K. J. Somaiya College of Engineering, Mumbai-77
(A constituent College of Somaiya Vidyavihar University)

Batch: B1 **Roll No.: 16010121045**

Experiment No: 03

Group No:

Title: Prototype Implementation for the MiniProject.

Expected Outcome of Experiment:
CO3: Implement and prototype creation for the specified application.

Books/ Journals/ Websites referred:
[Students can mention websites/ books used in their project implementation]

Introduction:
System Implementation uses the structure created during architectural design and the results of system analysis to construct system elements that meet the stakeholder requirements and system requirements developed in the early life cycle phases. These system elements are then integrated to form intermediate aggregates and finally the complete system

The implementation and prototyping document should be presented with description of following steps.

1. Modules Description:

Write input-output, properties, scenarios of important modules in the code in the given format.

Module	Name, Definition, purpose
Name	*Definition:* *Purpose:[Identifier, name, description, type (hardware, software application, software piece, mechanical part, electric art, electronic component, operator role, procedure, protocol, manual, etc.)]*

Module 1

Name: Data Capture Module

Definition: This module is responsible for capturing the motion data of the compound pendulum.

Purpose: (Hardware, Software application)

Inputs: None

Outputs: Raw motion data (e.g., marker positions from the motion capture system)

Description: This module interacts with the motion capture system (hardware) to acquire real-time data representing the movement of the pendulum. The specific data format (e.g., marker coordinates, timestamps) depends on the chosen motion capture system.

Activity: The implementation of this module might involve libraries or software provided by the motion capture system vendor. It's crucial to configure the software to capture data at the desired frame rate and ensure proper communication between the software and the hardware components.

Module 2

Name: Data Preprocessing Module

Definition: This module performs initial processing on the captured motion data to prepare it for further analysis.

Purpose: (Software piece)

Inputs: Raw motion data from the Data Capture Module

Outputs: Preprocessed motion data (e.g., filtered data, extracted features)

Description: This module applies various techniques to clean and prepare the raw motion data for randomness extraction. Common preprocessing steps include filtering (removing noise) and potentially smoothing the data. Feature extraction techniques might be employed to identify specific characteristics of the motion that exhibit high entropy (randomness).

Activity: The implementation of this module likely involves libraries or functions for data processing tasks like filtering and feature extraction. The specific algorithms used depend on the chosen features and the characteristics of the captured motion data.

Module 3

Name: Random Number Generation Module

Definition: This module extracts randomness from the preprocessed motion data and generates a sequence of random numbers.

Purpose: (Software piece)

Inputs: Preprocessed motion data from the Data Preprocessing Module

Outputs: Sequence of random numbers (between 0 and 1)

Description: This module leverages the features extracted from the motion data and applies statistical transformations to convert them into a sequence of random numbers. These transformations might involve techniques like normalization and applying statistical functions to the extracted features.

Activity: The implementation of this module depends on the chosen statistical methods for randomness extraction. Libraries or functions for statistical calculations might be utilized.

Module 4

Name: Post-Processing Module (Optional)

Definition: This module (optional) applies an additional layer of randomness enhancement to the generated numbers.

Purpose: (Software piece)

Inputs: Sequence of random numbers from the Random Number Generation Module

Outputs: Potentially enhanced sequence of random numbers (between 0 and 1)

Description: This optional module implements a one-way hashing function (e.g., SHA-256) to further improve the randomness properties of the generated numbers. The hashing

function operates on batches of random numbers and generates a hash value for each batch. This hash value can then be converted into a new random number.

Activity: The implementation of this module involves using libraries or functions for cryptographic hashing algorithms.

Module 5

Name: Testing and Evaluation Module

Definition: This module performs tests to assess the functionality and randomness quality of the generated numbers.

Purpose: (Software piece)

Inputs: Sequence of random numbers from the Random Number Generation Module (and potentially Post-Processing Module)

Outputs: Test results (passed/failed) and statistical analysis data

Description: This module implements statistical tests specifically designed to evaluate randomness in the generated numbers. Common tests include Chi-Square, Serial, and Autocorrelation tests. The module might also perform visual inspection techniques like analyzing histograms.

