SYSTEM SIMULATION AND MODELING LAB

(EC-715-F)

LAB MANUAL

VII SEMESTER

DRONACHARYA
College of Engineering

Department of Electronics & Computer Engg.
Dronacharya College Of Engineering
Khentawas, Gurgaon – 123506

SYSTEM SIMULATION AND MODELING LAB
LIST OF EXPERIMENTS

EXPERIMENT NO: 1

AIM: - Computer Generation of Random Numbers.

SOFTWARE/APPARATUS REQUIRED: - MATLAB R2013A**,** Personal Computer

THEORY: - When you create random numbers using software, the results are not random in a strict, mathematical sense. However, software applications, such as MATLAB®, use algorithms that make your results appear to be random and independent. The results also pass various statistical tests of randomness and independence. These apparently random and independent numbers are often described as *pseudorandom* and *pseudo independent*. You can use these numbers as if they are truly random and independent. One benefit of using pseudorandom, pseudo independent numbers is that you can repeat a random number calculation at any time. This can be useful in testing or diagnostic situations.
Although repeatability can be useful, it is possible to accidentally repeat your results when you really want different results. There are several ways to avoid this problem. The documentation contains several examples that show how to ensure that your results are different when that is your intention.

Use the `rand`, `randn`, and `randi` functions to create sequences pseudorandom numbers. Use the `rng` function to control the repeatability of your results. Use the `RandStream` class when you need more advanced control over random number generation.

Functions

`rand`	Uniformly distributed pseudorandom numbers
`randn`	Normally distributed pseudorandom numbers
`randi`	Uniformly distributed pseudorandom integers
`randperm`	Random permutation

DISCUSSION:-

This example shows how to avoid repeating the same random number arrays when MATLAB® restarts. This technique is useful when you want to combine results from the same random number commands executed different MATLAB sessions. All the random number functions, `rand`, `randn`, `randi`, and `randperm`, draw values from a shared random number generator. Every time you start MATLAB, the generator resets itself to the same state. Therefore, a command such as `rand(2,2)` returns the same result any time you execute it immediately following startup. Also, any script or function that calls the random number functions returns the same result whenever you restart.
One way to get different random numbers is to initialize the generator using a different seed every time. Doing so ensures that you don't repeat results from a previous session.

Execute the `rng('shuffle')` command once in your MATLAB session before calling any of the random number functions.

```
rng('shuffle')
```

You can execute this command in a MATLAB Command Window, or you can add it to your startup file, which is a special script that MATLAB executes every time you restart.

Now, execute a random number command.

```
A = rand(2,2);
```

Each time you call `rng('shuffle')`, it reseeds the generator using a different seed based on the current time. Alternatively, specify different seeds explicitly. For example,

```
rng(1);
A = rand(2,2);
rng(2);
B = rand(2,2);
```

Arrays `A` and `B` are different because the generator is initialized with a different seed before each call to the `rand` function.

QUIZ:-

Q1. What are random numbers?

EXPERIMENT NO: 2

AIM: - Chi-square goodness-of-fit test.
SOFTWARE/APPARATUS REQUIRED: - MATLAB R2013A**,** Personal Computer.

THEORY: -

PROGRAM:
```
h = chi2gof(x)
h = chi2gof(x,Name,Value)
```

DISCUSSION:-

`h = chi2gof(x)` returns a test decision for the null hypothesis that the data in vector `x` comes from a normal distribution with a mean and variance estimated from `x`, using the chi-square goodness-of-fit test. The alternative hypothesis is that the data does not come from such a distribution. The result `h` is `1` if the test rejects the null hypothesis at the 5% significance level, and `0` otherwise.

`h = chi2gof(x,Name,Value)` returns a test decision for the chi-square goodness-of-fit test with additional options specified by one or more name-value pair arguments. For example, you can test for a distribution other than normal, or change the significance level of the test.

QUIZ:-

Q1. What is chi square test?
Q2. What is goodness of fit test?

EXPERIMENT NO: 3

AIM: - One-sample Kolmogorov-Smirnov test

SOFTWARE/APPARATUS REQUIRED: - MATLAB R2013A**,** Personal Computer

THEORY: -

PROGRAM:-
```
h = kstest(x)
h = kstest(x,Name,Value)
```

DISCUSSION:- `h = kstest(x)` returns a test decision for the null hypothesis that the data in vector `x` comes from a standard normal distribution, against the alternative that it does not come from such a distribution, using the one-sample Kolmogorov-Smirnov test. The result `h` is `1` if the test rejects the null hypothesis at the 5% significance level, or `0` otherwise.

`h = kstest(x,Name,Value)` returns a test decision for the one-sample Kolmogorov-Smirnov test with additional options specified by one or more name-value pair arguments. For example, you can test for a distribution other than standard normal, change the significance level, or conduct a one-sided test.

`[h,p] = kstest(__)` also returns the *p*-value `p` of the hypothesis test, using any of the input arguments from the previous syntaxes.

`[h,p,ksstat,cv] = kstest(__)` also returns the value of the test statistic `ksstat` and the approximate critical value`cv` of the test.

QUIZ:-

Q1. Explain One-sample Kolmogorov-Smirnov test.

EXPERIMENT NO: 4

AIM: - Test for Standard Normal Distribution

SOFTWARE/APPARATUS REQUIRED: - MATLAB R2013A, Personal Computer

PROGRAM: Load the sample data. Create a vector containing the first column of the students' exam grades data.

```
load examgrades;
test1 = grades(:,1);
```

Test the null hypothesis that the data comes from a normal distribution with a mean of 75 and a standard deviation of 10. Use these parameters to center and scale each element of the data vector since, by default, `kstest` tests for a standard normal distribution.

```
x = (test1-75)/10;

h = kstest(x)
```

```
h =

        0
```

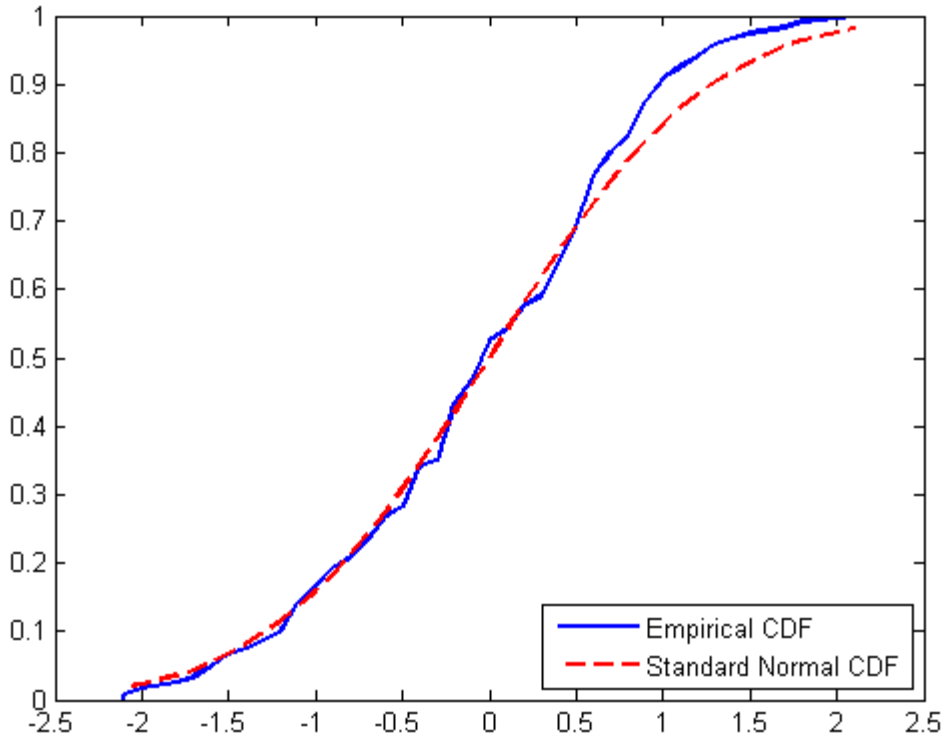
The returned value of `h = 0` indicates that `kstest` fails to reject the null hypothesis at the default 5% significance level. Plot the empirical cumulative distribution function (cdf) and the standard normal cdf for a visual comparison.

```
[f,x_values] = ecdf(x);

F = plot(x_values,f);

set(F,'LineWidth',2);
hold on;
G = plot(x_values,normcdf(x_values,0,1),'r-');
set(G,'LineWidth',2);
legend([F G],...
       'Empirical CDF','Standard Normal CDF',...
       'Location','SE');
```

RESULTS- The plot shows the similarity between the empirical CDF of the centered and scaled data vector and the CDF of the standard normal distribution



QUIZ:-

Q1. Define Standard Normal Distribution.

