Final Project Report
EME 154 A03

Jason Daniel Pieck
jdpieck@ucdavis.com
March 19, 2025

1. Project Description

The Automatic Padlock Opener Control (APOC) is a free-time system (FTS) designed to mount onto and open a padlock.

After initializing the system by entering the starting position, the user can operate in manual mode to input the lock combination manually. Alternatively, in automatic mode, the user can enter the three combination numbers on the keypad, allowing the system to enter the combination automatically.

If at any point the user wishes to reinitialize the system, they can do so through the setup menu. Once finished, they can exit the system.

2. Free-Time System (FTS)

The free-time system is modeled as shown in Figure 1.

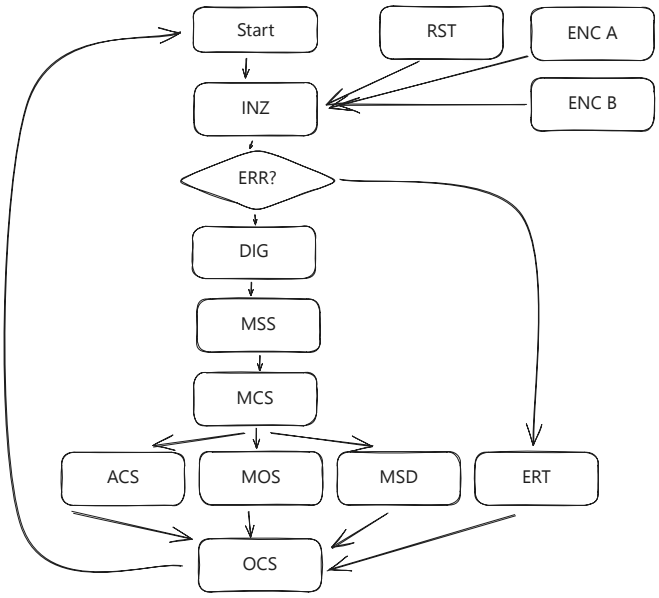

Figure 1

Based on the clock setup frequency, the system continuously runs, activating different modules based on user inputs and state information.

Encoder signals are measured using a timer-based interrupt, which determines the next instance of motor control based on the selected mode.

3. Project Circuit

The circuit diagram in Figure 2 was provided in the project documentation. The hardware setup is shown in Section 4.

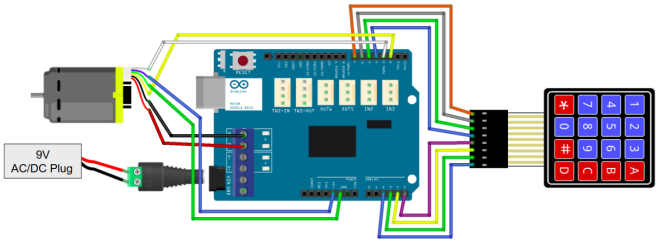

Figure 2

4. Hardware Setup

Below is the completed hardware setup.

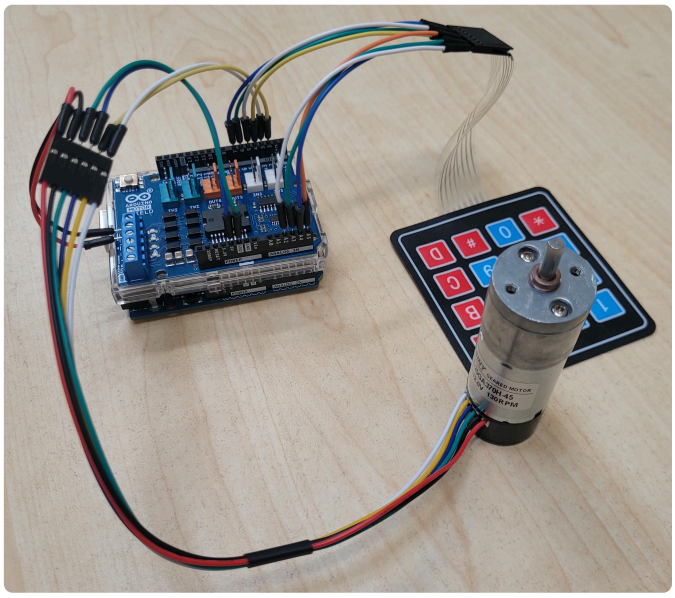

Figure 3

5. Human-Machine Interface

The Human-Machine Interface (HMI) operates entirely through the serial monitor. This means the Arduino must be connected to the Arduino IDE for the user to receive feedback and instructions.

5.1. Startup

When the system powers on, it sends the following message:

```
═══════ System Startup Initialized ═══════
═══ System Successfully Initialized! ═══
```

The first message helps the user recognize when a new session begins, and the second message signifies that the system started up successfully.

5.2. Setup

Once initialized, the system displays the following menu:

```
EME-154 Mechatronics    Free Time System
Jason Daniel Pieck
**************************************
            MACHINE SETUP MENU
**************************************
```

This is followed by a prompt asking the user to enter the starting position. If the user enters a position outside the dial range, the system will display an error message and indicate the valid range.

Once a valid dial position is entered, the system confirms the input and sets the position accordingly:

```
Input the initial dial position.
(2-digit) Combination Input: 88
Invalid Input! Input should be from 0-40

(2-digit) Combination Input: 24
Combination Number Accepted!

Dial Position Set to 24
```

Note that the logic used to validate the input range is the same in Automatic Mode and will return the same messages for out-of-range inputs.

