Final Project Report

Authors:

Mel Steppe #2133209

Sean Zhao #2167316

Date: 3/21/2025

EE 474 Winter 2025

Table of Contents

Introduction

The primary programming challenges that we address in our final project are utilizing various types of hardware and integrating them with a scheduler to create an RFID detection based cat feeder system on an Arduino ESP32. Before working on the project, we planned sections of code to tackle to ensure a more efficient and smooth integration of all portions of the project. We dove into the documentation of all hardware components and ensured that they were functioning on their own. We then drew schematics and planned out how the physical portion of the project would be laid out. Finally we integrated all of the hardware components together and used a task scheduler with two cores to create a fully functional automated cat feeder.

System Design and Implementation

Components
RTC: Real time clock device used to keep track of time so that the feeder can only dispense food once per specified intervals. Uses the RTClib library to record and report time.

Servo Motor: Servo motor used to control a blocking mechanism that can be controlled and opened to dispense food to the cat. Uses the ESP32Servo library for controlling the motor. Motor can be moved both clockwise and counterclockwise.

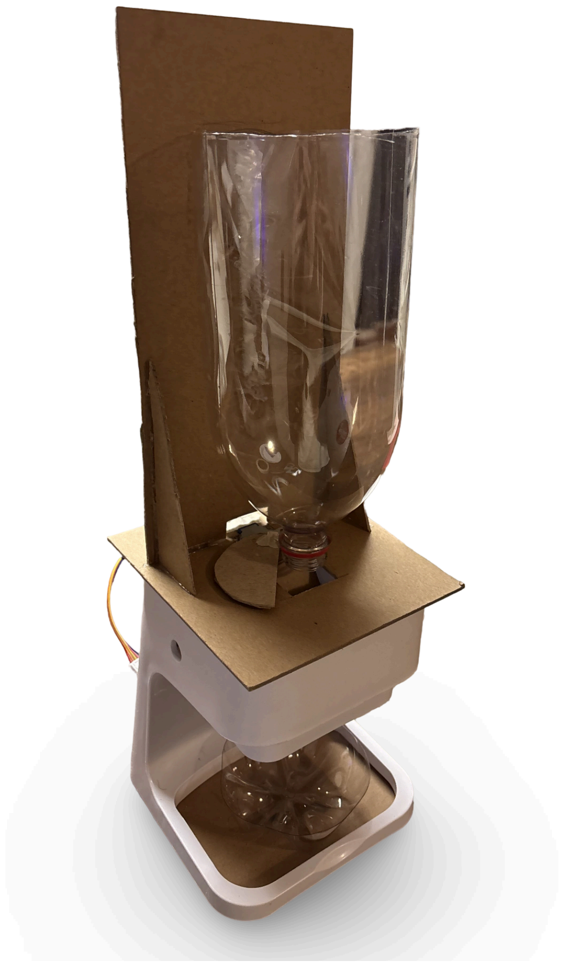
RFID Scanner: RFID scanner that will be used to identify specific RFID tags. Uses the MFRC522v2 library for controlling the scanner.

LCD Display: 16x2 LCD display that will be used to display information that the user can interface with. Uses the LiquidCrystal_I2C library for communication protocol with LCD display.

Pushbuttons: Push buttons that will be used for user interfacing to modify the time interval between feeding instances for the cats.

