# Design of the Cheese Vat System

## Objectives

The objective of this project is to design a fully automated Cheese Vat system. Process involved are Pasteurization, Water refill and drainage and cheese making and draining. To make cheese precisely, system will have precise temperature regulation, VFD based agitator control and efficient fluid flow using the pump.

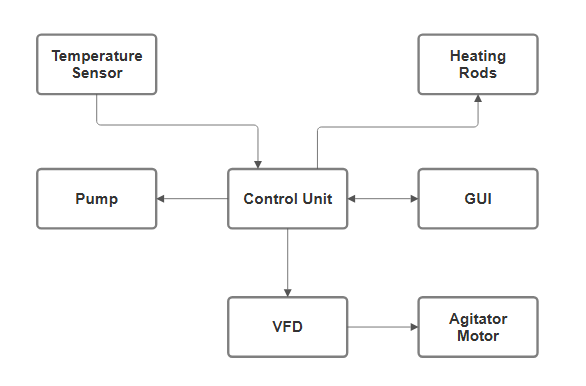
## Components

Basic components included in the design are:

1. Temperature sensor
2. Heating Rods
3. Pump
4. Variable Frequency Drive
5. 3 phase agitator motor and the gearbox
6. On/Off switches
7. Relays
8. Controller/Control unit
9. GUI

## System Design

### Block Diagram



### Control Topology

It breaks the control system into **4 layers**, each responsible for specific aspects of control, interaction, and decision-making:

#### ****Level 1: Hardware (Low level) Layer****

**Purpose:**This layer serves as the **foundation of the system**, providing a direct and simplified interface to the microcontroller’s physical hardware resources. It **abstracts low-level register manipulations** into readable and reusable functions, shielding the upper layers from hardware-specific details. The hardware layer makes it easier to **port code across different microcontrollers** or platforms by localizing hardware dependencies.

**Responsibilities:**

* Reading temperature sensor (ADC)
* Controlling relays (GPIO) for heating rods and pump
* Sending PWM/analog/digital commands to VFD for motor control
* GPIO toggles for direction control

#### ****Level 2: Device Control Layer****

**Purpose:**This layer focuses on managing **individual hardware components as modular devices**, building on top of the hardware level. It adds **intelligence to hardware control**, such as interpreting raw data, handling safety margins, and managing on/off control logic. It allows each hardware component like motors, sensors, relays, etc. to behave like a controlled unit with **defined behavior and interfaces**, making the code easier to reuse and scale.

**Responsibilities:**

* Calculate actual milk temperature from sensor voltage
* On/off control logic for heating rods
* Motor speed/direction control logic
* Pump control logic based on temperature or time

#### ****Level 3: High level****

**Purpose:**

This layer represents the **core functional brain of the system**. It integrates and coordinates multiple devices to execute higher-level tasks and processes. Whether it’s running a production cycle, managing heating/cooling phases, or implementing error handling and state transitions, this layer defines the **“what to do” and “when to do it” logic**. It keeps the system behavior consistent and organized based on **real-time input, feedback, and rules.**

**Responsibilities:**

* Implements the control logic for **Pasteurization, Cheese, Cooling**, etc.
* Coordinates multiple devices based on process state
* Applies safety checks and feedback evaluation

#### ****Level 4: User Interface****

**Purpose:**  
This topmost layer is responsible for **all interaction between the system and the user or external systems**. It handles user inputs, displays system status, and allows configuration of operational parameters. Whether via touchscreens, GUI panels, serial terminals, or web dashboards, it enables users to control, monitor, and understand the system in real time. It also manages scheduled operations, manual overrides, and real-time feedback visualization.

**Responsibilities:**

* GUI for selecting modes and setting parameters
* Display real-time system status
* Scheduled automation
* GUI screen to set desired temperature, motor speed
* "Scheduled Mode" logic that triggers processes based on time
* Manual override from the GUI

### Pseudo Code

Pseudo code for control system is following:

#### Initialize the system

Configure GPIO pins as outputs to control actuators (relays, direction pins). This enables the MCU to physically switch heating, pump, and motor direction.

