J&M Data Collection Automation Team – Project Documentation

Cal Poly Systems Optimizations Club '23-24

Table of Contents:

1. Introduction:

1.1 Abstract:

This technical and programming documentation paper was developed by Ethan Fischer, Ryan Dosanjh, Dakshesh Pasala, Eeshan Walia, and Giovanni Wang as a part of Cal Poly SOC. We have carefully designed the following instructions as a detailed reference for operators who intend to connect Arduino microcontrollers directly to a MySQL database for live data uploads during production days. More specifically, this system is designed to streamline the process of recording machine status codes and counting units produced, providing a comprehensive solution for efficient industrial monitoring and data management.

1.2 Cal Poly Systems Optimizations Club Team:

The California Polytechnic State University San Luis Obispo Systems Optimization Club (Cal Poly SOC) is a student-led and organized club associated with the Industrial and Manufacturing Engineering (IME) Department. Our club's primary purpose is to develop the next generation of professional Engineers through company sponsored projects and networking opportunities. Through such connections, students will learn to embody Efficiency, Experience, and Excellence in professional settings by practicing the completion of real world deliverables.

1.3 Introduction to Arduino:

Arduinos are powerful yet simple tools that can control and automate various tasks. They take inputs from the environment, process this information according to programmed instructions, and produce outputs to interact with the world. This makes them incredibly useful for a wide range of applications, from hobbyist projects to complex industrial systems.

- **Hardware:** An Arduino board consists of a microcontroller (the brain of the system), along with a variety of other components like voltage regulators, oscillators, and a USB connection for programming and power. The most common models include the Arduino Uno, Nano, and Mega. We will be using the Arduino Uno.

- **Programming:** You program an Arduino using the Arduino Integrated Development Environment (IDE) on your computer. The IDE allows you to write code in a simplified version of C++ and upload it to the Arduino via a USB cable. The code, known as a "sketch," tells the Arduino how to behave.

- **Inputs and Outputs:** Arduinos have pins that can be used for inputs and outputs. Inputs can come from sensors (like temperature sensors, light sensors, or buttons) that provide data to the Arduino. Outputs can control devices like LEDs, motors, or other electronics. For example, an input might be a button press, and an output could be turning on an LED. Our inputs for this project are mechanical inputs based on the machines our Arduino system is attached to.

- **Sensors and Actuators:** Sensors provide data to the Arduino, while actuators perform actions based on the Arduino's commands. For instance, a temperature sensor might send data to the Arduino, which then turns on a fan if the temperature exceeds a certain threshold. Our sensors in this project are primarily the RFID scanner.

- **Power:** Arduinos can be powered through a USB connection from a computer or through an external power supply. This flexibility makes them suitable for both stationary and portable projects.

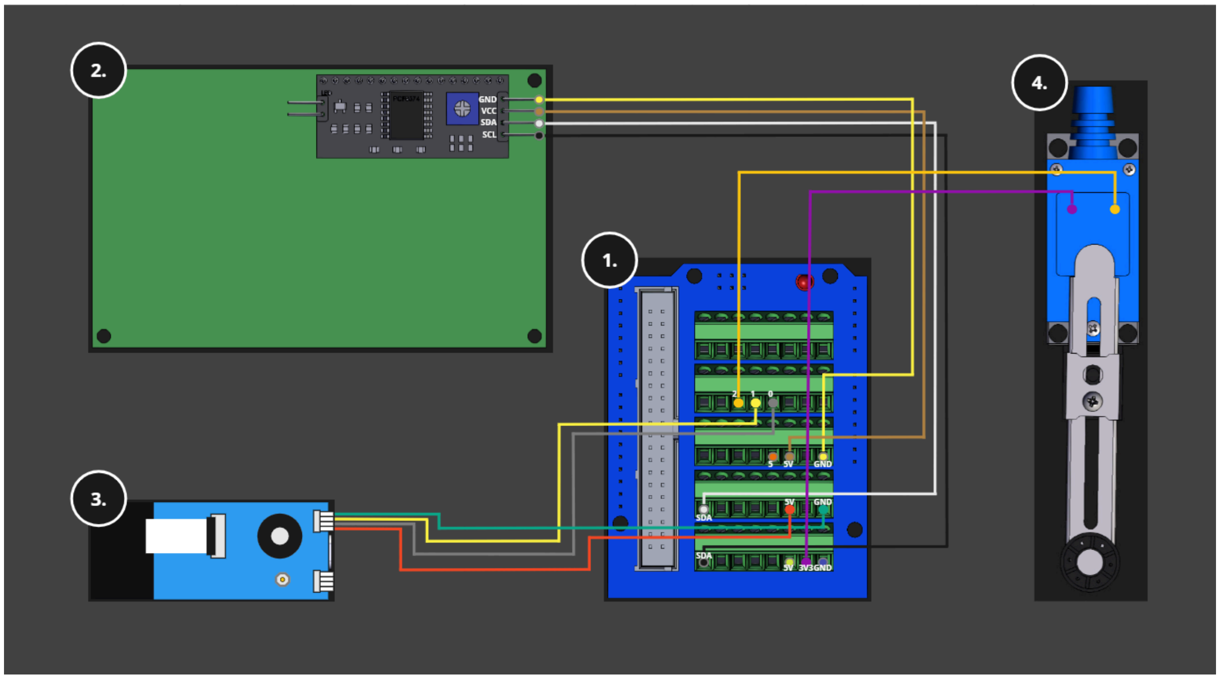
- **Communication:** Arduinos can communicate with other devices using various protocols like Serial, I2C, and SPI. This allows them to interface with other microcontrollers, computers, and a wide array of sensors and modules. Our Arduino system will be communicating with an external database through a configuration known as Flask API.

