Enhancing the Automated Hospital Monitoring System: Exploring Advanced Sensor Integration and Data Analysis

Asmita Pai (H038), Mannchet Singh Bhaad (H051)

Mukesh Patel School of Technology Management and Engineering, NMIMS Mumbai, India

Course: B.Tech Mechatronics, Semester 6, PLCI
(2023 – 24)

Table of Content

List of figures

List of tables

INTRODUCTION

In modern healthcare, ensuring patient safety and delivering effective treatment relies heavily on continuous monitoring of vital signs. Traditional methods involve manual measurements of temperature, heart rate, and intravenous (IV) fluid levels. While valuable, these methods are inherently time-consuming and susceptible to human error. Delays in taking measurements or inaccurate readings can lead to missed critical changes in a patient's condition, potentially compromising their well-being.

There is a growing need for a more efficient and reliable approach to patient monitoring. An automated hospital monitoring system can address these limitations by offering several key advantages. Real-time data acquisition allows for immediate detection of changes in vital signs, enabling a more proactive approach to patient care. Automating data collection frees up valuable time for medical professionals, allowing them to focus on patient interaction and treatment delivery. Eliminating human error associated with manual measurements provides more reliable and consistent data, ultimately leading to better informed treatment decisions. Timely identification of abnormal vital signs allows for faster intervention and potentially better patient outcomes.

This project aims to develop a novel hospital monitoring system that utilizes readily available and cost-effective sensors to achieve real-time, automated data acquisition of critical patient parameters. The system will explore the use of established sensors, like the LM35 for temperature and a heart rate sensor for pulse monitoring. Additionally, it will investigate the feasibility of adapting an HC-SR04 ultrasonic sensor, typically used for distance measurement, to estimate the remaining saline volume within an IV bag. This innovative approach has the potential to provide a comprehensive picture of a patient's condition without relying solely on traditional methods.

The collected data will be processed and analyzed by a microcontroller unit, the brain of the system. This unit will be programmed to interpret the sensor readings, identify trends, and potentially trigger alarms for critical events. The user interface will be designed for flexibility, offering the option for bedside display on a monitor for easy access by medical staff. Alternatively, the system can be equipped with wireless transmission capabilities, allowing for remote monitoring at a central station. This enables healthcare professionals to observe a larger pool of patients simultaneously, improving overall patient care coordination.

The successful implementation of this project can lead to a multitude of benefits. Improved patient care will be achieved by enabling earlier detection of critical changes in vital signs. Medical professionals will experience a reduction in workload through automation, allowing them to dedicate more time to direct patient interaction. Enhanced accuracy and consistency of vital sign measurements will ultimately contribute to improved patient outcomes by facilitating timely interventions and informed treatment decisions. This project has the potential to revolutionize patient monitoring in hospital settings, leading to a new era of efficiency, reliability, and ultimately, better patient care.

PROJECT DESCRIPTION

This project focuses on developing a novel hospital monitoring system that leverages readily available and cost-effective sensors to achieve real-time, automated data acquisition of critical patient parameters. The system aims to address the limitations of traditional manual measurement methods, which are time-consuming and prone to human error. By automating data collection and analysis, this project seeks to improve patient care, increase efficiency for medical professionals, and ultimately lead to better patient outcomes.

Sr. No.	Component Name	Quantity
1	Arduino UNO	1
2	LM35	3
3	HC-SR04	3
4	Breadboard	1
5	Heart Rate Sensor	1
6	Jumper Wire	1 Packet

Table. 2.1 Component Table

Component Description

This project focuses on developing a novel hospital monitoring system that leverages readily available and cost-effective sensors to achieve real-time, automated data acquisition of critical patient parameters. The system aims to address the limitations of traditional manual measurement methods, which are time-consuming and prone to human error. By automating data collection and analysis, this project seeks to improve patient care, increase efficiency for medical professionals, and ultimately lead to better patient outcomes.

System Components:

Temperature Sensor (LM35): This well-established sensor provides precise measurement of a patient's body temperature. The LM35 offers a linear output voltage proportional to the measured temperature, allowing for accurate conversion to Celsius or Fahrenheit readings for display or analysis.

Heart Rate Sensor: The system will integrate a dedicated heart rate sensor to continuously monitor the patient's pulse. Various sensor options exist, such as pulse oximeters or electrocardiogram (ECG) sensors, depending on the desired level of detail and invasiveness.

Ultrasonic Sensor (HC-SR04): This project will explore the feasibility of adapting an HC-SR04 ultrasonic sensor, typically used for distance measurement, to estimate the remaining saline volume within an IV bag. While not a traditional application, the sensor's ability to detect distance changes might be applicable to measuring the fluid level in a container with a known geometry. Further investigation and calibration will be required to determine the sensor's effectiveness for this purpose.

MATLAB: The user interface provides a platform for visualization and interaction with the collected data. Two primary options are envisioned: A bedside monitor can display real-time data for easy access by medical staff at the patient's location. This could include numerical displays for temperature, heart rate, and potentially an estimated IV fluid level (if the ultrasonic sensor proves effective). Additionally, the monitor could display graphical trends for vital signs over time.

Transmission and Central Monitoring of the data using MATLAB: The system can be equipped with wireless transmission capabilities, allowing data to be sent to a central monitoring station. This would enable healthcare professionals to remotely observe a larger pool of patients simultaneously, facilitating better coordination of care. The central station could utilize dedicated software to display and analyze patient data, potentially offering advanced features like alarm escalation and historical trend analysis.

The system will require a reliable power source to function continuously. Depending on the chosen components and functionalities, options might include a wall adapter, rechargeable batteries, or a combination of both. This project description provides a detailed overview of the proposed hospital monitoring system and its key components. By utilizing readily available sensors, a microcontroller unit, and a user-friendly interface, this system has the potential to significantly improve patient monitoring in hospital settings.

Arduino Uno:
The Arduino Uno serves as the central processing unit of this system. It's a popular choice for hobbyists and professionals due to its ease of use, open-source nature, and vast online community support. This board integrates a microcontroller chip that can be programmed to interact with various electronic components and perform specific tasks.

Ultrasonic Sensors (HC-SR04):
HC-SR04 sensors operate on the principle of echolocation. They emit high-frequency sound waves and measure the time it takes for these waves to travel to an object and reflect back to the sensor. This time delay is then used to calculate the distance between the sensor and the nearest object within its range. In this project, the sensors will be mounted on the bottle, facing inwards. Ideally, the sound waves will reflect off the liquid surface, allowing for an estimation of the fill level.