Initialize GPIO for heating rods, pump, direction control

Initialize ADC for temperature sensor input

Initialize communication interfaces (SPI, UART, RS-485) for GUI and VFD

Initialize GUI module (touchscreen or PC interface)

Initialize temperature sensors

Initialize motor drivers and VFD interface

Initialize RTC (Real-Time Clock) for scheduling

Set default mode to MANUAL

#### Set system variables

Define thresholds, motor parameters, and mode defaults. These values guide process control and safety.

Set minimum temperature Threshold

Set maximum temperature Threshold

Define agitator settings:

Pasteurization: speed = x RPM, direction = FORWARD

Cheese: speed = y RPM, direction = FORWARD

Cooling: speed = z RPM, direction = REVERSE

Load scheduled Tasks←get data from GUI or saved memory

Set current Mode ← GUI input or default (Manual/Auto/Scheduled)

Set default Heating Status OFF

Set default Pump Status OFF

#### Start the main loop

Begin continuous control logic that repeats as long as the system is active. Everything from reading sensors to updating the GUI happens here.

While (true):

#### Read inputs from GUI

Collect user inputs like selected mode, start/stop commands, and target values. These define what operation should occur during this loop iteration.

modeSelection ← GUI.getMode()

startCommand ← GUI.getStartStopCommand()

manualOverride ← GUI.getManualOverrideStatus()

targetTemperature ← GUI.getTargetTemperature()

agitatorSpeed ← GUI.getManualMotorSpeed()

agitatorDirection ← GUI.getManualMotorDirection()

#### Read current temperature sensor

Sample the analog signal and convert it into a temperature value. This is used to regulate heating and display live temperature to the user.

adcValue ← ReadADC(channel)

currentTemperature ← ConvertADCtoTemp(adcValue)

GUI.updateTemperature←(currentTemperature)

#### Implement heat control logic

Turn heating rods ON/OFF based on current vs. target temperature. It ensures proper thermal regulation according to the process phase.

If (currentTemperature < targetTemperature):

TurnOn(heatingRodRelay)

heatingStatus ← ON

Else:

TurnOff(heatingRodRelay)

heatingStatus ← OFF

#### Implement agitator motor control logic

It uses VFD to control agitator speed and direction based on mode or manual input and keeps the mixture consistent and follows process-specific parameters.

If (modeSelection in ["Pasteurization", "Cheese", "Cooling"]):

If (manualOverride is FALSE):

agitatorSpeed ← agitatorSettings[modeSelection].speed

agitatorDirection ← agitatorSettings[modeSelection].direction

VFD.setSpeed(agitatorSpeed)

SetDirection(agitatorDirection)

StartMotor()

Else:

StopMotor()

#### Implement pump control logic

Activate or deactivate the water circulation pump depending on the mode. It supports temperature control and drain processes.

If (modeSelection in ["Cooling", "Drain"]):

TurnOn(pumpRelay)

pumpStatus ← ON

Else:

TurnOff(pumpRelay)

pumpStatus ← OFF

#### Handling scheduled mode logic

Run predefined tasks at specific times using RTC. It allows fully automated process sequences without manual input.

If (currentMode == SCHEDULED):

currentTime ← RTC.getTime()

For each task in scheduledTasks:

If (task.time == currentTime AND task.notCompleted):

currentMode ← task.process

task.markAsCompleted()

Log("Scheduled process started:", task.process)

#### Handling auto mode logic

Apply intelligent logic for heat/stir profiles based on process phase. It simplifies operation and minimizes need for user intervention

If (currentMode == AUTO):

ApplyProcessLogicBasedOnPhase()

AdjustHeatAndStirBasedOnPhase(currentTemperature, elapsedTime)

CheckSafetyLimits()

UpdateTimers()

#### Update GUI with real time data

Show current status of temperature, motors, heating, and alarms. It gives the user immediate feedback on system activity.   
GUI.updateStatus({

"Temperature": currentTemperature,

"Motor Speed": agitatorSpeed,

"Motor Direction": agitatorDirection,

"Heating": heatingStatus,

"Pump": pumpStatus,

"Mode": currentMode,

"Alarms": CheckAlarms()

})

#### Restart the main loop

Return to the start of the control loop for the next iteration and keeps the system in continuous operation and monitoring.

## Layered Architecture with Corresponding Pseudocode & Files

#### 1. Hardware Layer

Purpose: Interact directly with physical devices (GPIO, ADC, UART, SPI, RTC, VFD).

