DAYANANDA SAGAR COLLEGE OF ENGINEERING
(An Autonomous Institute affiliated to VTU, Belagavi – 590018, Approved by AICTE & ISO 9001:2015 Certified)
Accredited by National Assessment & Accreditation Council (NAAC) with 'A' grade

A
Report on

SKILL DEVELOPMENT PROGRAM

ON SIGNAL & IMAGE PROCESSING WITH EMBEDDED HARDWARE INTEGRATION

(Duration: 15th March to 17th March)

Bachelor of Engineering – II Year BE (4th Semester)

Electronics & Communication Engineering

by

1DS23EC104:- LAHARI V
1DS23EC205 :-SHREYA S
1DS23EC206:- SHRIRAKSHA P
1DS23EC218 :-SRUSHTI M P

Coordinators: Prof. Kavita Guddad and Prof. Deepa N P

Department of Electronics & Communication Engineering
(An Autonomous College affiliated to VTU Belgaum, accredited by NBA & NAAC)
Shavige Malleshwara Hills, Kumaraswamy Layout,
Bengaluru-560078, Karnataka, India

2023-24

Table of Contents

1.SIGNAL PROCESSING

1.1.Introduction to MathWorks & MATLAB:

MathWorks is a leading developer of mathematical computing software, best known for MATLAB and Simulink.

- MATLAB (Matrix Laboratory): A high-level programming environment for
- numerical computation, visualization, and programming.
  Simulink: A graphical tool for modeling, simulating, and analyzing dynamic systems, widely used in control systems and embedded applications.

Key Features of MATLAB for Signal Processing:

- Built-in mathematical functions for signal analysis.
- Visualization tools such as plot(), spectrogram(), and fft().
- Extensive toolbox support, including the Signal Processing Toolbox for filtering,
- transformations, and statistical analysis.
  Hardware support for real-time signal acquisition and processing.

1.2.Overview of Signal Processing:

Signal processing is the analysis, manipulation, and interpretation of signals (such as audio, biomedical, and communication signals).

Types of Signals:

1. Continuous-Time (Analog) Signals: Represent real-world phenomena like ECG, sound waves, and temperature variations.
2. Discrete-Time (Digital) Signals: Processed in digital form, used in communication systems, image processing, and AI applications.

Key Concepts in Signal Processing:

- Time-domain analysis (e.g., waveform analysis using plot()).
- Frequency-domain analysis (e.g., Fourier Transform for analyzing frequency
- components).
- Filtering techniques (e.g., noise removal using FIR and IIR filters).
  Feature extraction and classification (e.g., using AI for automated signal analysis).

1.3.Signal Generation & Visualization

Signal Generation in MATLAB:

MATLAB allows generating synthetic signals for testing algorithms and models.

Example of generating a sine wave:

```
fs = 1000; % Sampling
frequency t = 0:1/fs:1-1/fs; %
Time vector
x = sin(2*pi*10*t); % 10 Hz sine wave
plot(t, x); grid on;
```

Visualization Techniques:

1. Time-Domain Representation
   - o plot() for continuous signals.
   - o stem() for discrete signals.
2. Frequency-Domain Representation
   - o Fourier Transform (fft()): Converts signals into frequency components.
   - o Spectrogram (spectrogram()): Visualizes time-frequency variations.

Example:

```
X = fft(x); % Compute FFT
f = fs*(0:(length(x)/2))/length(x); % Frequency
axis plot(f, abs(X(1:length(f)))); % Magnitude
spectrum
```

1.4.Signal Data Acquisition using Smartphone

MATLAB supports real-time signal acquisition from mobile sensors like an accelerometer,

gyroscope, and microphone.
Steps for Smartphone Data Acquisition:

1. Install the MATLAB Mobile App on your smartphone.
2. Connect it to MATLAB on your PC using mobiledev().
3. Collect sensor data (e.g., acceleration, sound) and process it.

Example of accelerometer data acquisition:
mobile = mobiledev; % Connect to mobile
[accX, accY, accZ] = accellog(mobile); % Get acceleration
data plot(accX);
Applications include motion analysis, gait recognition, and speech processing.

1.5.Working with Signal Analyzer:

Signal Analyzer is a MATLAB app that provides an interactive interface for analyzing signals.
Features:

- Time and Frequency Analysis: View signals in both domains.
- Filtering and Smoothing: Remove noise interactively.
- Spectral Analysis: Compute Power Spectral Density (PSD).
- Multichannel Signal Comparison: Compare multiple datasets.

