

NEW YORK UNIVERSITY

TERM PROJECT

SIMULATION TOOLS AND SOFTWARE FOR MECHATRONICS AND ROBOTICS

ME-GY 6923

SUBMITTED BY:

BHARATKUMAR PAVULURU

TANAYA BHAVE

ANGAD ANIL BORALKAR

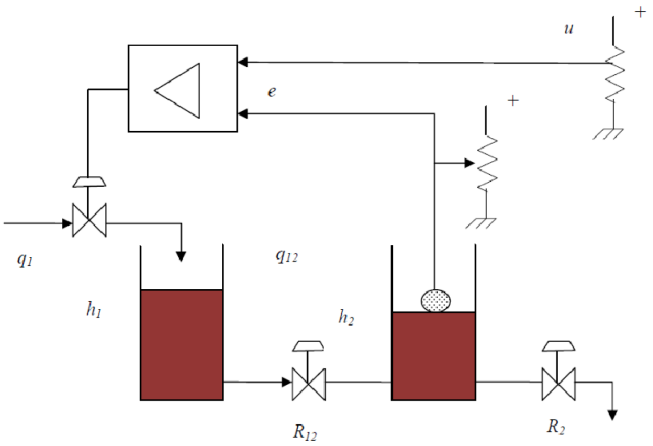
SYED AFFAN ALI

INDEX

PART I (SIMULINK)……(03)
Brief Description……(04)
Procedure……(04)
Schematic Diagrams……(06)
Block diagram of the SIMULINK model.……(06)
Block diagram for the Parameter e(t).……(07)
Block diagram for the Parameter $q_{12}(t)$.……(07)
Block diagram for the Parameter $q_e(t)$.……(08)
Block diagram for the Parameter $h_1(t)$.……(08)
Block diagram for the Parameter $h_2(t)$.……(09)
Graphs Obtained……(09)
Graph of $h_1(t)$ and $h_2(t)$ vs time.……(09)
Graph of $q_e(t)$ and $q_{12}(t)$ vs time.……(10)
Comments……(10)
PART II (MATLAB & VREP)……(11)
TASK 1……(11)
Procedure……(11)
MATLAB Code……(12)
MATLAB Output……(13)
Comment……13()
TASK 2……(14)
Procedure……(14)
MATLAB Code……(14)
MATLAB Output……(15)
Comment……(15)
TASK 3……(16)
Procedure……(16)
MATLAB Code……(16)
MATLAB Output……(17)
Comment……(17)
TASK 4……(18)
MATLAB Code……(18)
MATLAB Output……(18)
Commnet……(19)
TASK 5……(19)
Procedure……(20)
MATLAB Code……(20)
VREP Output……(21)
Comment……(21)

PART - I: SIMULINK

The system in Fig. 1 represents a height controller system for two tanks. The dynamic equations of the system include the nonlinear effects of the inlet flow $q_e(t)$ and the unidirectional flow between tanks $q_{12}(t)$.



The system equations are given as follows:

$$e(t) = K_t(u(t) - h_2(t))$$

$$q_e(t) = \begin{cases} min\,(K_g e(t), q_{emax}) & e(t) \geq 0 \\ 0 & e(t) < 0 \end{cases}$$

$$q_{12}(t) = \begin{cases} \dfrac{h_1(t) - h_2(t)}{R_{12}} & h_1(t) \geq h_2(t) \\ 0 & h_1(t) < h_2(t) \end{cases}$$

$$q_2(t) = \frac{h_2(t)}{R_2}$$

$$A_1 \cdot \frac{dh_1}{dt} = q_e(t) - q_{12}(t)$$

$$A_2 \cdot \frac{dh_2}{dt} = q_{12}(t) - q_2(t)$$

being $h_1(t)$, $h_2(t)$, $q_e(t)$, $q_{12}(t)$, $q_2(t)$ the tank-1 height, tank-2 height, input flow to tank 1, flow from tank-1 to tank-2, and output flow from tank-2, and R_{12} , R_2 , A_1, A_2 the hydraulic resistances and capacitances, K_t and K_g amplifier gains and q_{emax} a limiting value for q_e.

Simulate the dynamic behavior in SIMULINK for a reference input rectangular pulse train of height 2 m and period 20 s (with a duty cycle of 50% OFF and 50% ON) for the following parameter values R_{12} = 4, R_2 = 4, A_1 = 0.8, A_2 0.2, q_{emax} = 5, K_t = 1 and K_g = 5, plotting the evolution of $h_1(t)$, $h_2(t)$, $q_e(t)$, $q_{12}(t)$ and designing a block "subsystem" for each tank.

BRIEF DESCRIPTION :

The provided system is a height controller for the two water tanks. As described above, the system is bound by various parameters. The role of each parameter is as explained above and the numerical values of these parameters is as given. The task at hand is to develop a SIMULINK schematic of the system, satisfying each of the conditions given, to show the nonlinear effects of the inlet flow and unidirectional flow between the two tanks.

PROCEDURE:

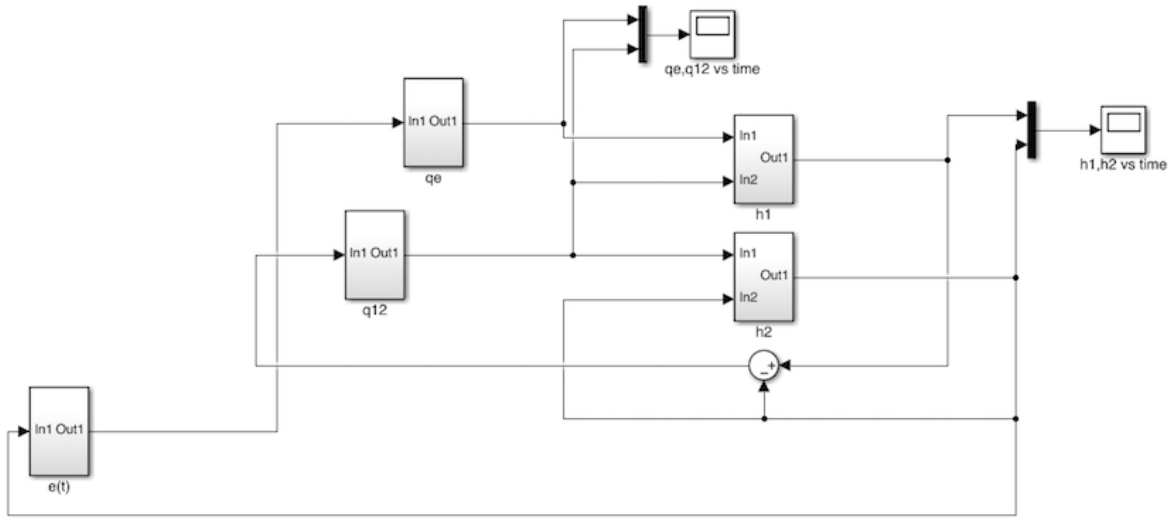
- As a first step, we analysed the conditions given in the problem and approached the SIMULINK model in such a way that each of the conditions is satisfied step-by-step.
- The first condition given is for the parameter e(t). In that respect, the first system block we made was that of e(t) with input as u(t).
- The input is given as a rectangular pulse of amplitude 2 m and period of 20 s. To get this required pulse we used a block from the SIMULINK library called the 'Pulse Generator' found under the 'Sources' tab. In this block, we set the amplitude of the pulse as 2 and the period as 20.
- After this, we subtract the parameter $h_2(t)$ from the output of the pulse generator block.This signal is then fed to a 'Gain' block found in the 'Commonly used blocks' tab of the SIMULINK library. The value of this gain is equal to the parameter K_t, as given in the problem. Thus we have successfully designed the e(t) block. After completing this, we select all the blocks used to construct the system block for parameter e(t) and create a subsystem with single output as the parameter e(t)
- Next we move to the second condition given in the problem for the parameter $q_e(t)$. To analyse $q_e(t)$, we need the parameter e(t) and check if this is *greater than/equal to* OR *less than* zero. Depending on this, we have the governing condition for $q_e(t)$. One condition being, if e(t) is *greater than/equal to* zero, we either select the value K_g*e(t) OR q_{emax}, whichever is minimum. Second condition states that if e(t) is *less than* zero then we select the value of $q_e(t)$ equal to zero.
- To construct the model for the first condition, we use a 'Gain' block (representing the parameter K_g) having input as parameter e(t). Then we use a constant block (representing the parameter q_{emax}) and assign the particular value to it.
- After this, we get a block 'MinMax' from the SIMULINK library and use it to get the minimum value of the two conditions described above. We connect the output of the gain block to one input terminals of the MinMax block and the output of the constant block to another input of the MinMax block.
- After this we set the condition of the MinMax block such that it gives the minimum of the two input values as the output. We then place a 'Switch' block dragging it from the

SIMULINK library and pasting it in the file. The switch block has three inputs and one output.

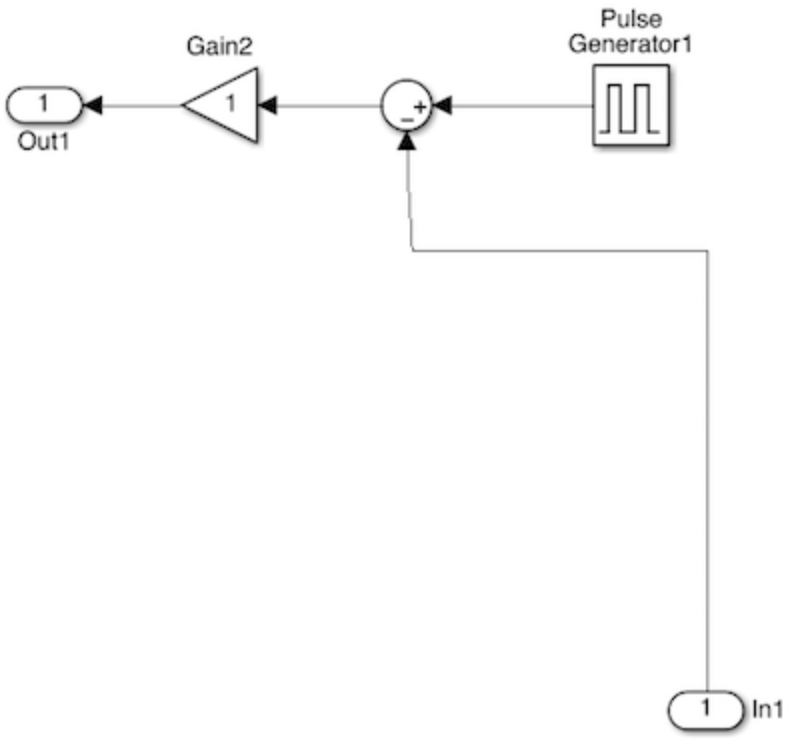
- As the name suggests, the switch block acts like a toggle between its inputs. The middle input is called the control port. It acts like a condition for the other two inputs. Double clicking the 'Switch' block reveals the properties of this block and we set the condition as 'u2>threshold' and set threshold value equal to zero.
- Out of the three inputs for the switch block, the first input is the output from the MinMax block. Second input (control port) is provided as e(t), as given in the problem. Third input is given as a constant block whose value is set to zero.
- The output of the 'Switch' block is the parameter $q_e(t)$ which satisfies all the conditions given in the problem. After completing this, we select all the blocks we used to define the parameter $q_e(t)$ and create a subsystem out of it with single output as the parameter $q_e(t)$.
- The next given parameter is $q_{12}(t)$. For this, we are given similar conditions as that of $q_e(t)$.
- To make a system model of this parameter, we have used a 'Switch' block to satisfy the first condition given for $q_{12}(t)$. In this switch block, of the three inputs, first input is the difference of $h_1(t)$ and $h_2(t)$ divided by the parameter R_{12}. To specify the value of R_{12} we have used a 'Gain' block.
- The second input (control port) for the switch block is only the difference between $h_1(t)$ and $h_2(t)$. The condition is set to 'u2>threshold' (threshold is set to zero).
- The third input for the switch block is given as a constant block with value zero. This is in accordance with the given conditions for the parameter $q_{12}(t)$.
- As a final output from the switch block, we get the parameter $q_{12}(t)$ which satisfies all the conditions given in the problem.
- After this, we select all the blocks used to define the parameter $q_{12}(t)$ and create a subsystem with single output as the parameter $q_{12}(t)$.
- The next parameter given is $q_2(t)$. To develop a model for it, we have integrated this parameter directly into the system block for parameter $h_2(t)$. For defining $h_2(t)$, we have used two inputs.
- One of the inputs is the parameter $q_{12}(t)$. The other parameter is $h_2(t)$ itself connected in feedback loop. For this we have used a 'Gain' block to define R_{12} (which will give $q_2(t)$ as output), another 'Gain' block to define parameter A_2 and an 'Integrator' block with output as the parameter $h_2(t)$.
- All these blocks are selected and a subsystem is created with single output as the parameter $h_2(t)$.
- The final block is that of parameter $h_1(t)$. For this, we have used two inputs, just as that for $h_2(t)$.