In summary, Arduinos are versatile, user-friendly microcontrollers that act as small computers, capable of interacting with their environment through various inputs and outputs.

1.4 System Overview:

The core of our Arduino system records various information through an Arduino microcontroller. Units produced are automatically recorded throughout the day, while any changes in machine status (such as a breakdown or malfunction) can be scanned by workers using a set of RFID tags. Each time a machine's status changes, the corresponding RFID tag can be manually scanned, capturing the information instantly and sending it to a separate database for management to review.

The following diagram details the exact layout and configurations of all required materials for the completed system:



1. **Arduino Breadboard:** The Arduino breadboard is a tool used to build and test circuits easily without soldering. It's a rectangular plastic board filled with tiny holes that hold electronic components and wires in place.

    - **Structure of a Breadboard:**
        - Rows and Columns: The breadboard is divided into rows and columns of holes. Each hole can hold the leg of a component or a wire.
        - Power Rails: The long rows along the sides of the breadboard are called power rails. They are usually marked with "+" and "-" to indicate positive and negative connections, providing power to your components.

        - Terminal Strips: The middle area of the breadboard contains rows of interconnected holes. Each row is electrically connected, meaning all the holes in a row can share the same electrical signal.

    - **Using a Breadboard with an Arduino:**
        - Connecting Components: You place components (like resistors, LEDs, or sensors) into the holes of the breadboard. Wires are used to connect these components to each other and to the Arduino.
        - Power and Ground: Connect the power (5V) and ground (GND) pins from the Arduino to the breadboard's power rails. This supplies the necessary electricity to your circuit.
        - Signal Wires: Use jumper wires to connect the input/output pins of the Arduino to the components on the breadboard. For example, you might connect a wire from an Arduino digital pin to an LED on the breadboard to control the LED with the Arduino.

    - **Monitoring Wire Connections:**
        - Continuity: Breadboards make it easy to change connections and see how they affect the circuit. You can visually follow the wires to ensure they connect the correct components.
        - Arduino Code: The Arduino can monitor connections through its code. For example, if you have a button on the breadboard connected to an Arduino pin, you can write code to check if the button is pressed and then respond accordingly.
        - Debugging: If something isn't working, you can easily move wires and components to different holes to troubleshoot and fix issues without needing to solder.

2. **Arduino LCD:** An LCD (Liquid Crystal Display) is a screen that can display text and numbers, which is useful for showing information directly from an Arduino.
    - **Understanding the LCD:**
        - Pins: The LCD has multiple pins for power, ground, data, and control signals. Typically, there are 16 pins, but sometimes fewer if it uses an I2C interface.
        - LiquidCrystal_I2C Library: Arduino has a built-in library called LiquidCrystal_I2C that simplifies controlling the LCD. This library provides functions to set up the LCD, write text, and control the display.