Example Use Case:

- Analyzing EEG (brainwave) signals for frequency components.
- Identifying ECG anomalies (e.g., arrhythmia detection).

To open the app:
signalAnalyzer;
Then, load the signal and perform various analyses.

1.6.ECG Signal Pre-processing:

ECG signals often contain noise from different sources. Pre-processing steps remove unwanted components.
Types of ECG Noise:

- Baseline Wander: Low-frequency drift due to respiration.
- Powerline Interference: 50/60 Hz noise from electrical sources.
- Muscle Artifacts: High-frequency noise from muscle movement.

Pre-processing Techniques:

1. Baseline Wander Removal:
   - o Apply high-pass filtering to eliminate low-frequency drift.

[b, a] = butter(2, 0.5/(500/2), 'high'); % High-pass filter
filteredECG = filtfilt(b, a, rawECG);

2. Notch Filtering for Powerline Noise (50/60 Hz):

d = designfilt('bandstopiir', 'FilterOrder', 2, 'HalfPowerFrequency1', 49, 'HalfPowerFrequency2', 51, 'DesignMethod', 'butter', 'SampleRate', 500);
filteredECG = filtfilt(d, rawECG);

3. Wavelet Transform for Noise Reduction:
   - o Uses Discrete Wavelet Transform (DWT) to remove noise while preserving important signal features.

Application: ECG analysis for heart rate variability (HRV) and arrhythmia detection.

1.7.Automating Filter Design:

Filtering is crucial in signal processing to remove noise and enhance signal quality. MATLAB provides automated tools for designing FIR and IIR filters.
Types of Filters:

1. FIR (Finite Impulse Response) Filters: Always stable, commonly used in audio and biomedical applications.

2. IIR (Infinite Impulse Response) Filters: More efficient but may introduce phase distortion.

Designing Filters in MATLAB:

Example of Low-Pass FIR Filter:

```
d = designfilt('lowpassfir', 'PassbandFrequency', 100, 'StopbandFrequency', 150,
'SampleRate', 1000);
fvtool(d); % Visualize filter
```

MATLAB also provides the Filter Designer App (filterDesigner) for interactive filter design.

Applications:

- Speech Processing: Removing background noise.
- Biomedical Applications: Enhancing ECG signals.
- Radar Systems: Extracting signals from clutter.

1.8.Human Activity Recognition with Machine Learning (Signal + AI):

Human Activity Recognition (HAR) involves classifying physical activities (e.g., walking, running, sitting) using sensor data.

Steps in HAR Using Machine Learning:

1. Data Collection:
   - o Accelerometer & gyroscope signals are recorded.
2. Feature Extraction:
   - o Mean, standard deviation, energy, entropy, and frequency domain features.
3. Model Training:
   - o Train classifiers like SVM, Decision Trees, or Neural Networks.
4. Evaluation:
   - o Accuracy, precision, recall, and confusion matrix analysis.

Example of Training an SVM Classifier in MATLAB:

```
X = [feature1, feature2, feature3]; % Extracted
features Y = activity_labels; % Walking, Running, etc.
model = fitcsvm(X, Y); % Train SVM
predictions = predict(model, X_test); % Predict on test data
```

Deep Learning with CNN for HAR:

- CNNs extract hierarchical features from raw signals.
- Pre-trained models like AlexNet or LSTM Networks can be used.

Applications:

- Healthcare: Fall detection in elderly patients.
- Sports Analytics: Motion tracking and performance assessment.
- Smart Devices: Gesture recognition for smartphones.

2. IMAGE PROCESSING

2.1.Introduction to Image Processing

Image Processing is the technique of analyzing, manipulating, and transforming images for various applications, including medical imaging, computer vision, artificial intelligence, and multimedia. It involves converting an image into digital form, processing it using mathematical operations, and extracting useful features for decision-making.

Types of Image Processing:

1. Analog Image Processing:
   - o Applied to physical images (e.g., photographs, X-rays).
2. Digital Image Processing:
   - o Performed using computers on digital images (e.g., satellite imagery, MRI scans).

Key Operations in Image Processing:

- Preprocessing: Noise removal, contrast enhancement.
- Transformation: Filtering, edge detection, segmentation.
- Analysis & Recognition: Object detection, AI-based classification.

Applications of Image Processing:

✔ Medical Imaging: MRI, CT scans, and X-rays analysis.

✔ Satellite Image Processing: Land cover classification, weather forecasting.

✔ Face Recognition: Biometric security systems.

