INDIVIDUAL ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT101-3-3-IOT

INTERNET OF THINGS: CONCEPTS AND APPLICATION

APD3F2308IT(CE), APD4F2308ME, APD3F2308CD(CYB), APD3F2308IT, APD3F2308IT(BIS), APD4F2308ME, APU3F2308IT(CE), APD3F2308CS(CYB), APD3F2308IT, APU3F2308IT(BIS)

HAND OUT DATE: 15th September 2023

HAND IN DATE: 1st December 2023

NAME	:	**AHMED SALAH AHMED ELHENEIDY**
STUDENT ID	:	**TP 059561**
ASSIGNMENT	:	**SMART PET HOUSE**
LECTURER	:	**KAMALANATHAN**

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION TO THE ARDUINO MEGA AND THE SYSTEM

In the ever-evolving landscape of embedded systems and microcontroller technologies, the Arduino Mega has emerged as a stalwart platform, offering a robust foundation for a myriad of innovative projects. Developed by the Arduino company, the Mega variant stands out among its counterparts, featuring the ATmega2560 microcontroller, which significantly expands the capabilities of the original Arduino architecture.

The Arduino Mega's appeal lies in its versatility, making it a go-to choose for both novice enthusiasts and seasoned professionals. With an impressive array of 54 digital input/output pins and 16 analog inputs, the Mega opens the door to a vast spectrum of project possibilities. Its substantial 256 KB of flash memory ensures ample storage space for complex codebases, enabling developers to tackle ambitious undertakings with confidence. These features, combined with an extended set of communication ports and compatibility with a diverse range of shields, position the Arduino Mega as a versatile and scalable microcontroller for a wide array of applications.

As part of the commitment to providing hands-on and practical learning experiences, APU Corporate Training has envisioned the SMART PET HOUSE project. This undertaking leverages Arduino Mega's capabilities to create an IoT-based system that addresses the evolving needs of pet care and home automation. By integrating a selection of sensors, actuators, and modules, the SMART PET HOUSE exemplifies the adaptability and functionality of the Arduino Mega in crafting intelligent and interconnected systems.

The SMART PET HOUSE project incorporates key components, each serving a specific purpose within the system. The DHT11 Temperature and Humidity Sensor ensures climate monitoring, providing essential data for maintaining a comfortable environment for pets. The 28BYJ-48 Stepper Motor, controlled by the ULN2003 Driver Board, manages window operations, showcasing the Mega's capability for precise motor control. The HC-SR04 Ultrasonic Distance Sensor, paired with the SG90 Micro Servo, automates the opening and closing of the pet house door based on proximity, adding an element of security and convenience. Additionally, the Single Channel 5V Relay Module and Micro Submersible Water Pump, facilitated by the water level sensor, handle the automated pet watering system.

This documentation serves as a comprehensive guide to the SMART PET HOUSE system, offering detailed insights into the hardware setup, code implementation, and the theoretical foundations supporting each module's functionality. By engaging with this project, participants in the APU Corporate Training program will not only gain practical experience in working with the Arduino Mega but will also develop a profound understanding of how this versatile microcontroller can be harnessed to create sophisticated IoT applications. Through the exploration of the SMART PET HOUSE, industrial professionals will be equipped to apply their knowledge in diverse contexts, fostering a deeper appreciation for the Arduino Mega's role in shaping the future of embedded technology and smart living solutions.

2. SYSTEM DESIGN

The SMART PET HOUSE project aims to create an intelligent and automated environment for pet care using Arduino microcontrollers. The system integrates various sensors and actuators to monitor and control the climate, water supply, and access points within a pet house. This document provides a comprehensive overview of the hardware components, software implementation, and operational aspects of the SMART PET HOUSE system.

2.1 System Architecture

Figure 1 below presents the Architecture Diagram of the SMART PET HOUSE project, skillfully illustrated using Fritzing. The diagram captures the intricate interconnection of components, providing a comprehensive overview of the project's circuitry. Through Fritzing's visual representation, the audience gains insight into the spatial arrangement and electrical relationships among Arduino microcontrollers, sensors, actuators, and other supporting modules. The Architecture Diagram serves as a valuable resource for understanding the systemic structure, fostering clarity regarding data flow, signal pathways, and power distribution within the SMART PET HOUSE. It stands as an essential tool for both instructional and reference purposes, offering a visual guide that aids in comprehending the project's hardware architecture and facilitating efficient communication of the design to diverse audiences.

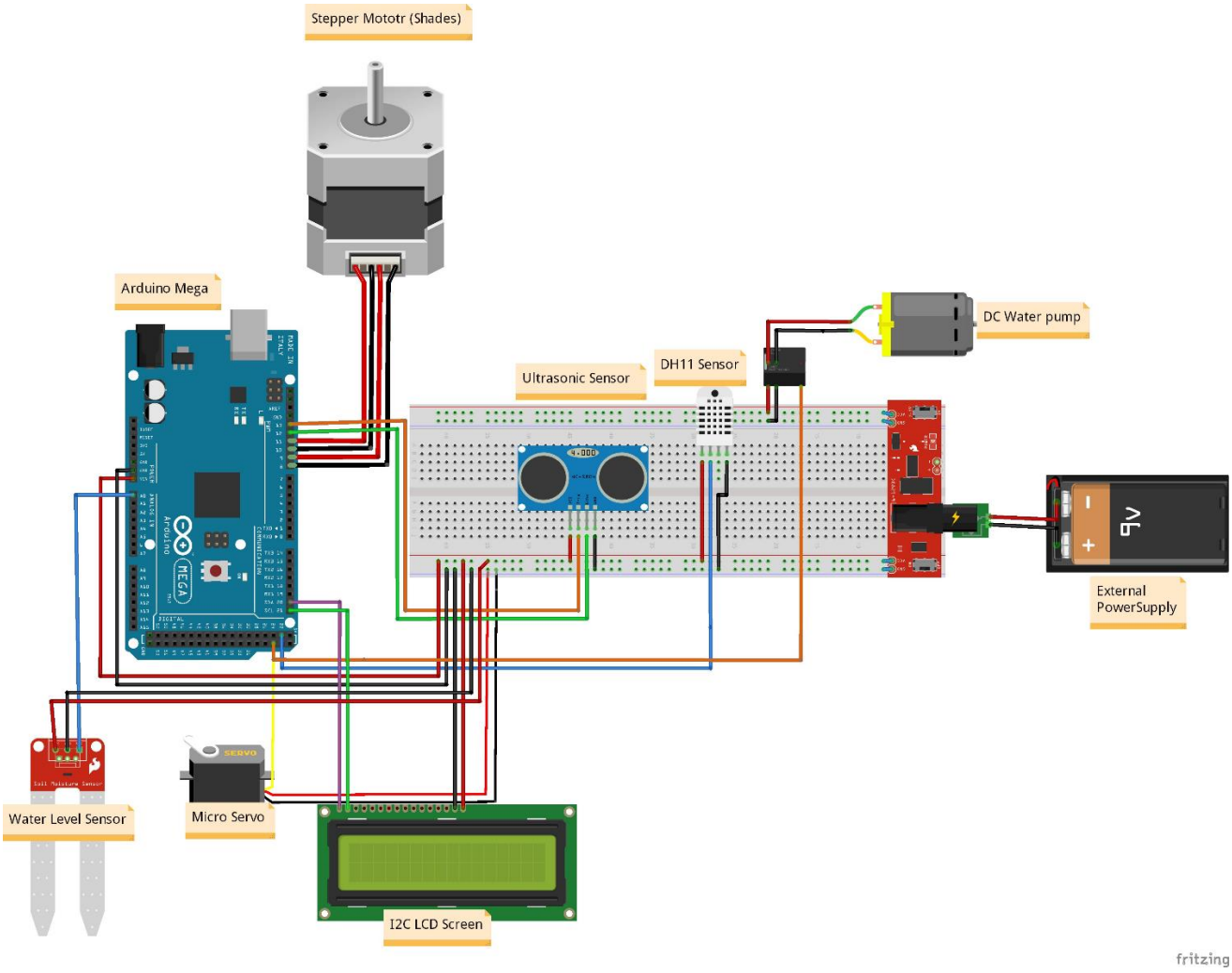

Figure 1 Architecture Diagram

2.1.1 Hardware Components

Table 1 below presents an overview of the hardware components utilized in the system. Each component serves a specific role in the SMART PET HOUSE project, collectively contributing to the functionality and automation of the pet care environment. From the central Arduino Mega microcontroller to the various sensors, actuators, and power supply modules, the table succinctly outlines the key elements that form the foundation of the system's architecture. This comprehensive summary aids in understanding the essential components and their respective functions within the SMART PET HOUSE, illustrating the

integration of technology to enhance the well-being of pets through intelligent monitoring and control.