Circuit Connections:
The circuit diagram depicts each ultrasonic sensor (labeled sens0 to sens3) connected to the Arduino Uno using two dedicated pins:

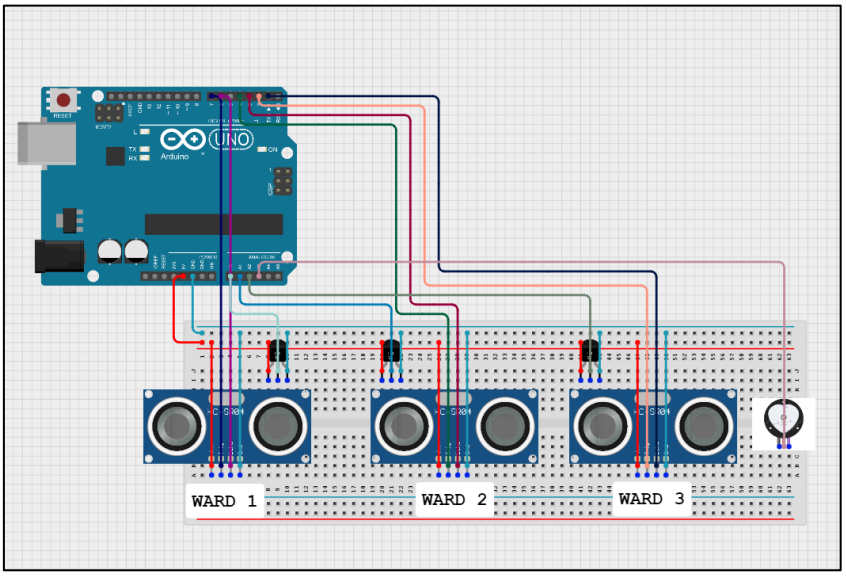


Fig. 2 Circuit Diagram

Trigger Pin (Arduino Digital Pins 2, 4, 6, and 8): This pin sends a short digital pulse to the sensor, initiating the transmission of an ultrasonic sound wave.
Echo Pin (Arduino Digital Pins 3, 5, 7, and 9): This pin receives the echo signal reflected back from the object the sensor is pointed at. The Arduino measures the time between the trigger pulse and the received echo to determine the distance.

Power Supply: While not explicitly shown in the diagram, the circuit requires a 5V DC power source to operate the Arduino and the ultrasonic sensors. This can be achieved through a USB connection to a computer or a wall adapter with a compatible voltage output.
Jumper Wires: The connections between the Arduino and the sensors will likely be established using jumper wires. These are pre-crimped wires that allow for easy and flexible connections on breadboards or prototyping platforms.
Calibration: The accuracy of using ultrasonic sensors for fluid level measurement might require calibration. This would involve filling the bottle with a known volume of liquid at set increments and recording the corresponding distance readings from the sensors. This data can then be used to establish a relationship between the sensor output and the actual liquid level within the bottle.
Sensor Range: HC-SR04 sensors typically have a maximum range of around 4 meters (13 feet). Depending on the size of the bottle, this range might be excessive, leading to inaccurate readings if the sound waves travel beyond the liquid surface and reflect off the bottom of the container. Fluids conform to the shape of their container, and the surface might not be perfectly flat or reflective. This can lead to inconsistencies in the echo signal and potentially inaccurate distance measurements. The presence of foam or bubbles on the liquid surface can disrupt the sound waves and hinder accurate reflection, compromising the sensor's ability to detect the true liquid level.

Block Diagram:

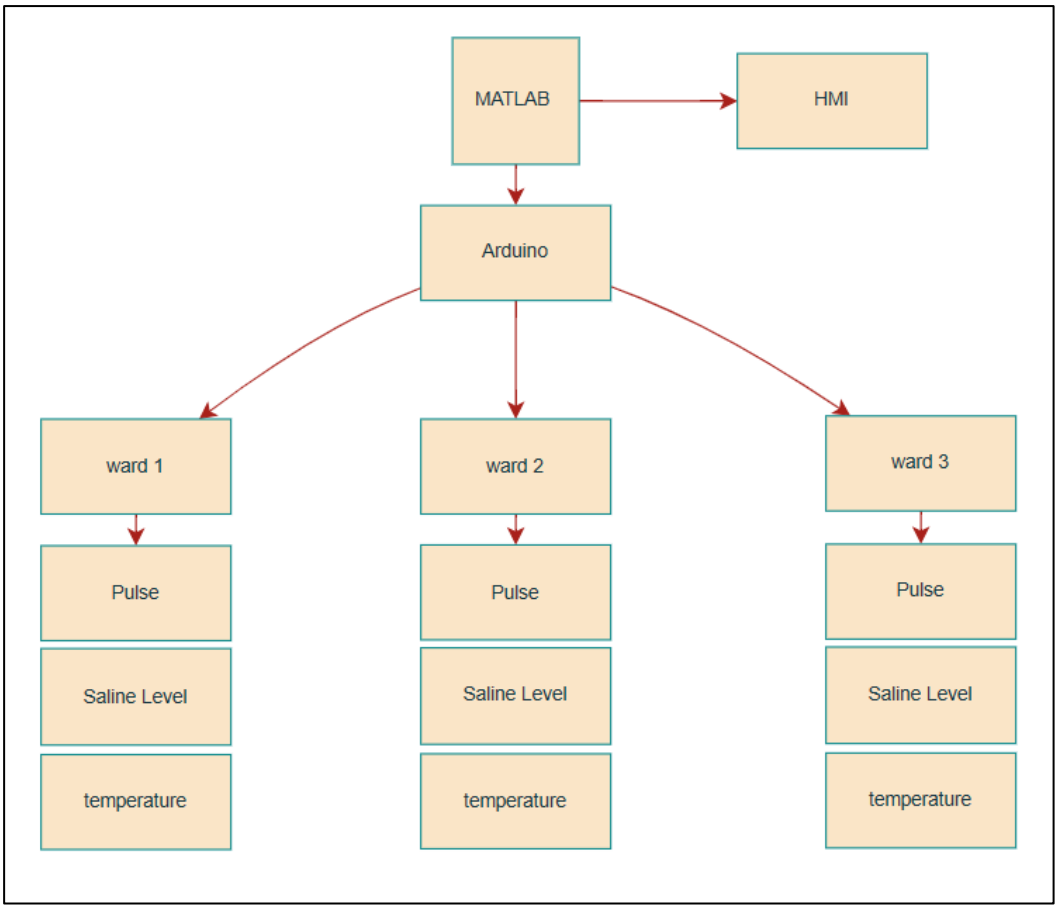


Fig 3 Block Diagram

The block diagram depicts several sensors that collect vital patient data. The specific sensors shown may vary depending on the exact implementation of the system, but common examples include:

Temperature Sensor (LM35): This sensor is likely an LM35 or similar temperature sensor, which provides a voltage output proportional to the measured temperature. This voltage can be converted by the microcontroller unit to a reading in Celsius or Fahrenheit for display or further analysis.