- **Monitoring and Displaying Data:**
  - Initial Setup: In the Arduino code, you initialize the LCD with the correct pin configuration.
  - Displaying Text: You can use functions like lcd.print() to display text or numbers on the screen. The required functions have been included in this document.

3. GM65 QR/Barcode Scanner: A QR/barcode scanner/reader is a device used to read data stored in barcode or QR codes. This technology is commonly used for access control, inventory tracking, and identification systems. We will be using barcode scanners to scan and load information for work orders and status codes in our current implementation, with plans for full use of scanners for all categorical variables in the future.

   - **Using Hardware Serial Connections:**
     - RX/TX Pins: The Arduino UNO has two pins set aside exclusively for hardware serial connectivity. The pins 0 (RX) and 1(TX) are used to communicate with external devices by using the Serial1 interface. This is the interface that we are using to communicate from the barcode scanner to the arduino.

4. **Arduino Limit Switch:** A limit switch is a type of sensor that can detect the presence or absence of an object, often used to limit the motion of a mechanical part. When integrated with an Arduino, a limit switch can be used for various applications such as safety interlocks, position detection, and event triggers.

   - **Understanding the Limit Switch:**
     - Mechanical Switch: A limit switch is a mechanical switch with an actuator lever that gets pressed when it comes into contact with an object.
     - Normally Open (NO) or Normally Closed (NC): The switch can be configured in either normally open (NO) or normally closed (NC) mode. In NO mode, the circuit is open until the switch is pressed. In NC mode, the circuit is closed until the switch is pressed.

   - **Connecting the Limit Switch to Arduino:**
     - Pins: The limit switch typically has three pins: common (COM), normally open (NO), and normally closed (NC). For a basic setup, you'll use the COM and NO pins.

        - Power and Ground: Connect one side of the switch to a digital pin on the Arduino and the other side to ground (GND). You can also connect a pull-up resistor between the digital pin and the 5V supply to ensure a stable signal.

    - **Using Code to Detect the Switch State:**
        - Digital Read: In the Arduino code, you can use the digitalRead function to check whether the switch is pressed or not.
        - State Change Detection: You can write code to perform actions when the switch state changes, such as turning on an LED or stopping a motor.

1.5 Required Materials:

- **Hardware:** Physical pieces for each component of our system.
    - Arduino IDE:
    - Arduino LCD:
    - Limit Switch - ME 8108
    - GM65 QR/Barcode Scanner
    - (X4) Jumper Wires - Female to male

- **Internal Software:**
    - Arduino IDE: https://www.arduino.cc/en/software
    - Arduino UNO R4 WiFi: https://store.arduino.cc/products/uno-r4-wifi
    - Python https://www.python.org/downloads/
    - Flask Program: *https://github.com/SOC-J-M/JM-Flask-App.git*
    - Arduino Program: *https://github.com/SOC-J-M/ArduinoCodeFinal.git*

- **Additional sites:** Supplementary information surrounding the hardware/software used in this project.
    - Uno R4 Wifi documentation: https://docs.arduino.cc/hardware/uno-r4-wifi/

2. Arduino IDE Setup:

2.1 Initializing Arduino IDE:

1. If you haven't already, download the Arduino IDE here: https://www.arduino.cc/en/software
   - For this project, you can download any version of the IDE. We tested it on both the latest version (2.3.2), and a legacy version (1.8.X).

2. Once the IDE is installed, plug in your Arduino to your machine, and allow the IDE to install the required drivers (automatically prompted).

2.2 Installing Arduino UNO R4:

1. To install the board package, go to Tools > Board > Board Manager. Search for Arduino UNO R4 Boards and install the latest version (1.1.0 or later).

2. To set your specific board: navigate to Tools > Board > Arduino Uno R4 Boards > Arduino UNO R4 WiFi

3. Make sure your board is connected to a port: Tools > Port > select whichever option has "(Arduino UNO R4)" next to it. The port connected to the Arduino will make it clearMake sure that your Arduino is connected to your computer.

4. In your Serial Monitor (Ctrl+Shift+M for Windows), make sure your baud rate is set to 9600. (Note: Serial Monitor is only for debugging purposes. Youwill not have it open when in production since the arduino will not be wired to a computer).