Table 1 Hardware Components in the SMART PET HOUSE Project

Component	**Description**
Arduino Mega	Central microcontroller for processing and controlling connected devices
Power Supply (Breadboard Power Module + 9V Battery)	External power supply for continuous operation
DHT11 Temperature and Humidity Sensor	Monitors climate conditions within the pet house
2x16 LCD with I2C Module	Displays real-time temperature and humidity data
5V 28BYJ-48 Stepper Motor + ULN2003 Driver Board	Controls window shades for climate control
HC-SR04 Ultrasonic Distance Sensor + SG90 Micro Servo	Automates door opening and closing based on pet presence
Single Channel 5V Relay Module + Micro Submersible Water Pump	Manages pet watering functionality
Jumper Wires and Breadboard	Facilitates the interconnection of components
Circuit Box Enclosure	Provides a secure housing for electronic components

The SMART PET HOUSE project incorporates a diverse array of hardware components to realize its intelligent and automated functionalities. The central processing unit, Arduino Mega, orchestrates the system's operations, supported by a reliable external power supply comprising the Breadboard Power Module and a 9V battery. Climate conditions are monitored by the DHT11 Temperature and Humidity Sensor, with real-time data displayed on the 2x16 LCD with I2C Module. Window shades are regulated by the 5V 28BYJ-48 Stepper Motor and ULN2003 Driver Board for optimal climate control. The door automation feature is enabled by the HC-SR04 Ultrasonic Distance Sensor and SG90 Micro Servo, ensuring secure access for pets. The Single Channel 5V Relay Module and Micro Submersible Water Pump manage the pet watering system, while Jumper Wires and Breadboard facilitate seamless interconnection. Finally, the Circuit Box Enclosure provides a protective housing for these components, ensuring the system's robustness and longevity.

a) Arduino Mega

The Arduino Mega serves as the brain of the SMART PET HOUSE system, overseeing data processing and controlling the entire network of connected devices. With its ample GPIO pins and processing power, the Arduino Mega ensures seamless integration and communication between various sensors and actuators.

b) Power Supply (Breadboard Power Module MD-102 + 9V Battery Snap + Max Energizer Battery 9V)

To guarantee continuous operation, an external power supply system is employed. The Breadboard Power Module, in conjunction with a 9V battery and Max Energizer Battery, ensures a reliable power source for the Arduino Mega and connected components.

c) DHT11 Temperature and Humidity Sensor

The DHT11 sensor plays a crucial role in climate monitoring within the pet house. It captures real-time data on temperature and humidity, providing essential information for pet owners to maintain a comfortable environment for their animals.

d) 2x16 LCD with I2C Module

The LCD with I2C module serves as the user interface, displaying pertinent climate information acquired from the DHT11 sensor. Its 2x16 configuration offers a compact and clear display for easy readability.

e) 5V 28BYJ-48 Stepper Motor + ULN2003 Driver Board

The stepper motor, coupled with the ULN2003 driver board, controls the window shades of the pet house. This functionality allows for automatic adjustment of the shades, contributing to the regulation of internal temperature and light conditions.

f) HC-SR04 Ultrasonic Distance Sensor + SG90 Micro Servo

The ultrasonic sensor detects the presence of a pet in the vicinity. When a pet is detected, the SG90 micro servo controls the automatic opening and closing of the pet house door, ensuring secure access and confinement as needed.

g) Single Channel 5V Relay Module + Micro Submersible Water Pump DC 5V + Water Level Sensor

The relay module, water pump, and water level sensor together form the watering system. The water level sensor monitors the water level in the pet's container, while the relay module controls the submersible water pump to maintain an adequate and continuous water supply.

h) Jumper Wires and Breadboard

Jumper wires and a breadboard facilitate the physical connection between components. The breadboard provides a prototyping platform for assembling the circuit, allowing for flexibility and ease of modification.

i) Circuit Box Enclosure

The circuit box enclosure houses all electronic components, ensuring a neat and secure environment. This enclosure protects the components from external elements, such as dust or moisture, enhancing the overall robustness and longevity of the system.

2.1.2 Software Components

a) Arduino IDE

The Arduino Integrated Development Environment (IDE) serves as the software platform for programming the Arduino Mega. It provides an intuitive interface for code development, uploading, and debugging.

b) Libraries

i. Wire.h

The Wire.h library plays a crucial role in enabling I2C (Inter-Integrated Circuit) communication between the Arduino Mega and the 2x16 LCD with I2C module. I2C communication allows for multiple devices to be connected on the same bus, reducing the number of required pins, and facilitating efficient communication between components. The implementation of Wire.h in the SMART PET HOUSE project enhances the overall system's modularity and connectivity.

ii. LiquidCrystal_I2C.h

LiquidCrystal_I2C.h is a dedicated library for the 2x16 LCD with I2C module. This library simplifies the complex task of interfacing with the LCD, making it more accessible for users with varying levels of expertise. By abstracting the intricacies of I2C communication and LCD control, LiquidCrystal_I2C.h streamlines the display interface, ensuring that real-time climate data is presented in a clear and user-friendly manner.

iii. DHT.h

The DHT.h library is instrumental in interfacing with the DHT11 Temperature and Humidity Sensor. This library abstracts low-level communication with the sensor, allowing users to easily read temperature and humidity data without delving into the intricacies of sensor communication protocols. The inclusion of DHT.h enhances the accuracy and reliability of climate data acquisition in the SMART PET HOUSE system.

iv. Servo.h

Servo.h is an essential library for controlling the SG90 micro servo motor responsible for automating the pet house door. This library provides a high-level interface for users to

precisely control the servo motor's movement and position. With Servo.h, the system ensures that the door operates smoothly and accurately based on the readings from the ultrasonic distance sensor, contributing to a seamless and responsive door automation mechanism.

c) Arduino Code

The heart of the SMART PET HOUSE system lies in the custom Arduino code. This code orchestrates the interactions between sensors and actuators, defining the logic for climate monitoring, window shade control, door automation, and the watering system. The code operates in a loop, ensuring continuous monitoring and control of the pet house environment. Refer to section 4 for more details.

2.1.3 System Functionality

a) Climate Monitoring

The DHT11 sensor, integrated into the SMART PET HOUSE, plays a pivotal role in continuously monitoring the temperature and humidity levels within the pet house environment. This sensor provides real-time data, enabling pet owners or caretakers to stay informed about the prevailing climate conditions. The LCD, equipped with the I2C module, serves as the user interface, displaying temperature and humidity values, ensuring that users can easily assess and respond to any fluctuations.

b) Window Shade Control

The climate control mechanism of the SMART PET HOUSE employs the 28BYJ-48 stepper motor in tandem with the ULN2003 driver board to manipulate the window shades. This automated system dynamically adjusts the shades based on environmental conditions, effectively regulating the amount of natural light and airflow entering the pet house. The intelligent adjustment of window shades contributes to maintaining an optimal and comfortable climate for the pet while minimizing energy consumption.

c) Door Automation

The SMART PET HOUSE prioritizes the safety and security of the pet by incorporating the HC-SR04 ultrasonic sensor for door automation. This sensor detects the

presence of the pet within its vicinity. The SG90 micro servo, acting as the door controller, responds to the ultrasonic sensor's input by automatically opening or closing the door. This seamless automation not only ensures convenient access for the pet but also safeguards against potential external threats, creating a secure and pet-friendly environment.

d) Watering System

Addressing the essential need for hydration, the SMART PET HOUSE integrates a sophisticated watering system. The water level sensor continuously monitors the water level in the pet's water container, providing valuable insights into the remaining water supply. The submersible water pump, under the control of the relay module, guarantees a consistent and reliable water supply to the pet. This automated watering system alleviates concerns about water scarcity, contributing to the overall well-being of the pet.