##### Header Files:

|  |  |
| --- | --- |
| **File** | **Pseudocode Responsibility** |
| gpio.h | Initialize GPIO for heating rods, pump, direction control TurnOn(heatingRodRelay), TurnOff(), pumpRelay control |
| adc.h | ReadADC(channel) |
| comms.h | Initialize SPI, UART, RS-485 |
| motordriver.h | SetDirection(), VFD.setSpeed(), StartMotor(), StopMotor() |
| rtc.h | RTC.getTime() |

##### Layer Pseudocode:

Initialize GPIO for heating rods, pump, direction control

Initialize ADC for temperature sensor input

Initialize communication interfaces (SPI, UART, RS-485)

Initialize motor drivers and VFD interface

Initialize RTC (Real-Time Clock) for scheduling

adcValue ← ReadADC(channel)

#### 2. Mid/Control Level Layer (control/)

Purpose: Translate raw hardware signals into usable data and apply device-level logic.

##### Header:

|  |  |
| --- | --- |
| **File** | **Pseudocode Responsibility** |
| TempControl.h | ConvertADCtoTemp(adcValue) Heating ON/OFF logic |
| MotorControl.h | Logic for agitatorSpeed, agitatorDirection Manual override check |
| PumpControl.h | ON/OFF logic for pumpRelay |
| SensorInterface.h | GUI.updateTemperature ← currentTemperature |

##### Layer Pseudocode:

currentTemperature ← ConvertADCtoTemp(adcValue)

GUI.updateTemperature ← (currentTemperature)

If (currentTemperature < targetTemperature):

TurnOn(heatingRodRelay)

heatingStatus ← ON

Else:

TurnOff(heatingRodRelay)

heatingStatus ← OFF

If (modeSelection in ["Pasteurization", "Cheese", "Cooling"]):

If (manualOverride is FALSE):

agitatorSpeed ← agitatorSettings[modeSelection].speed

agitatorDirection ← agitatorSettings[modeSelection].direction

VFD.setSpeed(agitatorSpeed)

SetDirection(agitatorDirection)

StartMotor()

Else:

StopMotor()

If (modeSelection in ["Cooling", "Drain"]):

TurnOn(pumpRelay)

pumpStatus ← ON

Else:

TurnOff(pumpRelay)

pumpStatus ← OFF

#### 3. High-Level Process Layer (process/)

Purpose: Handle process flows like Pasteurization, Scheduling, Auto logic, and Safety.

##### Header Files:

|  |  |
| --- | --- |
| **File** | **Pseudocode Responsibility** |
| ProcessSelect.h | Mode logic (Manual/Auto/Scheduled) |
| Scheduler.h | Load scheduled task logic |
| Config.h | Check limits, Apply the process logic based on phase |
| ProcessLogic.h | Heat and Stir profile Logic |

##### Layer Pseudocode:

Set minimum temperature Threshold

Set maximum temperature Threshold

Define agitator settings:

Pasteurization: speed = x RPM, direction = FORWARD

Cheese: speed = y RPM, direction = FORWARD

Cooling: speed = z RPM, direction = REVERSE

Load scheduled Tasks ← get data from GUI or saved memory

Set current Mode ← GUI input or default (Manual/Auto/Scheduled)

If (currentMode == SCHEDULED):

currentTime ← RTC.getTime()

For each task in scheduledTasks:

If (task.time == currentTime AND task.notCompleted):

currentMode ← task.process

task.markAsCompleted()

Log("Scheduled process started:", task.process)

If (currentMode == AUTO):

ApplyProcessLogicBasedOnPhase()

AdjustHeatAndStirBasedOnPhase(currentTemperature, elapsedTime)

CheckSafetyLimits()

UpdateTimers()

#### 4. User Interface Layer (GUI)

Purpose: Provide interactive display and input system for users.

