit (the lower the better), can tell us which is the more accurate delay. We may use *arxstruc* for this purpose:

```
V = arxstruc(z,zr,struc(2,2,1:10));
```

The loss function is inherently computed by comparing model data set *z* to validation data set *zr.* We now select that delay that gives the best fit for the reference set using:

```
nn =  selstruc(V,0)
```

Compare the estimation of the delay obtained with *cra* and *arxstruc*.

Choice of model structure

Now that the delay has been obtained, we will obtain the model order na and nb. As in the case for the delay, we check the fit for all 25 combinations of up to 5 b- parameters and up to 5 a-parameters, all with delay obtained earlier:

```
V = arxstruc(z,zr,struc(1:5,1:5,3));
```

The best fit for the reference data set is obtained for

```
nn = selstruc(V,0)
```

It is advisable to check yourself how much the fit is improved for the higher order models. We are more interested in the "settled-in" point (Figure 7.1). Beyond this point (see Figure 7.1), the improvement is marginal and yet we will have a more complicated model. You can use the following command:

```
nns = selstruc(V)
```

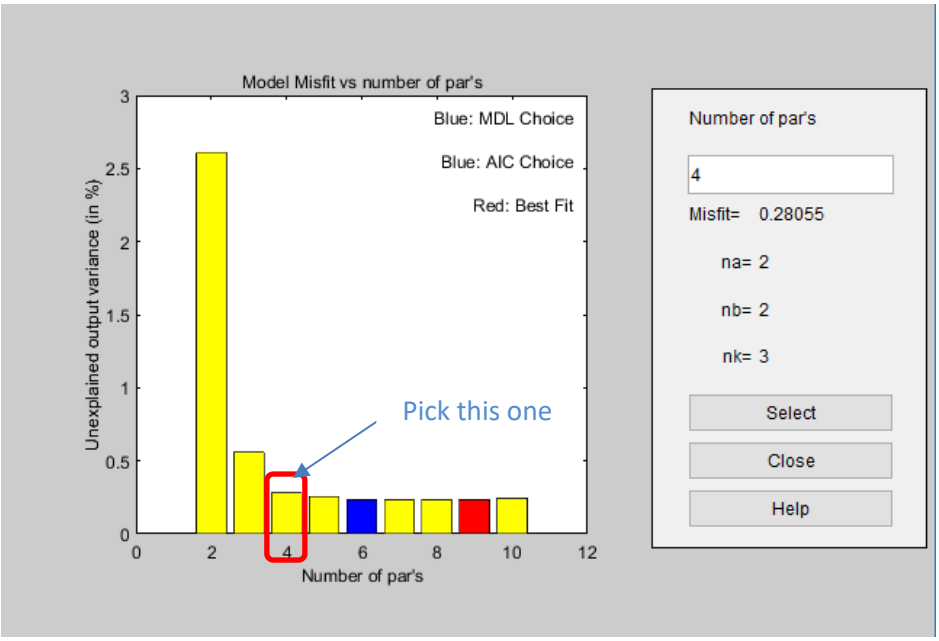

Figure 2.1 Selection of model order

The plot shows the fit as a function of the number of parameters used. You will then be asked to enter the number of parameters you think gives a sufficiently good fit. The routine then selects that structure with this many parameters that gives the best fit.

To ensure you are still on track, verify that the best fit is obtained for *nn* = [4 5 3], while we see that the improvement compared to *nn* = [2 2 3] is rather marginal.

Model computation

We will now compute the four parameters for the model corresponding to *nn* = [2 2 3] using

$$y(t) + a_1 y(t-T) + a_2 y(t-2T) = b_1 u(t-3T) + b_2 (t-4T) \qquad (2.2)$$

where *T* is the sampling interval. This ARX model predicts the value of output at time *t*, given previous values of *y* and *u*. The best values of the coefficients a_1, a_2, b_1, b_2 can be computed using the least squares method with:

```
th = arx(z,[2 2 3]);
```

The result is stored in the matrix *th* in a somewhat coded form. To specify the sampling interval, enter:

```
th = set(th, 'Ts',0.08);
```

Display the coefficient estimates and estimated standard deviations with:

```
present(th)
```

The System Identification Toolbox also include functions specific to other model structures e.g. ARMAX and Box-Jenkins models. The use of these functions is similar to *arx*. We shall compute the model parameters for a second-order ARMAX model. The steps are similar to the case of the ARX model:

```
am = armax(z,[2 2 2 3]);
am = set(am, 'Ts',0.08);
present(am);
am.Report.Fit.LossFcn
```

Compare the loss function for the ARMAX model to that of the ARX model.

Model validation

You may wonder how well the model fits the data. We will check the residuals ("leftovers") of this model, i.e., what is left unexplained by the model.

```
resid(z,th);
```

The 99% confidence intervals for these values are also computed and displayed as dotted red curves. We see that the residual are quite small compared to the signal level of the output.

Another simple test is to run a simulation whereby real input data is fed into the model, and to compare the simulated output with the actual measured output. For this, select a portion of the data that was not used to build the model, for example, from sample 501-1000 (called cross-validation):

```
u = detrend(u(501:1000));
y = detrend(y(501:1000));
ysim=sim(u,th);
plot([y(100:500) ysim(100:500)])
```

Print the result and attach to your report. Comment on the comparison.

Frequency response

If you want to know the frequency response, you can compute the frequency function of the model and present it as a Bode plot by entering:

```
gth = th2ff(th);
bodeplot(gth)
```

Step response

Finally, plot the step response of the model. The model comes with an estimate of its own uncertainty. Ten different step responses are computed and graphed. They correspond to "possible" models, drawn from the distribution of the true system.

(There appear to be some issues with the new MATLAB Version. For now, please comment out the line of specifying sampling interval, % *th = sett(th, 0.08)* before the following steps)

```
step = ones(30,1);
idsimsd(step,th)
```

Print the step responses.

7.2 Working on the Assigned System

Now, you should be ready to identify an adequate model of the following assigned system.

$$y[k] = q^{-3}\frac{0.3 + 0.2q^{-1}}{1 - 0.9q^{-1} + 0.2q^{-2}}u[k] + \frac{1}{1 - 0.9q^{-1} + 0.2q^{-2}}e[k] \tag{2.3}$$

Use the steps listed in the above to identify the mode (2.3), observe and discuss the identification discrepancy between the identified model and the true model.