2.1.4 System Operation

The SMART PET HOUSE functions seamlessly in a loop, executing a series of tasks to ensure the efficient operation of its components. The key operational tasks include:

a) Water Level Control

The system monitors the water level in the pet's water container at regular intervals. If the water level falls below a predefined threshold, the relay module activates the submersible water pump, replenishing the water supply and ensuring the pet's hydration needs are met.

b) Ultrasonic Sensor

The HC-SR04 ultrasonic sensor operates continuously to detect the presence of the pet. Depending on the sensor's input, the SG90 micro servo controls the door, automatically opening or closing to allow or restrict the pet's access.

c) Temperature and Humidity Monitoring

At predefined intervals, the DHT11 sensor collects temperature and humidity data from the pet house environment. This information is then displayed on the LCD, providing users with real-time insights into the prevailing climate conditions.

d) Stepper Motor Control

The 28BYJ-48 stepper motor, responsible for regulating the window shades, operates based on a predefined algorithm. The system adjusts the shades to optimize temperature and lighting conditions within the pet house, contributing to the pet's comfort.

3. PROTOTYPE SCREENSHOT

Figure 2 below provides a detailed glimpse into the front view of the SMART PET HOUSE prototype. The LCD interface takes center stage, showcasing real-time information such as temperature and humidity readings for effective environmental monitoring. Additionally, the shades motor, a pivotal component for window control, is prominently featured. The clear visualization in this front view offers insight into the user interface and the mechanisms responsible for regulating the internal environment of the pet house.

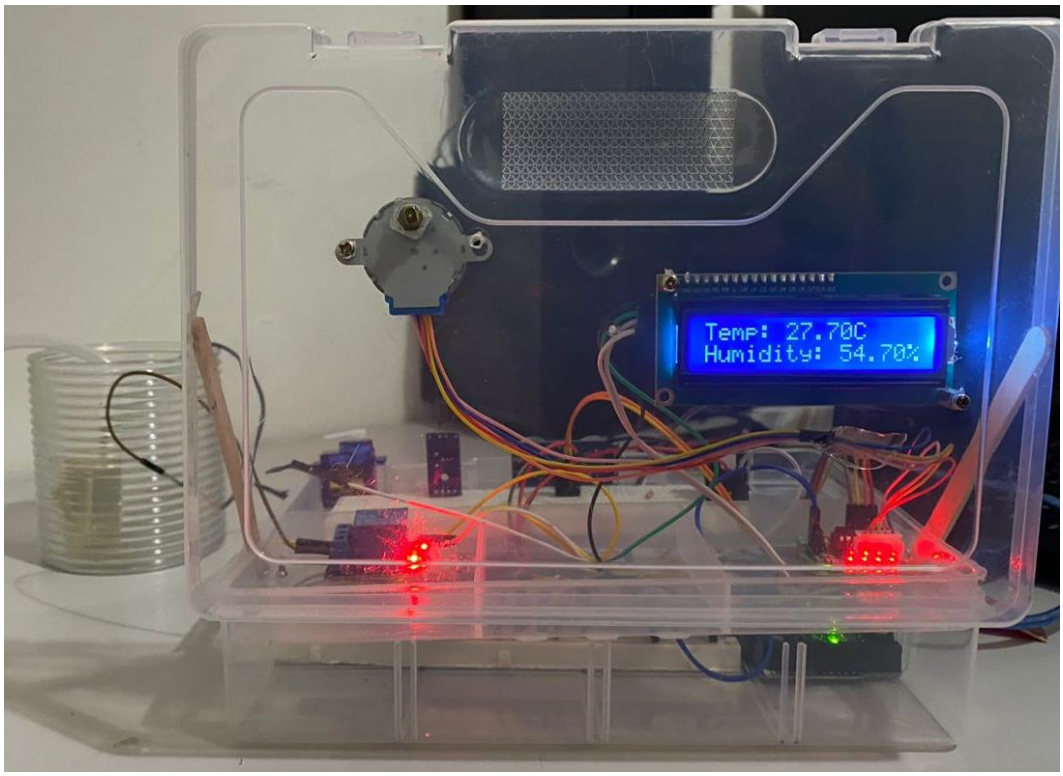

Figure 2 Prototype Front View

Figure 3 illustrated below displays, the top view of the prototype unfolds a comprehensive overview of all integrated components. Notably, the motor driver, relay, water pump, and water level sensor are distinctly visible. This view provides a holistic perspective on the arrangement and interaction of these crucial elements. The inclusion of the shades motor driver, the relay for water control, and the water pump with its associated level sensor emphasizes the multi-faceted functionality embedded in the SMART PET HOUSE, illustrating its capability to address diverse aspects of pet care.

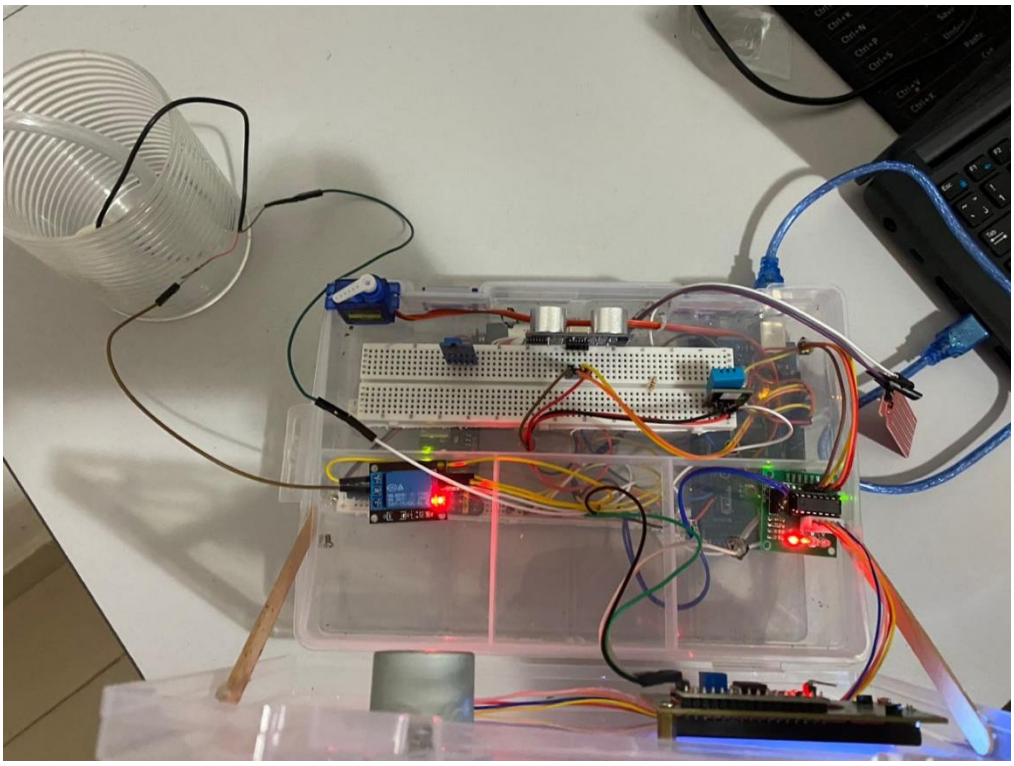

Figure 3 Prototype Top View

The back view, as depicted below in Figure 4, unveils essential components crucial for the project's functionality. Here, the ultrasonic sensor and micro servo for the door mechanism are evident. The inclusion of sensors adds a layer of sophistication, enabling the SMART PET HOUSE to respond to external stimuli intelligently. This view not only highlights the technical aspects of the ultrasonic sensor and micro servo but also underscores their strategic placement within the overall design. The door mechanism, when coupled with these sensors, enhances the system's responsiveness and adaptability.