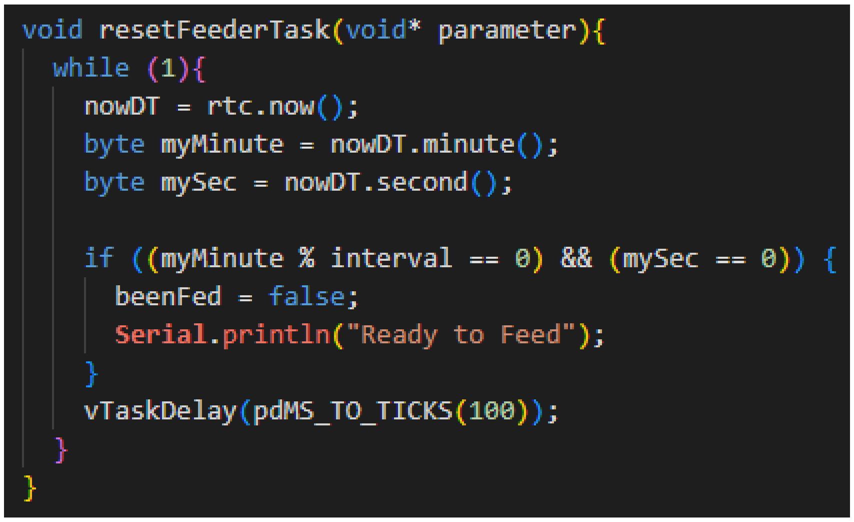
Design
Our system works by having the user specify a time interval for cats to be fed. The default interval is set to 5 minutes for demonstration purposes but the final application would use hours as the units for time. The user can interact with two push buttons to either increase or decrease the interval by 1 unit. Everytime the user modifies the interval, the new time interval will be displayed on the LCD screen. The RFID scanner is constantly reading and if a whitelisted RFID tag is detected, the servo motor will open as long as it has not already been opened during the specified time interval. The RTC is constantly noting the time and once a new time interval is reached, a boolean flag that keeps track of whether or not the feeder has opened already is reset.

The physical framework for the system was created using cardboard, chipboard, a 2L bottle, and the base from a plastic cereal dispenser. The model can be seen in the following image:



Link to Video Demonstration
https://www.youtube.com/shorts/-5rXxeRCHnI

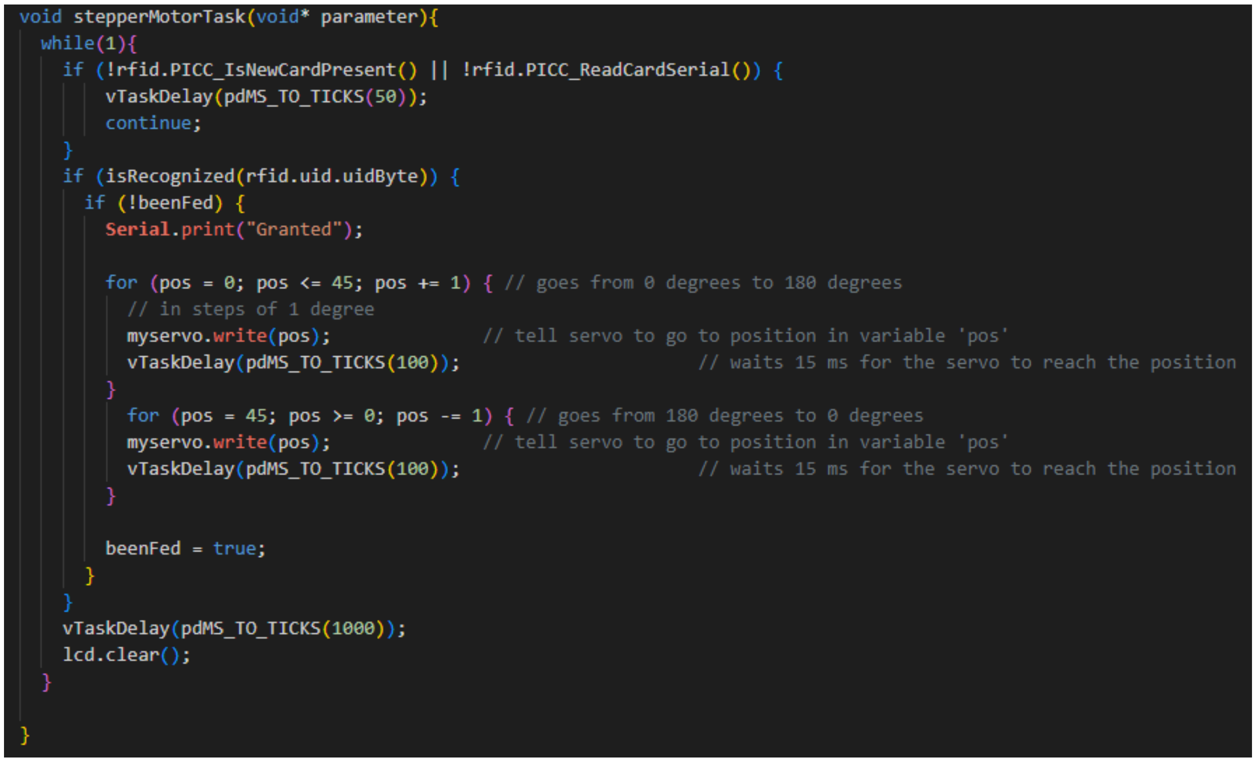
Code Structure



```
void resetFeederTask(void* parameter){
  while (1){
    nowDT = rtc.now();
    byte myMinute = nowDT.minute();
    byte mySec = nowDT.second();

    if ((myMinute % interval == 0) && (mySec == 0)) {
      beenFed = false;
      Serial.println("Ready to Feed");
    }
    vTaskDelay(pdMS_TO_TICKS(100));
  }
}
```