EXPERIMENT NO: 5

AIM: - Testing Random Number Generators.

SOFTWARE/APPARATUS REQUIRED: - MATLAB R2013A**,** Personal Computer

Theroy:- Random numbers are widely used ingredient in the simulation of almost all discrete systems. Simulation languages generate random numbers that are used to generate event times and other random variables. Random number generators have applications in gambling, statistical sampling, computer simulation, cryptography, completely randomized design and other areas where producing an unpredictable result is desirable. The generation of pseudo random numbers is an important and common task in computer programming.

Syntax

- `h = runstest(x)`
- `h = runstest(x,v)`
- `h = runstest(x,'ud')`
- `h = runstest(__,Name,Value)`
- `[h,p,stats] = runstest(__)`

Description

`h = runstest(x)` returns a test decision for the null hypothesis that the values in the data vector `x` come in random order, against the alternative that they do not. The test is based on the number of runs of consecutive values above or below the mean of `x`. The result `h` is `1` if the test rejects the null hypothesis at the 5% significance level, or `0` otherwise.

`h = runstest(x,v)` returns a test decision based on the number of runs of consecutive values above or below the specified reference value `v`. Values exactly equal to `v` are discarded.

`h = runstest(x,'ud')` returns a test decision based on the number of runs up or down. Too few runs indicate a trend, while too many runs indicate an oscillation. Values exactly equal to the preceding value are discarded.

`h = runstest(__,Name,Value)` returns a test decision using additional options specified by one or more name-value pair arguments. For example, you can change the significance level of the test, specify the algorithm used to calculate the *p*-value, or conduct a one-sided test.

`[h,p,stats] = runstest(__)` also returns the *p*-value of the test `p`, and a structure `stats` containing additional data about the test.

Examples

Test Data for Randomness Using Sample Median

Input Arguments

x — Data vectorvector of scalar values
v — Reference valuemean of `x` (default) | scalar value

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name`must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as`Name1,Value1,...,NameN,ValueN`.

Example:

'Alpha' — Significance level`0.05` (default) | scalar value in the range (0,1)
'Method' — Method used to compute *p*-value`'exact'` | `'approximate'`
'Tail' — Type of alternative hypothesis`'both'` (default) | `'right'` | `'left'`

Output Arguments

h — Hypothesis test result`1` | `0`
p — *p*-valuescalar value in the range [0,1]
stats — Test datastructure

QUIZ:-

Q1. How can you test random numbers?

EXPERIMENT NO: 6

AIM: - Monte-Carlo Simulation.

SOFTWARE/APPARATUS REQUIRED: - MATLAB R2013A**,** Personal Computer

THEORY: **-** Monte Carlo simulation is a method for exploring the sensitivity of a complex system by varying parameters within statistical constraints. These systems can include financial, physical, and mathematical models that are simulated in a loop, with statistical uncertainty between simulations. The results from the simulation are analyzed to determine the characteristics of the system.

Common tasks for performing Monte Carlo analysis include:

- Varying uncertain parameters for your model
- Creating dynamic simulations and alter parameters with statistical uncertainty
- Creating a Monte Carlo simulation to model a complex dynamic system
- Distributing simulations between processor cores and individual PCs to speed analysis
- Analyzing data through robust plotting and advanced statistical methods

Syntax

```
simsd(sys,data)
simsd(sys,data,N)
simsd(sys,data,N,opt)
y = simsd(sys,data,N,opt)
[y,y_sd] = simsd(sys,data,N,opt)
```

Description

`simsd(sys,data)` simulates and plots the response of 10 perturbed realizations of the identified model, `sys`. Simulation input data, `data`, is used to compute the simulated response.
The parameters of the perturbed realizations are consistent with the parameter covariance of the original model, `sys`.
`simsd(sys,data,N)` simulates and plots the response of `N` perturbed realizations of the identified model, `sys`.
`simsd(sys,data,N,opt)` simulates the system response using the option set, `opt`, to specify simulation behavior.
`y = simsd(sys,data,N,opt)` returns the simulation result as a cell array, `y`. No simulated response plot is produced.
`[y,y_sd] = simsd(sys,data,N,opt)` also returns the estimated standard deviation, `y_sd`, for the simulated response.
The parameter changes in the randomly selected models are scaled to be small (ca 0.1%) compared to the parameter values. The response changes are then scaled up to correspond to one standard deviation. The scaling does not apply to free delays of `idproc` or `idtf` models.

Input Arguments

`sys`	Identified linear model.
`data`	Simulation input data. Specify `data` as a time- or frequency-domain `iddata` object, with input channels only. For time-domain simulation of discrete-time systems, `data` may also be specified as a matrix whose columns correspond to each input channel.
`N`	Number of perturbed realizations for simulation. Specify `N` as a positive integer. **Default:** `10`
`opt`	Simulation options. opt is an option set, created using `simsdOptions`, that specifies options including: Signal offsets Initial condition handling Additive noise

Output Arguments

`y`	Simulated response. `y` is a cell array of *N*+1 elements, where *N* is the number of perturbed realizations. `y{1}` contains the nominal response for `sys`. The remaining elements contain the simulated response for the *N* perturbed realizations.
`y_sd`	Estimated standard deviation of the simulated response. `y_sd` is derived by averaging the simulations results in `y`.

QUIZ:-

Q1. What is monte carlo simulation?

Q2. What are the applications of monte carlo simulation?

EXPERIMENT NO: 7

AIM: - Simulation of Single Server Queuing System.