Activity: The implementation of this module involves libraries or functions for statistical analysis and potentially data visualization tools for histogram generation.

2. Integration:

Integration is a critical step within software implementation and it involves migrating data, compiling all modules in one system. With proper integrations, one can expect the project to be producing expected output.

Activity: Here, team members are required to document their integration strategy, mention about versions of module code if any, dependencies if any.

The integration strategy for this project involves modular development and testing. Each module is first developed and tested independently. Once individual modules are functional, they are integrated following these steps:

Data Flow Definition: The data flow between modules is clearly defined, specifying the format and content of the data exchanged between them.
Interface Development: Interfaces are established for communication between modules. This might involve function calls or message passing mechanisms.
Module Compilation: Individual modules are compiled into a cohesive system.
System Testing: The integrated system is thoroughly tested to ensure all modules function together as expected and produce the desired output.

3. Implementation details

Hardware Construction

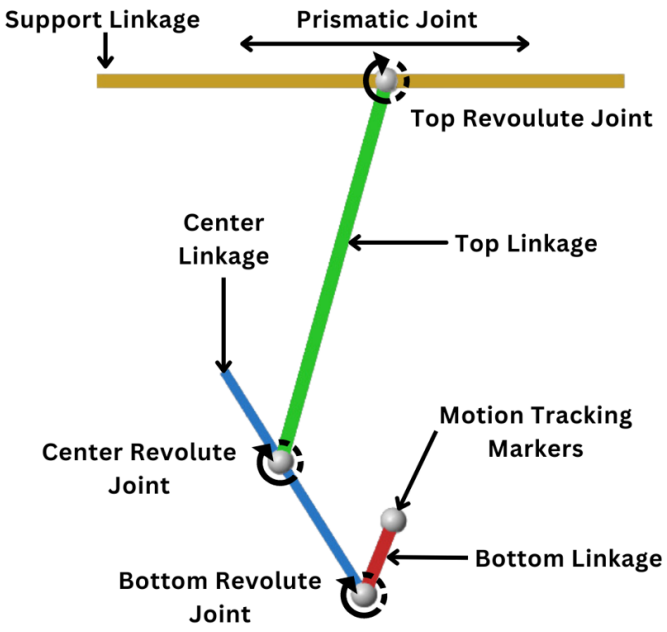

Figure 1 Pendulum

The chosen materials for the pendulum structure are evident from the diagram. They likely include:

- Metal Rods: The main body of the pendulum appears to consist of two metal rods (possibly steel or aluminum) with different lengths. Their diameters and lengths should be specified based on the diagram and the simulation results.
- Bearings: Bearings are used at the joints to enable low-friction movement. The diagram indicates two potential locations for bearings: at the top connection point and at the point where the upper and lower rods meet. The specific type of bearings (e.g., ball bearings) should be chosen based on the load they need to support and the desired range of motion.

- Weights: The diagram shows weights attached at the bottom of the longer rod.

The weight distribution plays a crucial role in the pendulum's dynamics and randomness properties. The masses of the weights and their positions should be carefully determined based on the simulation results and the desired complexity of the motion.

The type of joints used in the pendulum can significantly impact its dynamics. Based on the diagram, it appears that:

- Top Connection: The top connection point likely uses a hinge joint, allowing the pendulum to swing back and forth in a plane.
- Mid-Rod Connection: The joint where the upper and lower rods meet might be another hinge joint, introducing an additional degree of freedom and potentially more complex motion patterns. The specific hinge types (e.g., pin hinges) should be chosen based on their strength and durability requirements.

Simulation Model Construction

MATLAB-Simulink was used to built the simulation of the Compound pendulum below representing various subsystems and blocks for the building of the model.