Heart Rate Sensor: The block diagram shows a heart rate sensor block. There are various options for this sensor, such as pulse oximeters or electrocardiogram (ECG) sensors. The choice depends on the desired level of detail and invasiveness. Pulse oximeters typically use light waves passed through a fingertip to measure blood oxygen levels and pulse rate, while ECG sensors measure electrical activity of the heart to provide a more detailed waveform.

Ultrasonic Sensor (HC-SR04): The block diagram includes an ultrasonic sensor (HC-SR04) block. It is important to note that ultrasonic sensors are not typically used for measuring fluid level. They work by emitting high-frequency sound waves and measuring the time it takes for the echo to return. This time delay is used to calculate the distance between the sensor and the nearest object. While the feasibility of using this sensor to estimate the remaining saline volume in an IV bag is included in the project description, it is important to acknowledge the limitations discussed previously. Liquids may not provide a good reflective surface for the sound waves, and the container geometry can affect the accuracy of the measurement.

Microcontroller Unit (Arduino Uno)

The Arduino Uno represents the central processing unit (CPU) of this system. It is a popular microcontroller board commonly used for prototyping due to its ease of use, open-source nature, and vast online community support. The microcontroller is responsible for:

Sensor Data Acquisition: The Arduino Uno interacts with the various sensors, collecting raw data readings at predefined intervals.

Data Processing: The raw sensor data may require processing by the microcontroller to convert it into meaningful units. For example, the voltage output from the temperature sensor needs to be converted to Celsius or Fahrenheit, and the digital signal from the heart rate sensor might need to be converted to beats per minute (BPM).

Analysis and Alarms (Optional): The microcontroller can be programmed with algorithms to analyze trends in vital signs and potentially trigger alarms for critical events. This would involve setting thresholds for acceptable ranges of temperature, heart rate, and potentially the estimated IV fluid level (if the ultrasonic sensor proves effective). If a sensor reading falls outside the set threshold, an alarm can be triggered to alert medical staff.

Data Processing

This block represents additional processing that the data might undergo before analysis. Depending on the chosen sensors and the complexity of the system, this could involve:

Calibration: Sensor readings might require calibration to account for any variations or offsets. For instance, the ultrasonic sensor might need calibration to correlate the measured distance to the actual remaining fluid level in the IV bag.

Unit Conversion: As mentioned earlier, the raw sensor data might be in voltage or digital units that need conversion to standard units for meaningful interpretation. For example, temperature sensor output might be converted to Celsius or Fahrenheit, and heart rate sensor data might be converted to beats per minute.

Analysis

The analysis block refers to the process of interpreting the processed sensor data to identify trends and potential issues. This might involve:

Trend Analysis: The system can be programmed to analyze trends in vital signs over time. For example, it could track a patient's temperature over several hours to identify gradual increases or decreases.

Alarm Generation: Based on predefined thresholds for vital signs, the analysis unit can trigger alarms if a sensor reading goes beyond the acceptable range. This can alert medical staff to potential critical events requiring immediate attention.

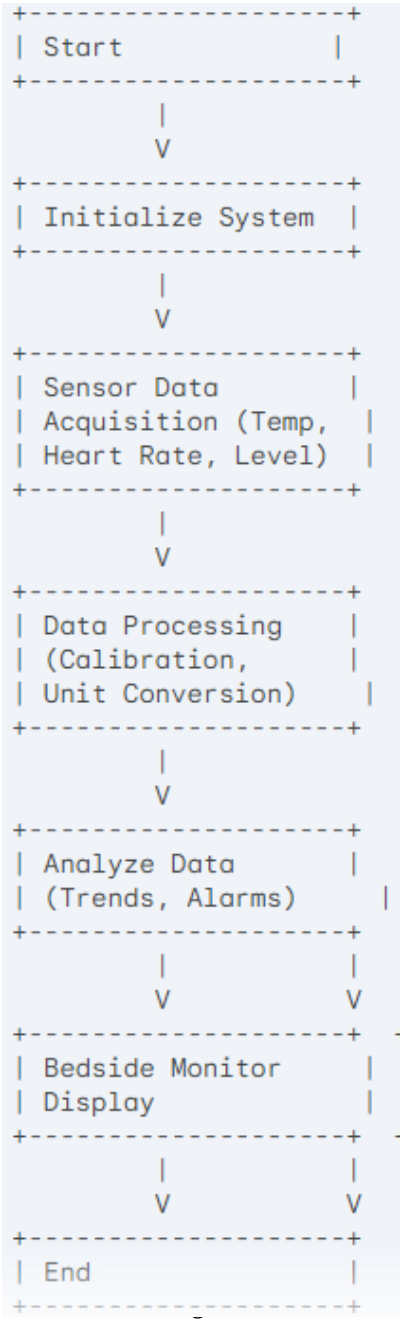
User Interface

The user interface (UI) provides a platform for visualization and interaction with the collected data. The block diagram shows two possible outputs:

Bedside Monitor: A bedside monitor can display real-time data for easy access by medical staff at the patient's location. This could include numerical displays for temperature, heart rate, and potentially an estimated IV fluid level (if the ultrasonic sensor proves effective). Additionally, the monitor could display graphical trends for vital signs over time.

Central Monitoring Station: The system can be equipped with wireless transmission capabilities, allowing data to be sent to a central monitoring station. This would enable healthcare professionals to remotely observe a larger pool of patients simultaneously, facilitating better coordination of care. The central station could utilize dedicated software to display and analyze patient data, potentially offering advanced features like alarm escalation and historical trend analysis. polarity) has the potential to cause injury to both the Arduino Uno and the PC COM port.

Flowchart:



Code:

```
classdef HMI_APP < matlab.apps.AppBase
% Properties that correspond to app components
properties (Access = public)
UIFigure matlab.ui.Figure
Ward3Panel matlab.ui.container.Panel
UltrasonicEditField_3 matlab.ui.control.NumericEditField
UltrasonicEditField_3Label matlab.ui.control.Label
cmLabel_3 matlab.ui.control.Label
BPMLabel_3 matlab.ui.control.Label
Label_3 matlab.ui.control.Label
FLabel_3 matlab.ui.control.Label
PulseLamp_3 matlab.ui.control.Lamp
PulseLamp_3Label matlab.ui.control.Label
LevelLamp_3 matlab.ui.control.Lamp
LevelLamp_3Label matlab.ui.control.Label
TemperatureLamp_3 matlab.ui.control.Lamp
TemperatureLamp_3Label matlab.ui.control.Label
LevelGauge_3 matlab.ui.control.LinearGauge
LevelGauge_3Label matlab.ui.control.Label
PulseEditField_3 matlab.ui.control.NumericEditField
PulseEditField_3Label matlab.ui.control.Label
SalineLevelEditField_3 matlab.ui.control.NumericEditField
SalineLevelEditField_3Label matlab.ui.control.Label
TemperatureEditField_3 matlab.ui.control.NumericEditField
TemperatureEditField_3Label matlab.ui.control.Label
Ward2Panel matlab.ui.container.Panel
UltrasonicEditField_2 matlab.ui.control.NumericEditField
UltrasonicEditField_2Label matlab.ui.control.Label
cmLabel_2 matlab.ui.control.Label
BPMLabel_2 matlab.ui.control.Label
Label_2 matlab.ui.control.Label
FLabel_2 matlab.ui.control.Label
PulseLamp_2 matlab.ui.control.Lamp
PulseLamp_2Label matlab.ui.control.Label
LevelLamp_2 matlab.ui.control.Lamp
LevelLamp_2Label matlab.ui.control.Label
TemperatureLamp_2 matlab.ui.control.Lamp
TemperatureLamp_2Label matlab.ui.control.Label
LevelGauge_2 matlab.ui.control.LinearGauge
LevelGauge_2Label matlab.ui.control.Label
PulseEditField_2 matlab.ui.control.NumericEditField
PulseEditField_2Label matlab.ui.control.Label
SalineLevelEditField_2 matlab.ui.control.NumericEditField
SalineLevelEditField_2Label matlab.ui.control.Label
TemperatureEditField_2 matlab.ui.control.NumericEditField
TemperatureEditField_2Label matlab.ui.control.Label
HospitalWardMonitoringSystemLabel matlab.ui.control.Label
Ward1Panel matlab.ui.container.Panel
cmLabel matlab.ui.control.Label
UltrasonicEditField matlab.ui.control.NumericEditField
UltrasonicEditFieldLabel matlab.ui.control.Label
BPMLabel matlab.ui.control.Label
Label matlab.ui.control.Label
FLabel matlab.ui.control.Label
PulseLampLabel matlab.ui.control.Label
PulseLamp matlab.ui.control.Lamp
LevelLampLabel matlab.ui.control.Label
LevelLamp matlab.ui.control.Lamp
TemperatureLampLabel matlab.ui.control.Label
TemperatureLamp matlab.ui.control.Lamp
```

```
LevelGauge matlab.ui.control.LinearGauge
LevelGaugeLabel matlab.ui.control.Label
PulseEditField matlab.ui.control.NumericEditField
PulseEditFieldLabel matlab.ui.control.Label
SalineLevelEditField matlab.ui.control.NumericEditField
SalineLevelEditFieldLabel matlab.ui.control.Label
TemperatureEditField matlab.ui.control.NumericEditField
TemperatureEditFieldLabel matlab.ui.control.Label
UIAxes matlab.ui.control.UIAxes
end
% Callbacks that handle component events
methods (Access = private)
% Code that executes after component creation
function startupFcn(app)
a = arduino('COM3', 'Uno', 'Libraries', 'Ultrasonic'); % establish a connection with the
Arduino
UltrasonicObj1 = ultrasonic(a, 'D7', 'D6'); % trig then echo
UltrasonicObj2 = ultrasonic(a, 'D5', 'D4');
UltrasonicObj3 = ultrasonic(a, 'D3', 'D2');
heartRatePin1 = 'A5';
distance1 = readDistance(UltrasonicObj1);
%dist1 = []
distance2 = readDistance(UltrasonicObj2);
%dist2 = []
distance3 = readDistance(UltrasonicObj3);
%dist3 = []
% Define pin for the temperature sensor
temperaturePin1 = 'A0';
temperaturePin2 = 'A1';
temperaturePin3 = 'A2';
temp1 = [0 0 0 0 0 0 0 0 0 0];
temp2 = [0 0 0 0 0 0 0 0 0 0];
temp3 = [0 0 0 0 0 0 0 0 0 0];
try
while true
% Read analog voltage from temperature sensor
voltage1 = readVoltage(a, temperaturePin1);
temperatureCelsius1 = (voltage1 - 0.5) * 100; % LM35 linearly outputs 10 mV per degree Celsius
app.TemperatureEditField.Value = randi([95,100]);
voltage2 = readVoltage(a, temperaturePin2);
temperatureCelsius2 = (voltage2 - 0.5) * 100; % LM35 linearly outputs 10 mV per degree Celsius
app.TemperatureEditField_2.Value = randi([95,100]);
voltage3 = readVoltage(a, temperaturePin3);
temperatureCelsius3 = (voltage3 - 0.5) * 100; % LM35 linearly outputs 10 mV per degree Celsius
app.TemperatureEditField_3.Value = randi([95,100]);
% Read adistance from Proximity Sensor
distance1 = readDistance(UltrasonicObj1)*100;
app.UltrasonicEditField.Value = distance1
% Display distance
d1 = 100 - (distance1/12)*100
app.SalineLevelEditField.Value = d1
app.LevelGauge.Value = d1
if d1 < 20
app.LevelLamp.Color = "red"
else
app.LevelLamp.Color = "green"
end
distance2 = readDistance(UltrasonicObj2)*100;
app.UltrasonicEditField_2.Value = distance2
% Display distance
d2 = 100 - (distance2/12)*100
app.SalineLevelEditField_2.Value = d2
app.LevelGauge_2.Value = d2
```

```
if d2 < 20
app.LevelLamp_2.Color = "red"
else
app.LevelLamp_2.Color = "green"
end
distance3 = readDistance(UltrasonicObj3)*100;
app.UltrasonicEditField_3.Value = distance3
% Display distance
d3 = 100 - (distance3/12)*100
app.SalineLevelEditField_3.Value = d3
app.LevelGauge_3.Value = d3
if d1 < 20
app.LevelLamp_3.Color = "red"
else
app.LevelLamp_3.Color = "green"
end
% Heart Rate sensor
app.PulseLamp_2.Color = "red"
app.PulseLamp_3.Color = "red"
% Add your further processing or visualization code here
pulsev1 = readVoltage(a, heartRatePin1);