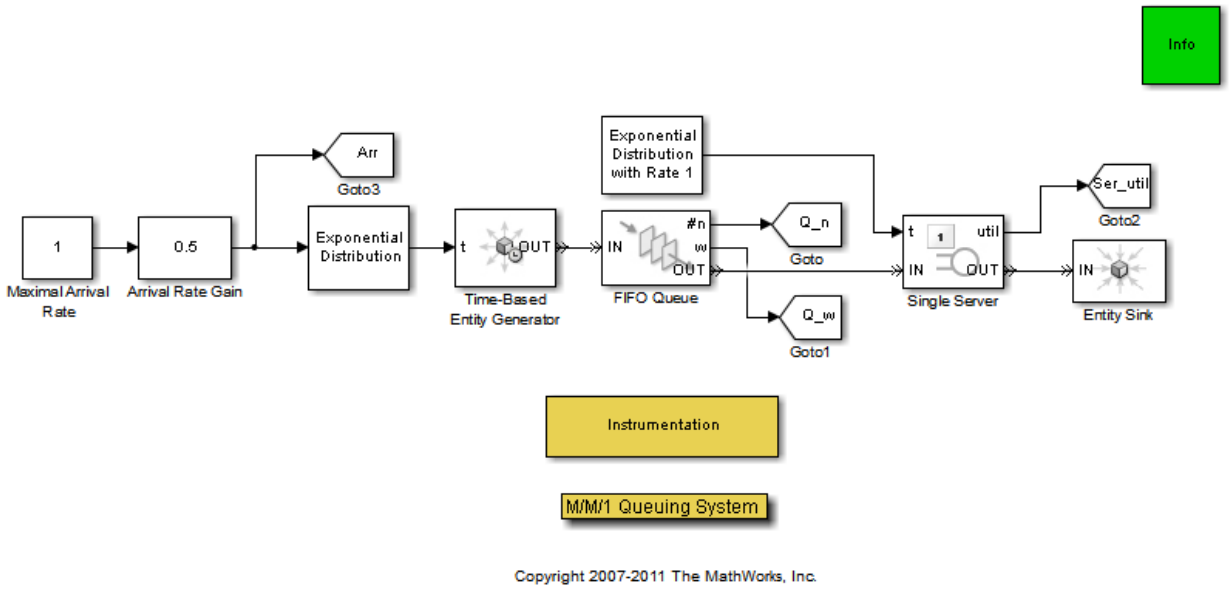
SOFTWARE/APPARATUS REQUIRED: - MATLAB R2013A**,** Personal Computer

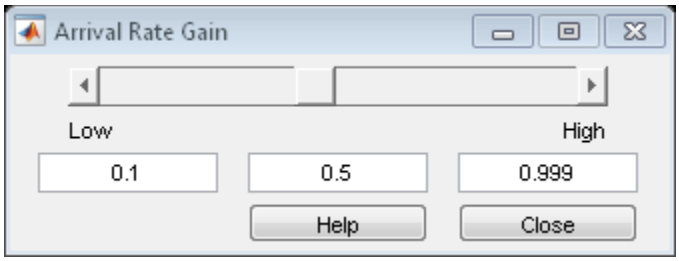
THEORY: - This example shows how to model a single-queue single-server system with a single traffic source and an infinite storage capacity. In the notation, the M stands for Markovian; M/M/1 means that the system has a Poisson arrival process, an exponential service time distribution, and one server. Queuing theory provides exact theoretical results for some performance measures of an M/M/1 queuing system and this model makes it easy to compare empirical results with the corresponding theoretical results.

Structure

The model includes the components listed below:

- **Time Based Entity Generator block:** It models a Poisson arrival process by generating entities (also known as "customers" in queuing theory).
- **Exponential Interarrival Time Distribution subsystem:** It creates a signal representing the interarrival times for the generated entities. The interarrival time of a Poisson arrival process is an exponential random variable.
- **FIFO Queue block:** It stores entities that have yet to be served.
- **Single Server block:** It models a server whose service time has an exponential distribution.