2.3 Initializing Arduino Code Using GitHub:

1. Navigate to the Github link above and download the .ino file (final.ino).
   a. Click on the file > "Download raw file" (top right of the code block)

2. Open the file with your Arduino IDE

3. You will be prompted with a message to create a new directory for the .ino file. Select "yes" to confirm this. The Arduino IDE will automatically place the .ino file in the correct location within the Arduino directory.

4. Once the code has been installed, the Port of the arduino may have changed. Follow the same instructions from part 3 in the “Installing Arduino UNO R4” section to make sure that the correct port is specified.

2.4 Installing Required Library:

1. From GitHub, download the LiquidCrystal_I2C.zip file.

2. To install the library: Sketch > Include Library > Add .ZIP Library

2.5 Running the Code:

1. Once the required libraries are installed, on the top left of your IDE, select the “Upload” button to send the code to your Arduino. NOTE: The program will not function properly yet as the Python API has not been connected! Please continue with the setup guide below.

3. Python API Setup:

3.1 Setting Up Python Flask:

- The Python API utilizes Flask, which is a web framework for python, designed to easily build web applications that handle routing, request handling, etc.
- For our purposes, Flask was used to build a RESTful API to handle requests to send data from each Arduino.

3.2 Performing Preliminary Setup:

1. First, install a stable version of Python from here. Our development environment was set up using Python 3.11.x, however, any recent version of Python will work as Flask is a well updated package.

2. Check python installation by running the following command: python --version

3. Next, make sure to check that pip is installed. Versions of Python newer than 3.4.x come with pip installed already. In the case that it isn’t, follow the instructions below to install:

- Checking if pip is installed: pip --version

Install pip:

a. Navigate to this page: https://bootstrap.pypa.io/get-pip.py
b. Right click the page and click 'Save As' and save it to some directory in your computer under the name 'get-pip.py'.
c. Open a terminal/command prompt and cd (cd [folder name]) to the directory that it is saved in.
d. Once in the correct directory, run the following command:
   python get-pip.py

3.3 Setting Up Flask:

- Once Python and pip are installed, the preliminary setup is complete!
- Now, in order to get the Flask app running on your machine, install the program code from GitHub.

  To Install:

  1. Open a terminal/command prompt and navigate to the directory in which you want to save the Flask program file. (use cd [directory name])
  2. If "git" is not already installed, install it from the download page.
  3. Once git is installed, run the following command:
     git clone https://github.com/SOC-J-M/JM-Flask-App.git

- Once the files are installed on your machine, you can proceed to setting up the environment.

  Setup (the following instructions are also available in the GitHub ReadMe file):

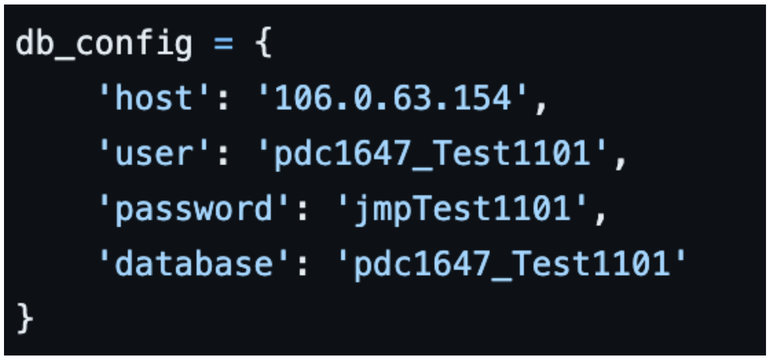
  1. Open a terminal/command prompt and navigate to the directory containing the Flask application.
  2. Once there, you will see a file called "requirements.txt". This file contains all the libraries needed to run the Flask application.
  3. Run the following command to install all the requirements at once:
     pip install -r requirements.txt
  4. You will see pip installing all the required libraries in your terminal/command prompt. Once the installer is complete, your setup is complete!

3.4 Running the Program:

- Once the environment is set up, and all the libraries are installed, there are only a few things that need to be noted before you are ready to go.
- The Flask application is used to communicate with the database, so **the database credentials need to be updated in order for the API to work**.