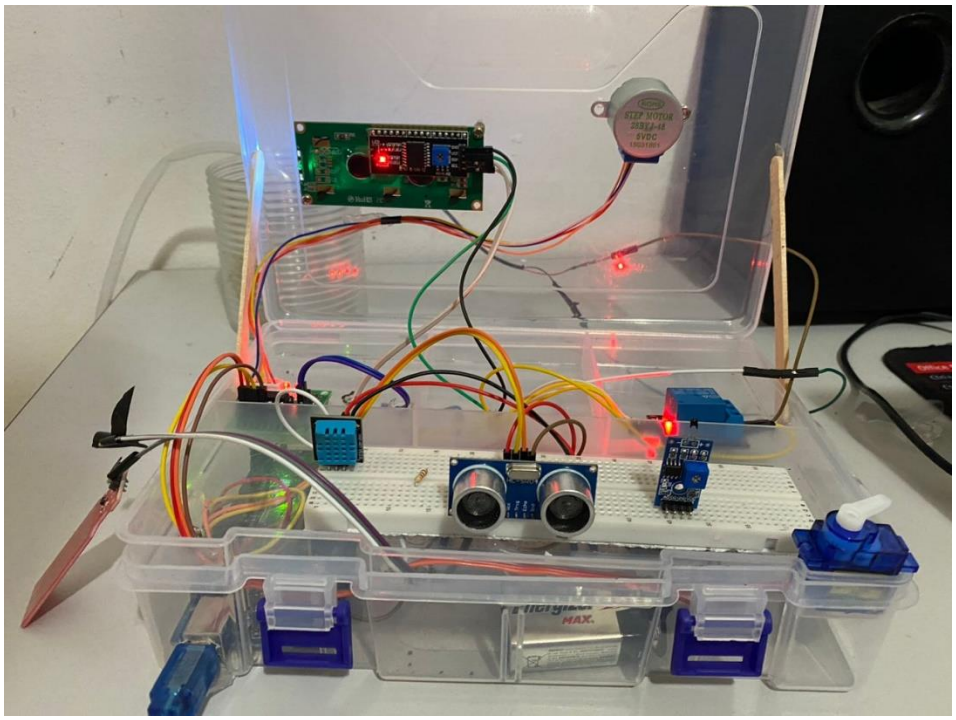

Figure 4 Prototype Back View

4. SOURCE CODE

4.1 Code Segments And Explanation

The present code segment is an Arduino sketch intended for facilitating the implementation of a SMART PET HOUSE project. The code is organized in a modular manner, leveraging several libraries to address specific functionalities crucial to the project.

Figure 5 below illustrates The included libraries are as follows: Wire.h for I2C communication, LiquidCrystal_I2C.h for controlling I2C-based LCD displays, DHT.h for interfacing with DHT temperature and humidity sensors, and Servo.h for managing servo motors.

Subsequently, in lines 6 and 7 two constants are defined for the DHT sensor configuration. The DHTPIN constant designates the specific digital pin (pin 22) to which the DHT sensor is connected, while DHTTYPE signifies the type of DHT sensor in use, denoted as DHT11.

Further, in lines 9 to 12 the code designates specific digital pins for the stepper motor. The pins are identified as IN1, IN2, IN3, and IN4, serving as inputs for the stepper motor driver (ULN2003). These pins are instrumental in dictating the direction and steps of the stepper motor, thereby facilitating the control of window shades within the SMART PET HOUSE project.

sketch_dec9a.ino

```
1   #include <Wire.h>
2   #include <LiquidCrystal_I2C.h>
3   #include <DHT.h>
4   #include <Servo.h>
5   // Define pin for DHT sensor
6   #define DHTPIN 22
7   #define DHTTYPE DHT11
8   // Define pins for stepper motor
9   #define IN1  8
10  #define IN2  9
11  #define IN3  10
12  #define IN4  11
```

Figure 5 Code Segment Part 1

Figure 6 below illustrates part 2 of the source code introduces variables and pin assignments to orchestrate the functionality of the SMART PET HOUSE project, employing an Arduino microcontroller. Key elements include the control of a stepper motor, water level sensor and relay, RGB LED with a sound sensor, an ultrasonic sensor coupled with a servo motor, and the initialization of a DHT sensor and an LCD display.

The **Steps** variable is utilized to keep track of the stepper motor's step count, while the **Direction** boolean variable denotes the direction of rotation, with **true** representing forward and **false** indicating reverse. **last_time**, **currentMillis**, and **time** are variables employed for temporal tracking, contributing to the precise control of the stepper motor. **steps_left** denotes the remaining steps for the stepper motor, and **lastStepperMove** tracks the last time the stepper motor was engaged.

Pin assignments are introduced for the water level sensor (**waterLevelPin**), relay (**relayPin**), sound sensor (**soundSensorPin**), RGB LED components (**redPin**, **greenPin**, **bluePin**), ultrasonic sensor (**trigPin**, **echoPin**), and the servo motor (**servoPin**). A **Servo** object (**myServo**) is instantiated for the servo motor.

Furthermore, instances of the DHT sensor (**dht**) and the LiquidCrystal_I2C display (**lcd**) are initialized, where **DHTPIN** and **DHTTYPE** constants are employed to specify the pin and type of the DHT sensor.

To manage the update intervals for various sensor readings and actuator control, several **unsigned long** variables, such as **lastWaterLevelCheck**, **lastUltrasonicCheck**, **lastTemperatureHumidityCheck**, and **lastDirectionChangeTime**, are introduced. These variables serve to optimize resource utilization and maintain a systematic approach to the periodic execution of sensor readings and actuation commands.

sketch_dec9a.ino ●

```
13  // Stepper motor variables
14  int Steps = 0;
15  boolean Direction = true;
16  unsigned long last_time;
17  unsigned long currentMillis;
18  int steps_left = 4095;
19  long time;
20  // Define pins for water level sensor and relay
21  const int waterLevelPin = A0;
22  const int relayPin = 23;
23  // Define pins for ultrasonic sensor and servo motor
24  const int trigPin = 13;
25  const int echoPin = 12;
26  const int servoPin = 28;
27  Servo myServo;
28  // Initialize DHT sensor and LCD
29  DHT dht(DHTPIN, DHTTYPE);
30  LiquidCrystal_I2C lcd(0x27, 16, 2);
31  // Variables to keep track of last update times
32  unsigned long lastWaterLevelCheck = 0;
33  unsigned long lastUltrasonicCheck = 0;
34  unsigned long lastTemperatureHumidityCheck = 0;
35  unsigned long lastStepperMove = 0;
36  unsigned long lastDirectionChangeTime = 0;
37
```

Figure 6 Code Segment Part 2

Figure 7 below presents source code part 3 delineates the setup and main loop functions of the Arduino sketch for the SMART PET HOUSE project. This part of the code is designed to initialize pins, sensors, actuators, and external devices, as well as to orchestrate the primary functionalities within a continuous loop

The **setup()** function serves as the initialization routine, executed once at the commencement of the program. Serial communication is initiated at a baud rate of 115200 for potential debugging and monitoring purposes. Subsequently, pin modes are configured for the stepper motor (IN1 to IN4), water level sensor, relay, RGB LED components (redPin, greenPin, bluePin), ultrasonic sensor (trigPin, echoPin), and the servo motor. The LCD display is initialized and its backlight activated. Finally, the DHT sensor is initialized using the **dht.begin()** method.