The reset feeder task works by utilizing the rtc module to read in the current minute and second of the system. Once the minute reaches a valid interval length and the respective seconds are 0, a global boolean flag beenFed is reset to false. For example, if the user sets an interval of 10 minutes and the current time is 1:25, once the clock hits 1:30 beenFed will be set to false.



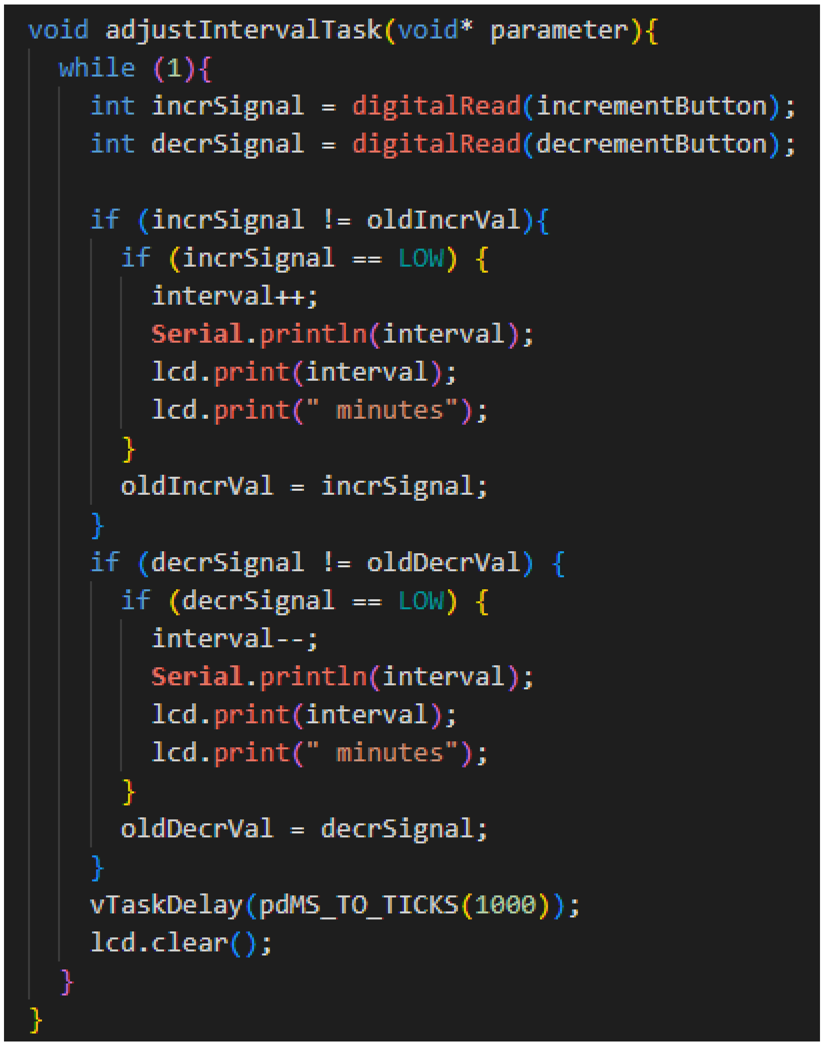
```
void stepperMotorTask(void* parameter){
  while(1){
    if (!rfid.PICC_IsNewCardPresent() || !rfid.PICC_ReadCardSerial()) {
        vTaskDelay(pdMS_TO_TICKS(50));
        continue;
    }
    if (isRecognized(rfid.uid.uidByte)) {
      if (!beenFed) {
        Serial.print("Granted");

        for (pos = 0; pos <= 45; pos += 1) { // goes from 0 degrees to 180 degrees
          // in steps of 1 degree
          myservo.write(pos);              // tell servo to go to position in variable 'pos'
          vTaskDelay(pdMS_TO_TICKS(100));                    // waits 15 ms for the servo to reach the position
        }
          for (pos = 45; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
          myservo.write(pos);              // tell servo to go to position in variable 'pos'
          vTaskDelay(pdMS_TO_TICKS(100));                    // waits 15 ms for the servo to reach the position
        }

        beenFed = true;
      }
    }
    vTaskDelay(pdMS_TO_TICKS(1000));
    lcd.clear();
  }

}
```