<u>**To Update Credentials:**</u>

a. Navigate to the directory with your Flask API and open the 'app.py' file using a code editor or text editor of your choice. We like to use VSCode, however there is no requirement for this step.
b. At the top of the file, there is a dictionary called 'db_config'. This contains all the credentials for the database that we want to communicate to. Find your database credentials and update the 'host', 'user', 'password', and 'database' fields according to your database management system.



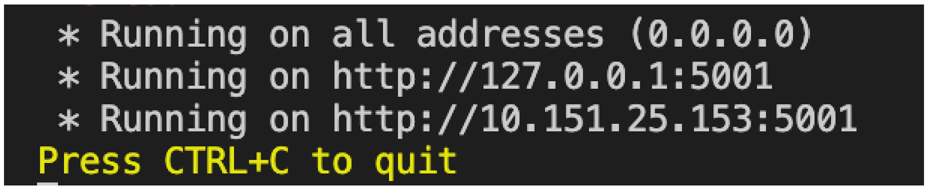
```
db_config = {
    'host': '106.0.63.154',
    'user': 'pdc1647_Test1101',
    'password': 'jmpTest1101',
    'database': 'pdc1647_Test1101'
}
```

c. Once the database credentials are updated, you are ready to go!

<u>**To Run:**</u>

1. Open a terminal/command prompt, and navigate to the directory with the code for the flask application.
2. Run the following command: flask run --host = 0.0.0.0
    a. *If on macOS*: add a port argument to specify a different port like so: flask run --host = 0.0.0.0 --port = 5001
    b. *NOTE*: The port used can vary from machine to machine. Make sure to check which ports are open for use on your machine. Flask default runs on port 5000, however on macOS, when using the host=0.0.0.0 argument, you often run into an error since port 5000 is normally used for "AirDrop".

3. Once the program is running, the *third line* starting with "Running on" is the PUBLIC IP that the flask app is running on (any device on your network should be able to access it). Take note of this value as it will be needed for the Arduino program setup.



```
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5001
 * Running on http://10.151.25.153:5001
Press CTRL+C to quit
```

3.5 Usage and Troubleshooting:

1. The easiest way to test if the Flask app is running is by testing it via another device on the same network.
2. Access your cell phone or another laptop/computer and connect to the same WiFi that your machine running the server is connected to.
3. Navigate to a web browser and type in the full URL in the third line of the Flask output, followed by the route "/values" (http://10.151.25.153:5001/values in the above example).
    a. This is a simple GET route to check if the API is functioning correctly.
    b. NOTE: Make sure to type "http" and not "https". This makes sure not to run into an error with encryption and security mismatches between the browser and the server.
4. You should be able to see the message "Success" on your screen. If this message isn't displayed, make sure that you have added the --host=0.0.0.0 argument, that you have specified "http", and that you are on the same network as the server.

4. Arduino LCD Setup:

4.1 Performing SDA and SCL Connections:

1. Identify the SDA and SCL Pins on the LCD: Look at the pins on your LCD. They are usually labeled. Find the pins labeled "SDA" and "SCL."

2. Identify the SDA and SCL Pins on the Screw Terminal Block Breakout: Locate the SDA and SCL pins on your breakout board. These should also be labeled.

3. Connect the SDA Pin: Take a jumper wire and connect one end to the SDA pin on the breakout board. Connect the other end of this wire to the SDA pin on the LCD.

4. Connect the SCL Pin:Take another jumper wire and connect one end to the SCL pin on the breakout board. Connect the other end of this wire to the SCL pin on the LCD.
    - This setup will allow for stable I2C communication between the Arduino and the LCD.

4.2 Performing Power Connections:

1. Identify the Ground and 5V Pins on the LCD: Locate the pins on your LCD labeled "GND" (Ground) and "VCC" (Voltage Common Collector, which is 5V).

2. Identify the Ground and 5V Pins on the Screw Terminal Block Breakout: Find the corresponding ground (GND) and 5V pins on your breakout board.

3. Connect the Ground Pin: Take a jumper wire and connect one end to the GND pin on the breakout board. Connect the other end of this wire to the GND pin on the LCD.

4. Connect the 5V Pin: Take another jumper wire and connect one end to the 5V pin on the breakout board. Connect the other end of this wire to the VCC pin on the LCD.
    - This will ensure that your LCD is properly powered with 5V.