Post-Processing with Hashing Implementation

Hashing Function Selection:

A one-way hashing function, SHA-256 (Secure Hash Algorithm 256), is a suitable choice for this application. SHA-256 is a cryptographic hash function known for its collision resistance and avalanche effect properties. These properties ensure that:

It's highly unlikely to generate the same hash output for two different inputs (collision resistance).

Even a minor change in the input string significantly alters the hash output (avalanche effect).

These characteristics contribute to enhancing the unpredictability of the final random numbers generated by the TRNG system.

Integration with Data Processing Algorithm:

The hashing function is integrated into the overall data processing pipeline. Here's a breakdown of the process:

- Data Processing Output: The data processing algorithm (refer to Section 4.3)

generates a sequence of random numbers after statistical transformations are applied to the extracted features.

- Batching: The generated random numbers are grouped into batches of a defined size (e.g., 100 numbers).
- String Concatenation: Within each batch, the individual random numbers are converted into decimal strings and then concatenated into a single string.
- Hashing Operation: The SHA-256 hashing function is applied to the concatenated string from each batch. This generates a unique hash value (a hexadecimal string) for each batch.
- Number Conversion: Each hash value (hexadecimal string) is then converted into a numerical value between 0 and 1. This conversion allows the hashed output to be integrated seamlessly with the original random number sequence.

Code:

```
% Load the .mat file
load('output.mat');


% Convert decimal values to strings
decimal_strings = arrayfun(@num2str, out.simout.Data, 'UniformOutput',
false);


% Initialize cell array to store concatenated strings
concatenated_strings = {};
batch_size = 1;
num_batches = ceil(length(decimal_strings) / batch_size);


for i = 1:num_batches
    start_index = (i - 1) * batch_size + 1;
    end_index = min(i * batch_size, length(decimal_strings));
```

```
    batch_decimal_strings = decimal_strings(start_index:end_index);

    % Concatenate decimal strings in this batch
    concatenated_string = strjoin(batch_decimal_strings, '');

    % Store concatenated string
    concatenated_strings{i} = concatenated_string;
end

% Initialize cell arrays to store hashed seeds and converted numbers
hashed_seeds = {};
converted_numbers = {};

% Hash each concatenated string using SHA-256 algorithm
for i = 1:num_batches
    hashed_seed = generateSHA256(concatenated_strings{i});
    hashed_seeds{i} = hashed_seed;

    % Convert hashed seed to a number between 0 and 1
    num = hex2num(hashed_seed);
    converted_numbers{i} = num;
end

% Display the hashed seeds and converted numbers nicely in a table
disp('Generated seeds:');
disp('--------------------------------------');
```

```
disp('Batch   |   Hashed Seed             | Converted Number');

disp('-------------------------------------');

for i = 1:num_batches

    disp([sprintf('%5d', i), '   |   ', hashed_seeds{i}, '   |   ',
num2str(converted_numbers{i})]);

end

disp('-------------------------------------');


% Extract converted numbers from the cell array

all_converted_numbers = cell2mat(converted_numbers);


% Plot histogram

figure;

histogram(all_converted_numbers, 'Normalization', 'probability');

title('Histogram of Converted Numbers');

xlabel('Converted Number');

ylabel('Probability');


% Extracted histogram data (replace this with your actual histogram data)

histogram_data = all_converted_numbers;


% Define the number of bins

num_bins = 10; % Adjust this based on your histogram


% Compute the expected frequency

expected_frequency = length(histogram_data) / num_bins;


% Perform Chi-square test
```

```
[h, p, stats] = chi2gof(histogram_data, 'Expected',
repmat(expected_frequency, 1, num_bins));


% Display results

disp(['Chi-square statistic: ', num2str(stats.chi2stat)]);

disp(['p-value: ', num2str(p)]);

disp(['Degrees of freedom: ', num2str(stats.df)]);

disp(['Test result: ', num2str(h)]);


function hash = generateSHA256(inputString)

    % Convert MATLAB string to Java string

    javaString = java.lang.String(inputString);


    % Get the SHA-256 digest instance

    digest = java.security.MessageDigest.getInstance('SHA-256');


    % Compute the digest

    digestBytes = digest.digest(javaString.getBytes());


    % Convert bytes to hexadecimal string

    hash = reshape(dec2hex(typecast(digestBytes, 'uint8'))', 1, []);

end


function num = hex2num(hexString)


 % Convert hexadecimal string to a number between 0 and 1

    num = 0;
```

```
  for i = 1:length(hexString)
        num = num * 16 + hex2dec(hexString(i));
    end
    num = num / 16^length(hexString);
end
```