3. System Simulation Using PID Control

Simulation is a cost effective way of verifying the performance of a system based on its model. It does not require experimentation with the actual system which may not be possible due to various reasons such as costs and safety issues. Furthermore, with amplitude and time scaling, simulation can help to speed up the design process, by providing prediction of a system performance in a very short time via powerful computer systems. In this section, students will first work on a modelling and simulation example, where simulation is used to predict the outcome of a hypothetic scenario. Next, students will familiarise themselves with how simulation can help in control design once a model becomes available, using a commonly available tool in the form of Microsoft Excel spreadsheet.

3.1 Predator-Prey Modelling and Simulation

In this part, we will first experiment with how a model can be simulated and the results used for analysis and prediction purposes. Let us digress from the typical engineering problems to consider a predator-prey model.

Two animal species (lynx and hare) are in a predator-prey situation. We are interested in deriving a model which shows the variations in the surviving number of each species with time.

Let

$N_1(t)$ denotes the number of lynx at time *t*

$N_2(t)$ denotes the number of hare at time *t*

λ_1 denotes the birth rate of lynx (number per unit time per lynx)

λ_2 denotes the birth rate of hare (number per unit time per hare)

μ_1 denotes the mortality rate of lynx (number per unit time per lynx)

μ_2 denotes the mortality rate of hare (number per unit time per hare)

Since lynx needs hare to survive, μ_1 reduces with N_2 (less lynx will die per unit time when number of hare increases). We assume a simple proportional relationship:

$$\mu_1 = \gamma_1 - \alpha_1 N_2, \quad \alpha_1 < 0$$

Since hare is eaten by lynx, μ_2 increases with N_1 (more hare will die when number of lynx increases). We assume a simple proportional relationship:

$$\mu_2 = \gamma_2 - \alpha_2 N_1, \quad \alpha_2 > 0$$

Thus, derive the predator-prey model relating $\frac{\mathrm{d}N_1}{\mathrm{d}t}$ and $\frac{\mathrm{d}N_1}{\mathrm{d}t}$ to N_1 and N_2. Verify that you obtain the following results:

$$\frac{\mathrm{d}}{\mathrm{d}t}N_1(t) = (\lambda_1 - \gamma_1)N_1(t) + \alpha_1 N_1(t)N_2(t)$$

$$\frac{\mathrm{d}}{\mathrm{d}t}N_2(t) = (\lambda_2 - \gamma_2)N_2(t) + \alpha_2 N_1(t)N_2(t)$$

Let us use some reasonable numbers for the various constants for simulation.

If there is no hare (N_2=0), number of lynx is expected to decrease, i.e. $\frac{\mathrm{d}N_1}{\mathrm{d}t} < 0$. Therefore, $\lambda_1 - \gamma_1 < 0$. Let $\lambda_1 = 1, \gamma_1 = 2$.

If there is no lynx (N_1=0), number of hare is expected to increase, i.e. $\frac{\mathrm{d}N_2}{\mathrm{d}t} < 0$. Therefore, $\lambda_2 - \gamma_2 > 0$. Let $\lambda_2 = 2, \gamma_2 = 1$.

Let $\alpha_1 = \alpha_2 = 1$.

Thus, the models become:

$$\frac{\mathrm{d}}{\mathrm{d}t}N_1(t) = -N_1(t) + N_1(t)N_2(t)$$

$$\frac{\mathrm{d}}{\mathrm{d}t}N_2(t) = N_2(t) - N_1(t)N_2(t)$$

Now, let us simulate based on the models. We need to convert the models to discrete time to simulate on computers.

Show that using a simple Euler's (forward difference) method, the following discrete-time models can be obtained: $N_1\big((k+1)T\big) = N_1(kT) + T(-N_1(kT) + N_1(kT)N_2(kT))$ $N_2\big((k+1)T\big) = N_2(kT) + T(N_2(kT) - N_1(kT)N_2(kT))$

Assuming an initial population of $N_1(0) = N_2(0) = 2$ (in thousands) and sampling interval of T=0.01, simulate the models using any languages of your choice (MATLAB, C, PASCAL, etc.).

Log your results in your final report and comment.

To make the simulation more interesting, let us also simulate a disturbance. Assume a disaster (wild fire) broke out at time t=10 which reduces the population of both species by 50% at that time.

Log your results in your final report and comment.

Do you expect the overall population of the species to decrease after years have passed as a result of the fire? Check if the prediction from your simulation tallies with your expectation.

3.2 Simulation of a PID Control System

Proportional + Integral + Derivative controllers are widely used in industry, due to their relative simplicity and ability to control a wide range of processes reasonably well. This part of the project investigates the main features of these controllers by analyzing the responses of each component through simulation on commonly available spreadsheet software: Microsoft EXCEL.

In order to illustrate the main concepts of this type of controller, the development proceeds

in stages; generating first a simple first order plus delay process model, and then a P, PI, and finally full PID controller.

The process model used in this simulation is the standard "first order lag plus delay" model used in many process control applications as follows:

$$Y(s) = \frac{Ae^{-sL}}{1+sT}U(s) \tag{3.1}$$

where *U*(s) corresponds to the input to the process, *Y*(s) corresponds to the output of the process, *T* is the time constant, *A* is the gain, and *L* is the time delay of the process (see Figure 8.1).

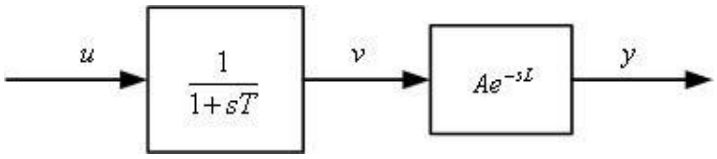

Figure 3.1: Process model

Suggest any non-parametric method where simple estimates of *T*, *L*, and *A* can be obtained.

The model in Equation (8.1) is a time-continuous model. In order to simulate it on a computer, the model must first be discretized using numerical methods.

The time delay (e^{-sL}) can be represented by integer samples of delay:

$$y_k = v_{k-n} \tag{3.2}$$

where y_k is the process output at sample time k, and v_{k-n} is the output from the lag model at sample time $k-n$. A time delay is *L* will correspond to samples, where *h* is the sampling interval.