- One input is the parameter $q_{12}(t)$ and the other is the parameter $q_e(t)$. As the condition given for $h_2(t)$, we have taken the difference of $q_{12}(t)$ and $q_e(t)$ and divided it by the parameter A_1. Parameter A_1 is defined using a 'Gain' block.
- The output of the gain block is fed to an 'Integrator' block which gives the output as the parameter $h_1(t)$.
- After this, all the blocks used to define the parameter $h_1(t)$ are selected and a subsystem is made with single output as the parameter $h_1(t)$.
- To get the plots of $q_{12}(t)$ and $q_e(t)$, we have used a 'Mux' block to get a single vector representing both $q_{12}(t)$ and $q_e(t)$.
- To get the graph, we have connected the output of this block to the 'Scope' block. Thus we get a plot of $q_{12}(t)$ and $q_e(t)$ versus time.
- Similarly we obtain the plot of $h_1(t)$ and $h_2(t)$ versus time.
- This marks the completion of the first part (SIMULINK) of the term project.

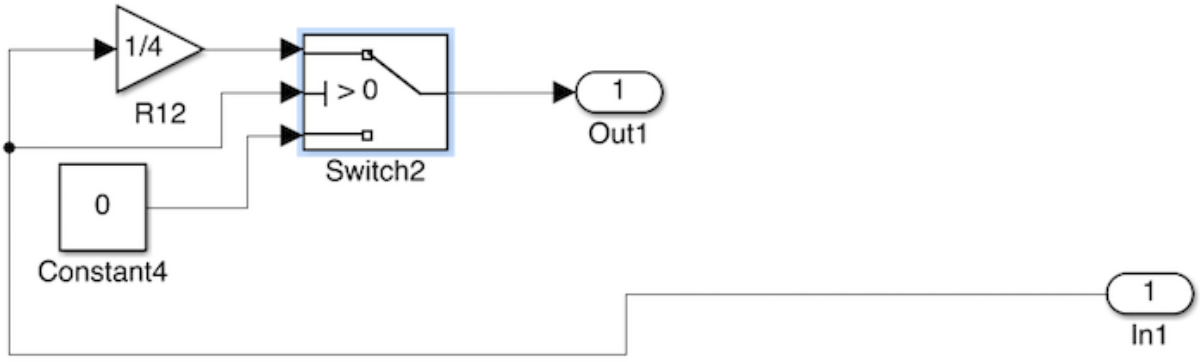
Schematic Diagrams:



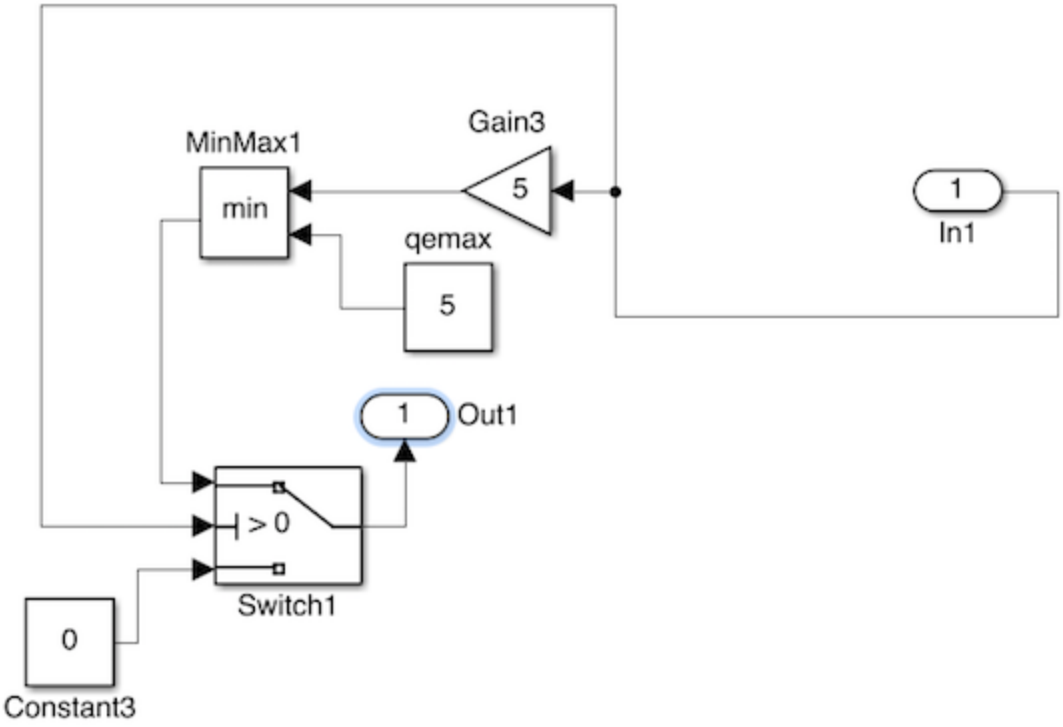
Block diagram of the SIMULINK model.