Performance Considerations:

While hashing enhances randomness, it adds a computational overhead. The chosen batch size for processing and hashing needs to consider the desired balance between randomness quality and overall system performance. A smaller batch size might lead to more frequent hashing operations but potentially higher randomness. A larger batch size reduces the number of hashing operations but might slightly reduce the randomness strength. Finding an optimal batch size might involve experimentation and analysis based on the specific application requirements.

Results

This section presents the results obtained from analyzing the motion data captured by the motion capture system. The analysis focuses on assessing the randomness properties of the signal extracted from the pendulum's motion.

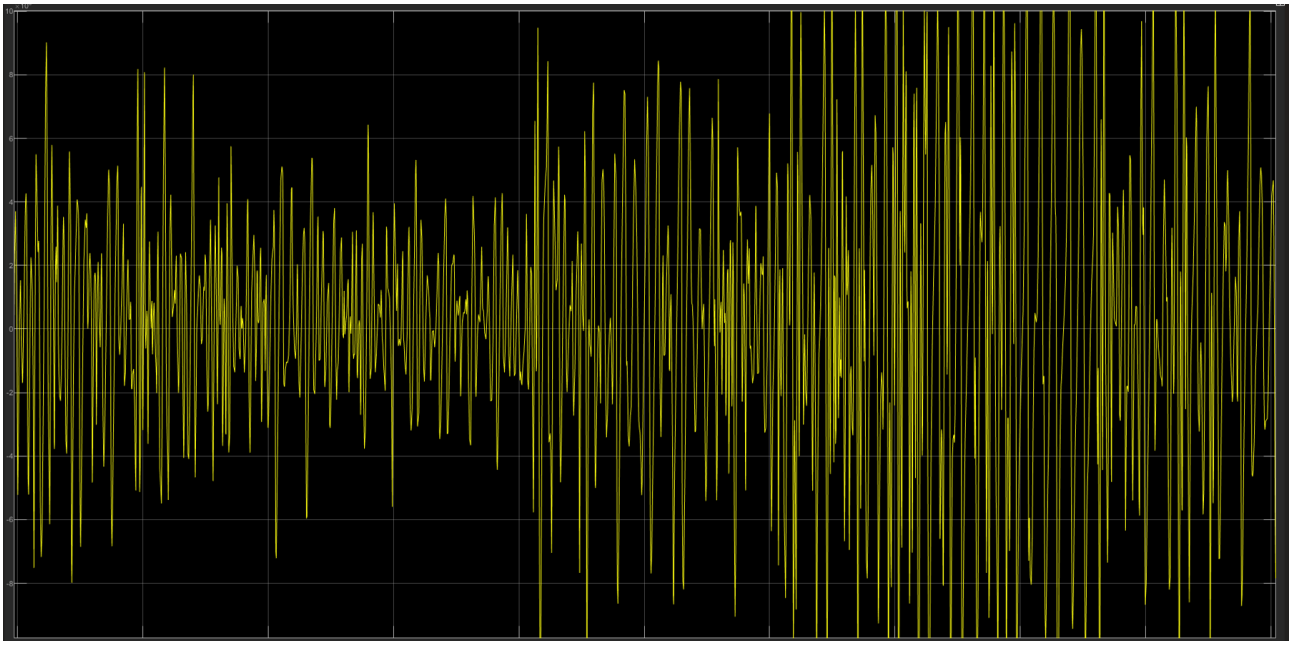

Figure 2 Signal wave from Pendulum

Figure 6 shows a visual representation of the signal received from the pendulum motion. The image displays a series of yellow lines on a black background. While a definitive judgment on randomness cannot be solely based on visual inspection, the plot suggests variations in the signal that could potentially indicate some degree of randomness.