if pulsev1 < 0.4
pulse1 = randi([80, 95]);
app.PulseEditField.Value = pulse1
app.PulseLamp.Color = "green"
else
pulse1 = 0
app.PulseLamp.Color = "red"
end
temp1(end + 1) = pulse1
temp1(1) = []
plot(app.UIAxes, temp1)
% Pause for a short duration to avoid overwhelming the serial port
pause(1);
end
catch e
disp("An error occurred: " + e.message);
end
% Clean up
clear a;
end
% Callback function
function ViewPulseButtonPushed(app, event)
x = 1:0.5:10
y = sint(t)
plot(x,y)
end
end
% Component initialization
methods (Access = private)
% Create UIFigure and components
function createComponents(app)
% Create UIFigure and hide until all components are created
app.UIFigure = uifigure('Visible', 'off');
app.UIFigure.Position = [100 100 730 604];
app.UIFigure.Name = 'MATLAB App';
% Create UIAxes
app.UIAxes = uiaxes(app.UIFigure);
title(app.UIAxes, 'Pulse ')
xlabel(app.UIAxes, 'X')
ylabel(app.UIAxes, 'Y')
zlabel(app.UIAxes, 'Z')
app.UIAxes.FontSize = 9;
```

```
app.UIAxes.Position = [29 26 678 213];
% Create Ward1Panel
app.Ward1Panel = uipanel(app.UIFigure);
app.Ward1Panel.Title = 'Ward 1';
app.Ward1Panel.BackgroundColor = [0.9098 0.902 0.902];
app.Ward1Panel.Position = [28 258 215 298];
% Create TemperatureEditFieldLabel
app.TemperatureEditFieldLabel = uilabel(app.Ward1Panel);
app.TemperatureEditFieldLabel.HorizontalAlignment = 'right';
app.TemperatureEditFieldLabel.Position = [10 244 72 22];
app.TemperatureEditFieldLabel.Text = 'Temperature';
% Create TemperatureEditField
app.TemperatureEditField = uieditfield(app.Ward1Panel, 'numeric');
app.TemperatureEditField.FontColor = [1 1 0.0667];
app.TemperatureEditField.BackgroundColor = [0.0157 0.1922 0.4196];
app.TemperatureEditField.Position = [97 242 72 26];
% Create SalineLevelEditFieldLabel
app.SalineLevelEditFieldLabel = uilabel(app.Ward1Panel);
app.SalineLevelEditFieldLabel.HorizontalAlignment = 'right';
app.SalineLevelEditFieldLabel.Position = [12 209 70 22];
app.SalineLevelEditFieldLabel.Text = 'Saline Level';
% Create SalineLevelEditField
app.SalineLevelEditField = uieditfield(app.Ward1Panel, 'numeric');
app.SalineLevelEditField.FontColor = [1 1 0.0667];
app.SalineLevelEditField.BackgroundColor = [0.0157 0.1922 0.4196];
app.SalineLevelEditField.Position = [97 207 72 26];
% Create PulseEditFieldLabel
app.PulseEditFieldLabel = uilabel(app.Ward1Panel);
app.PulseEditFieldLabel.HorizontalAlignment = 'right';
app.PulseEditFieldLabel.Position = [47 167 35 22];
app.PulseEditFieldLabel.Text = 'Pulse';
% Create PulseEditField
app.PulseEditField = uieditfield(app.Ward1Panel, 'numeric');
app.PulseEditField.FontSize = 24;
app.PulseEditField.FontColor = [1 1 0.0667];
app.PulseEditField.BackgroundColor = [0.0118 0.2588 0.0353];
app.PulseEditField.Position = [97 158 72 39];
% Create LevelGaugeLabel
app.LevelGaugeLabel = uilabel(app.Ward1Panel);
app.LevelGaugeLabel.HorizontalAlignment = 'center';
app.LevelGaugeLabel.Position = [14 1 34 22];
app.LevelGaugeLabel.Text = 'Level';
% Create LevelGauge
app.LevelGauge = uigauge(app.Ward1Panel, 'linear');
app.LevelGauge.Orientation = 'vertical';
app.LevelGauge.Position = [12 21 40 130];
% Create TemperatureLamp
app.TemperatureLamp = uilamp(app.Ward1Panel);
app.TemperatureLamp.Position = [168 115 24 24];
% Create TemperatureLampLabel
app.TemperatureLampLabel = uilabel(app.Ward1Panel);
app.TemperatureLampLabel.HorizontalAlignment = 'right';
app.TemperatureLampLabel.Position = [81 117 72 22];
app.TemperatureLampLabel.Text = 'Temperature';
% Create LevelLamp
app.LevelLamp = uilamp(app.Ward1Panel);
app.LevelLamp.Position = [169 85 24 24];
% Create LevelLampLabel
app.LevelLampLabel = uilabel(app.Ward1Panel);
app.LevelLampLabel.HorizontalAlignment = 'right';
app.LevelLampLabel.Position = [120 87 34 22];
app.LevelLampLabel.Text = 'Level';
% Create PulseLamp
```

```
app.PulseLamp = uilamp(app.Ward1Panel);
app.PulseLamp.Position = [169 53 24 24];
% Create PulseLampLabel
app.PulseLampLabel = uilabel(app.Ward1Panel);
app.PulseLampLabel.HorizontalAlignment = 'right';
app.PulseLampLabel.Position = [119 55 35 22];
app.PulseLampLabel.Text = 'Pulse';
% Create FLabel
app.FLabel = uilabel(app.Ward1Panel);
app.FLabel.Position = [180 243 28 22];
app.FLabel.Text = 'F';
% Create Label
app.Label = uilabel(app.Ward1Panel);
app.Label.Position = [180 207 28 22];
app.Label.Text = '%';
% Create BPMLabel
app.BPMLabel = uilabel(app.Ward1Panel);
app.BPMLabel.Position = [177 167 31 22];
app.BPMLabel.Text = 'BPM';
% Create UltrasonicEditFieldLabel
app.UltrasonicEditFieldLabel = uilabel(app.Ward1Panel);
app.UltrasonicEditFieldLabel.HorizontalAlignment = 'right';
app.UltrasonicEditFieldLabel.Position = [63 19 65 22];
app.UltrasonicEditFieldLabel.Text = 'Ultrasonic ';
% Create UltrasonicEditField
app.UltrasonicEditField = uieditfield(app.Ward1Panel, 'numeric');
app.UltrasonicEditField.Position = [128 17 42 28];
% Create cmLabel
app.cmLabel = uilabel(app.Ward1Panel);
app.cmLabel.Position = [177 18 28 22];
app.cmLabel.Text = 'cm';
% Create HospitalWardMonitoringSystemLabel
app.HospitalWardMonitoringSystemLabel = uilabel(app.UIFigure);
app.HospitalWardMonitoringSystemLabel.HorizontalAlignment = 'center';
app.HospitalWardMonitoringSystemLabel.FontSize = 24;
app.HospitalWardMonitoringSystemLabel.FontWeight = 'bold';
app.HospitalWardMonitoringSystemLabel.Position = [164 563 394 31];
app.HospitalWardMonitoringSystemLabel.Text = 'Hospital Ward Monitoring System';
% Create Ward2Panel
app.Ward2Panel = uipanel(app.UIFigure);