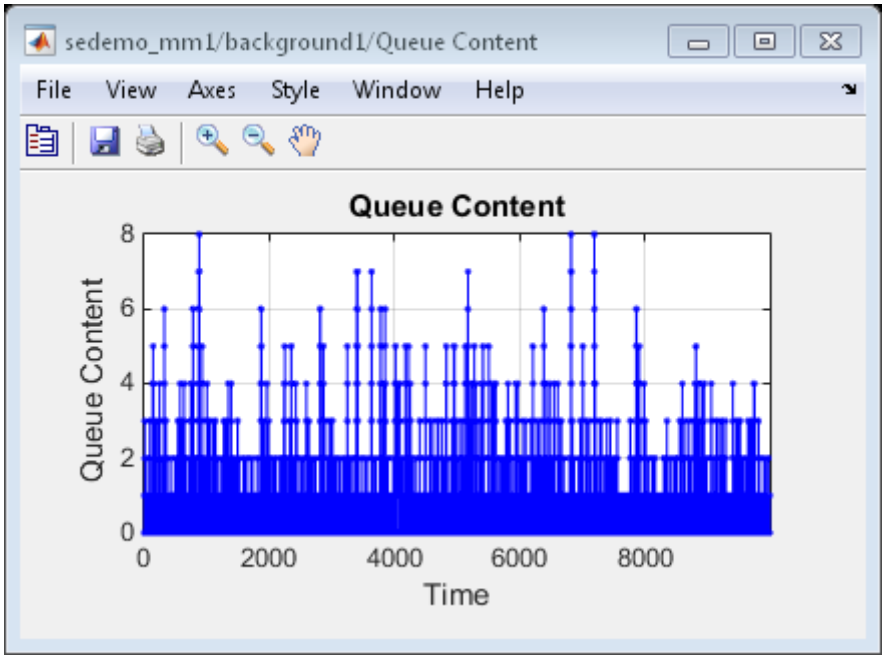


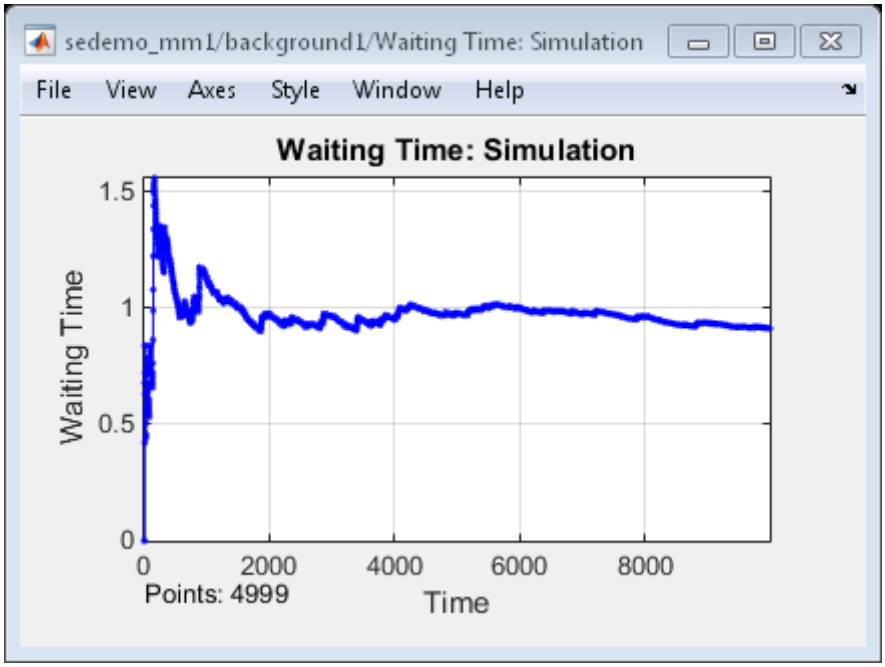


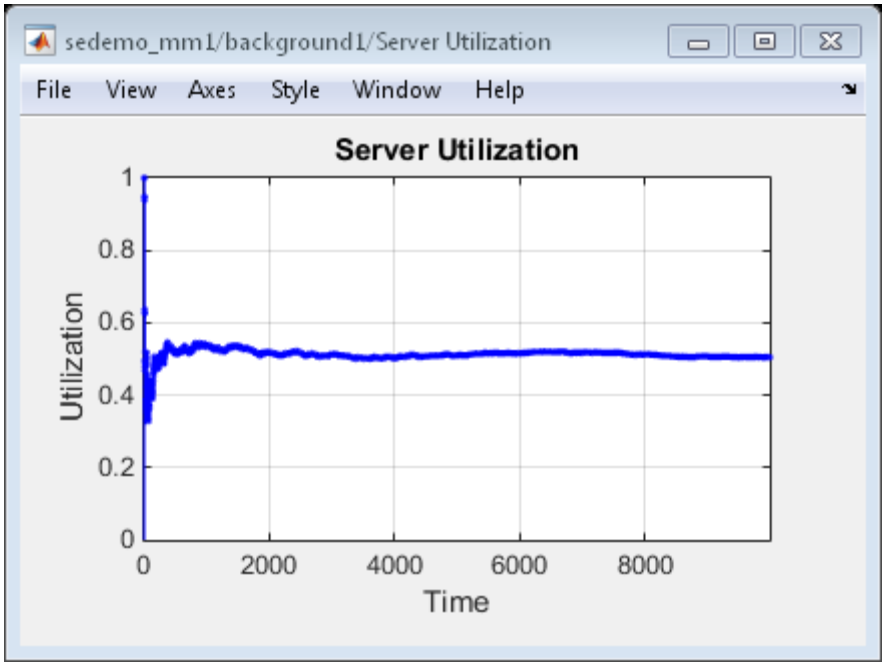
Results

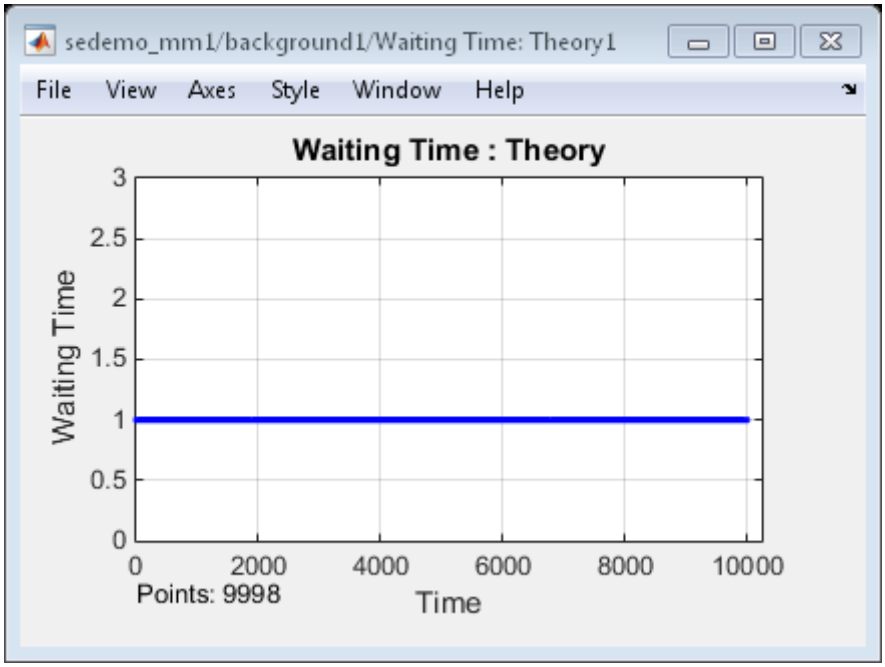
The model includes these visual ways to understand its performance:

- Display blocks that show the waiting time in the queue and the server utilization
- A scope showing the number of entities (customers) in the queue at any given time
- A scope showing the theoretical and empirical values of the waiting time in the queue, on a single set of axes. You can use this plot to see how the empirical values evolve during the simulation and compare them with the theoretical value.




sedemo_mm1/background1/Waiting Time: Simulation
File View Axes Style Window Help
Waiting Time: Simulation
Waiting Time
1.5
1
0.5
0
0 2000 4000 6000 8000
Points: 4999
Time


sedemo_mm1/background1/Server Utilization
File View Axes Style Window Help
Server Utilization
Utilization
1
0.8
0.6
0.4
0.2
0
0 2000 4000 6000 8000
Time



QUIZ:-

Q1. What is Single Server Queuing System?

EXPERIMENT NO: 8

AIM: - Simulation of Two-Server Queuing System.

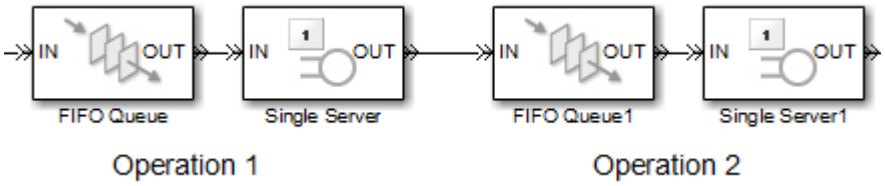
SOFTWARE/APPARATUS REQUIRED: - MATLAB R2013A**,** Personal Computer

THEORY: - Here are some examples of ways to combine FIFO Queue and Single Server blocks to model different situations:

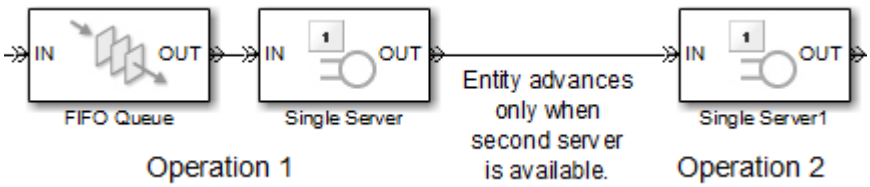
- Serial Queue-Server Pairs
- Parallel Queue-Server Pairs as Alternatives
- Parallel Queue-Server Pairs in Multicasting
- Serial Connection of Queues
- Parallel Connection of Queues

Serial Queue-Server Pairs

Two queue-server pairs connected in series represent successive operations that an entity undergoes. For example, parts on an assembly line are processed sequentially by two machines.

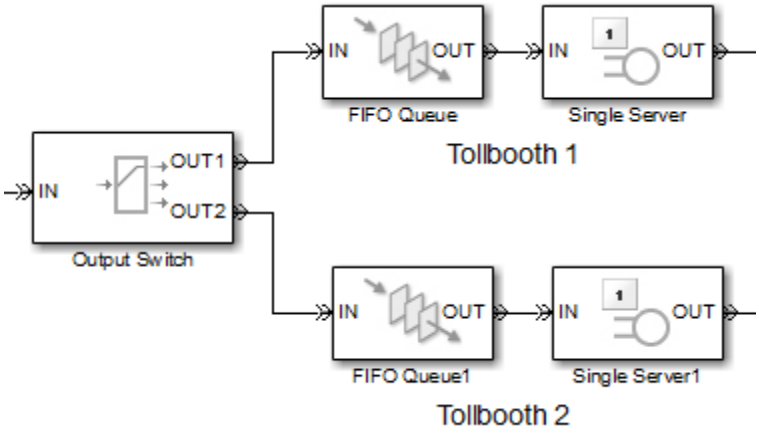


While you might alternatively model the situation as a pair of servers without a queue between them, the absence of the queue means that if the first server completes service on an entity before the second server is available, the entity must stay in the first server past the end of service and the first server cannot accept a new entity for service until the second server becomes available.