The first order lag (with time constant *T*) will be simulated using the difference equation. This is obtained via a backward difference approximation by replacing *s* with $\frac{1-q^{-1}}{h}$, resulting in:

$$v_k = v_{k-1} + (u_k - v_{k-1})\frac{h}{h+T} \tag{3.3}$$

where h is the time interval between subsequent samples k and $k-1$.

Show how Equation (8.3) is obtained.

The closed-loop control system with a PID controller is shown in Figure 8.2. The PID controller transfer function is given, in the time and frequency domains, as follows:

$$u(t) = G\left(x + \frac{1}{T_i}\int_0^t x d\tau + T_d\frac{dx}{dt}\right) \tag{3.4}$$

$$U(s) = G\left(1 + \frac{1}{sT_i} + sT_d\right)X(s) \tag{3.5}$$

where x is the error between the set-point, w, and the process value ,y, i.e., $x = w - y$, and u is the controller output.

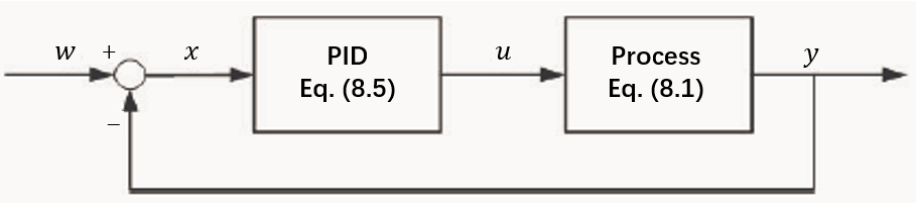

Figure 3.2: Closed-loop PID control of the process

This controller may be implemented as the sum of the three separate components. The proportional component is as follows:

$$P_k = Gx_k \tag{3.6}$$

The integral component may be implemented in discrete time, as an accumulator, as follows:

$$I_k = I_{k-1} + P_k \frac{h}{T_i} \tag{3.7}$$

and the derivative component may be implemented in difference form as follows:

$$D_k = (P_k - P_{k-1}) \frac{T_d}{h} \tag{3.8}$$

Show how Equation (8.7) and Equation (8.8) are obtained.

These three components are then added to give the PID controller output, u.

Preparation

Open Microsoft Excel application to conduct the simulation.

Procedures

Time (Column A)

In Column A (label it as Time in A1), generate a time axis for your simulation, which runs from time 0 to time 400 in 0.5 second increments. *The value of 0.5 is therefore the sampling interval* (h).

To do this, select spreadsheet cell A10 and enter the value 0. Press the down arrow to enter this data and select the next cell (A11), then enters 0.5. Select the two cells (A10 and A11), then position the mouse to point at the handle in the lower-right corner of selected range. The pointer will change to a cross. Drag the cross to the end of range (A810) to fill with the sequence of time and release the mouse button. You should now have a column of 800 time samples running from 0 to 400 seconds.

Setpoint (Column B)

The next step is to generate a setpoint profile for the PID control system. In this spreadsheet, the setpoint signal is in Column B (label it as w in B1). Set the setpoint from 0 at time 0, to 60 at time 0.5 until time 200, and then down to 40 at time 200.5 until time 400. Do this by using the same method as described for Column A.

Error (Column E)

To generate an error signal ($x_k = w - y_k$), enter an equation in Column E (label it as x_k in E1). The error signal is generated by subtracting the setpoint (w) from the process output, which is generated in Column D (label it as y_k in D1).

To do this, generate the first sample of the process output by selecting cell E11 and enter the following formula:

=B11-D11

When you press the return key, you will notice that the cell now contains the numerical difference between the values in the two preceding columns. Since Column D contains no data at this time, it defaults to 0, resulting in an error which is equal to the setpoint.

Position the mouse to point at the handle in the lower-right corner of cell E11; the pointer changes to a cross. Drag the cross to the end of range (E810) to fill Column E with the sequence of error and release the mouse button. You should now have 800 error signals in Column E.

Process Output (Column D)

Formulae in spreadsheet cells can refer to any other cells, and you will use this feature in generating the process output (Column D). *We will choose a delay of 2.5 second (5 samples) and a process gain (A) of 2 (see Equation (8.2)).*

Select column D10 and enter the following formula:

=2*C5

where Column C is v_k in Equation (8.2) (label it as v_k in C1). The formula you just entered reflects the gain and the delay of the process. Copy this formula from cell D10 to D810. Since there is no data in Column C, it defaults to 0.

You can test that this 'delay' works by entering any data into cells C10 to C20. You should see the result (delayed by 2.5 seconds and amplified by 2) in Column D.

Lag (Column F) and Controller Output (Column G)

Set the lag time constant in Column F (label it as T in F1) to 100 seconds (from F10 to F810).

Column G (label it as u_k in G1) is the controller output. The value is dependent on the calculation of Equation (8.6) – (8.8), with other specification, such as saturation, if any. For now, we will leave it empty.

Process (Column C)

Column C (label it as v_k in C1) represent the process/plant (see Figure 8.1). Select cell C11 and enter a formula corresponding to the difference equation in Equation (8.3). The input to your lag should be the controller output u_k for the corresponding time sample, i.e., cell G11. Fill in Column C with this equation.

P+I+D Controller (Column I)

Before designing our PID controller, we will configure the platform of the PID controller.

Column J is the proportional controller (label it as P_k in J1), column L is the integral controller (label it as I_k in L1), and column N is the derivative controller (label it as D_k in N1).

Column I is the PID controller (label it as PID in I1). Bearing in mind that PID is the summation of P, I, and D (see Equation (8.4) and (8.5)), configure Column I to give the sum P+I+D (sum of proportional term in Column J, integral term in Column L, and derivative term in Column N).

Now that we have configured the platform of PID controller, we can implement it to control the process. At this moment, we do not consider any constraint, such as saturation. Therefore, the controller u_k (Column G) is set as the same as PID controller (Column I).

Proportional Gain (Column H) and Proportional Controller (Column J)

In our first step, only P controller is used, with a gain determined by the value in the proportional gain (Column H, label it as G in H1). Generate the proportional term in Column J according to the formula of Equation(8.6). Set the proportional gain as 10.

Column L and Column N contain no data at the moment and they default to 0.