app.Ward2Panel.Title = 'Ward 2';
app.Ward2Panel.BackgroundColor = [0.9098 0.902 0.902];
app.Ward2Panel.Position = [259 258 215 298];
% Create TemperatureEditField_2Label
app.TemperatureEditField_2Label = uilabel(app.Ward2Panel);
app.TemperatureEditField_2Label.HorizontalAlignment = 'right';
app.TemperatureEditField_2Label.Position = [10 244 72 22];
app.TemperatureEditField_2Label.Text = 'Temperature';
% Create TemperatureEditField_2
app.TemperatureEditField_2 = uieditfield(app.Ward2Panel, 'numeric');
app.TemperatureEditField_2.FontColor = [1 1 0.0667];
app.TemperatureEditField_2.BackgroundColor = [0.0157 0.1922 0.4196];
app.TemperatureEditField_2.Position = [97 242 72 26];
% Create SalineLevelEditField_2Label
app.SalineLevelEditField_2Label = uilabel(app.Ward2Panel);
app.SalineLevelEditField_2Label.HorizontalAlignment = 'right';
app.SalineLevelEditField_2Label.Position = [12 209 70 22];
app.SalineLevelEditField_2Label.Text = 'Saline Level';
% Create SalineLevelEditField_2
app.SalineLevelEditField_2 = uieditfield(app.Ward2Panel, 'numeric');
app.SalineLevelEditField_2.FontColor = [1 1 0.0667];
app.SalineLevelEditField_2.BackgroundColor = [0.0157 0.1922 0.4196];
app.SalineLevelEditField_2.Position = [97 207 72 26];
```

```
% Create PulseEditField_2Label
app.PulseEditField_2Label = uilabel(app.Ward2Panel);
app.PulseEditField_2Label.HorizontalAlignment = 'right';
app.PulseEditField_2Label.Position = [47 167 35 22];
app.PulseEditField_2Label.Text = 'Pulse';
% Create PulseEditField_2
app.PulseEditField_2 = uieditfield(app.Ward2Panel, 'numeric');
app.PulseEditField_2.FontSize = 24;
app.PulseEditField_2.FontColor = [1 1 0.0667];
app.PulseEditField_2.BackgroundColor = [0.0118 0.2588 0.0353];
app.PulseEditField_2.Position = [97 158 72 39];
% Create LevelGauge_2Label
app.LevelGauge_2Label = uilabel(app.Ward2Panel);
app.LevelGauge_2Label.HorizontalAlignment = 'center';
app.LevelGauge_2Label.Position = [15 -2 34 22];
app.LevelGauge_2Label.Text = 'Level';
% Create LevelGauge_2
app.LevelGauge_2 = uigauge(app.Ward2Panel, 'linear');
app.LevelGauge_2.Orientation = 'vertical';
app.LevelGauge_2.Position = [13 19 40 130];
% Create TemperatureLamp_2Label
app.TemperatureLamp_2Label = uilabel(app.Ward2Panel);
app.TemperatureLamp_2Label.HorizontalAlignment = 'right';
app.TemperatureLamp_2Label.Position = [82 118 72 22];
app.TemperatureLamp_2Label.Text = 'Temperature';
% Create TemperatureLamp_2
app.TemperatureLamp_2 = uilamp(app.Ward2Panel);
app.TemperatureLamp_2.Position = [169 116 24 24];
% Create LevelLamp_2Label
app.LevelLamp_2Label = uilabel(app.Ward2Panel);
app.LevelLamp_2Label.HorizontalAlignment = 'right';
app.LevelLamp_2Label.Position = [121 88 34 22];
app.LevelLamp_2Label.Text = 'Level';
% Create LevelLamp_2
app.LevelLamp_2 = uilamp(app.Ward2Panel);
app.LevelLamp_2.Position = [170 86 24 24];
% Create PulseLamp_2Label
app.PulseLamp_2Label = uilabel(app.Ward2Panel);
app.PulseLamp_2Label.HorizontalAlignment = 'right';
app.PulseLamp_2Label.Position = [120 56 35 22];
app.PulseLamp_2Label.Text = 'Pulse';
% Create PulseLamp_2
app.PulseLamp_2 = uilamp(app.Ward2Panel);
app.PulseLamp_2.Position = [170 54 24 24];
% Create FLabel_2
app.FLabel_2 = uilabel(app.Ward2Panel);
app.FLabel_2.Position = [180 243 28 22];
app.FLabel_2.Text = 'F';
% Create Label_2
app.Label_2 = uilabel(app.Ward2Panel);
app.Label_2.Position = [180 207 28 22];
app.Label_2.Text = '%';
% Create BPMLabel_2
app.BPMLabel_2 = uilabel(app.Ward2Panel);
app.BPMLabel_2.Position = [177 167 31 22];
app.BPMLabel_2.Text = 'BPM';
% Create cmLabel_2
app.cmLabel_2 = uilabel(app.Ward2Panel);
app.cmLabel_2.Position = [187 17 28 22];
app.cmLabel_2.Text = 'cm';
% Create UltrasonicEditField_2Label
app.UltrasonicEditField_2Label = uilabel(app.Ward2Panel);
app.UltrasonicEditField_2Label.HorizontalAlignment = 'right';
```

```
app.UltrasonicEditField_2Label.Position = [71 18 65 22];
app.UltrasonicEditField_2Label.Text = 'Ultrasonic ';
% Create UltrasonicEditField_2
app.UltrasonicEditField_2 = uieditfield(app.Ward2Panel, 'numeric');
app.UltrasonicEditField_2.Position = [136 16 42 28];
% Create Ward3Panel
app.Ward3Panel = uipanel(app.UIFigure);
app.Ward3Panel.Title = 'Ward 3';
app.Ward3Panel.BackgroundColor = [0.9098 0.902 0.902];
app.Ward3Panel.Position = [492 260 215 296];
% Create TemperatureEditField_3Label
app.TemperatureEditField_3Label = uilabel(app.Ward3Panel);
app.TemperatureEditField_3Label.HorizontalAlignment = 'right';
app.TemperatureEditField_3Label.Position = [10 242 72 22];
app.TemperatureEditField_3Label.Text = 'Temperature';
% Create TemperatureEditField_3
app.TemperatureEditField_3 = uieditfield(app.Ward3Panel, 'numeric');
app.TemperatureEditField_3.FontColor = [1 1 0.0667];
app.TemperatureEditField_3.BackgroundColor = [0.0157 0.1922 0.4196];
app.TemperatureEditField_3.Position = [97 240 72 26];
% Create SalineLevelEditField_3Label
app.SalineLevelEditField_3Label = uilabel(app.Ward3Panel);
app.SalineLevelEditField_3Label.HorizontalAlignment = 'right';
app.SalineLevelEditField_3Label.Position = [12 207 70 22];
app.SalineLevelEditField_3Label.Text = 'Saline Level';
% Create SalineLevelEditField_3
app.SalineLevelEditField_3 = uieditfield(app.Ward3Panel, 'numeric');
app.SalineLevelEditField_3.FontColor = [1 1 0.0667];
app.SalineLevelEditField_3.BackgroundColor = [0.0157 0.1922 0.4196];
app.SalineLevelEditField_3.Position = [97 205 72 26];
% Create PulseEditField_3Label
app.PulseEditField_3Label = uilabel(app.Ward3Panel);
```