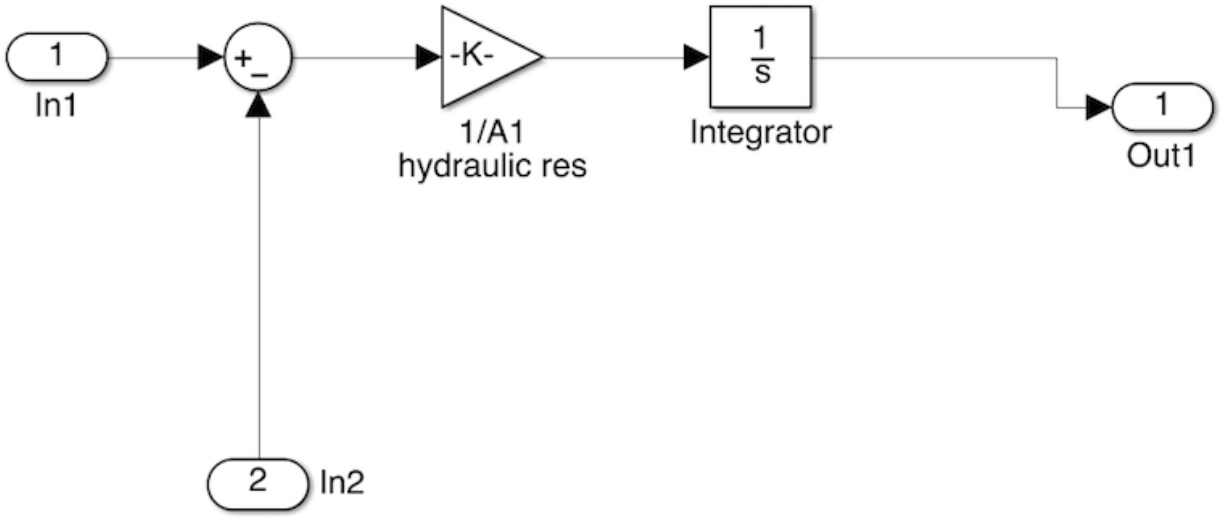
Block diagram for the Parameter e(t).



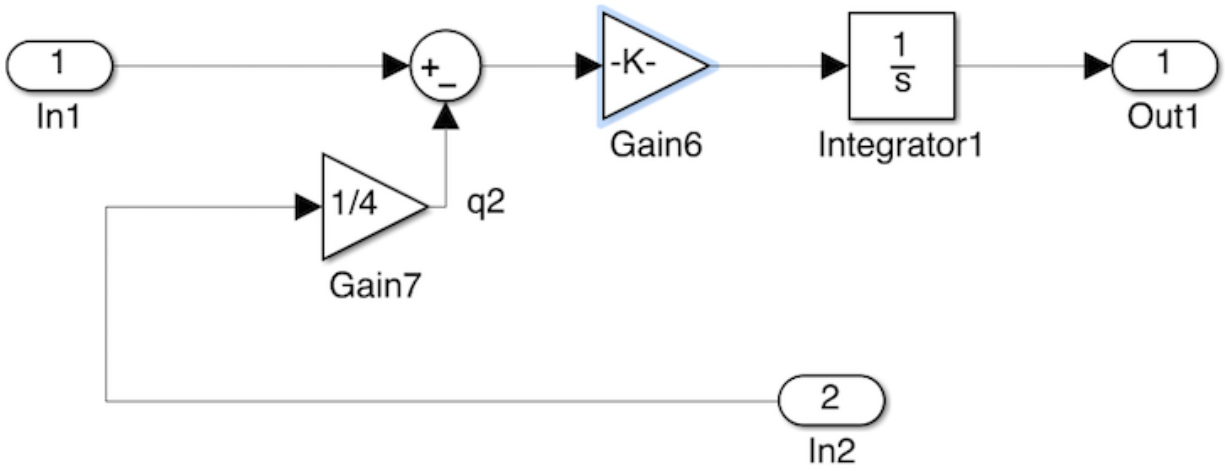
Block diagram for the Parameter $q_{12}(t)$.



Block diagram for the Parameter $q_e(t)$.