The **loop()** function encapsulates the iterative execution of the primary operational logic. Different functions are invoked within the loop to manage distinct tasks, fostering modular code organization, and enhancing code readability. The functions include **handleWaterLevelControl()** for monitoring and controlling water levels, **handleUltrasonicSensor()** for ultrasonic distance measurements, **handleTemperatureHumidityMonitoring()** for obtaining climate data from the DHT sensor, and **handleStepperMotorControl()** for governing the stepper motor's actions. This modular structure facilitates the addition of supplementary tasks as necessitated by the project's evolving requirements.

The **loop()** function perpetuates its execution, ensuring the continual surveillance and regulation of pertinent parameters for the SMART PET HOUSE project. It provides a comprehensive and adaptable framework for integrating diverse sensors and actuators, demonstrating an efficient and maintainable approach to IoT-based pet management system development.

sketch_dec9a.ino

```
// Setup function
void setup() {
  Serial.begin(115200);
  // Setup pins for stepper motor
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  // Setup pins for water level sensor and relay
  pinMode(waterLevelPin, INPUT);
  pinMode(relayPin, OUTPUT);
  // Setup pins for RGB LED and sound sensor
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
  // Setup pins for ultrasonic sensor and servo motor
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  myServo.attach(servoPin);
  // Initialize LCD
  lcd.init();
  lcd.backlight();
  // Initialize DHT sensor
  dht.begin();
}

// Main loop function
void loop() {
  // Call different functions to handle various tasks
  handleWaterLevelControl();
  handleUltrasonicSensor();
  handleTemperatureHumidityMonitoring();
  handleStepperMotorControl();
  // Add other tasks as needed...
}
```

Figure 7 Code Segment Part 3

Figure 8 below displays code segment part 4 defines a function named **handleWaterLevelControl()**, responsible for monitoring and controlling the water level in the SMART PET HOUSE project.

The **handleWaterLevelControl()** function is designed to manage the water level within the SMART PET HOUSE system. It operates on a periodic basis, checking the water level at intervals of 2 seconds to ensure timely and responsive control. This interval is controlled by the comparison between the current time, obtained through **millis()**, and the previously recorded time of the water level check.

Upon initiating the water level assessment, the function employs the **analogRead()** method to retrieve analog data from the water level sensor connected to the specified pin (**waterLevelPin**). Subsequently, the obtained analog value is mapped using the **map()** function to a range indicative of the water level. This mapping allows for the conversion of the raw sensor reading to a more comprehensible and manageable scale, ranging from 1 to 10.

Following the mapping procedure, the function orchestrates the control of a relay (connected to **relayPin**) based on the derived water level. If the mapped water level falls below a threshold of 5, the relay is activated (**HIGH**), signifying a need to pump water into the designated area. The function then outputs a corresponding message to the serial monitor, conveying the state of the relay and the current water level. Conversely, if the mapped water level surpasses a threshold of 10, indicating an adequate water level, the relay is deactivated (**LOW**), and a corresponding message is printed to the serial monitor.

This function embodies a pivotal aspect of the SMART PET HOUSE project, ensuring the maintenance of optimal water levels for pet-related activities. It demonstrates a structured and responsive approach to water level control, contributing to the overall functionality and efficiency of the IoT-based pet management system.

sketch_dec9a.ino

```
81  void handleWaterLevelControl() {
82    unsigned long currentMillis = millis();
83
84    // Check water level every 2 seconds
85    if (currentMillis - lastWaterLevelCheck >= 2000) {
86      lastWaterLevelCheck = currentMillis;
87
88      // Read water level sensor and map the value
89      int waterLevel = analogRead(waterLevelPin);
90      int mappedLevel = map(waterLevel, 200, 430, 1, 10);
91
92      // Control relay based on water level
93      if (mappedLevel < 5) {
94        digitalWrite(relayPin, HIGH);
95        Serial.println("Relay ON - Water level: " + String(mappedLevel));
96      } else if (mappedLevel >= 10) {
97        digitalWrite(relayPin, LOW);
98        Serial.println("Relay OFF - Water level: " + String(mappedLevel));
99      }
100   }
101 }
```

Figure 8 Code Segment Part 4

Figure 9 below illustrates The code segment part 5 defines a function named **handleUltrasonicSensor()**, responsible for managing the ultrasonic sensor within the SMART PET HOUSE project.

The **handleUltrasonicSensor()** function is designed to oversee the functionality of the ultrasonic sensor within the SMART PET HOUSE system. Operating at a high frequency, the function performs distance measurements at regular intervals, with a check occurring every 100 milliseconds. This periodicity is regulated through the comparison between the current time, obtained via **millis()**, and the timestamp of the last ultrasonic sensor check.

Upon execution, the function initiates the distance measurement process using the ultrasonic sensor. This involves triggering the sensor by briefly setting the trigger pin (**trigPin**) to a high state for 10 microseconds. The subsequent pulse duration is then captured using **pulseIn()** on the echo pin (**echoPin**). The measured duration is converted into distance, employing the speed of sound in air (0.034 cm/μs) and halving the result to account for the round-trip travel of the ultrasonic signal.

Based on the obtained distance, the function proceeds to control the position of a servo motor (**myServo**). If the calculated distance is less than or equal to 10 units, indicative of an object or obstruction in close proximity, the servo motor is positioned at 90 degrees. Conversely, if the distance exceeds 10 units, the servo motor is set to 0 degrees. This servo motor control mechanism is implemented to automate the movement of a component within the SMART PET HOUSE, possibly serving a role in door or barrier management based on the proximity of objects.

sketch_dec9a.ino

```
102
103  // Function to handle ultrasonic sensor
104  void handleUltrasonicSensor() {
105    unsigned long currentMillis = millis();
106
107    // Read ultrasonic sensor every 100 milliseconds
108    if (currentMillis - lastUltrasonicCheck >= 100) {
109      lastUltrasonicCheck = currentMillis;
110
111      // Measure distance using ultrasonic sensor
112      long duration, distance;
113      digitalWrite(trigPin, LOW);
114      delayMicroseconds(2);
115      digitalWrite(trigPin, HIGH);
116      delayMicroseconds(10);
117      digitalWrite(trigPin, LOW);
118      duration = pulseIn(echoPin, HIGH);
119      distance = duration * 0.034 / 2;
120
121      // Control servo motor based on distance
122      if (distance <= 10) {
123        myServo.write(90);
124      } else {
125        myServo.write(0);
126      }
127    }
128  }
```

Figure 9 Code Segment Part 5

Figure 10 below showcases The code snipped part 6 defines a function named **handleTemperatureHumidityMonitoring()**, responsible for overseeing the monitoring of temperature and humidity within the SMART PET HOUSE project.

The **handleTemperatureHumidityMonitoring()** function is designed to manage the temperature and humidity monitoring aspects within the SMART PET HOUSE system. Executing at regular intervals, the function reads temperature and humidity data from a DHT sensor and updates the information on an attached LCD display. The temporal control of this function is facilitated by the comparison between the current time, obtained through **millis()**, and the timestamp of the last temperature and humidity check.

The function follows a periodicity of 2 seconds for readings, ensuring a balance between data currency and resource efficiency. Upon initiation, the function employs the DHT sensor's **readHumidity()** and **readTemperature()** methods to obtain real-time humidity and temperature values. Subsequently, these values are presented on a 2x16 LCD display through dedicated cursor manipulations.