✔ Industrial Automation: Defect detection in manufacturing.

2.2.Importing, Exporting & Visualizing Image Data

MATLAB provides built-in functions to handle images in multiple formats (JPEG, PNG, BMP, TIFF).

Importing an Image into MATLAB:

To read an image and store it as a matrix:

```
img = imread('image.jpg'); % Load the
image imshow(img); % Display the image
```

Understanding Image Representation in MATLAB:

- Grayscale Image: 2D matrix where each value represents intensity (0 = black, 255 =
- white).
- RGB Image: 3D matrix (m x n x 3), where each layer represents Red, Green, Blue channels.
  Binary Image: Each pixel is either 0 (black) or 1 (white).

Exporting an Image:

Save the processed image in different formats:

```
imwrite(img, 'output.png'); % Save image as
```

PNG Visualizing Image Data:

- Displaying the histogram of pixel intensity:

```
imhist(img); % Display intensity distribution
```

- Plotting 3D surface of intensity variations:

```
surf(double(img));
```

2.3.Image Enhancement & Thresholding

Image Enhancement Techniques:

1. Contrast Enhancement: Improves image clarity.

```
enhanced_img = imadjust(img, [0.3 0.7], []); % Adjust contrast
imshow(enhanced_img);
```

2. Histogram Equalization: Distributes pixel intensities more evenly.

```
equalized_img = histeq(img);
imshow(equalized_img);
```

3. Noise Removal Using Filters:
   - o Median Filter (Good for salt-and-pepper noise):

```
filtered_img = medfilt2(img);
imshow(filtered_img);
```

   - o Gaussian Filter (Removes Gaussian noise while preserving edges):

```
filtered_img = imgaussfilt(img, 2);
```

Thresholding (Binary Image Conversion) Thresholding converts an image intobinary

form, useful for segmentation and feature extraction. Global Thresholding:

```
gray_img = rgb2gray(img); % Convert to grayscale
bw_img = imbinarize(gray_img, 0.5); % Convert to binary using threshold
imshow(bw_img);
```

Adaptive Thresholding (Otsu's Method):
Automatically determines an optimal threshold.

```
level = graythresh(gray_img);
bw_img = imbinarize(gray_img, level);
imshow(bw_img);
```

Application of Thresholding:

✔ License plate recognition

✔ Medical imaging (tumor detection)

✔ Object segmentation

2.4.Edge Detection & Morphology Operations

Edge Detection

Edge detection is used to extract boundaries and features of objects in an image.

Common Edge Detection Techniques:

1. Sobel Operator
   - o Detects edges using derivatives in horizontal & vertical directions.

```
edges = edge(gray_img, 'sobel');
imshow(edges);
```

2. Prewitt Operator
   - o Similar to Sobel but less sensitive to noise.

```
edges = edge(gray_img, 'prewitt');
```

3. Canny Edge Detection
   - o Most effective as it reduces noise and detects strong edges.

```
edges = edge(gray_img, 'canny');
imshow(edges);
```

Morphological Operations

Morphological operations help refine images by modifying object shapes.

Basic Morphological Operations:

1. Dilation (Expands Object Boundaries)

```
se = strel('disk', 3); % Create a structuring element
dilated_img = imdilate(bw_img, se);
imshow(dilated_img);
```

2. Erosion (Removes Small Noises & Shrinks Objects)

```
eroded_img = imerode(bw_img, se);
```

3. Opening (Erosion followed by Dilation – Removes Noise)

```
opened_img = imopen(bw_img, se);
```

4. Closing (Dilation followed by Erosion – Fills Gaps)

```
closed_img = imclose(bw_img, se);
```

Applications of Edge Detection & Morphology:

✔ Face detection

✔ Object segmentation

✔ Automated medical diagnosis

2.5.Image Classification using CNN Model (Image + AI)

Introduction to CNN (Convolutional Neural Networks)

CNNs are deep learning models designed for image classification, object detection, and pattern recognition.