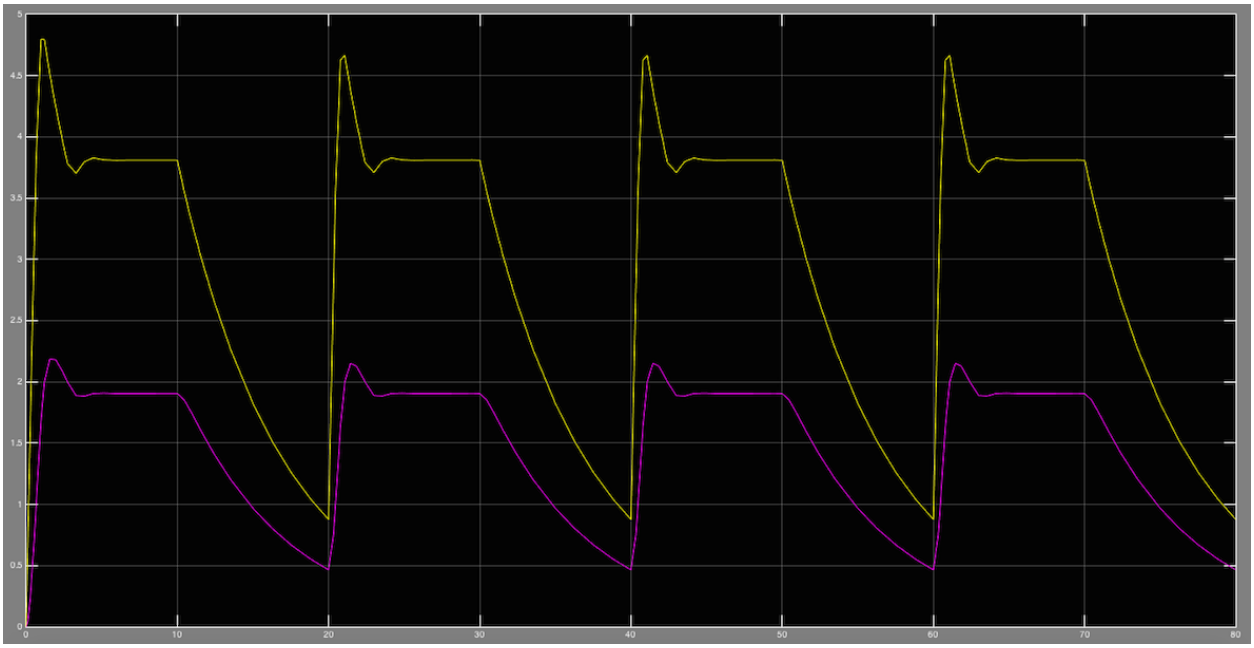


Block diagram for the Parameter $h_1(t)$.

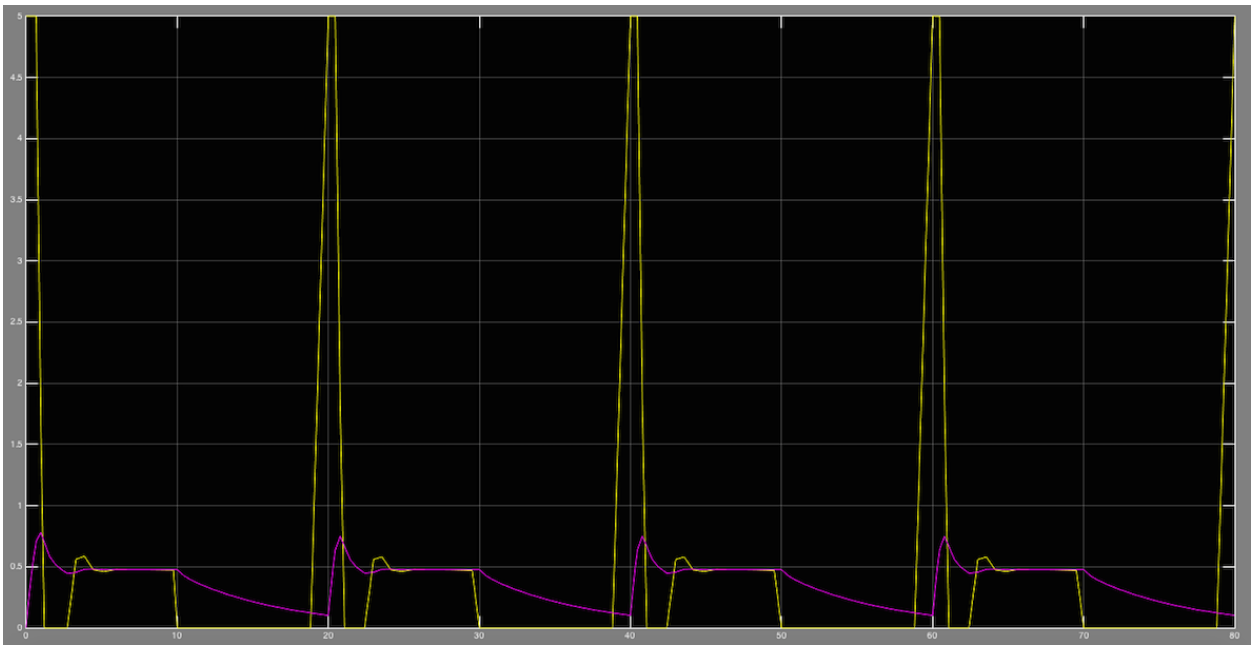


Block diagram for the Parameter $h_2(t)$.

Graphs Obtained:



Graph of $h_1(t)$ and $h_2(t)$ vs time.



Graph of $q_e(t)$ and $q_{12}(t)$ vs time.

COMMENTS:

- As seen from the graphs obtained, we have successfully created a SIMULINK model of the given dynamic system and obtained satisfactory results.

PART - II

MATLAB AND VREP

The exercise is to show that if a robot is given a map of the linear representations, it can localize itself based on the linear structures it perceives.

TASK I

The first task involves the extended Kalman filter for localization and is structured into a prediction step and update step. We predict the location of robot based on its former position and the executed movement.

State Prediction :

We allow x to be the state of the best estimated position of the robot. The control input to drive the robot is given by u.

In the first MATLAB code that is provided, is transitionFunction.m, it extracts the prior state of robot, x at time t-1 from the input data and predicts the state of robot, x at time t.

The function accepts the previous state of robot, control input and the distance between the wheels and gives the updated state of robot's position(f), the Jacobian of updated position with respect to the previous state (F_x) and control input (F_u). The variables used are in coherence with the data given by the book.