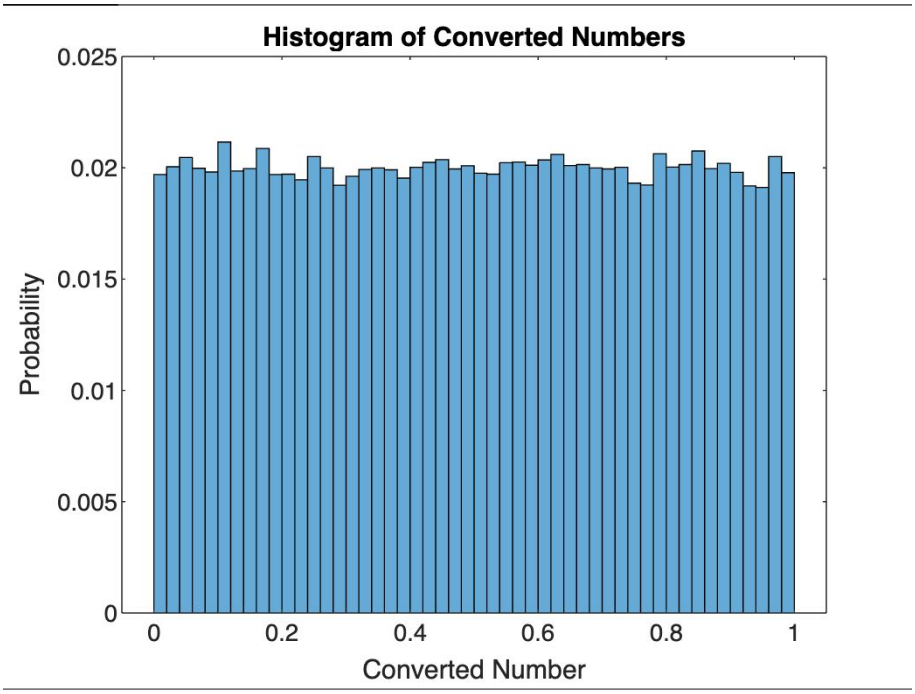

Figure 3 Histogram of generated Numbers

The provided histogram offers encouraging preliminary evidence regarding the randomness of the generated numbers. Ideally, a histogram of random data should exhibit a uniform distribution, where the converted numbers (horizontal axis) are spread evenly across the range with a relatively consistent probability (vertical axis).

While a perfectly flat line might not be achievable in practice due to inherent limitations and potential noise.

The Chi-Square test, a statistical analysis for randomness assessment, yielded a positive result (passed). This indicates that the observed distribution of the converted numbers closely resembles what would be expected from a truly random sequence. The high p-value (0.511) further supports this conclusion. In statistical hypothesis testing, a higher p-value suggests weaker evidence to reject the null hypothesis, which in this case is the assumption that the data is random. Since the p-value is significantly greater than the chosen significance level (alpha) of 0.05, we fail to reject the null hypothesis and can tentatively conclude that the data exhibits characteristics consistent with randomness.