5.3. Idle

After entering a valid combination, the following menu appears, allowing access to the system's functionalities:

```
EME-154 Mechatronics    Free Time System
Jason Daniel Pieck
**************************************
            OPERATION MENU
**************************************
1. Manual Operation
2. Automatic Operation
3. Machine Data Set-Up
5. Exit
**************************************
```

5.4. Manual

When the user selects manual mode, the following menu is displayed:

```
EME-154 Mechatronics    Free Time System
Jason Daniel Pieck
**************************************
            MANUAL OPERATION MENU
**************************************
1. Move One Tick CW
2. Move One Tick CCW
5. Exit
**************************************
Manual Mode Accepted
Dial Position is at: 24
```

When selecting `1` or `2` on the keypad, the system moves accordingly and provides feedback:

```
Moving 1 Tick CW
Dial Position is at: 25

Moving 1 Tick CW
Dial Position is at: 26

Moving 1 Tick CCW
Dial Position is at: 25
```

Selecting `5` exits back to the Idle Menu.

5.5. Automatic

Upon selecting Automatic Mode, the following menu is displayed:

```
EME-154 Mechatronics    Free Time System
Jason Daniel Pieck
**************************************
            AUTOMATIC MENU
**************************************
Please enter the lock combination.
```

The system prompts the user to input each combination number, verifying their validity as in Setup Mode:

```
~~~~ Combination 1 ~~~~
(2-digit) Combination Input: 10
Combination Number Accepted!

~~~~ Combination 2 ~~~~
(2-digit) Combination Input: 20
Combination Number Accepted!

~~~~ Combination 3 ~~~~
(2-digit) Combination Input: 30
Combination Number Accepted!
```

Once all three numbers are accepted, the system enters the combination, adding required rotations for the first and second numbers.

After completion, the user is prompted to exit automatic mode by pressing `#` on the keypad, returning to the Idle Menu:

```
Moving to: 10

Moving to: 20

Moving to: 30

You can pull the shackle now!
Press # to Exit
```

5.6. Exit

If the user opts to exit the system from the Idle Menu, the following message is displayed:

```
═══════════ System Exit ═══════════
```

6. System Development

This system was developed over two weeks after acquiring the hardware.

Although the project was based on provided code, large sections were rewritten, allowing a focus on developing Automatic Mode and tuning motor control parameters, including PI control.

6.1. Code Accessibility

To improve readability and debugging efficiency, several steps were taken.

Before coding, I renamed FTS functions to full, descriptive names, making their purpose clear without needing a reference key. Serial Monitor messages were also heavily utilized to track function outputs and execution, aiding debugging.

6.2. Localizing Function Logic

Redundant code was consolidated into single functions. For example, the combination validation logic initially appeared separately in Setup Mode and three times in Automatic Mode. By creating a single function, `getCombination()`, I was able to streamline modifications and reduce errors.

6.3. Git Version Management

After facing development setbacks, I began using Git for version control. Successful changes were committed, providing stable return points and facilitating troubleshooting.

You can view the full repository here.

7. Encountered Challenges

Below are some notable challenges faced during development.

7.1. Inverted PWM Setup

Initially, in Manual Mode, the motor failed to stop properly. Debugging via Serial Monitor revealed that the stop condition in `drivePulses()` was incorrectly set, causing the motor to continue running unexpectedly.

Swapping comparison operators resolved the issue, ensuring the motor stopped as intended when the correct conditions were met.

7.2. Memory Issues

During testing, menu text would disappear randomly, leading to unexpected behavior in system responses. Investigation revealed that the Arduino was exceeding its dynamic memory limit, likely due to excessive use of `Serial.print()` statements and large string allocations.

Bellow is the Ouput Message that the Arduino IDE would display when the strings started disappearing.

```
Sketch uses 11626 bytes (36%) of program
storage space. Maximum is 32256 bytes.

Global variables use 1689 bytes (82%) of
dynamic memory, leaving 359 bytes for local
variables. Maximum is 2048 bytes.
```

By reducing `Serial.print()` commands and staying below 1688 bytes of dynamic memory usage, these issues were resolved, resulting in stable menu operation.

7.3. Motor Control Tuning

Fine-tuning the motor control was the most time-consuming aspect of the project. While the system performed well for full rotations, it consistently over-rotated on short movements due what I believe to be accumulated errors in step calculations. Introducing proportional-integral (PI) control helped mitigate these errors, but unexpected behavior persisted, requiring a complete rewrite of the ISR and `drivePulses()` functions.

After multiple iterations and lots debugging, the system was successfully able to enter lock combinations accurately in Automatic Mode, reliably stopping near the correct positions.

8. Comments & Feedback

Overall, I found this project to be highly rewarding, providing valuable hands-on experience in mechatronics and motor control development. The combination of hardware and software challenges made for an engaging learning experience.

I think that it would have been great is students have the ability to take home the physical lock systems to test on. Since the motor response is greatly affected by its load, the final demonstration involved me having to change a load of parameters last minute.

Additionally, the provided codebase would benefit from improved structure and organization. Splitting it into multiple files, such as additional C++ and header files, would enhance modularity, making it easier for students to read, understand, and modify the code. Implementing a more structured approach to function calls and state management would also contribute to improved maintainability and scalability.