We employ the odometry model to predict the position of robot from the previous state and then update the state.

$$\hat{x}_t = f(x_{t-1}, u_t) = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} \frac{\Delta s_r + \Delta s_l}{2} \cos\left(\theta_{t-1} + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\ \frac{\Delta s_r + \Delta s_l}{2} \sin\left(\theta_{t-1} + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\ \frac{\Delta s_r - \Delta s_l}{b} \end{bmatrix},$$

Jacobian of the state has been calculated by using the formulae :

$$F_p = \nabla_p f = \nabla_p (f^T) = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial f}{\partial \theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\Delta s \sin(\theta + \Delta\theta/2) \\ 0 & 1 & \Delta s \cos(\theta + \Delta\theta/2) \\ 0 & 0 & 1 \end{bmatrix},$$

The following MATLAB code is used:

```
function [f, F_x, F_u] = transitionFunction(x,u, b)
% [f, F_x, F_u] = transitionFunction(x,u,b) predicts the state x at time t
given
% the state at time t-1 and the input u at time t. F_x denotes the
Jacobian
% of the state transition function with respect to the state evaluated at
% the state and input provided. F_u denotes the Jacobian of the state
% transition function with respect to the input evaluated at the state and
% input provided.
% State and input are defined according to "Introduction to Autonomous
Mobile Robots", pp. 337

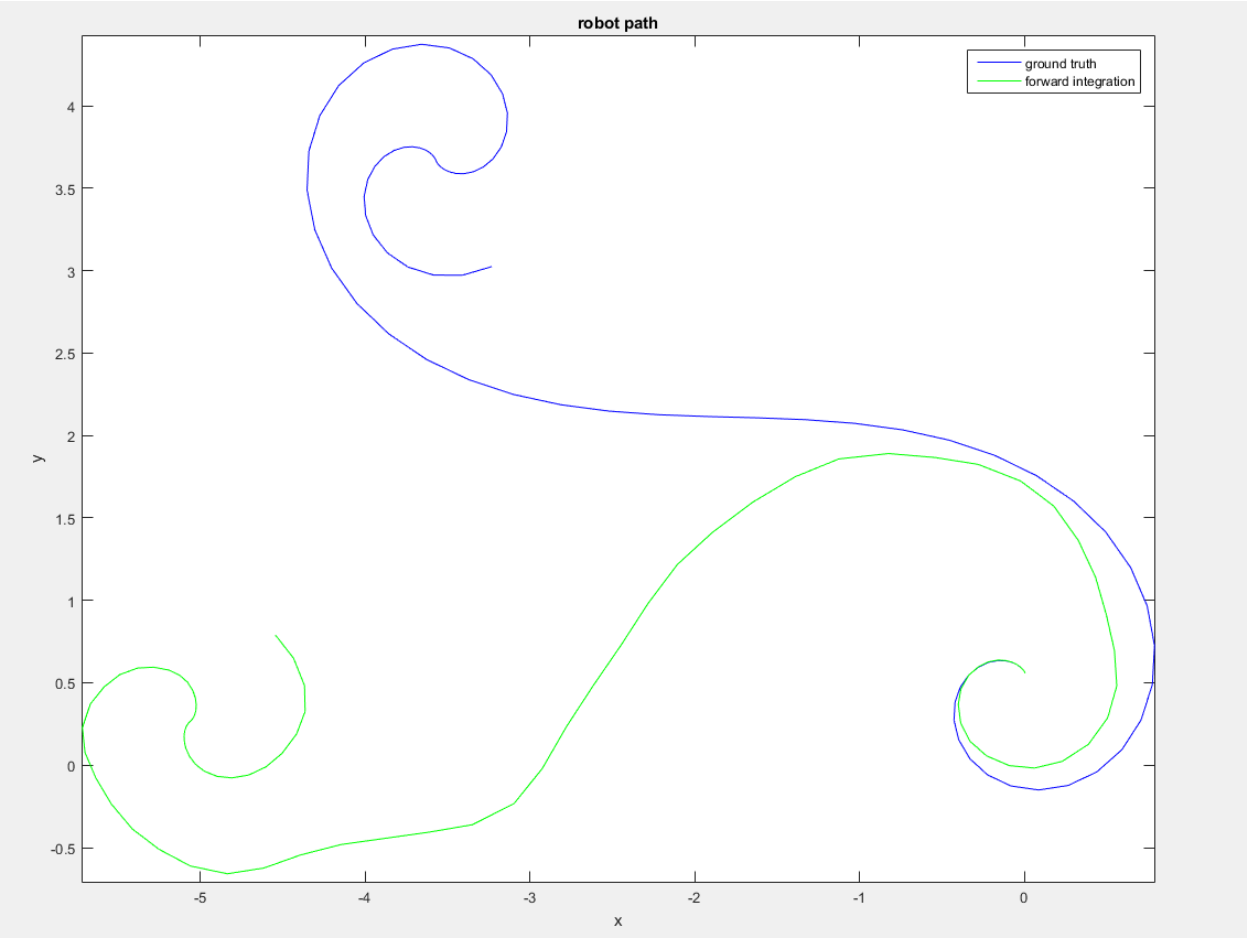
%initializing and defining the parameters
dsl = u(1,1);
dsr = u(2,1);
ds = (dsl+dsr)/2;
dtheta = (dsr-dsl)/b;
xu = zeros(3,1);

%measuring the state prediction f
xu(1,1) = x(1,1) + ((dsl+dsr)/2)* cos(x(3,1) + (dsr-dsl)/(2*b));
xu(2,1) = x(2,1) + ((dsl+dsr)/2)* sin(x(3,1) + (dsr-dsl)/(2*b));
xu(3,1) = x(3,1) + (dsr-dsl)/b;
f = xu;

%computing jacobian F_x and F_u
F_x=[1 0 -ds * sin(x(3,1) + dtheta/2);0 1 ds * cos(x(3,1) + dtheta/2);0 0
1];
F_u(1,2) = 0.5 * cos(x(3,1) + dtheta/2) - (ds/(2*b))*sin(x(3,1) +
dtheta/2);
F_u(1,1) = 0.5 * cos(x(3,1) + dtheta/2) + (ds/(2*b))*sin(x(3,1) +
dtheta/2);
F_u(2,2) = 0.5 * sin(x(3,1) + dtheta/2) + (ds/(2*b))*cos(x(3,1) +
dtheta/2);
F_u(2,1) = 0.5 * sin(x(3,1) + dtheta/2) - (ds/(2*b))*cos(x(3,1) +
dtheta/2);
F_u(3,2) = 1/b;
F_u(3,1) = -1/b;

end
```

The MATLAB code gives the following output :



The graph hence generated shows two paths. The blue path indicates the predicted trajectory of the robot whereas the green path shows the actual trajectory the robot follows after the updation of the position of the robot.