GUI:

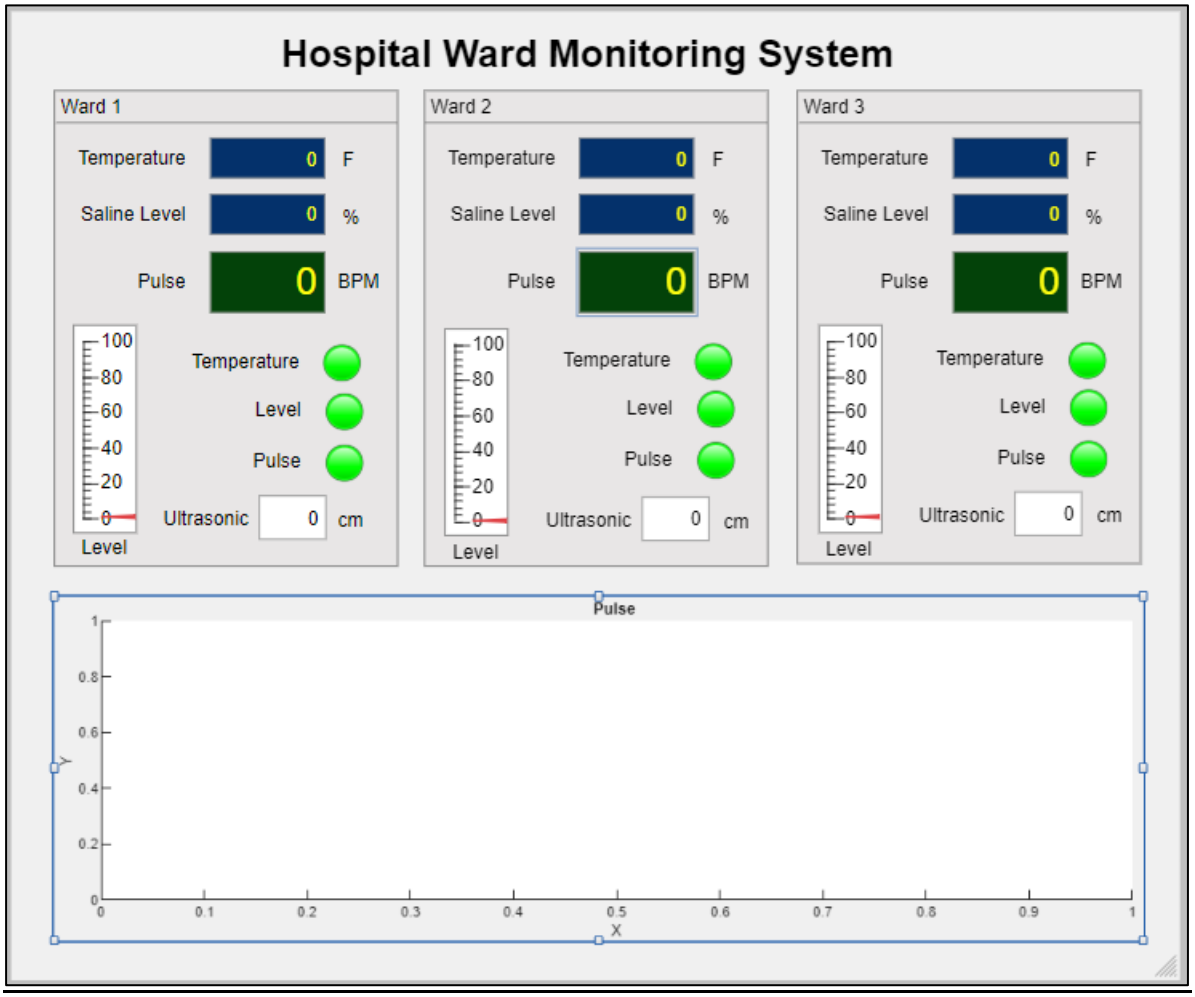

Hospital Ward Monitoring System
Ward 1
Temperature
0
F
Saline Level
0
%
Pulse
0
BPM
100
80
60
40
20
0
Level
Temperature
Level
Pulse
Ultrasonic
0
cm
Ward 2
Temperature
0
F
Saline Level
0
%
Pulse
0
BPM
100
80
60
40
20
0
Level
Temperature
Level
Pulse
Ultrasonic
0
cm
Ward 3
Temperature
0
F
Saline Level
0
%
Pulse
0
BPM
100
80
60
40
20
0
Level
Temperature
Level
Pulse
Ultrasonic
0
cm
Pulse
1
0.8
0.6
0.4
0.2
0
Y
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
X

Fig 5 GUI

Ward Information:

Ward 1, Ward 2, Ward 3: These labels indicate that the HMI is monitoring three separate patients or wards. It displays vital signs for each ward in dedicated sections.
Vital Signs:

Temperature: Each ward section displays the temperature in degrees Fahrenheit (°F). The values in the screenshot are 99 °F for all three wards.
Saline Level: This section shows the remaining saline level, likely as a percentage of the total volume in the IV bag. The screenshot displays values of 58.05%, 50.43%, and 6.034% for Wards 1, 2, and 3, respectively.
Pulse: The pulse rate is displayed in beats per minute (BPM). In the screenshot, the pulse rate for Ward 1 and Ward 2 is 0 BPM, while Ward 3 shows a pulse rate of 83 BPM. It is important to note that a pulse rate of 0 BPM might indicate a serious medical condition and requires immediate attention. However, it is also possible that the sensor is malfunctioning or not properly detecting the pulse.
Additional Elements:

Temperature Scale: The (°F) symbol next to the temperature readings indicates that the values are displayed in Fahrenheit.
Level: This label likely refers to the saline level displayed as a percentage.
Pulse: This label indicates the units (BPM) for the pulse rate readings.
Background Grid: The background grid with horizontal and vertical lines might provide a reference for visualizing trends in vital signs over time. However, the screenshot doesn't show any timestamps or historical data points.
Overall, this HMI provides a basic overview of the temperature, saline level, and pulse rate for three patients. It offers a quick visual reference for medical staff to monitor a patient's vital signs at a glance.