After filling Column J, the control loop will be closed with proportional only controller and a gain of 10.

Select the data in Column D and Column B with 'mouse + Ctrl', and then select 'Chart' from the 'Insert' menu to create a chart. Select Line type, which is a smooth line without points. Choose the data in Column A as the x-axis. You may choose other options in the chart menu to improve the presentation of your chart.

You should now have the graph of the closed-loop response of the system to setpoint steps from 0 → 60 → 40 against time.

Highlight this graph and print it. Comment on the resulting control error magnitude in your report.

Integral Time (Column K) and Integral Controller (Column L)

The second step will be to include the integral controller. The integral time is given in Column K (label it as T_i in K1). Add integral action to the controller by inserting the formula given in Equation (8.7) in Column L. Set the integral time as 30 seconds.

Create a chart similar to the proportional section to view the result and attach it to your report.

Create another chart to display the process output, y_k, error signal, x_k, controller output, u_k, and integral output, I_k, against time, and attach this chart to your report. Explain the new control performance and the values of x_k, I_k, and u_k when the control loop has stabilized at each set-point value.

Derivative Time (Column M) and Derivative Controller (Column N)

The third step will be to introduce the derivative controller. The derivative time is given in

Column M (label it as T_d in M1). Add derivative action to the controller by inserting the formula given in Equation (8.8) in Column N. Set the derivative time as 2.5. Observe especially the overshoot and the initial derivative kick when the setpoint changes.

Create the charts similar to the integral section and attach them to your report.

Tuning

The process is moderately tuned in the above simulation. Experiment by varying the Gain,T_i, and T_d values to achieve an improved performance in terms of faster response and small overshoot. Describe any improvements measured. Ziegler- Nichols tuning rules can be used to obtain good PID parameters based on the model of Equation (8.1).

Discuss all the points mentioned in the above notes and answer all the questions. The results of each stage of this simulation should be logged.

Simulation of the Actual System of a Ground Vehicle

Based on the steps given previously, simulate the performance of the system of the following ground vehicle with PID control in Microsoft Excel.

$$M\dot{v}(t) = u(t) - B_e v(t) + e(t) \tag{3.9}$$

where $v(t)$ is the velocity of the vehicle, $u(t)$ is the traction of the vehicle, $e(t)$ is the external disturbance, M is the mass, B_e is the damping coefficient, and M is the vehicle mass. We define $v(t)$ as the output to be controlled and $u(t)$ is the input.

(1). Discretize the above model (3.9) using backward difference equation.

(2). Design a PID control low using Ziegler-Nichols tuning rule to make the velocity track the desired value of $10\,m/s$. Set the disturbance as a constant, i.e., $e(t) = 100N$

The system parameters are defined as follows: $M = 1000kg$, $B_e = 15$, sampling interval = $0.1s$, and initial velocity = $0\,m/s$.

4. Adaptive Control

Adaptive control is concerned about controlling unknown systems using the time-varying adaptive laws that can tune the control gains online. In this section, a guided example will first be given to help students get familiar with the design steps for adaptive control with full state measurable, and then students should design adaptive control laws for a given system in Matlab/Simulink.

4.1 A Guided Example

We consider the adaptive control problem for a DC motor. The control objective is to make the angular position of the DC motor track the desired position. The overall blocks in Simulink is shown below.

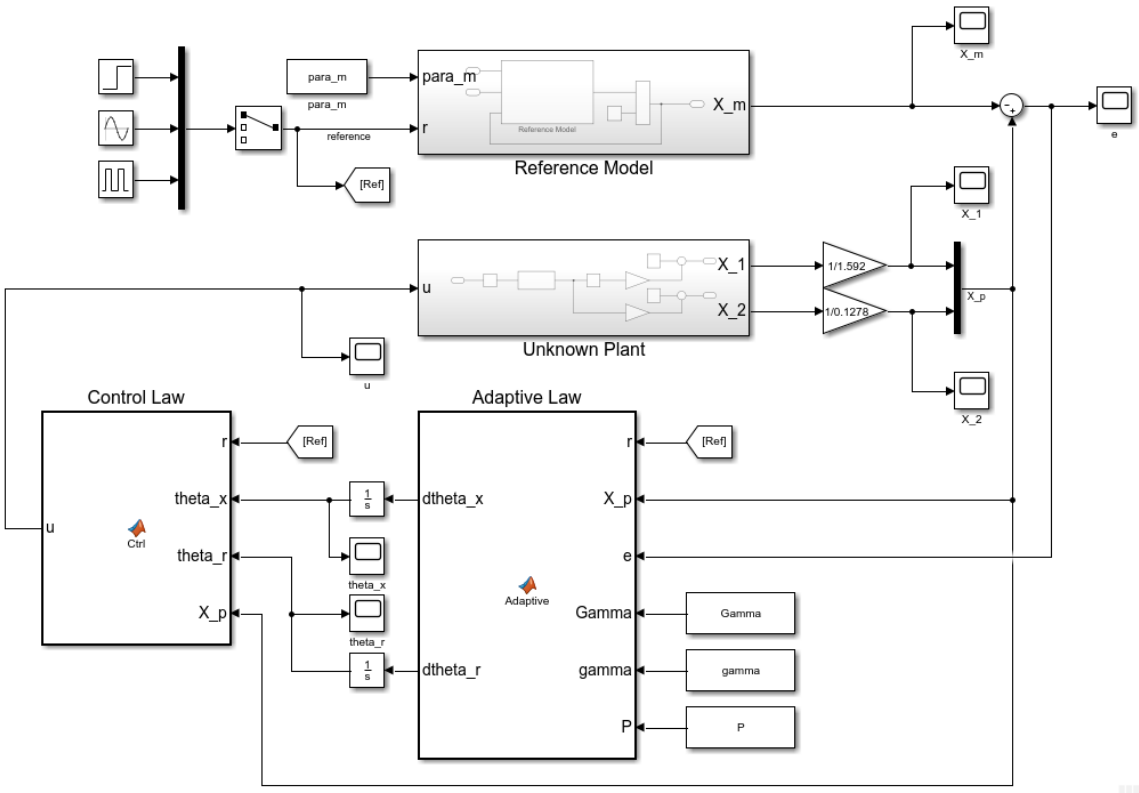

Figure. 4.1. Overall framework of adaptive control for a DC motor in Simulink

Procedures

Model of DC Motor

A DC motor can be simplified as the following second-order dynamics model.

$$\frac{Y(s)}{U(s)} = \frac{K}{s(1+\tau s)} \tag{4.1}$$

where $K = 6.2$, $\tau = 0.25$, Y is the angular position of the motor, and U is the control input which is voltage. We define the angular position and its time-derivative as the system state, i.e., $\boldsymbol{x}_p = [y, \dot{y}]$ There are two types of sensors for a DC motor, i.e. potentiometer for measuring the angular position, and tachometer for measuring the angular velocity. Note that these sensors measure the state in voltage, so we need to convert the measurements to the actual angular position and velocity. This DC model is used for simulation purpose only and cannot be used in control design.