COMMENT :

- As seen from the trajectory diagram, we have successfully completed the assigned task and obtained satisfactory results.

TASK II:

The second task is to edit the 'measurementFunction.m' file involving parameterization of lines. The parameterization applied acts on the output of our perception system as well as entries of the map M given in the input file. The map M contains all linear structures in its operating environment.

PROCEDURE:

From the given function, we extract 'x' and 'm' from the given data (in the file validationData.mat file). Here 'x' is prior state and 'm' denotes map lines. Also 'h' is the measurement prediction. These are the uncertainties of line parameters 'alpha' and 'r'. 'h' is in model state from :

$$\hat{z}_t^j = \begin{bmatrix} \hat{\alpha}_t^j \\ \hat{r}_t^j \end{bmatrix} = h^j(\hat{x}_t, m^j) = \begin{bmatrix} {}^{\{W\}}\alpha_t^j - \hat{\theta}_t \\ {}^{\{W\}}r_t^j - (\hat{x}_t \cos({}^{\{W\}}\alpha_t^j) + \hat{y}_t \sin({}^{\{W\}}\alpha_t^j)) \end{bmatrix},$$

The function also generates the jacobian of h with respect to the prior state x as H_x

$$H^j = \begin{bmatrix} \frac{\partial \alpha_t^j}{\partial \hat{x}} & \frac{\partial \alpha_t^j}{\partial \hat{y}} & \frac{\partial \alpha_t^j}{\partial \hat{\theta}} \\ \frac{\partial r_t^j}{\partial \hat{x}} & \frac{\partial r_t^j}{\partial \hat{y}} & \frac{\partial r_t^j}{\partial \hat{\theta}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & -1 \\ -\cos({}^{\{W\}}\alpha_t^j) & -\sin({}^{\{W\}}\alpha_t^j) & 0 \end{bmatrix},$$

The following MATLAB code is used:

```
function [h, H_x] = measurementFunction(x, m)
% [h, H_x] = measurementFunction(x, m) returns the predicted measurement
% given a state x and a single map entry m. H_x denotes the Jacobian of
the
% measurement function with respect to the state evaluated at the state
% provided.
% Map entry and state are defined according to "Introduction to Autonomous
Mobile Robots" pp. 337

%computing the state updates
h = [m(1)-x(3);m(2)-(x(1)*cos(m(1))+x(2)*sin(m(1)))];

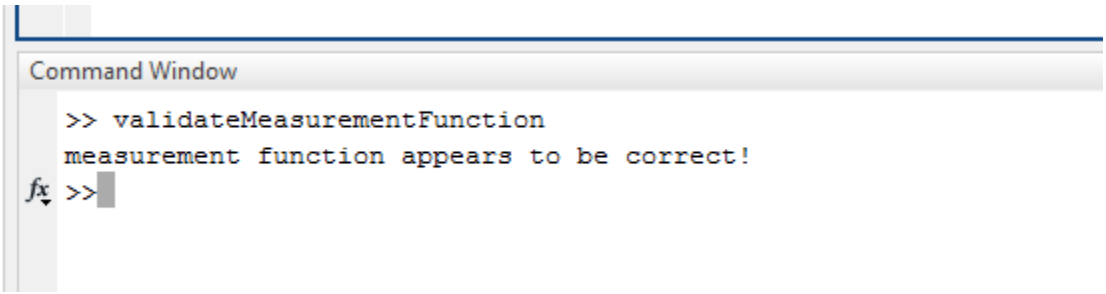
%computing the jacobian for m and x
```

```
H_x = [0, 0, -1; -cos(m(1)),-sin(m(1)),0];

%we callbach normalize parameter function
[h(1), h(2), isRNegated] = normalizeLineParameters(h(1), h(2));

if isRNegated
    H_x(2, :) = - H_x(2, :);
end
```

The output generated by the following code :



COMMENT:

- Thus we have completed the given task and as it is displayed in the command window of MATLAB, our measurement function appears to be correct.

TASK III

The third task requires us to update the establishments of associations between observations and the map entries correctly. We employ the Mahalanobis distance between a predicted measurement and an observation to fulfil the task. The difference between the predicted and observed measurement.

To find the Mahalanobis distance, we use the innovation covariance which is calculated as:

$$\mathbf{v}_t^{ij} = \mathbf{z}_t^j - \hat{\mathbf{z}}_t^i$$

The innovation covariance:

$$\mathbf{\Sigma}_{IN_t}^{ij} = \hat{\mathbf{H}}_t^i \cdot \hat{\mathbf{P}}_t \cdot \hat{\mathbf{H}}_t^{iT} + \mathbf{R}_t^j$$

Mahalanobis distance is calculated using:

$$d_t^{ij} = \mathbf{v}_t^{ijT} \cdot \left(\mathbf{\Sigma}_{IN_t}^{ij}\right)^{-1} \cdot \mathbf{v}_t^{ij}$$

The MATLAB code is:

```
function [h, H_x] = measurementFunction(x, m)
% [h, H_x] = measurementFunction(x, m) returns the predicted measurement
% given a state x and a single map entry m. H_x denotes the Jacobian of
the
% measurement function with respect to the state evaluated at the state
% provided.
% Map entry and state are defined according to "Introduction to Autonomous
Mobile Robots" pp. 337

%computing the state updates
h = [m(1)-x(3);m(2)-(x(1)*cos(m(1))+x(2)*sin(m(1)))];

%computing the jacobian for m and x
H_x = [0, 0, -1; -cos(m(1)),-sin(m(1)),0];

%we callbach normalize parameter function
[h(1), h(2), isRNegated] = normalizeLineParameters(h(1), h(2));

if isRNegated
    H_x(2, :) = - H_x(2, :);
end
```

The Output is as:

```
>> validateAssociations
measurement function appears to be correct!
association function appears to be correct!
```

COMMENT:

- The output displaying that function appears to be correct serves as a validation for our approach.