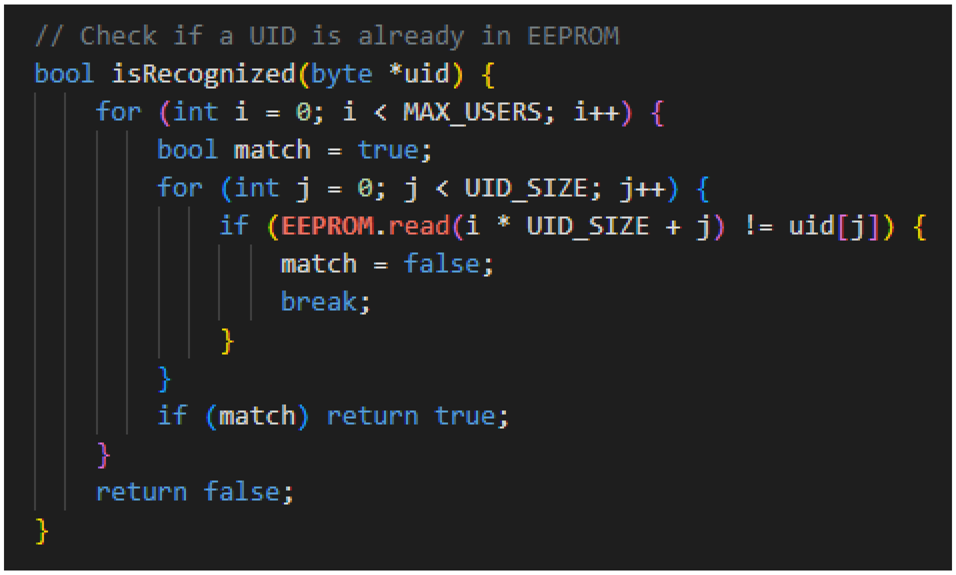
The stepper motor task is the main task that controls the servo motor. The first if statement of the task checks to see if the RFID tag is recognized or not. If it is not, the program continues and will loop back to the if statement. Once a recognized RFID tag is present. The program checks to see if the beenFed flag is true, indicating that the cat has already been fed during this current interval. If it is false the servo motor opens and closes to allow food to fall through and the beenFed flag is then set to true to prevent the cat from eating multiple times in one interval.



```
void adjustIntervalTask(void* parameter){
  while (1){
    int incrSignal = digitalRead(incrementButton);
    int decrSignal = digitalRead(decrementButton);

    if (incrSignal != oldIncrVal){
      if (incrSignal == LOW) {
        interval++;
        Serial.println(interval);
        lcd.print(interval);
        lcd.print(" minutes");
      }
      oldIncrVal = incrSignal;
    }
    if (decrSignal != oldDecrVal) {
      if (decrSignal == LOW) {
        interval--;
        Serial.println(interval);
        lcd.print(interval);
        lcd.print(" minutes");
      }
      oldDecrVal = decrSignal;
    }
    vTaskDelay(pdMS_TO_TICKS(1000));
    lcd.clear();
  }
}
```