4.3 Performing Limit Switch Connections:

1. Identify the Pins on the Limit Switch: Limit switches typically have two pins for connection.

2. Connect One Wire to the 3.3V Pin on the Screw Terminal Block Breakout: Take a jumper wire and connect one end to one of the pins on the limit switch. Connect the other end of this wire to the 3.3V pin on the Screw Terminal Block Breakout. This will provide power to the switch.

3. Connect the Other Wire to Digital Pin 2: Take another jumper wire and connect one end to the remaining pin on the limit switch. Connect the other end of this wire to digital pin 2 on the Screw Terminal Block Breakout. This will read the state of the switch.
    - This setup ensures that when the switch is pressed, it changes the state detected by the arduino, allowing you to count the presses.

4.4 Usage and troubleshooting:

- If you encounter issues with the LCD display or the limit switch not working, here are some steps to help you identify and resolve the problem:

1. Check Connections: Ensure all connections are secure and in their designated locations. Verify that the SDA and SCL lines from the Screw Terminal Block Breakout are correctly connected to the SDA and SCL pins on the LCD. Confirm that the ground from the breakout is connected to the ground pin on the LCD and that the 5V from the breakout is connected to the VCC on the LCD.

2. Power Supply: Make sure the LCD is receiving 5V power and not 3.3V. The LCD requires 5V to operate correctly, and using 3.3V can result in malfunction or no display.

3. Limit Switch Connections: Verify that one wire from the limit switch is connected to the 3.3V pin on the Arduino and the other wire is connected to digital pin 2. Also, ensure the internal pull-up resistor is enabled in your code using pinMode(2, INPUT_PULLUP);.

4. Library and Code: Double-check that the necessary libraries (Wire.h and LiquidCrystal_I2C.h) are included and correctly initialized in your code. Ensure there are no syntax errors or logical mistakes in your code that could affect the functionality.

5. Debounce Issues: If the limit switch is overly sensitive or not counting correctly, verify that the debounce logic in your code is correctly implemented. Ensure you have a delay to prevent multiple counts from a single press.
    - By following these troubleshooting steps, you should be able to diagnose and fix common issues with the LCD display and limit switch in your project.

5. GM65 QR/Barcode Scanner Setup:

5.1 Connect to Arduino:

The GM65 QR/barcode scanner has connectivity via USB and UART. Since we are connecting to Arduino, we want to use UART connection over RX/TX pins. Connect the GM65 QR/barcode scanner to 5V and GND on the Arduino; connect the RX pin on GM65 to Pin 0 on Arduino, connect the TX pin on GM65 to Pin 1 on Arduino.

5.2 Datasheet:

Access the GM65 QR/barcode scanner datasheet here. The barcode scanner settings are set by scanning the QR codes in the datasheet. The factory default settings are incompatible with the Arduino so we have to make some adjustments.

5.3 Settings:

1. Reset to factory default: go to page 4, section "**1.5 Reset**". Scan the QR code labeled **"Reset"**. This ensures no other settings have been modified other than the ones necessary.

2. Enable series communication: go to page 5, section **"2.1 Series Communication Interface"**. Scan the QR code labeled **"Series Output"**. This allows the scanner to communicate with the Arduino over serial.

3. Set baud rate settlement: still on page 5, section **"2.1 Series Communication Interface"**, scan the QR code labeled **"9600bps(Default)"**. This sets the preferred communication channel used by the Arduino, mainly for debugging purposes when opening the Serial Monitor.

4. (Optional) Enable manual scanning mode: This won't affect the functionality of the scanner, but may be preferred in certain use cases. By default, the GM65 QR/barcode scanner is set to continuous read mode; this means it will stay on and scan at all

times. This increases the likelihood of accidental scans. Manual scanning will require a toggle key (located on top of component) to be held in order to scan. To enable, go to page 9, section **"3.3 Manual Mode"**. Scan the QR code labeled **"Manual Mode".**

5. (Optional) Set time settlement for single read **continuous mode only**:
Alternatively, you may wish to change the break time between scans in continuous mode to prevent duplicate scanning. Go to page 7, section **"3.1 Continuous Mode"**, subsection **"Break time settlement"** to set your preferred break time.