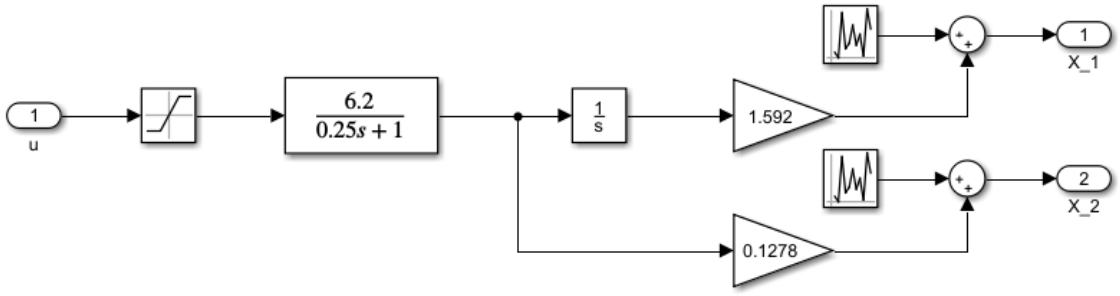

Figure. 4.2. Blocks of the unknown DC motor in Simulink

Build blocks and connect them together, as shown in Fig.4.2, to form the unknown plant to be controlled. To simulate the measurement noise, we can add uniform random number to the state. Two numbers 1.592 and 0.1278 are the calibration coefficients of the sensors. **This means that the outputs of the sensors need to be related to the actual angular position and velocity**. The saturation block in the forward pass is used to limit the input to the range of -5 to 5. Select all the blocks that you have built and right click to select 'Create Subsystem from Selection', so you will get exactly what is shown as 'Unknown Plant' in Fig.4.2.

Similar steps are applied to building the reference model, as shown below.

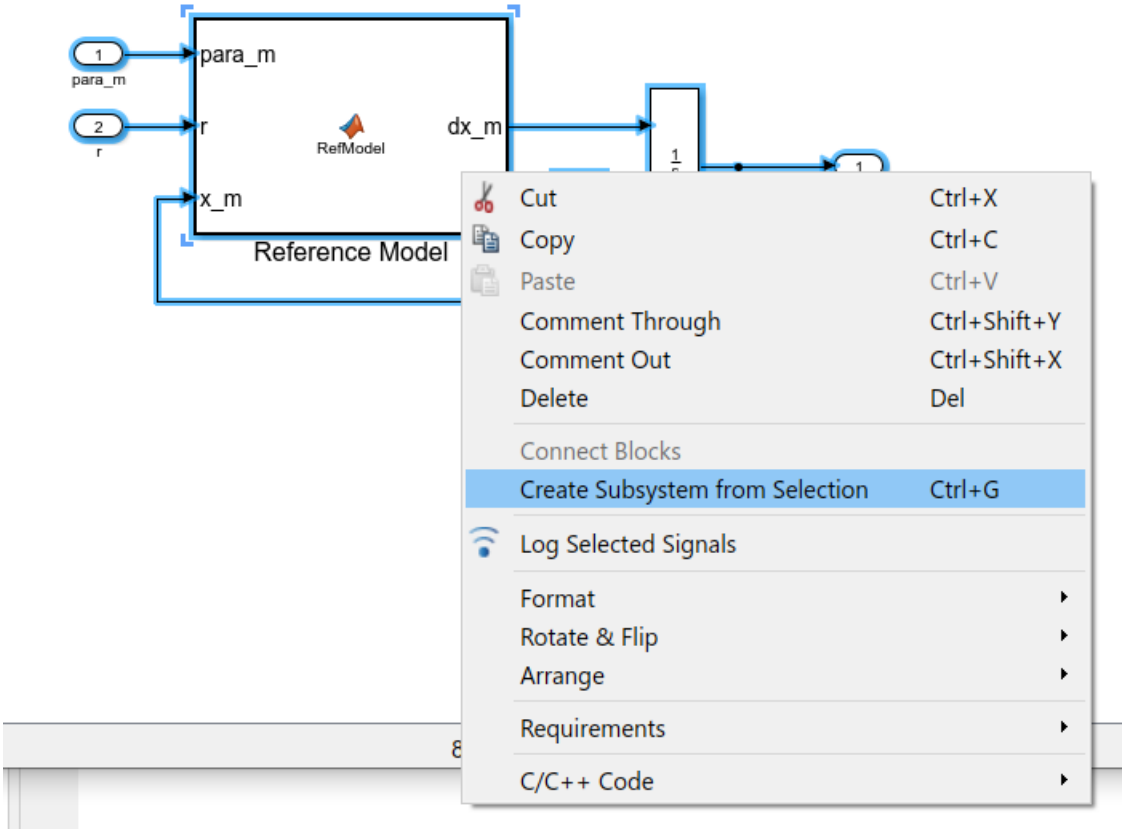

Figure. 4.3. Building subsystem for the selected blocks

Since the DC model (4.1) is unknown to the designer, we need a stable reference model to generate a reference state $\boldsymbol{x}_m$ that is a dynamic response to the desired position. As

such, the control objective is converted to making the state of the unknown DC model track the reference state.

$$\text{minimize } \boldsymbol{e} = \boldsymbol{x}_p - \boldsymbol{x}_m \tag{4.2}$$

We use 'Matlab Function' block to construct the reference model. Go to the top bar of the Simulink, click 'view', and then select 'Library Browser'. You can find the block under the menu of 'User-Defined Function', as shown below.