The LCD display is cleared, and the temperature and humidity values are sequentially printed on the two rows of the display. The first row exhibits the temperature in degrees Celsius, while the second row displays the humidity percentage. This real-time display provides valuable insights into the environmental conditions within the SMART PET HOUSE, facilitating comprehensive monitoring and management.

sketch_dec9a.ino

```
130  // Function to handle temperature and humidity monitoring
131  void handleTemperatureHumidityMonitoring() {
132    unsigned long currentMillis = millis();
133
134    // Read temperature and humidity every 2 seconds
135    if (currentMillis - lastTemperatureHumidityCheck >= 2000) {
136      lastTemperatureHumidityCheck = currentMillis;
137
138      // Read temperature and humidity from DHT sensor
139      float humidity = dht.readHumidity();
140      float temperature = dht.readTemperature();
141
142      // Display temperature and humidity on LCD
143      lcd.clear();
144      lcd.setCursor(0, 0);
145      lcd.print("Temp: ");
146      lcd.print(temperature);
147      lcd.print("C");
148
149      lcd.setCursor(0, 1);
150      lcd.print("Humidity: ");
151      lcd.print(humidity);
152      lcd.print("%");
153    }
154  }
```

Figure 10 Code Segment Part 6

Figure 11 below illustrates The source code segment part 7 defines a function named **handleStepperMotorControl()**, responsible for orchestrating the control of a stepper motor within the SMART PET HOUSE project.

The **handleStepperMotorControl()** function is designed to manage the control and movement of a stepper motor within the SMART PET HOUSE system. The primary goal is to move the stepper motor in a specified direction, controlling the number of steps taken. The function operates within a loop, continually checking the number of remaining steps to execute.

The function utilizes the **micros()** function to obtain the current time in microseconds. It checks whether there are remaining steps (**steps_left > 0**) to be executed. If so, it evaluates whether a certain time interval (1000 microseconds) has elapsed since the last

step. If the condition is met, the **stepper()** function is called with a parameter of 1, indicating a step in the current direction. The execution time and step count are updated accordingly.

Upon completing the specified number of steps (**steps_left** reaching 0), the function transitions to an alternate branch. Here, the code monitors the duration since the last change in direction (**millis() - lastDirectionChangeTime**). If 10 seconds have elapsed, the direction of the stepper motor is inverted (**Direction = !Direction**), and the step count is reset to its initial value (**4095**). This mechanism ensures a periodic change in the direction of stepper motor movement, contributing to the overall dynamics of the SMART PET HOUSE system.

The function incorporates serial communication commands (**Serial.println()**) for informative debugging purposes, allowing the monitoring and logging of relevant details during the stepper motor control process. This ensures transparency in system behavior and aids in troubleshooting or debugging efforts.

sketch_dec9a.ino

```
156  // Function to handle stepper motor control
157  void handleStepperMotorControl() {
158    unsigned long currentMillis = micros();
159
160    // Move the stepper motor if steps are remaining
161    if (steps_left > 0) {
162      if (currentMillis - last_time >= 1000) {
163        stepper(1);
164        time = time + micros() - last_time;
165        last_time = micros();
166        steps_left--;
167      }
168    } else {
169      // Change direction after waiting for 10 seconds
170      Serial.println(time);
171      if (millis() - lastDirectionChangeTime >= 10000) {
172        Serial.println("Waited 10 seconds, changing direction...");
173        Direction = !Direction;
174        steps_left = 4095;
175        lastDirectionChangeTime = millis();
176      } else {
177        Serial.println("Waiting 10 seconds before changing direction...");
178      }
179    }
180  }
```

Figure 11 Code Segment Part 7

Figure 12 below presents code segment part 8 defines a function named **stepper(int xw)**, responsible for controlling the movement and direction of a stepper motor within the SMART PET HOUSE project.

The **stepper(int xw)** function serves as the control mechanism for the stepper motor, directing its movement in accordance with the specified number of steps (**xw**). The function operates within a **for** loop, iterating for the designated number of steps.

Within the loop, a **switch** statement governs the activation of specific coils in the stepper motor. The **Steps** variable, representing the current step in the sequence, dictates the corresponding pattern of coil activations. Each case in the switch statement corresponds to a specific step in the stepper motor sequence, determining which coils are set to a high (**HIGH**) or low (**LOW**) state. This sequence is pivotal in achieving controlled and precise stepper motor movement.

The coil activation patterns for each case are structured in a manner that aligns with the standard sequence for driving a stepper motor. This sequence ensures a unidirectional stepwise motion, crucial for applications like window shade control or other controlled movements within the SMART PET HOUSE.

Following the coil activation, the **SetDirection()** function is invoked. Although the specific implementation of the **SetDirection()** function is not provided in the code snippet, it can be inferred that it plays a role in managing the direction of the stepper motor's rotation. This interaction with the direction component enhances the versatility of the stepper motor control, allowing for both forward and reverse movement.

The **stepper()** function demonstrates an organized and modular approach to stepper motor control, encapsulating the intricate details of coil activations and step sequences within a separate function. This design facilitates code readability, maintenance, and adaptability to different applications within the SMART PET HOUSE system.

sketch_dec9a.ino

```
182  // Function to control the stepper motor
183  void stepper(int xw) {
184    for (int x = 0; x < xw; x++) {
185      switch (Steps) {
186        case 0:
187          digitalWrite(IN1, LOW);
188          digitalWrite(IN2, LOW);
189          digitalWrite(IN3, LOW);
190          digitalWrite(IN4, HIGH);
191          break;
192        case 1:
193          digitalWrite(IN1, LOW);
194          digitalWrite(IN2, LOW);
195          digitalWrite(IN3, HIGH);
196          digitalWrite(IN4, HIGH);
197          break;
198        case 2:
199          digitalWrite(IN1, LOW);
200          digitalWrite(IN2, LOW);
201          digitalWrite(IN3, HIGH);
202          digitalWrite(IN4, LOW);
203          break;
204        case 3:
205          digitalWrite(IN1, LOW);
206          digitalWrite(IN2, HIGH);
207          digitalWrite(IN3, HIGH);
208          digitalWrite(IN4, LOW);
209          break;
210        case 4:
211          digitalWrite(IN1, LOW);
212          digitalWrite(IN2, HIGH);
213          digitalWrite(IN3, LOW);
214          digitalWrite(IN4, LOW);
215          break;
216        case 5:
217          digitalWrite(IN1, HIGH);
218          digitalWrite(IN2, HIGH);
219          digitalWrite(IN3, LOW);
220          digitalWrite(IN4, LOW);
221          break;
```

Figure 12 Code Segment Part 8

Figure 13 below displays code segment part 9 defines a function named **SetDirection()**, responsible for managing the direction of the stepper motor in the SMART PET HOUSE project.

The **SetDirection()** function is designed to control the direction of the stepper motor's rotation within the SMART PET HOUSE system. It operates based on the value of the **Direction** variable, which serves as a boolean flag (1 for true, 0 for false) indicating the desired direction of the stepper motor.

The function employs a series of conditional statements to determine the appropriate adjustment to the **Steps** variable, which represents the current step in the stepper motor sequence. If the **Direction** is set to 1 (indicating forward direction), the **Steps** variable is incremented. Conversely, if the **Direction** is set to 0 (indicating reverse direction), the **Steps** variable is decremented.

Additionally, the function includes boundary checks to ensure that the **Steps** variable remains within the valid range of 0 to 7, inclusive. If **Steps** exceeds 7, it is reset to 0, and if it falls below 0, it is set to 7. These checks ensure that the stepper motor sequence remains within a predefined range, preventing overflow or underflow issues.

sketch_dec9a.ino

```
222         case 6:
223           digitalWrite(IN1, HIGH);
224           digitalWrite(IN2, LOW);
225           digitalWrite(IN3, LOW);
226           digitalWrite(IN4, LOW);
227           break;
228         case 7:
229           digitalWrite(IN1, HIGH);
230           digitalWrite(IN2, LOW);
231           digitalWrite(IN3, LOW);
232           digitalWrite(IN4, HIGH);
233           break;
234         default:
235           digitalWrite(IN1, LOW);
236           digitalWrite(IN2, LOW);
237           digitalWrite(IN3, LOW);
238           digitalWrite(IN4, LOW);
239           break;
240       }
241       SetDirection();
242     }
243   }
244
245   // Function to set the direction of the stepper motor
246   void SetDirection() {
247     if (Direction == 1) {
248       Steps++;
249     }
250     if (Direction == 0) {
251       Steps--;
252     }
253     if (Steps > 7) {
254       Steps = 0;
255     }
256     if (Steps < 0) {
257       Steps = 7;
258     }
259   }
```

Figure 13 Code Segment Part 9

4.2 Summary

The above code constitutes an Arduino sketch for the SMART PET HOUSE project, an IoT-based system designed to monitor and manage various environmental aspects for pets. The code leverages multiple sensors and actuators, including a DHT temperature and humidity sensor, an ultrasonic distance sensor, a stepper motor for window shade control, a water level sensor for automatic pet watering, and an RGB LED with a sound sensor for additional features. Additionally, an LCD display is employed to provide real-time feedback on temperature, humidity, and other relevant data.