5.4 Usage and Troubleshooting:

- The barcode scanner at any given time takes work order numbers and status codes. Reference the QR status codes here. Any other QR/barcodes scanned will be ignored and not recorded by the Arduino.

- If a new work order number is scanned, the Arduino assumes a new job has begun. If a previous work order is already running, the Arduino terminates it, assumes the job has completed, and pushes the remaining record of the old work order. The global time, part quantity, and status are all reset before the Arduino begins collecting data again.

- If a new status code is scanned, the Arduino pushes the current record immediately and updates the status of the work order. It waits until either a new work order has been scanned or the status is changed back to "NORMAL". Upon change, it will push a record and resume collection as normal.

- If a work order/status code is scanned that is already present (e.g. work order 1234 is running and GM65 scans 1234 again), it will be ignored.

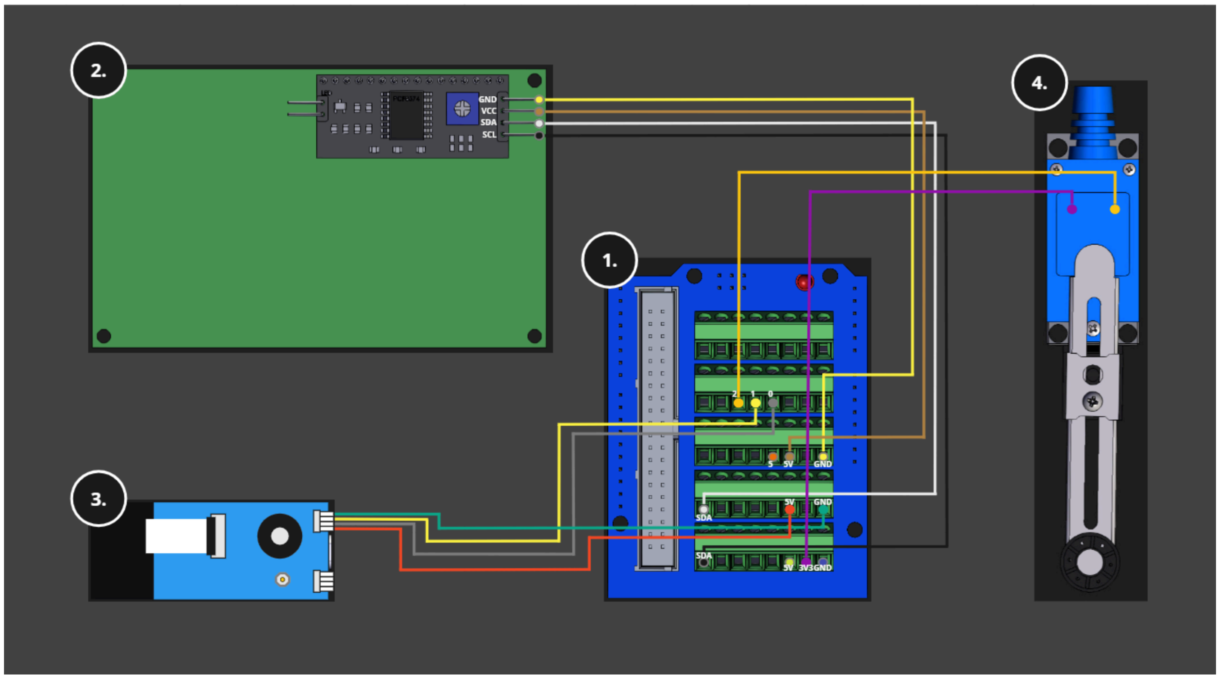
- If the GM65 QR/barcode scanner is not working, here are some steps to identify and resolve the problem:

1. Reset to factory default settings and try step 5.2 again. If the component is on and scanning but the Arduino is not reading it, it's possible one of the settings were not initialized correctly on the scanner.

2. Check wired connections to Arduino. Ensure you are connected to 5V and GND, and that the RX pin (GM65) is set to pin 0 (Arduino) and the TX pin (GM65) is set to pin 1 (Arduino). **They cannot be connected to any other pin on the Arduino.**

3. Try a different GM65 QR/barcode scanner. It's possible the component is damaged, in which case it's worth trying out a new component.