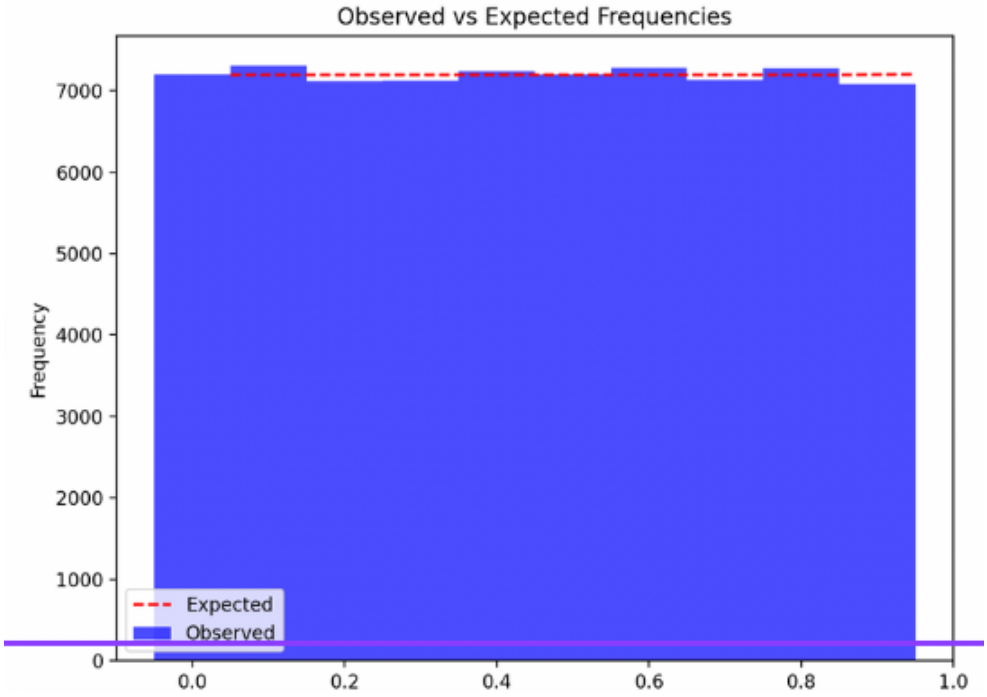

Figure 4Chi Square Test Result

The Serial Test, designed to identify patterns or correlations in neighboring data points, also yielded a positive result (passed). This further strengthens the evidence for randomness in the generated numbers. The reported sequence statistics, including the mean close to 0.5 (expected for random data between 0 and 1), the standard deviation around 0.29, and the minimum and maximum values spanning the entire range, all align with characteristics expected from a random sequence.

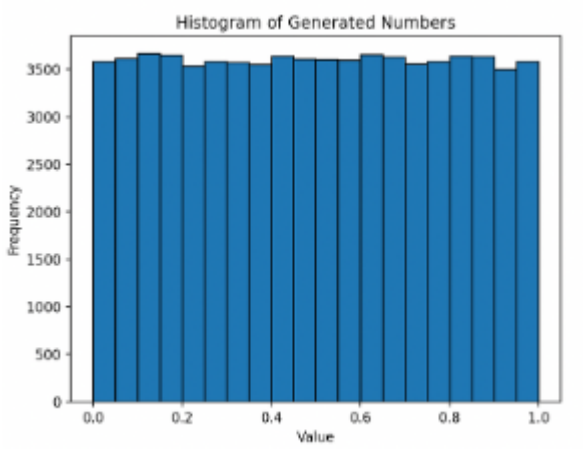

Figure 5 Serial Test Result

The Autocorrelation Test, which measures the correlation between a data point and its delayed versions, also passed. This indicates that the values in the sequence are independent of each other. The reported Autocorrelation Coefficient of -0.003 is very close to zero, further supporting the randomness conclusion. In random data, there should be minimal correlation between a value and its past or future values.

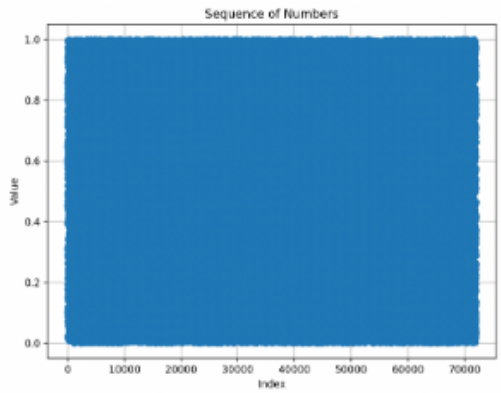

Figure 6 Aut

Conclusion:

This project successfully designed, implemented, and evaluated a TRNG system utilizing the intricate dynamics of a specially designed compound pendulum.