Steps for Image Classification using CNN in MATLAB:

Step 1: Load & Preprocess Dataset

```
digitDatasetPath = 'path_to_dataset';
imds = imageDatastore(digitDatasetPath, 'IncludeSubfolders', true, 'LabelSource',
```

'foldernames');
Step 2: Define CNN Architecture

```
layers = [
  imageInputLayer([28 28 1]) % Input Layer
  convolution2dLayer(3,8,'Padding','same') % Convolution Layer
  batchNormalizationLayer % Normalization
  reluLayer % Activation Function
  maxPooling2dLayer(2,'Stride',2) % Pooling Layer
  fullyConnectedLayer(10) % Fully Connected Layer
  softmaxLayer % Softmax Function
  classificationLayer]; % Output Layer
```

Step 3: Train the Model

```
options = trainingOptions('sgdm', 'MaxEpochs', 10, 'MiniBatchSize', 64, 'Verbose', true);
cnnModel = trainNetwork(imds, layers, options);
```

Step 4: Test & Evaluate Performance YPred

```
= classify(cnnModel, testImages);
accuracy = sum(YPred == testLabels)/numel(testLabels);
disp(['Classification Accuracy: ', num2str(accuracy*100), '%']);
```

Applications of CNN in Image Processing:

✔ Handwritten digit recognition (MNIST dataset)

✔ Medical image classification (X-ray, MRI)

✔ Real-time object detection (Autonomous vehicles)

3. EMBEDDED HARDWARE INTEGRATION WITH MATLAB & SIMULINK

3.1.Introduction to Embedded Hardware Integration

Embedded hardware integration refers to the ability to interface MATLAB & Simulink with microcontrollers, development boards, and embedded systems for real-time data acquisition, control, and automation.

Use of MATLAB & Simulink for Embedded Systems:

✔ Rapid Prototyping: Speeds up testing and deployment.

✔ Graphical Programming (Simulink): No need for complex coding.

✔ Hardware Compatibility: Supports microcontrollers (Arduino, STM32), FPGAs, and Raspberry Pi.

✔ Real-Time Data Processing: Used in IoT, robotics, and control systems.

Commonly Used Embedded Hardware:

- Microcontrollers: Arduino, ESP32, STM32.
- FPGAs: Xilinx, Intel FPGA.
- SoCs & SBCs: Raspberry Pi, NVIDIA Jetson.
- Sensors: Temperature, pressure, accelerometers, cameras.

3.2.Hardware Support Packages in MATLAB

MATLAB provides Hardware Support Packages to simplify communication with embedded systems.

How to Install a Hardware Support Package?

1. Open MATLAB.
2. Go to Home → Add-Ons → Get Hardware Support Packages.
3. Search for the required hardware (e.g., “Arduino Support for MATLAB”).
4. Click Install and follow the setup instructions.

Common Hardware Support Packages:

Hardware	MATLAB & Simulink Support
Arduino	Support Package

Features

Read/write
GPIO, ADC,
I2C,

Hardware	Support Package	Features
	Package	SPI
Raspberry Pi	MATLAB & Simulink Support Package	Camera, GPIO, Audio Processing
STM32	Embedded Coder Support	Direct code generation
ESP8266/ESP32	I o T S up p o r t P a c k a g e	W i - F i a n d c lo u d in te r at ion
NI Data Acquisition	D a ta A cq u i s i ti o n T o o lbox	H i g h - s p e e d d a ta lo gg in g

3.3.Setting up Hardware with MATLAB & Simulink

To communicate with embedded hardware, MATLAB provides MATLAB Support Package APIs and Simulink blocks.

3.3.1.Setting up Arduino with MATLAB

1. Connect Arduino to the PC via USB.
2. Install the Arduino Support Package (arduino).
3. Create an object in MATLAB:

```
a = arduino('COM3', 'Uno'); % Connect Arduino on COM3
```

3.3.2.Setting up Raspberry Pi with MATLAB

1. Install the Raspberry Pi Support Package.
2. Connect Raspberry Pi to MATLAB over Wi-Fi or Ethernet.
3. Create a connection:

```
r = raspi;
```

3.3.3.Using Simulink for Embedded Hardware

1. Open Simulink (simulink command).
2. Create a New Model and select the target hardware.
3. Use GPIO blocks to interact with sensors and actuators.
4. Deploy the Simulink model to hardware.

3.4.Data Acquisition from Pins & Sensors

3.4.1.Reading Sensor Data from Arduino

Example: Reading Temperature Sensor (LM35) Data

```
a = arduino('COM3', 'Uno');
temp = readVoltage(a, 'A0') * 100; % Convert voltage to temperature
disp(['Temperature: ', num2str(temp), ' °C']);
```

3.4.2.Using Raspberry Pi for Data Logging

Example: Reading Data from a DHT11 Temperature & Humidity Sensor

```
r = raspi;
sensor = dht11(r, 4); % DHT11 connected to GPIO4
[temp, hum] = read(sensor);
disp(['Temperature: ', num2str(temp), '°C, Humidity: ', num2str(hum), '%']);
```

3.4.3.Real-Time Data Plotting

```
for i = 1:100
```

```
    temp = readVoltage(a, 'A0') * 100;
    plot(i, temp, 'ro'); hold on;
    pause(0.5);
  end
```

3.5.Hands-on IoT Applications with MATLAB

IoT (Internet of Things) involves connecting sensors, devices, and embedded hardware to the internet for remote monitoring and control.