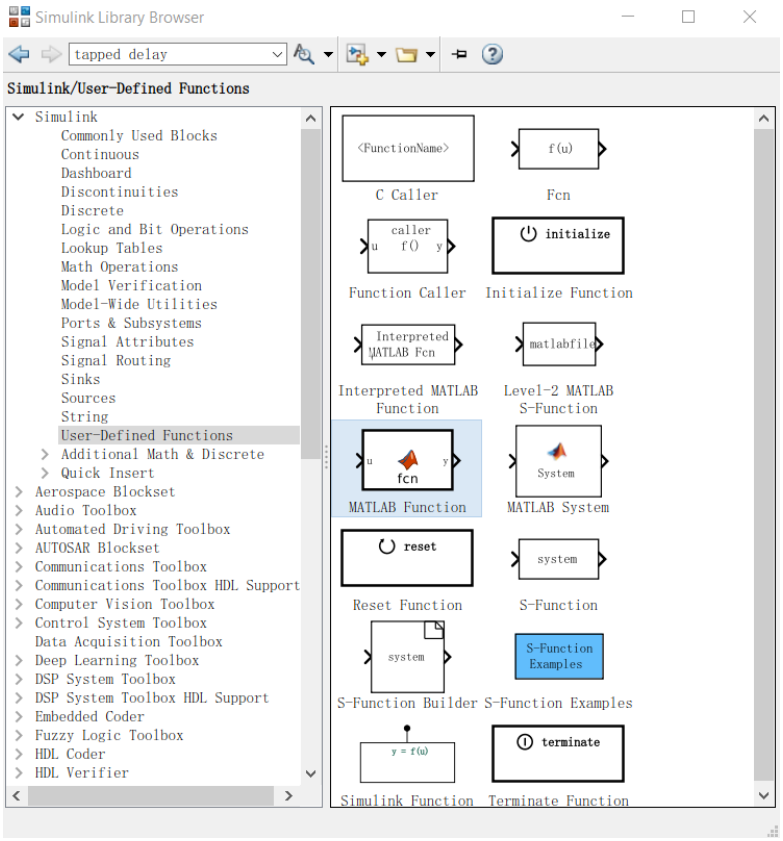

Figure 4.4. MATLAB Function in Simulink

Double click the selected 'MATLAB Function' block, enter the following code that represents a reference model in the corresponding m file.

```
function dx_m = RefModel(para_m, r, x_m)
ratio_m = para_m(1);
w_m     = para_m(2);
A_m     = [0,  1;
           -w_m^2, -2*ratio_m*w_m];
b       = [0; 1];
g_m     = w_m^2;
dx_m    = A_m*x_m+g_m*b*r;
```

Figure 4.5. Matlab code of the reference model

The '`para_m`' is the parameters of the reference model to be designed, '`r`' is the desired

angular position, and '`x_m`' is the state of the reference model.

Adaptive Control Law

The control law and adaptive law are written in two 'MATLAB Function' blocks. The adaptive law takes the following form.

$$\theta_x = \int_0^t -sgn(g)\ \Gamma\, \boldsymbol{e}^T \boldsymbol{P}\boldsymbol{b}\boldsymbol{x}_p \tag{4.4}$$

$$\theta_r = \int_0^t -sgn(g)\gamma\, \boldsymbol{e}^T \boldsymbol{P}\boldsymbol{b}r \tag{4.5}$$

where $\boldsymbol{\Gamma}$ is a positive definite matrix and γ is a positive constant, both of which are tuning parameters to be designed, $\boldsymbol{P}$ is a positive definite matrix obtained by solving the following Lyapunov equation.

$$\boldsymbol{A}_m^T \boldsymbol{P} + \boldsymbol{P}\boldsymbol{A}_m = -\boldsymbol{Q} \tag{4.6}$$

where $\boldsymbol{Q}$ is a positive definite matrix.

With the time-varying control gains θ_x and θ_r, the control law can be obtained by

$$u = \boldsymbol{\theta}_x^T \boldsymbol{x}_p + \theta_r r \tag{4.7}$$

Compute the perfect control gains using the following matching conditions.

$$\boldsymbol{A}_p + g\boldsymbol{b}\theta_x^{*T} = \boldsymbol{A}_m \tag{4.8}$$

$$g\boldsymbol{b}\theta_r^* = g_m\boldsymbol{b} \tag{4.9}$$

The above computation of the perfect gains can be done manually or using Matlab code.

Build two 'MATLAB Function' blocks named 'Adaptive Law' and 'Control Law' respectively. Enter the following code in respective m file of the block.

```
function [dtheta_x, dtheta_r] = Adaptive(r, X_p, e, Gamma, gamma, P)
sgn_g = 1;
b     = [0;1];
dtheta_x = -sgn_g*Gamma*(e.'*P*b)*X_p;
dtheta_r = -sgn_g*gamma*(e.'*P*b)*r;
```

Figure 4.6. Matlab code of the adaptive law

```
function u = Ctrl(r, theta_x, theta_r, X_p)
u = theta_x.'*X_p + theta_r * r;
```

Figure 4.7. Matlab code of the control law

Design of Parameters

Finally, we need to design the parameters used in the control law and the reference model, i.e., determining the values of $\zeta_m, \omega_m, \boldsymbol{Q}$, Γ, and γ. Build a new m file and enter the following code.

```
% Compute the control parameters
clear all
% parameters for reference model
ratio_m = 1;
w_m     = 2;
para_m  = [ratio_m, w_m];
A_m     = [0, 1;
           -w_m^2, -2*ratio_m*w_m];
Q       = diag([1, 100]);
P       = lyap(A_m, Q);
% parameters for adaptive law
Gamma = diag([1, 1]);
gamma = 10;
```

Figure 4.8. Matlab code of designing the parameters

4.2 Simulation of Adaptive Control of a DC Motor

Simulate the above guided example in Simulink by changing the parameters in Fig.4.8. Discuss any observation you have from the simulation. In addition, compare the control gains with the perfect gains when the tracking error ***e*** is minimized and becomes stable, and answer the following question by giving your detailed explanation.

Is it true that the control gains have to be exactly the perfect gains when the tracking error is minimized? Why?