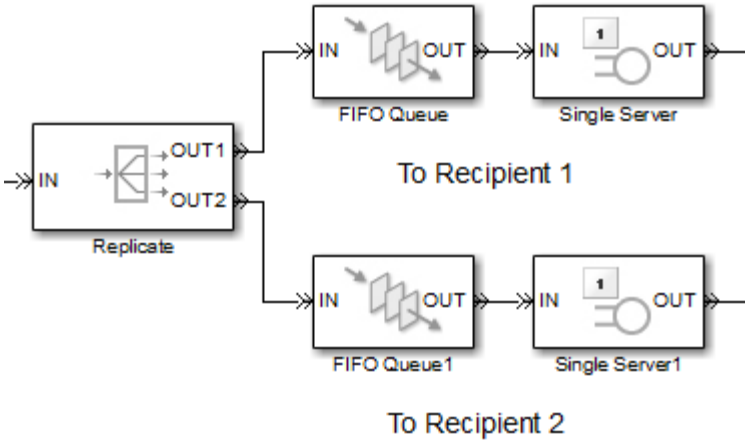
Parallel Queue-Server Pairs as Alternatives

Two queue-server pairs connected in parallel, in which each entity arrives at one or the other, represent alternative operations. For example, vehicles wait in line for one of several tollbooths at a toll plaza.



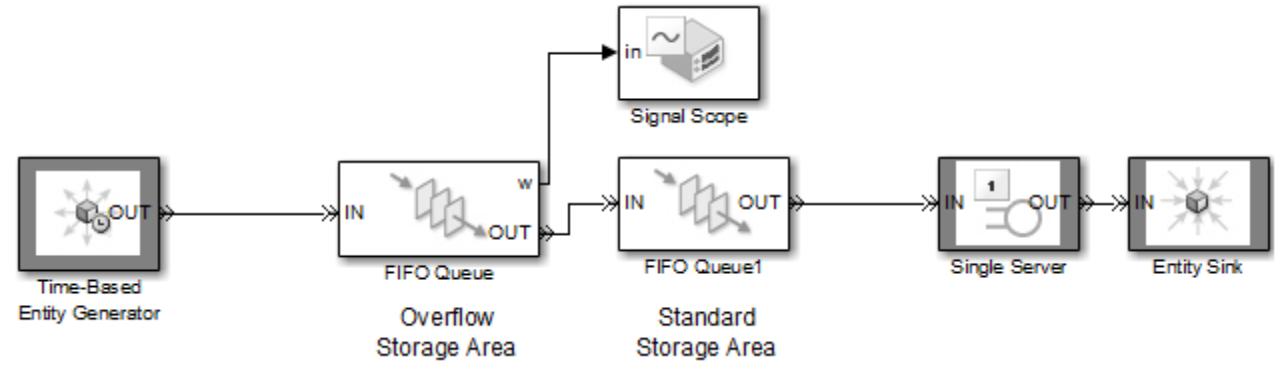
Parallel Queue-Server Pairs in Multicasting

Two queue-server pairs connected in parallel, in which a copy of each entity arrives at both, represent a multicasting situation such as sending a message to multiple recipients. Note that copying entities might not make sense in some applications.



Serial Connection of Queues

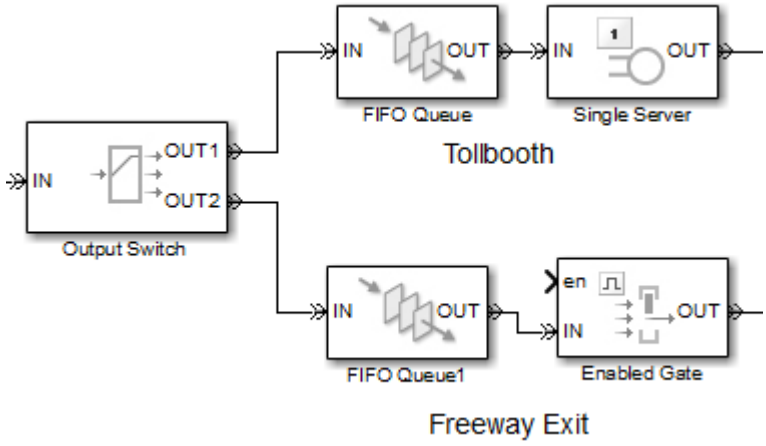
Two queues connected in series might be useful if you are using entities to model items that physically experience two distinct sets of conditions while in storage. For example, additional inventory items that overflow one storage area have to stay in another storage area in which a less well-regulated temperature affects the items' long-term quality. Modeling the two storage areas as distinct queue blocks facilitates viewing the average length of time that entities stay in the overflow storage area.



A similar example is in Example of a Logical Queue, except that the example there does not suggest any physical distinction between the two queues.

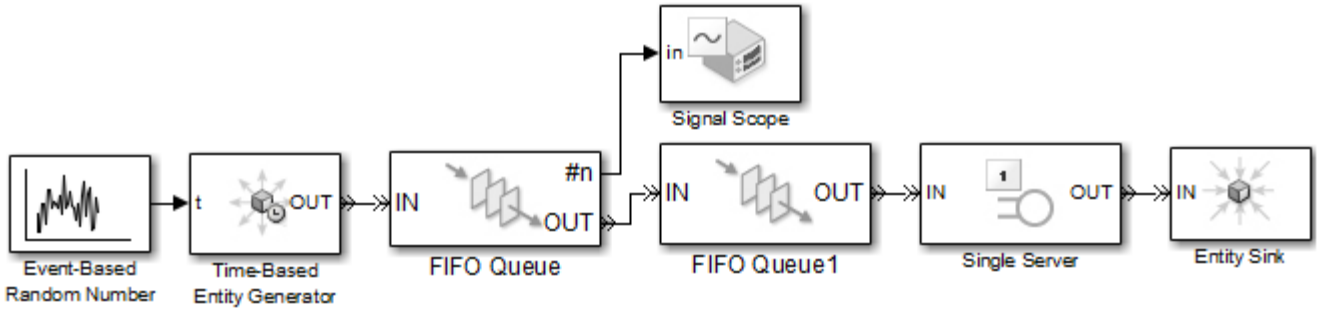
Parallel Connection of Queues

Two queues connected in parallel, in which each entity arrives at one or the other, represent alternative paths for waiting. The paths might lead to different operations, such as a line of vehicles waiting for a tollbooth modeled and a line of vehicles waiting on a jammed exit ramp of the freeway. You might model the tollbooth as a server and the traffic jam as a gate.



Example of a Logical Queue

Suppose you are modeling a queue that can physically hold 100 entities and you want to determine what proportion of the time the queue length exceeds 10. You can model the long queue as a pair of shorter queues connected in series. The shorter queues have length 90 and 10(open model).



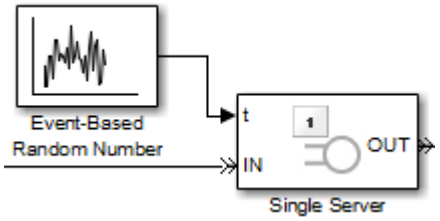
Although the division of the long queue into two shorter queues has no basis in physical reality, it enables you to gather statistics specifically related to one of the shorter queues. In particular, you can view the queue length signal (**#n**) of the queue having length 90. If the signal is positive over a nonzero time interval, then the length-90 queue contains an entity that cannot advance to the length-10 queue. This means that the length-10 queue is full. As a result, the physical length-100 queue contains more than 10 items. Determining the proportion of time the physical queue length exceeds 10 is equivalent to determining the proportion of time the queue length signal of the logical length-90 queue exceeds 0.

Vary the Service Time of a Server

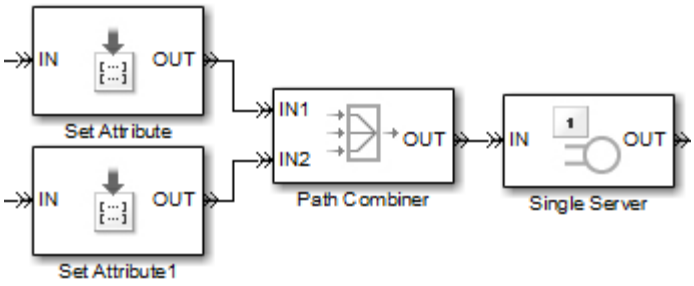
The subsystem described in Lesson 3: Add Event-Based Behavior includes an Infinite Server block that serves each entity for a random amount of time. The random duration is the value of a signal that serves as an input to the Infinite Server block. Similarly, the Single Server block can read the service time from a signal, which enables you to vary the service time for each entity that arrives at the server.