The “adjust interval task” works by constantly reading two buttons. If there is a change that indicates a button was pressed and set to LOW, the global time interval variable will be either incremented or decremented.



```
// Check if a UID is already in EEPROM
bool isRecognized(byte *uid) {
    for (int i = 0; i < MAX_USERS; i++) {
        bool match = true;
        for (int j = 0; j < UID_SIZE; j++) {
            if (EEPROM.read(i * UID_SIZE + j) != uid[j]) {
                match = false;
                break;
            }
        }
        if (match) return true;
    }
    return false;
}
```

The isRecognized function is a pivotal component of our system as it determines whether or not a scanned RFID tag is recognized on the whitelist. It utilizes EEPROM which is a non volatile read only memory that can be reprogrammed with pulsed voltages. If the RFID tag matches any of the entries on the EEPROM the function will return true to indicate a match, if not it will return false.

Discussion

As we were working through the project, we encountered many technical difficulties with the hardware. The RTC and servo motor documentations that we found online were only compatible with a standard Arduino and not the ESP32, meaning we had to do deeper research to find more nuanced libraries that were compatible with the ESP32 and still allowed us to control the hardware in ways that we wanted to.

Another difficulty we encountered was working with the EEPROM in recognizing RFID tags. As it is a read only memory, removing invalid or unwanted tags from the EEPROM was extremely difficult and time consuming.

Another challenge we would like to comment on was the motor draining current from the LCD display. As both pieces of hardware needed 5V to operate, we found that the 5V supplied from the ESP32 often got drained by the servo motor whenever the gate was opened to feed the cats, causing the LCD screen to dim significantly or sometimes print strange characters to the display. To solve this problem, we used a 9V battery in conjunction with resistors to create a voltage divider to supply the motor with its independent stream of electricity.

Division of Work

As the final project did not contain parts, the division of work is a bit harder to comment on. Sean solely worked on the programming portion of the system and getting the hardware components to function as intended as well as setting up the scheduler. Mel was in charge of creating the physical structure of the system using cardboard and a cereal dispenser bought off of Amazon. Mel also helped tackle the arduous debugging process for the final system design, and the initial testing to figure out how each peripheral worked. This division of labor allowed both members to have their own responsibilities and tasks to work on at their own pace without being blocked by other dependencies at the start, before working together to complete the final design.

Suggestions

We believe that providing students with proper documentation for their respective devices could be a way to save lots of time and confusion for everyone. While it was not hard to find documentation online, oftentimes we were unsure whether or not the library we were using was compatible with the ESP32 or had other dependencies that may not have been listed.

We also initially wanted to use the IR remote for our system but found that the remote that we got from the pack did not come with any battery, other groups that choose the interface pack also faced these issues.

Conclusions

This project successfully implemented an RFID activated automatic cat feeder system by synthesizing the embedded systems knowledge we gained during the previous laboratories. By referencing our past use of peripherals such as the LCD display screen, we effectively implemented three novel peripherals including the RTC module, RFID scanner module, and a Servo motor. This experience has reinforced our learning and highlighted the importance of flexibility in an embedded systems project as we often had to pivot from our original plan when peripherals or libraries did not work as we originally anticipated. This was especially evident with the motor, as we changed from using the step motor to the servo motor after realizing the servo would work more effectively for the physical model we created.

References

[1]S. Santos, “ESP32: Guide for DS3231 Real Time Clock Module (RTC) | Random Nerd Tutorials,” *Random Nerd Tutorials*, Mar. 12, 2025. https://randomnerdtutorials.com/esp32-ds3231-real-time-clock-arduino/ (accessed Mar. 22, 2025).

[2]“OSSLibraries/Arduino_MFRC522v2,” *GitHub*, Feb. 22, 2024. https://github.com/OSSLibraries/Arduino_MFRC522v2

[3]D. Workshop, “Using Servo Motors with the ESP32,” *DroneBot Workshop*, Jul. 23, 2020. https://dronebotworkshop.com/esp32-servo/

Appendices

Total Number of hours Spent

We spent around 25 hours working on the final project.