The entire code operates within the main loop, invoking these functions at specified intervals to achieve a well-coordinated and responsive system. The use of **millis()** and **micros()** functions for timing allows the system to function smoothly without interruptions. This non-blocking approach ensures that different tasks are executed at the appropriate intervals without impeding the overall performance of the SMART PET HOUSE system. The integration of modular functions and efficient timing mechanisms contributes to the system's ability to work in real-time, providing a seamless and dynamic environment for pet care.

The main components of the code are organized into distinct functions:

1. **handleWaterLevelControl():**
    - Monitors water levels using an analog sensor.
    - Maps sensor values to a specific range for effective control.
    - Activates a relay to control water flow based on mapped values.
2. **handleUltrasonicSensor():**
    - Utilizes an ultrasonic sensor to measure distances periodically.
    - Adjusts the position of a servo motor based on measured distances.
    - This function facilitates automated responses to nearby objects.
3. **handleTemperatureHumidityMonitoring():**

- Reads temperature and humidity data from a DHT sensor.
- Updates the information on an LCD display every 2 seconds.
- Provides real-time feedback on environmental conditions.

4. **handleStepperMotorControl():**
    - Manages the movement of a stepper motor in a controlled manner.
    - Incorporates dynamic direction changes after a specified time interval.
    - Ensures the precise and automated control of the stepper motor.
5. **stepper(int xw):**
    - Controls the stepper motor's step sequence based on the specified number of steps.
    - Utilizes a switch-case structure to activate specific coils in a defined sequence.
    - Works in conjunction with the SetDirection() function.
6. **SetDirection():**
    - Determines the direction of the stepper motor's rotation.
    - Adjusts the step sequence based on the Direction variable.

5. LIMITATION

The SMART PET HOUSE project, while showcasing innovative features for pet care and environmental monitoring, is not without its limitations. One notable constraint pertains to the accuracy of sensors integrated into the system, such as the DHT temperature and humidity sensor and the ultrasonic distance sensor. The potential for minor discrepancies in sensor readings may impact the precision of environmental monitoring, leading to variations in the reported data.

Moreover, the reliability of hardware components, including the stepper motor, servo motor, and other electronic parts, can influence the overall dependability of the system. Mechanical wear and tear or electrical issues over time may necessitate periodic maintenance and replacement of components, adding to the project's long-term considerations.

Power consumption emerges as another limitation, especially concerning continuous sensor readings, motor control, and display updates. The impact on energy efficiency becomes significant, particularly when utilizing battery-powered solutions. Balancing functionality with power efficiency becomes crucial to address these concerns and extend the system's operational lifespan.

Sensor calibration poses challenges as well, requiring meticulous adjustment to ensure accurate readings. Calibration complexities may arise due to variations in environmental conditions, demanding user intervention and technical expertise for optimal sensor performance.

Real-time data transmission, if a requisite aspect of the project, may encounter challenges related to network connectivity, latency, and potential interruptions. This can affect the reliability of remote monitoring and control functionalities, warranting careful consideration of the project's communication infrastructure.

The incorporation of a user interface, such as a mobile app or web dashboard, introduces complexity. Ensuring a user-friendly design becomes essential, particularly for individuals less familiar with technology. Clear instructions and an intuitive interface are critical for successful user adoption and interaction.

The project's applicability to diverse pets might be limited, as certain specific needs of different animals may not be fully addressed. Customization to accommodate individual pet requirements may be necessary, acknowledging that a one-size-fits-all approach might not be suitable for all pet owners.

Regular maintenance is essential to address wear and tear of mechanical components, ensure proper sensor functioning, and update software as needed. Users should be aware of and willing to perform routine maintenance tasks to sustain the system's optimal performance.

Cost considerations may impact the project's accessibility, particularly if high-quality sensors or motors are employed. Evaluating the costs against the desired features and functionalities becomes essential to strike a balance between affordability and performance.

Lastly, the current design may present challenges in terms of scalability for larger pet environments or the integration of additional features. Scaling up the project might necessitate a redesign to accommodate the increased complexity and scope of the system.

6. FUTURE ENHANCEMENT

As the SMART PET HOUSE project lays a foundation for intelligent pet care, several potential avenues for future enhancements can further elevate its capabilities and user experience.

- **Integration of AI and Machine Learning:**

Future iterations could leverage artificial intelligence (AI) and machine learning algorithms to analyze historical data patterns. This could enable the system to learn and adapt to the specific needs and behaviors of individual pets, offering personalized care and more efficient environmental control.

- **Expandable Modular System:**

Designing the SMART PET HOUSE as an expandable and modular system would allow users to seamlessly integrate additional sensors or actuators as needed. This flexibility ensures that the system can adapt to evolving requirements and accommodate a broader range of pet care scenarios.

- **Advanced Environmental Monitoring:**

Enhancements in sensor technology could enable the integration of more advanced environmental monitoring capabilities. For instance, gas sensors could detect air quality, ensuring a healthier living environment for pets. This expansion of monitoring parameters contributes to a comprehensive understanding of the pet's surroundings.

- **Mobile Application with Remote Control:**

Developing a dedicated mobile application for remote monitoring and control would empower users to manage the SMART PET HOUSE from anywhere. Real-time alerts, historical data visualization, and the ability to adjust settings remotely enhance convenience and accessibility for pet owners.

- **Energy-Efficient Solutions:**

Implementing energy-efficient solutions, such as low-power sensors and optimization algorithms, could extend the system's battery life or reduce overall power

consumption. This would be particularly beneficial for users relying on battery-operated configurations, ensuring prolonged system uptime.

- **Enhanced User Interface and Interactivity:**

Improvements to the user interface, including the addition of touchscreens or voice commands, can enhance the overall user experience. Introducing interactive features that allow pets to interact with the system, such as automated treat dispensers or interactive toys, adds an extra layer of engagement.

By pursuing these future enhancements, the SMART PET HOUSE project can evolve into a more sophisticated and adaptive pet care solution, aligning with technological advancements and the evolving needs of pet owners. Continuous innovation ensures that the system remains at the forefront of intelligent pet care, delivering enhanced functionality and user satisfaction.

CONCLUSION

In the realm of IoT-based projects, the development and demonstration of the SMART PET HOUSE stand as a testament to the capabilities of Arduino microcontrollers. As an instructor at APU Corporate Training, tasked with providing instruction to industrial professionals, the objective was to create an engaging and instructive project centered around Arduino microcontrollers. The project aimed to showcase the integration of various sensors and actuators, offering a practical demonstration of IoT principles while addressing real-world applications.

One of the significant achievements of the SMART PET HOUSE project lies in its comprehensive approach to pet care and environmental monitoring. By leveraging the versatility of Arduino microcontrollers, the project successfully integrated a range of sensors, including the DHT temperature and humidity sensor, ultrasonic distance sensor, and water level sensor. These sensors, in conjunction with actuators like the stepper motor and servo motor, demonstrated a holistic solution for monitoring and enhancing the living environment for pets.

The modular design of the project, with distinct functions handling specific tasks, contributes to its educational value. This modular approach allows industrial professionals to dissect and understand individual components, fostering a deeper comprehension of Arduino programming and IoT concepts. The project documentation provides a detailed guide, enhancing the learning experience for professionals who seek to delve into the intricacies of the Arduino ecosystem.