5. Trajectory Tracking Control of a Marine Vehicle

Trajectory tracking plays a vital role in the wide applications of small autonomous underwater vehicles (AUVs) that are used in ocean, river, and lake explorations. In this section, we consider a trajectory tracking control problem of a small autonomous underwater vehicle using feedback linearization control technique. The structure of the underwater vehicle and the coordinates are defined below.

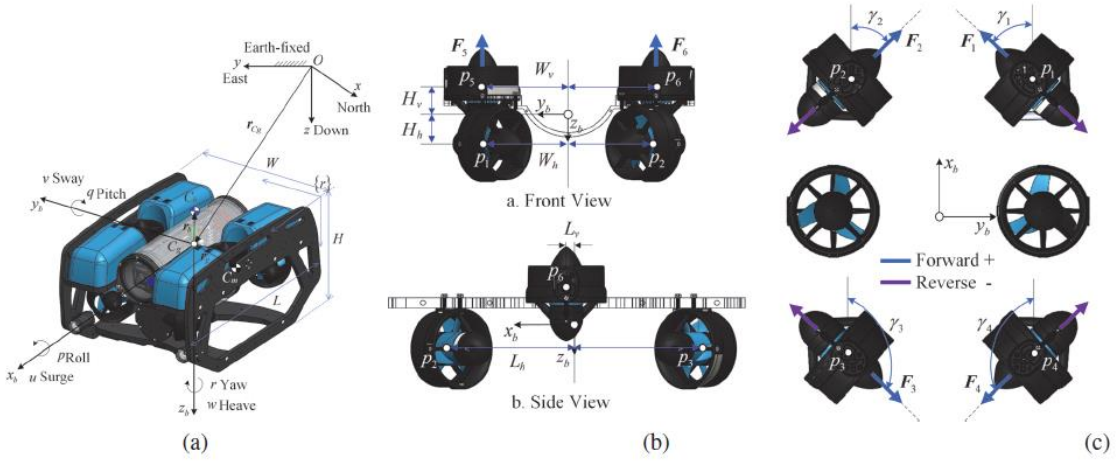

Figure 5.1. Overview of a small autonomous underwater vehicle

5.1 A Guided Example

To simplify the control design, we focus our attention on the horizontal motion. The planar dynamics of the vehicle takes the following form.

$$\ddot{\boldsymbol{\eta}}_h = -\boldsymbol{M}_{\eta}^{-1}\boldsymbol{C}_{\eta}\dot{\boldsymbol{\eta}}_h + \boldsymbol{M}_{\eta}^{-1}\boldsymbol{B}_{\eta}\boldsymbol{F}_h + \boldsymbol{d}_M \tag{5.1}$$

where $\boldsymbol{\eta}_h = [x, y, \psi]$, x and y are the planar coordinates of the center of mass in inertial frame, ψ is the yaw angle, $\boldsymbol{F}_h = [F_1, F_2, F_3, F_4]$ is the control force vector generated by four motors as shown in Fig.5.1.(c), $\boldsymbol{d}_M = -\boldsymbol{M}_A\dot{\boldsymbol{v}}_r - \boldsymbol{C}_{hA}\boldsymbol{v}_r - \boldsymbol{D}_h\boldsymbol{v}_r$ is the planar disturbance caused by ocean currents, the matrices $\boldsymbol{M}_h, \boldsymbol{C}_h$, and $\boldsymbol{B}_h$ are defined below.

$$\boldsymbol{M}_{\eta} = \boldsymbol{J}_h\boldsymbol{M}_{hRB}\boldsymbol{J}_h^{-1} = \begin{bmatrix} m & 0 & -m(y_g c\psi + x_g s\psi) \\ 0 & m & m(x_g c\psi - y_g s\psi) \\ -m(y_g c\psi + x_g s\psi) & m(x_g c\psi - y_g s\psi) & I_z \end{bmatrix},$$

$$\boldsymbol{C}_{\eta} = \boldsymbol{J}_h\boldsymbol{C}_{hRB}(\boldsymbol{v}_h^b)\boldsymbol{J}_h^{-1} - \boldsymbol{J}_h\boldsymbol{M}_{hRB}\boldsymbol{J}_h^{-1}\dot{\boldsymbol{J}}_h\boldsymbol{J}_h^{-1} = \begin{bmatrix} 0 & mr & -mc\psi(v + rx_g) - ms\psi(u - ry_g) \\ -mr & 0 & mc\psi(u - ry_g) - ms\psi(v + rx_g) \\ m(vc\psi + us\psi) & -m(uc\psi - vs\psi) & 0 \end{bmatrix},$$

$$\boldsymbol{B}_{\eta} = \boldsymbol{J}_h\boldsymbol{B}_h = \begin{bmatrix} c(\gamma_1 + \psi) & c\gamma_2 c\psi - s\gamma_1 s\psi & c\gamma_3 c\psi - s\gamma_1 s\psi & c\gamma_4 c\psi - s\gamma_1 s\psi \\ s(\gamma_1 + \psi) & c\psi s\gamma_1 + c\gamma_2 s\psi & c\psi s\gamma_1 + c\gamma_3 s\psi & c\psi s\gamma_1 + c\gamma_4 s\psi \\ L_h s\gamma_1 - W_h c\gamma_1 & W_h c\gamma_2 + L_h s\gamma_2 & W_h c\gamma_3 - L_h s\gamma_3 & -W_h c\gamma_4 - L_h s\gamma_4 \end{bmatrix},$$

where $c(\cdot)$ and $s(\cdot)$ represent $cos(\cdot)$ and $sin(\cdot)$, respectively, m is the mass, (x_g, y_g) is the coordinate of the center of mass in body frame, I_z is the moment of inertia around z axis, $\boldsymbol{\nu}_h = [u, v, r]$ is the velocity vector in body frame, (L_h, W_h) is the coordinate of motor in body frame, $(\gamma_1, \gamma_2, \gamma_3, \gamma_4)$ are constant angles of motors in body frame.