6. System Usage:



6.1 Overview:

Reference the diagram above for a system overview

1. Arduino UNO R4 WiFi
   - Main IoT device, connecting GM65 QR/Barcode scanner, LCD display, limit switch, and Flask API
   - Data inputs: GM65 QR/Barcode scanner (work order numbers, status codes), limit switch (number of parts made per cycle per work order)
   - Data outputs: LCD display (updates elapsed time, work order number, status codes), Flask API (sends formatted record data to be processed before sending to database)

2. LCD display
   - Updates work order, elapsed time, and status for informing operators
   - Controlled by Arduino

3. GM65 QR/Barcode scanner
    - Scans work orders and status codes and sends them to the Arduino. All other scans are ignored.

4. Limit switch
    - Mechanical switch attached to the press machines; incrementally counts the number of parts made at any given time.

5. Flask API (not shown)
    - Reads and processes data from the Arduino each cycle
    - Pushes the data to the SQL database for a completed record

6.2 System initialization:

- Wire up the setup as described in this document

- **Before uploading the Arduino code to the arduino, make sure to specify the time between record pushes to the database. This can be defined in the pushTime variable at the top of the Arduino program.** *Note that this value is in milliseconds, so 10,000 corresponds to 10 seconds, 600,000 corresponds to 10 minutes, etc.*

- Initialize Flask API and upload Arduino code (reminder: modify WiFi, password, server IP and server host variables as indicated)

- Power on the Arduino, either through wired connection or battery power.

- The system is ready to use!

6.3 Interacting with the system:

1. Once the code is uploaded to the Arduino, it should trigger the setup function, causing the LCD to light up and display a blank value for work order and time. This indicates that the arduino is waiting for a work order to be scanned.

2. Scan a valid work order.

    a. This will trigger the start of a work order for the Arduino

3. To indicate a change in status, scan one of the barcodes corresponding to the respective status that you would like to switch to.

4. In order to end a work order and start another, simply scan a new barcode corresponding to the next work order.

6.4 Troubleshooting:

<u>IF THE ARDUINO IS NOT PUSHING DATA:</u>

- Plug the arduino into a computer and open the Serial Monitor via the Arduino IDE (Tools > Serial Monitor). Check the logs and verify that once the code is uploaded, that the Arduino displays a confirmation for WiFi connectivity. If WiFi is the issue, make sure to check the SSID and PASS variables for accurate information. If the WiFi is not the issue, wait to see if the Arduino displays an error connecting to the server, or if there is a bad response from the Flask server itself.
    - In the case of an error connecting to the client, verify that the Flask server is visible to devices over the network (See section 3.5)
    - If the server is returning a bad request, view the actual runtime of the Flask server to see what issues it is returning. This can be viewed in the terminal/command prompt that was used to start the Flask server. The issue likely lies in a bad query, so make sure to check the query that is being printed out in the terminal to see where the issue lies. The most probable issue is a barcode that includes spaces, which causes an error when running the POST call.

<u>IF THE QUANTITY VALUES ARE INCORRECT:</u>

- The most likely culprit for incorrect part quantities in the database is loose connections.
- Look at the limit switch connected to the arduino and make sure that all the wires are properly connected.
- If the limit switch looks fine, take a look at the screw breakout terminal mounted on the Arduino to make sure that the wires are properly fastened.

<u>IF THE BARCODE IS NOT SCANNING:</u>

- View the detailed troubleshooting guide for the GM65 barcode scanner in section 5.4.

7. Closing and acknowledgements:

Key takeaways:
Our Arduino system integrates several entry level components to perform an action typically done by human workers. Its machine accuracy offers significant improvements in operational efficiency and data accuracy to the production floor and provided an in-depth exploration of integrated electrical systems for all members of our team. Additionally, the project allowed for additional exposure to the connections between complex electrical systems and online databases.

Acknowledgements:
We extend our heartfelt gratitude to our project sponsor, Ash Green, whose support and vision have been instrumental in the successful realization of this project. Thank you for your invaluable contribution and commitment to innovation.

We would also like to commend CalPoly SOC for the incredible opportunities we have been able to experience as a team, and for those that the organization will continue to offer for its students. It is important that we credit our Project Director Nick Quattrocci in particular, whose guidance throughout critical design reviews have ensured that our results demonstrate adherence to SOC's core values of Efficiency, Experience, and Excellence.