The incorporation of real-time monitoring and control mechanisms further underscores the project's practicality. The SMART PET HOUSE not only showcases the capabilities of Arduino microcontrollers but also emphasizes their applicability in real-world scenarios. The seamless integration of sensors and actuators, coupled with efficient timing mechanisms, ensures that the project operates smoothly without interruptions, aligning with the requirements of industrial professionals seeking reliable and robust solutions.

Furthermore, the SMART PET HOUSE project lays a foundation for future enhancements and iterations. The inclusion of AI and machine learning, expansion of

monitoring capabilities, and improvements in energy efficiency present exciting avenues for further exploration. The project, designed for demonstration purposes, not only fulfills its immediate educational objectives but also sparks curiosity and innovation, encouraging professionals to envision and implement their IoT-based solutions.

In conclusion, the SMART PET HOUSE project serves as a successful demonstration of IoT principles using Arduino microcontrollers. Its achievements in modularity, real-time operation, and practical application make it an invaluable educational tool for industrial professionals seeking to advance their understanding of Arduino-based IoT projects. As the instructor at APU Corporate Training, the creation of this project and its accompanying documentation represents a step toward empowering professionals to navigate and excel in the realm of IoT development.

REFERENCES

Arduino - home. (n.d.). https://www.arduino.cc/

Arduino and stepper motor configurations | Arduino Documentation. (n.d.). https://docs.arduino.cc/learn/electronics/stepper-motors

Johnrickman. (n.d.). *GitHub - johnrickman/LiquidCrystal_I2C: LiquidCrystal Arduino library for the DFRobot I2C LCD displays*. GitHub. https://github.com/johnrickman/LiquidCrystal_I2C

Saputro, A. D., & Yantidewi, M. (2021). Analysis of air temperature and humidity in Kedunggalar against BMKG data based on DHT11 sensor. *Journal of Physics, 1805*(1), 012045. https://doi.org/10.1088/1742-6596/1805/1/012045

Servo - arduino reference. (n.d.). https://www.arduino.cc/reference/en/libraries/servo/

Welcome to Fritzing. (n.d.). https://fritzing.org/

APPENDICES

Appendix A: Source Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <DHT.h>
#include <Servo.h>
// Define pin for DHT sensor
#define DHTPIN 22
#define DHTTYPE DHT11
// Define pins for stepper motor
#define IN1  8
#define IN2  9
#define IN3  10
#define IN4  11
// Stepper motor variables
int Steps = 0;
boolean Direction = true;
unsigned long last_time;
unsigned long currentMillis;
int steps_left = 4095;
long time;
// Define pins for water level sensor and relay
const int waterLevelPin = A0;
const int relayPin = 23;
// Define pins for ultrasonic sensor and servo motor
const int trigPin = 13;
const int echoPin = 12;
const int servoPin = 28;
Servo myServo;
// Initialize DHT sensor and LCD
DHT dht(DHTPIN, DHTTYPE);
LiquidCrystal_I2C lcd(0x27, 16, 2);
// Variables to keep track of last update times
unsigned long lastWaterLevelCheck = 0;
unsigned long lastUltrasonicCheck = 0;
unsigned long lastTemperatureHumidityCheck = 0;
unsigned long lastStepperMove = 0;
unsigned long lastDirectionChangeTime = 0;
```

```
// Setup function
void setup() {
  Serial.begin(115200);
  // Setup pins for stepper motor
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);s
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  // Setup pins for water level sensor and relay
  pinMode(waterLevelPin, INPUT);
  pinMode(relayPin, OUTPUT);
  // Setup pins for RGB LED and sound sensor
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
  // Setup pins for ultrasonic sensor and servo motor
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  myServo.attach(servoPin);
  // Initialize LCD
  lcd.init();
  lcd.backlight();
  // Initialize DHT sensor
  dht.begin();
}

// Main loop function
void loop() {
  // Call different functions to handle various tasks
  handleWaterLevelControl();
  handleUltrasonicSensor();
  handleTemperatureHumidityMonitoring();
  handleStepperMotorControl();
  // Add other tasks as needed...
}

// Function to handle water level control
```

```
void handleWaterLevelControl() {
  unsigned long currentMillis = millis();

  // Check water level every 2 seconds
  if (currentMillis - lastWaterLevelCheck >= 2000) {
    lastWaterLevelCheck = currentMillis;

    // Read water level sensor and map the value
    int waterLevel = analogRead(waterLevelPin);
    int mappedLevel = map(waterLevel, 200, 430, 1, 10);

    // Control relay based on water level
    if (mappedLevel < 5) {
      digitalWrite(relayPin, HIGH);
      Serial.println("Relay ON - Water level: " +
String(mappedLevel));
    } else if (mappedLevel >= 10) {
      digitalWrite(relayPin, LOW);
      Serial.println("Relay OFF - Water level: " +
String(mappedLevel));
    }
  }
}

// Function to handle ultrasonic sensor
void handleUltrasonicSensor() {
  unsigned long currentMillis = millis();

  // Read ultrasonic sensor every 100 milliseconds
  if (currentMillis - lastUltrasonicCheck >= 100) {
    lastUltrasonicCheck = currentMillis;

    // Measure distance using ultrasonic sensor
    long duration, distance;
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
```

```
    duration = pulseIn(echoPin, HIGH);
    distance = duration * 0.034 / 2;

    // Control servo motor based on distance
    if (distance <= 10) {
      myServo.write(90);
    } else {
      myServo.write(0);
    }
  }
}

// Function to handle temperature and humidity monitoring
void handleTemperatureHumidityMonitoring() {
  unsigned long currentMillis = millis();

  // Read temperature and humidity every 2 seconds
  if (currentMillis - lastTemperatureHumidityCheck >= 2000) {
    lastTemperatureHumidityCheck = currentMillis;

    // Read temperature and humidity from DHT sensor
    float humidity = dht.readHumidity();
    float temperature = dht.readTemperature();

    // Display temperature and humidity on LCD
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Temp: ");
    lcd.print(temperature);
    lcd.print("C");

    lcd.setCursor(0, 1);
    lcd.print("Humidity: ");
    lcd.print(humidity);
    lcd.print("%");
  }
}

// Function to handle stepper motor control
```

```
void handleStepperMotorControl() {
  unsigned long currentMillis = micros();

  // Move the stepper motor if steps are remaining
  if (steps_left > 0) {
    if (currentMillis - last_time >= 1000) {
      stepper(1);
      time = time + micros() - last_time;
      last_time = micros();
      steps_left--;
    }
  } else {
    // Change direction after waiting for 10 seconds
    Serial.println(time);
    if (millis() - lastDirectionChangeTime >= 10000) {
      Serial.println("Waited 10 seconds, changing direction...");
      Direction = !Direction;
      steps_left = 4095;
      lastDirectionChangeTime = millis();
    } else {
      Serial.println("Waiting 10 seconds before changing
direction...");
    }
  }
}

// Function to control the stepper motor
void stepper(int xw) {
  for (int x = 0; x < xw; x++) {
    switch (Steps) {
      case 0:
        digitalWrite(IN1, LOW);
        digitalWrite(IN2, LOW);
        digitalWrite(IN3, LOW);
        digitalWrite(IN4, HIGH);
        break;
      case 1:
        digitalWrite(IN1, LOW);
        digitalWrite(IN2, LOW);
```

```
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, HIGH);
  break;
case 2:
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  break;
case 3:
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  break;
case 4:
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  break;
case 5:
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  break;
case 6:
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  break;
case 7:
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  break;
```

```
      default:
        digitalWrite(IN1, LOW);
        digitalWrite(IN2, LOW);
        digitalWrite(IN3, LOW);
        digitalWrite(IN4, LOW);
        break;
    }
    SetDirection();
  }
}

// Function to set the direction of the stepper motor
void SetDirection() {
  if (Direction == 1) {
    Steps++;
  }
  if (Direction == 0) {
    Steps--;
  }
  if (Steps > 7) {
    Steps = 0;
  }
  if (Steps < 0) {
    Steps = 7;
  }
}
```