TASK IV

Once we have all the measurements obtained from the previous three tasks i.e. including the estimated state and noise. The idea is that the cycle predict / update, predict / update, ... is repeated for as many time steps as we like. Now In order for validation, a sequence of

observations, an initial value for the state estimates that can just be our first observation, an initial value for the prediction error.

The MATLAb code along with the output is presented below.

The MATLAB code is:

```
function [x_, P_] = filterStep(x, P, u, Z, R, M, k, g, b)
% [x_, P_] = filterStep(x, P, u, z, R, M, k, g, b)
% returns an a posteriori estimate of the state and its covariance
x_prior=x;
P_prior=P;

if size(Z,2) == 0
    x_ = x_prior;
    P_ = P_prior;
    return;
end

%covariance of noise
Q=[k*abs(u(1,:)) 0; 0 k*abs(u(2,:))];


[f, F_x, F_u] = transitionFunction(x,u, b);
x=f;

%computing covariance of robot motion
P =F_x*P*F_x'+ F_u* Q*F_u';

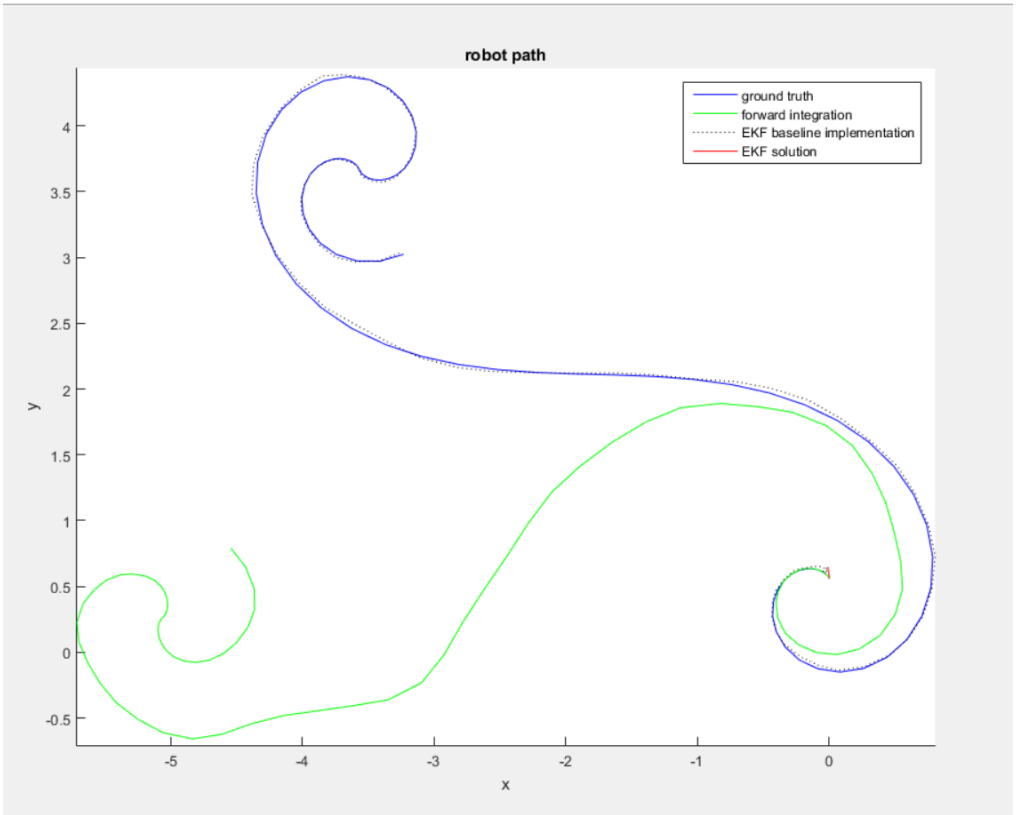

[v, H, R] = associateMeasurements(x, P, Z, R, M, g);

R = blockDiagonal(R);
i=size(v,2);

H=permute(H,[1 3 2]);
H=reshape(H,2*i,3);
v=reshape(v,2*i,1);


In=H*P*H'+R;
kt=P*H'*inv(In);
x_=x+kt*v;
P_=P-kt*In*kt';
end
```

The output generated by the above mentioned code is :



COMMENT:

- Since we had an additional sensor output and more accurate map, our solution represents that it estimated the information better than the before.

TASK V

Since we had the data from line extraction and EKF localization separately, we combine them for together implementation.

We write the function incremntalLocalization.m that takes the input of all the states combined with the control inputs and the map. The function in the end returns the latest state of the robot.

The MATLAB Code along with the output is shown below:

The MATLAB Code is:

```
function [x_posterori, P_posterori] = incrementalLocalization(x, P, u, S,
M, params, k, g, b)
% [x_posterori, P_posterori] = incrementalLocalization(x, P, u, S, R, M,
% k, b, g) returns the a posterori estimate of the state and its
covariance,
% given the previous state estimate, control inputs, laser measurements
and the map

C_TR = diag([repmat(0.1^2, 1, size(S, 2)) repmat(0.1^2, 1, size(S, 2))]);


[Z,R,segends]=extractLinesPolar(S(1,:),S(2,:),C_TR,params);


[x_posterori, P_posterori] = filterStep(x, P, u, Z, R, M, k, g, b);

end
```

The output based on the above function is