Some scenarios in which you might use a varying service time are as follows:

- You want the service time to come from a random number generator. In this case, set the Single Server block's **Service time from**parameter to `Signal port t` and use the Event-Based Random Number block to generate the input signal for the Single Server block. Be aware that some distributions can produce negative numbers, which are not valid service times.



- You want the service time to come from data attached to each entity. In this case, set the Single Server block's **Service time from**parameter to `Attribute` and set **Attribute name** to the name of the attribute containing the service time. An example is in the figure below.



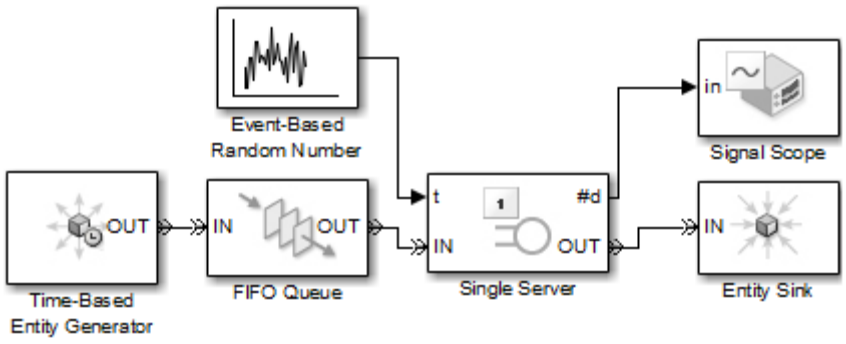
To learn more about attaching data to entities, see Set Entity Attributes.

- You want the service time to arise from dynamics of the simulation. In this case, set the Single Server block's **Service time from** parameter to `Signal port t` and create a signal whose value at the time an entity arrives at the server is equal to the desired service time for that entity.
If the signal representing the service time is an event-based signal such as the output of a Get Attribute block, ensure that the signal's updates occur before the entity arrives at the server. For common problems and troubleshooting tips, see Unexpected Use of Old Value of Signal.

Random Service Times in a Queuing System

Open the model that you created in Build a Discrete-Event Model, or enter `simeventsdocex('doc_dd1')` in the MATLAB® Command Window to open a prebuilt version of the same model. By examining the Single Server block's **Service time from** and **Service time** parameters, you can see that the block is configured to use a constant service time of 1 second. To use a random service time instead, follow these steps:

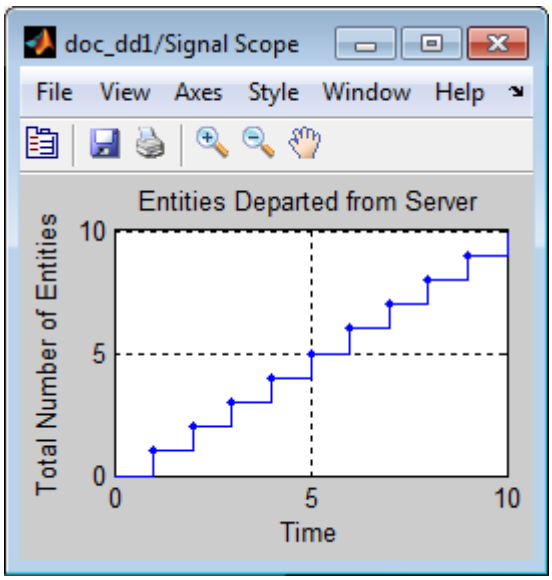
1. Set **Service time from** to `Signal port t`. This causes the block to have a signal input port labeled **t**.
2. From the Signal Generators sublibrary of the Generators library, drag the Event-Based Random Number block into the model window and connect it to the Single Server block's signal input port labeled **t**.



3. Run the simulation and note how the plot differs from the one corresponding to constant service times (shown in Results of the Simulation).

RESULTS OF THE SIMULATION

When the simulation runs, the Signal Scope block opens a window containing a plot. The horizontal axis represents the times at which entities depart from the server, while the vertical axis represents the total number of entities that have departed from the server.



After an entity departs from the Single Server block, the block updates its output signal at the **#d** port. The updated values are reflected in the plot and highlighted with plotting markers. From the plot, you can make these observations:

- Until *T*=1, no entities depart from the server. This is because it takes one second for the server to process the first entity.
- Starting at *T*=1, the plot is a stairstep plot. The stairs have height 1 because the server processes one entity at a time, so entities depart one at a time. The stairs have width equal to the constant service time, which is one second.

QUIZ:-

Q1. What is Two-Server Queuing System?

EXPERIMENT NO: 9

AIM: - Simulate and control a conveyor belt system

SOFTWARE/APPARATUS REQUIRED: - MATLAB R2013A, Personal Computer

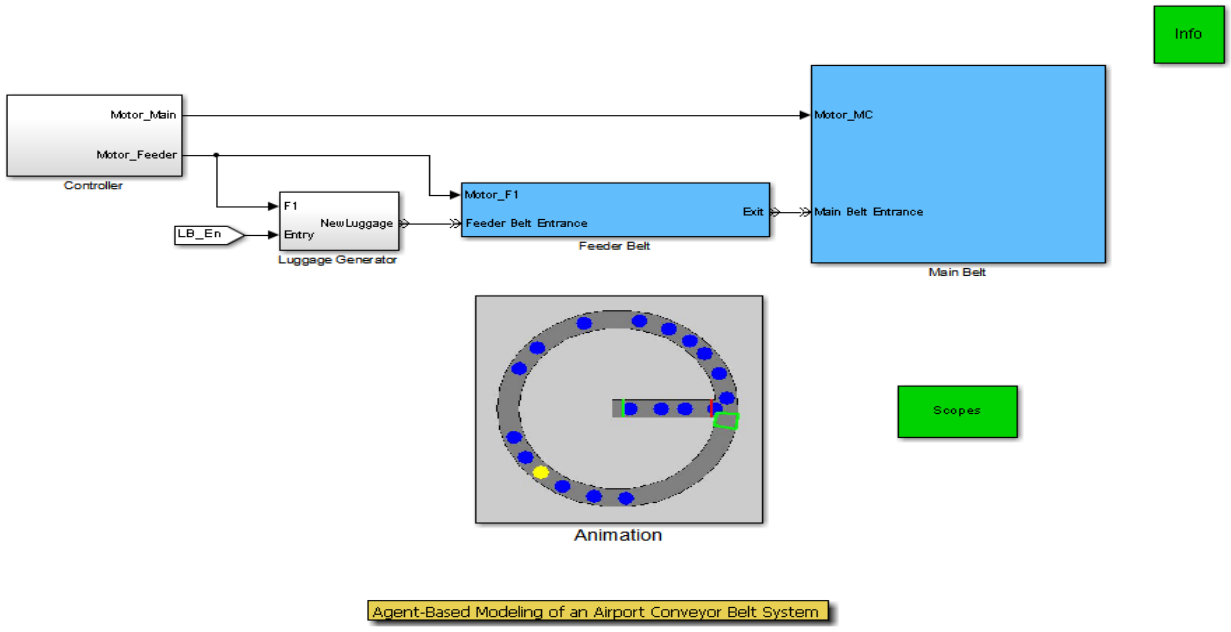
THEORY: - This example shows how event-driven modeling in SimEvents® can be integrated with Stateflow® and Simulink®, to simulate and control a conveyor belt system. In this example, the conveyor belt system shown represents the type used in an airport to carry luggage.