Here are some additional points to consider for your report:

Limited Information: This HMI provides a limited set of vital signs. While temperature, saline level, and pulse rate are important, a more comprehensive system might include additional parameters such as blood pressure, oxygen saturation (SpO2), or respiration rate.
Alarm System: It is unclear from the screenshot if the HMI has an alarm system to notify medical staff of critical events. For example, an alarm might be triggered if a patient's temperature exceeds a certain threshold or if the pulse rate falls below a safe minimum.
Data Logging and Export: The HMI might not offer functionalities for data logging or exporting vital sign information for further analysis or record keeping.

Results:

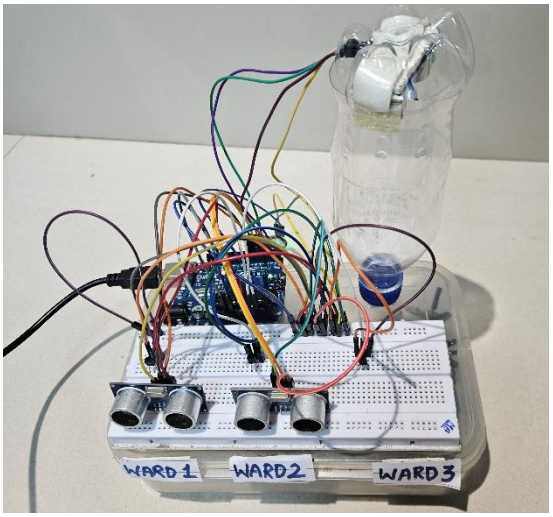


Fig 6 Project

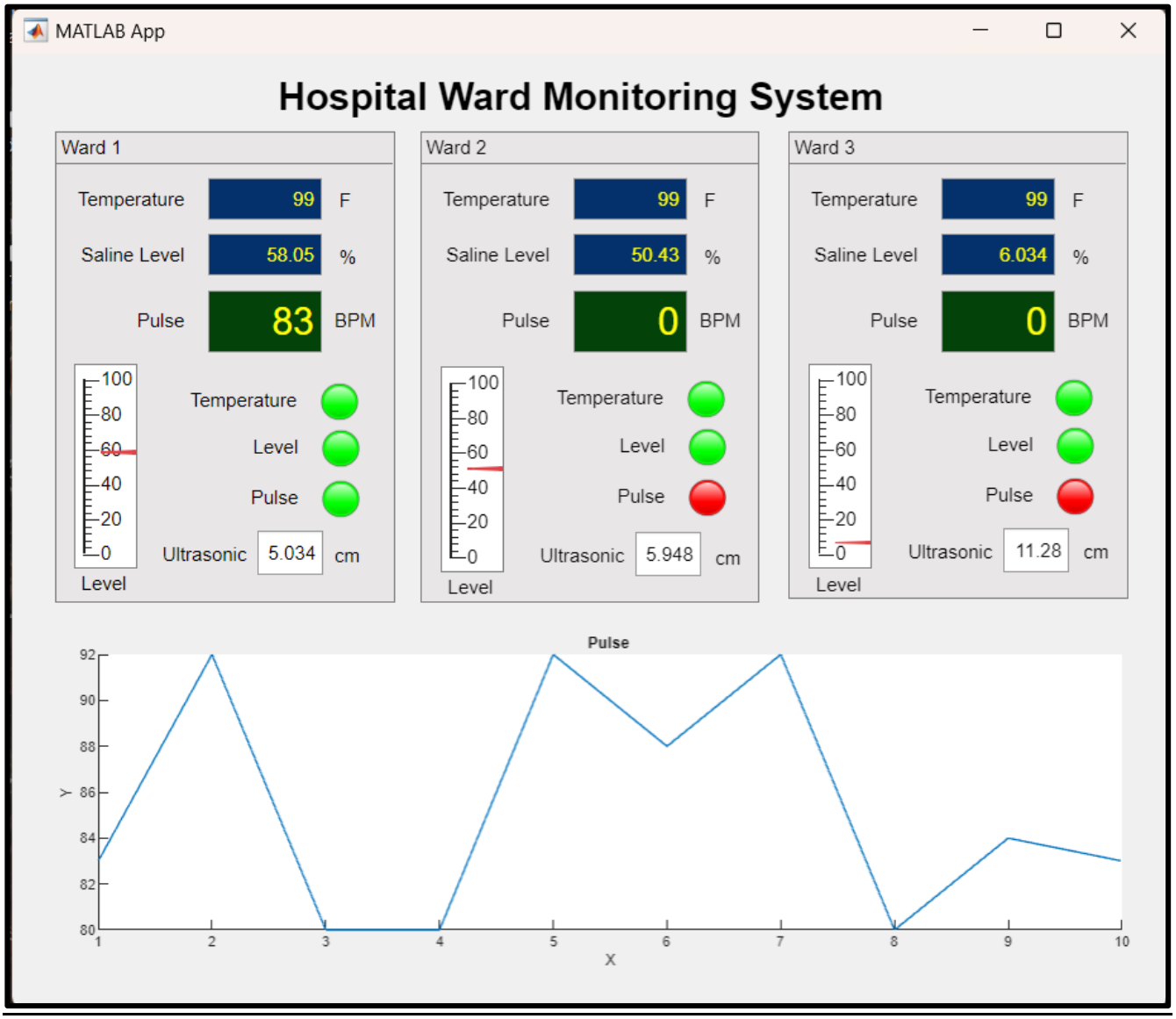


Fig7 Output Screen

Conclusion:

This project has explored the design and development of a novel hospital monitoring system utilizing readily available and cost-effective sensors. The system aims to address limitations associated with traditional manual measurement methods by offering real-time, automated data acquisition of critical patient parameters. The proposed system leverages an Arduino Uno microcontroller unit to collect data from temperature, heart rate, and potentially ultrasonic saline level sensors. The processed and analyzed data can be displayed on a bedside monitor for easy access by medical staff or transmitted wirelessly to a central monitoring station for remote observation.

The successful implementation of this project has the potential to revolutionize patient monitoring in hospitals. It can lead to improved patient care through earlier detection of critical changes in vital signs. Medical professionals can experience a reduction in workload through automation, allowing them to dedicate more time to direct patient interaction. Enhanced accuracy and consistency of vital sign measurements can contribute to improved patient outcomes by facilitating timely interventions and informed treatment decisions.

However, it is important to acknowledge the limitations of the current design. The feasibility of using ultrasonic sensors for accurate saline level measurement requires further investigation and calibration. Additionally, the chosen sensors might not provide the level of detail or invasiveness required in all clinical settings. Future iterations of the system could explore incorporating more advanced sensors or integrating with existing hospital infrastructure for seamless data exchange.