Given the reference trajectory $\boldsymbol{\eta}_{ref}, \dot{\boldsymbol{\eta}}_{ref}$, and $\ddot{\boldsymbol{\eta}}_{ref}$, the control objective of the trajectory tracking problem is to minimize the tracking error defined by the difference between the vehicle state and the reference.

$$\text{minimize } \boldsymbol{e} = \boldsymbol{\eta}_h - \boldsymbol{\eta}_{ref} \tag{5.2}$$

The feedback linearization control consists of a feedback term that drives the tracking error to zero, and a feedforward term that directly cancels out the nonlinear part of the vehicle model (5.1). We design the feedback term as a PID controller. Therefore, the control law takes the following form.

$$\boldsymbol{F}_h = \underbrace{\boldsymbol{B}_\eta^+ \boldsymbol{C}_\eta \dot{\boldsymbol{\eta}}_h + \boldsymbol{B}_\eta^+ \boldsymbol{M}_\eta \ddot{\boldsymbol{\eta}}_{ref}}_{\text{feedforward}} - \underbrace{\boldsymbol{B}_\eta^+ \boldsymbol{M}_\eta \left(k_p \boldsymbol{e} + k_i \int \boldsymbol{e} + k_d \dot{\boldsymbol{e}} \right)}_{\text{feedback}} \tag{5.3}$$

where $\boldsymbol{B}_\eta^+$ is Moore-Penrose inverse of $\boldsymbol{B}_\eta$, k_p, k_i, and k_d are the control gains.

5.2 Simulation of Trajectory Tracking Control of a Marine Vehicle

Simulate the above guided example in Simulink using 'MATLAB Function' introduced in the section 4. The reference trajectory is a circle on the water surface, defined as:

$$\boldsymbol{\eta}_{\text{ref}} = \begin{bmatrix} R_{\text{ref}} \cos\varphi \\ R_{\text{ref}} (1 + \sin\varphi) \\ \pi/2 + \varphi \end{bmatrix} \tag{5.4}$$

where $\varphi = \varphi_0 + r_{ref} t$ is the desired yaw angle, φ_0 is the initial desired angle, r_{ref} is the desired angular velocity, t is time, and R_{ref} is the circle radius.

In simulation, a 1st-order Gauss-Markov process is used to generate the unknown average current velocity. Therefore, the instant magnitude of current velocity V_c can be defined as:

$$\dot{V}_c(t) = \begin{cases} \omega(t) & V_{\min} \le V_c(t) \le V_{\max} \\ -\omega(t) & \text{otherwise,} \end{cases} \tag{5.5}$$

where $\omega(t)$ is a Gaussian white noise, V_{min} and V_{max} are the boundaries of current velocity. The initial current velocity should be $V_c(0) = \frac{1}{2}(V_{min} + V_{max})$.

The irrotational current velocity vector in body frame can be defined in terms of the yaw angle ψ and the sideslip angle β.

$$\boldsymbol{\nu}_{hc}^b = \begin{bmatrix} V_c \cos(\beta - \psi) \\ V_c \sin(\beta - \psi) \\ 0 \end{bmatrix} \tag{5.6}$$

where β is set as zero.

In the definition of disturbance $\boldsymbol{d}_M = -\boldsymbol{M}_A\dot{\boldsymbol{v}}_r - \boldsymbol{C}_{hA}\boldsymbol{v}_r - \boldsymbol{D}_h\boldsymbol{v}_r$, $\boldsymbol{v}_r = \boldsymbol{v}_h - \boldsymbol{v}_{hc}^b$ is the relative velocity in body frame. $\boldsymbol{v}_h$ is related to $\dot{\boldsymbol{\eta}}_h$ by the following kinematics model.

$$\dot{\boldsymbol{\eta}}_h = \boldsymbol{J}_h\boldsymbol{v}_h \tag{5.7}$$

where $\boldsymbol{J}_h$ is Jacobian matrix defined by

$$\boldsymbol{J}_h = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.8}$$

The matrices in $\boldsymbol{d}_M$ are defined by

$$\boldsymbol{M}_A = -\mathrm{diag}\,(X_{\dot{u}}, Y_{\dot{v}}, N_{\dot{r}})\,,\ \boldsymbol{C}_{hA}\,(\boldsymbol{v}) = \begin{bmatrix} 0 & 0 & Y_{\dot{v}}v \\ 0 & 0 & -X_{\dot{u}}u \\ -Y_{\dot{v}}v & X_{\dot{u}}u & 0 \end{bmatrix} \tag{5.9}$$

$$\boldsymbol{D}_h = -diag(X_u, Y_v, N_r) \tag{5.10}$$

where $X_u, Y_v, N_r, X_{\dot{u}}, Y_{\dot{v},}, N_{\dot{r}}$ are hydrodynamics coefficients.

All parameters required in simulation are set as below.

$m = 10, I_z = 2, x_g = y_g = 0.01, L_h = 0.145, W_h = 0.1, \gamma_1 = -\frac{\pi}{4}, \gamma_2 = \frac{\pi}{4}, \gamma_3 = \frac{3\pi}{4}, \gamma_4 = -\frac{3\pi}{4}$

$X_u = -7.2, Y_v = -7.7, N_r = -3, X_{\dot{u}} = -2.9, Y_{\dot{v},} = -3, N_{\dot{r}} = -3.3, R_{ref} = 5, r_{ref} = \frac{\pi}{60},$

$\varphi_0 = -\frac{\pi}{2}, V_{min} = 0, V_{max} = 0.1$

You need to do:

(1). Set different initial conditions of $\boldsymbol{\eta}_h$ and $\boldsymbol{v}_h$, as well as different V_{max};

(2). Tune the PID control gains to observe the tracking performance and also monitor the motor control force;

(3). Plot the vehicle planar trajectory, tracking error, and four control forces;

(4). Discuss any observation you have from the simulation.

6. Report

The results of each stage of the project should be logged, and important displays printed. Important data such as, the operating point, must be included in the report. In particular, there should be concise explanations and interpretation of the results thus obtained.

References

- Lecture notes for EE4307 (Control System Design and Simulation)
- L.Ljung. System Identification - Theory for the User, Prentice-Hall, Englewood Cliffs, N.J., 1987.
- T. Soderstrom and P. Stoica System identification, Prentice-Hall International, London, 1989.
- Astrom, K.J. & Wittenmark, B., “Computer-Controlled Systems”, 2nd Edition, Prentice Hall, 1990
- Rolf Johansson, System Modeling & Identification, Prentice-Hall, Englewood Cliffs, N.J., 1993
- Darrell Williamson, Digital Control and Implementation, Prentice-Hall International, Sydney, 1991.