Structure of Model

The model contains the following subsystem blocks:

- A Control subsystem containing Stateflow® charts that execute control of the overall conveyor belt system.
- An Animation subsystem that provides a 2D visualization of the system dynamics.
- A Luggage Generator subsystem that generates SimEvents® entities to represent discrete items of luggage.
- A Feeder Belt subsystem that represents the linear conveyor belt used to load items of luggage into the system. The feeder belt has fixed entry and exit points and a fixed length.
- A Main Belt subsystem that represents the rotational belt used to accumulate items and circulate them in a closed-loop until they are picked up by passengers.

The Feeder Belt and Main Belt subsystems contain both SimEvents® and Simulink® blocks.



SimEvents® Component of Model

The Luggage Generator subsystem contains a Time-Based Entity Generator block that generates entities-representing discrete items of luggage-in a time-based fashion. The time duration between entities is based on a uniform and independent-and-identically-distributed (IID) random sequence.

Both the Feeder Belt and Main Belt subsystems contain a SimEvents® N-Server block that can service a number of entities at any time.

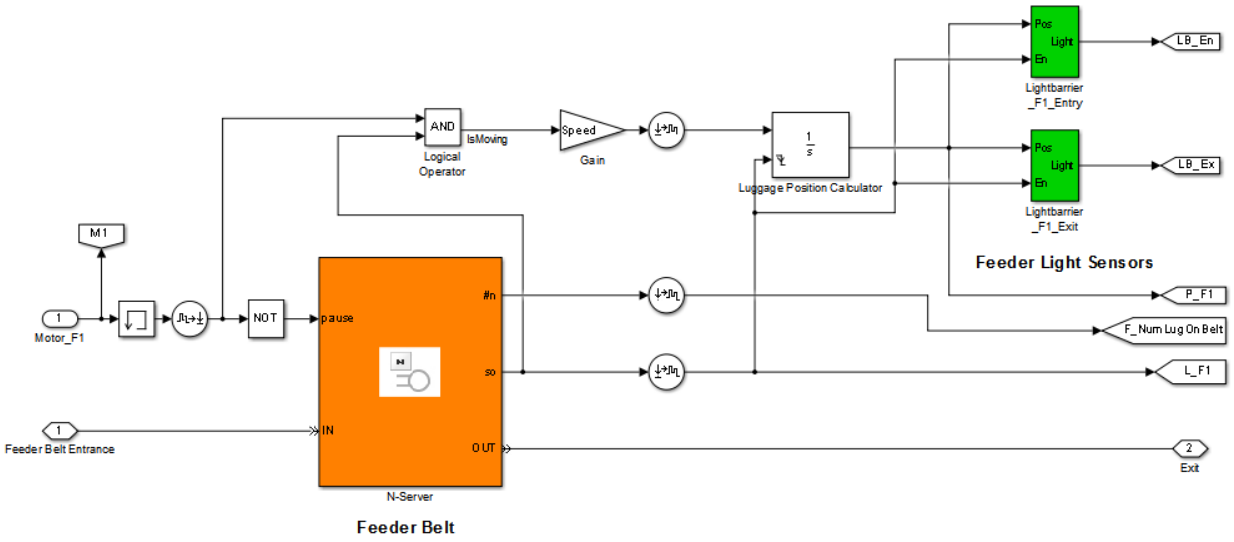
For each belt subsystem, the value of N specified for the N-Server block represents the capacity of the belt.

Each N-Server block uses built-in server control and server-occupancy-monitoring capabilities to pause service when likely collisions are detected by the control unit, or when an emergency stop of the system is requested by an operator.

Simulink® Component of Model

Simulink® blocks are used to:

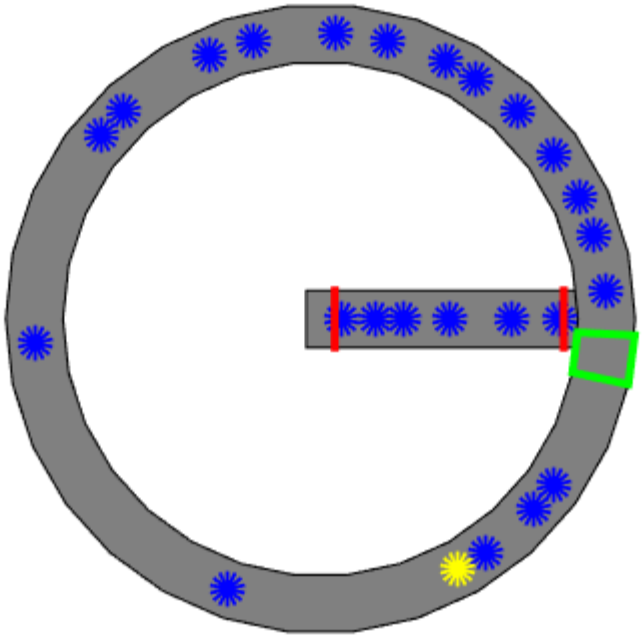
- Calculate the position of items on both belts, based on a fixed belt speed.
- Simulate the dynamics of both belts. These dynamics are used by the Animation subsystem to provide a visualization of the complete moving system.
- Model the behavior of light sensors that detect items of luggage at the entry and exit points of the feeder belt, and at the junction of the feeder belt with the main belt.



RESULTS :- The simulation produces the following results:

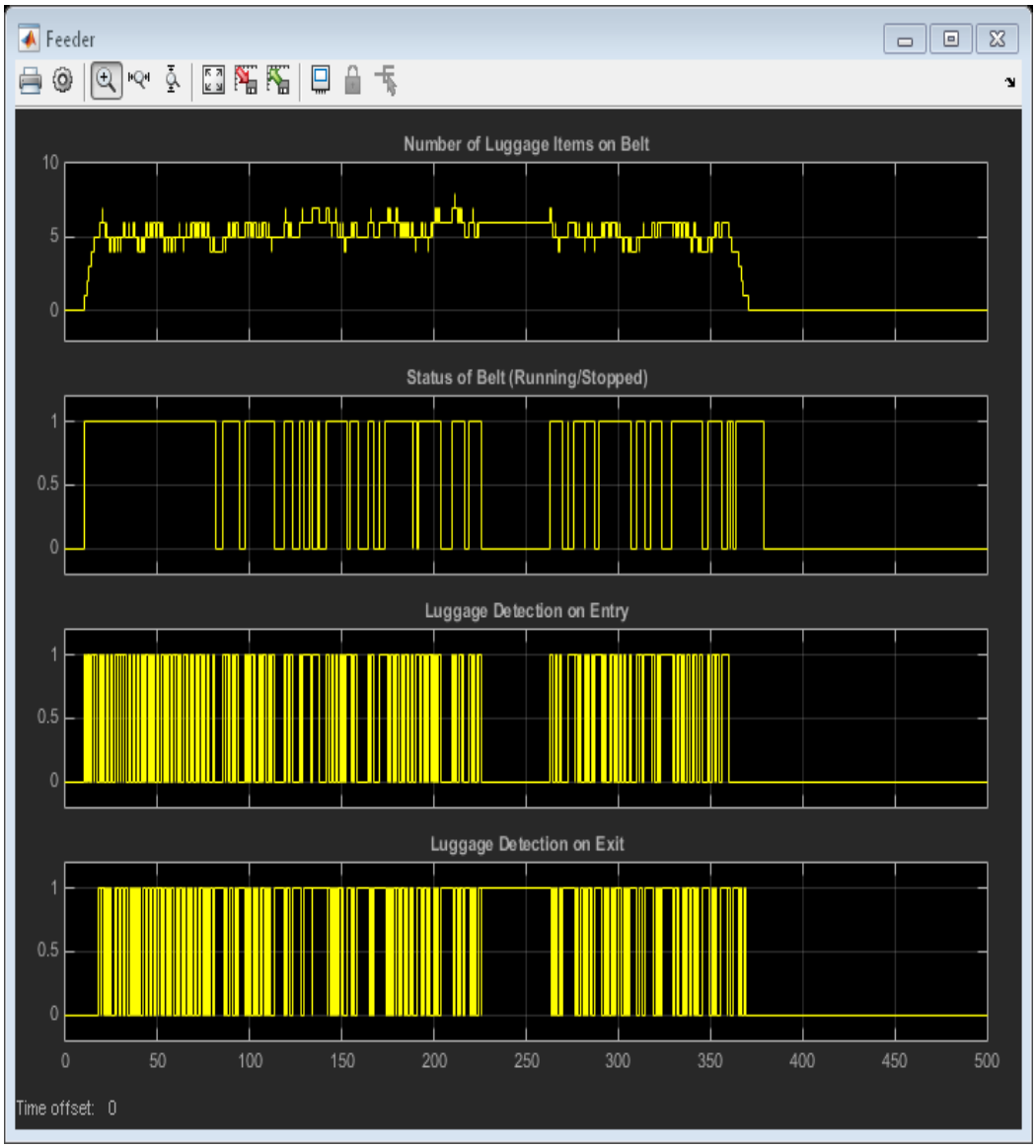
Animation (2D) showing:

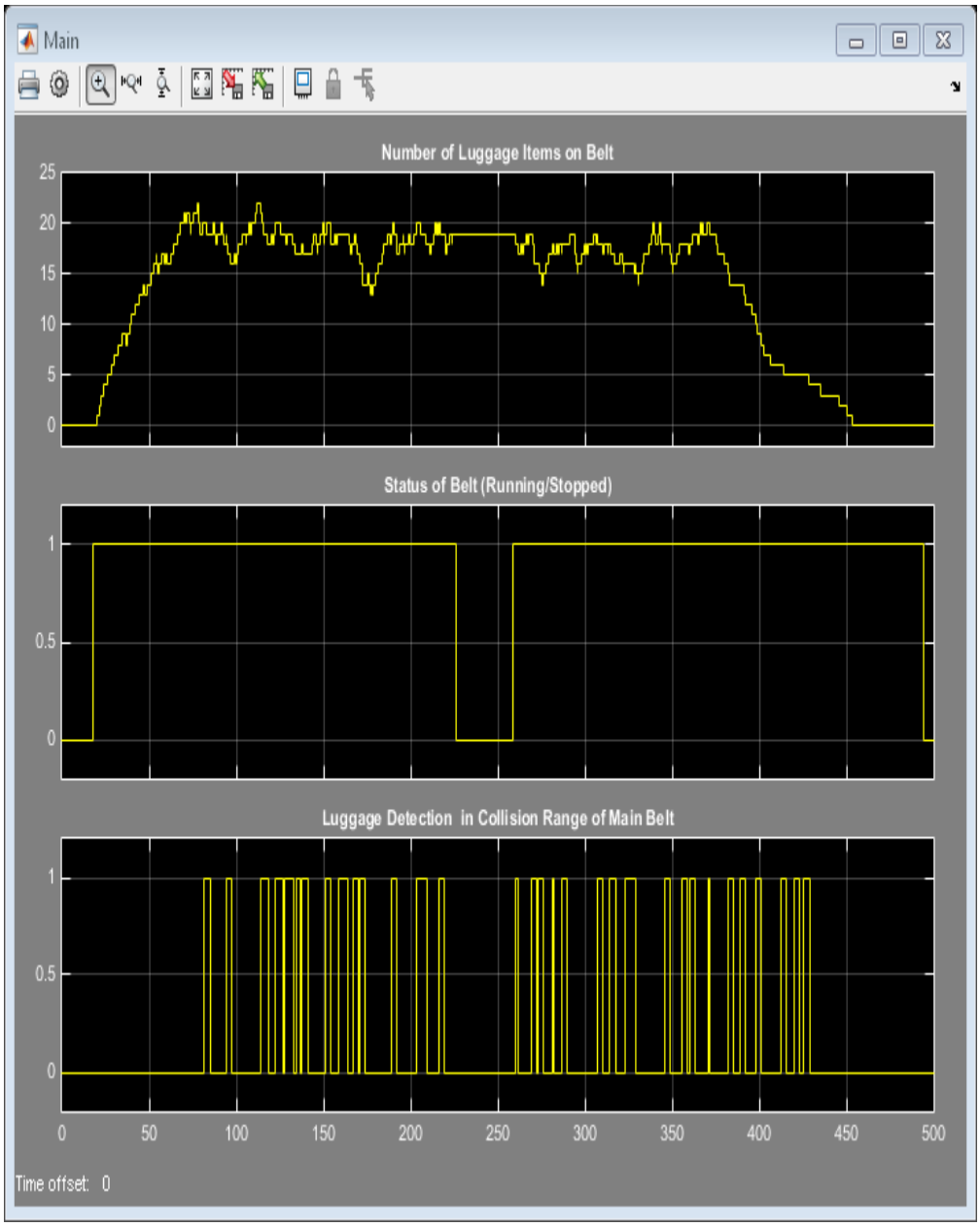
- The motion of items on both conveyor belts.
- Activities such as the loading of items by operators, transfer of items from the feeder belt to the main belt, emergency stopping of the system by an operator, and removal of luggage items by passengers.
- The status of the supervisory control unit, such as any existing blockages at the entry point of the feeder belt or active collision control at the junction of both belts.



Plots showing:

- The changing number of items on each belt.
- The operating status of each belt, such as running, paused, etc.
- The detection of items by light sensors on each belt.


Feeder
Number of Luggage Items on Belt
10
5
0
Status of Belt (Running/Stopped)
1
0.5
0
Luggage Detection on Entry
1
0.5
0
Luggage Detection on Exit
1
0.5
0
0
50
100
150
200
250
300
350
400
450
500
Time offset: 0



QUIZ:-

Q1. Define conveyor belt system.

EXPERIMENT NO: 10

AIM: - Two-sample Kolmogorov-Smirnov test.

SOFTWARE/APPARATUS REQUIRED: - MATLAB R2014b**,** Personal Computer

PROGRAM:
```
h = kstest2(x1,x2)
h = kstest2(x1,x2,Name,Value)
```

DISCUSSION:- `h = kstest2(x1,x2)` returns a test decision for the null hypothesis that the data in vectors `x1` and `x2` are from the same continuous distribution, using the two-sample Kolmogorov-Smirnov test. The alternative hypothesis is that `x1` and `x2` are from different continuous distributions. The result `h` is `1` if the test rejects the null hypothesis at the 5% significance level, and `0` otherwise.

`h = kstest2(x1,x2,Name,Value)` returns a test decision for a two-sample Kolmogorov-Smirnov test with additional options specified by one or more name-value pair arguments. For example, you can change the significance level or conduct a one-sided test.

`[h,p] = kstest2(__)` also returns the asymptotic *p*-value `p`, using any of the input arguments from the previous syntaxes.

`[h,p,ks2stat] = kstest2(__)` also returns the test statistic `ks2stat`.

QUIZ:-

Q1. What is Two-sample Kolmogorov-Smirnov test?