3.5.1. IoT Architecture in MATLAB

Component	Example
Edge Device	Arduino, ESP32, Raspberry Pi
Cloud Platform	Thingspeak, AWS IoT, Firebase
Data Processing	MATLAB & Simulink
Visualization	MATLAB GUI, Dashboards

3.5.2. IoT Applications in MATLAB

(a) Uploading Sensor Data to ThingSpeak

ThingSpeak is a MATLAB cloud platform for IoT analytics.

Steps:

1. Sign up at www.thingspeak.com.
2. Create a new channel and get the API Key.
3. Send sensor data from Arduino:

```
writeAPIKey = 'YOUR_API_KEY';
temperature = readVoltage(a, 'A0') * 100;
url = ['https://api.thingspeak.com/update?api_key=', writeAPIKey, '&field1=',
num2str(temperature)];
webwrite(url);
```

(b) Controlling a Device via MQTT (ESP32 + MATLAB)

1. Install MQTT Support in MATLAB.
2. Connect ESP32 to MQTT Broker (mqtt.eclipse.org).
3. Control an LED using MQTT:

```
mqttClient = mqtt('tcp://mqtt.eclipse.org');
subscribe(mqttClient, 'home/led');
write(mqttClient, 'home/led', 'ON');
```

(c) Real-Time Face Detection with Raspberry Pi Camera

```
r = raspi;
cam = cameraboard(r, 'Resolution', '640x480');
img = snapshot(cam);
faceDetector = vision.CascadeObjectDetector();
bbox = step(faceDetector, img);
imshow(insertObjectAnnotation(img, 'rectangle', bbox, 'Face'));
```

Applications of MATLAB in IoT:

✔ Smart Home Automation (ESP32, MQTT)

✔ Smart Agriculture (Weather Monitoring)

✔ Industrial IoT (Vibration Analysis)

Project Report

This project demonstrates how to use MATLAB to isolate and highlight blue regions in an image using the HSV (Hue, Saturation, Value) color model. The idea is to retain the blue parts of an image in color while turning all other areas grayscale.
Below is a step-by-step explanation of the code used in this project:

```
img = imread('waterbreathing.jpeg');
```

This line loads the image file named waterbreathing.jpeg into the variable img. It serves as the input for the entire image processing operation.

```
hsv_img = rgb2hsv(img);
```

This converts the original RGB image to HSV color space. HSV is often preferred in color detection tasks because the 'Hue' component directly represents color type.

```
H = hsv_img(:, :, 1); % Hue
S = hsv_img(:, :, 2); % Saturation V =
hsv_img(:, :, 3); % Value
```

This separates the HSV image into its three channels:
H captures the hue (color),
S captures the saturation (intensity of color), and
V captures the value or brightness.

```
blue_mask = (H >= 0.55 & H ◂ = 0.75) & (S > 0.2) & (V > 0.2);
```

This creates a logical mask that identifies pixels in the blue range. It selects pixels with:
Hue values between 0.55 and 0.75 (typical for blue tones),
Moderate to high saturation (above 0.2), and
Adequate brightness (value also above 0.2).

```
gray_img = repmat(rgb2gray(img), [1, 1, 3]);
```

This converts the original RGB image into grayscale using rgb2gray, and then replicates it across all three color channels. This is done so that the grayscale image has the same dimensions as the original RGB image.

```
result_img = img;
result_img(~repmat(blue_mask, [1, 1, 3])) = gray_img(~repmat(blue_mask, [1, 1, 3]));
```

A new image result_img is initialized as a copy of the original image. Then, all pixels not identified as blue (using the inverse of blue_mask) are replaced by corresponding grayscale pixels. This creates the effect where only the blue parts of the image are in color.

```
figure;
imshow(img);
title('Original Image');
```

This creates a figure window and displays the original image on the left side (first subplot). It's used for comparison purposes.

```
figure;
imshow(result_img);
title('Processed Image with HSV-based Blue Detection');
```

Displays the processed image on the right side (second subplot), showing the blue parts in color and all other areas in grayscale.