##### Headers:

|  |  |
| --- | --- |
| **File** | **Pseudocode Responsibility** |
| gui\_interface.h | GUI.getMode(), GUI.getStartStopCommand(), GUI.getTargetTemperature() |
| gui\_update.h | GUI.updateStatus(), GUI.updateTemperature() |

##### Layer Pseudocode:

Start the main loop

While (true):

modeSelection ← GUI.getMode()

startCommand ← GUI.getStartStopCommand()

manualOverride ← GUI.getManualOverrideStatus()

targetTemperature ← GUI.getTargetTemperature()

agitatorSpeed ← GUI.getManualMotorSpeed()

agitatorDirection ← GUI.getManualMotorDirection()

GUI.updateStatus({

"Temperature": currentTemperature,

"Motor Speed": agitatorSpeed,

"Motor Direction": agitatorDirection,

"Heating": heatingStatus,

"Pump": pumpStatus,

"Mode": currentMode,

"Alarms": CheckAlarms()

})

## Components Description

### Temperature Sensor

K-type Thermocouple Temperature Sensor will be used to control the Heating rods. |It is a widely used temperature sensor known for its broad operating range and durability in various industrial environments.

#### Detection Principle

A Type K thermocouple operates based on the Seebeck effect, where a voltage (electromotive force) is generated at the junction of two dissimilar metals when there is a temperature difference between that junction and the other ends of the metals. This voltage is proportional to the temperature difference and can be measured to determine temperature.​

The Type K thermocouple consists of:

* **Positive leg (Chromel):** Approximately 90% nickel and 10% chromium.
* **Negative leg (Alumel):** Approximately 95% nickel, 2% aluminum, 2% manganese, and 1% silicon.​

This combination provides a sensitivity of approximately 41 µV/°C.

#### **Integration into the system**

The Type K thermocouple would be integrated into the controller circuitry in the following way:

##### ****Sensor Placement****

The **thermocouple probe** is inserted into the cheese vat in a position that represents the bulk temperature of the liquid (milk or curd). It must be **sealed and food-safe**, often housed in a stainless-steel sheath.

##### ****Signal Conditioning Circuit****

Since the thermocouple produces a **very low millivolt signal**, it is connected to a **thermocouple amplifier module** such as **MAX31855**, **MAX6675** or an **instrumentation amplifier** with cold junction compensation. These modules Amplify the signal, convert it to a **digital SPI-compatible signal** and include **cold junction compensation**

##### ****Connection to Microcontroller****

The digital output from the thermocouple module (e.g., SPI pins: MISO, SCK, CS) is connected to the **microcontroller**. The microcontroller reads the temperature value at regular intervals.

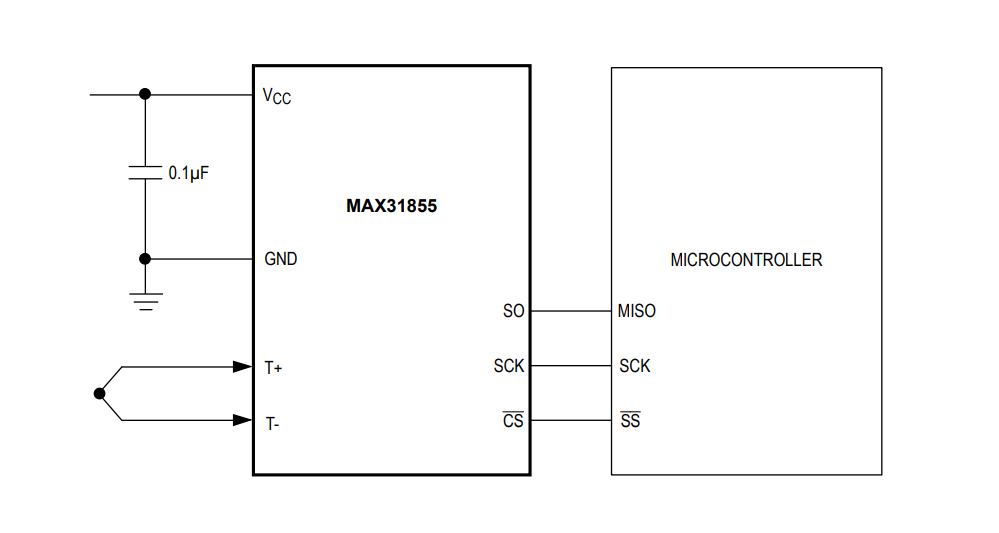
##### ****Control Logic****

The microcontroller uses this data to:

* + Control **heating elements**
  + Display temperature on an **GUI panel**
  + Trigger alarms if temperature goes out of bounds

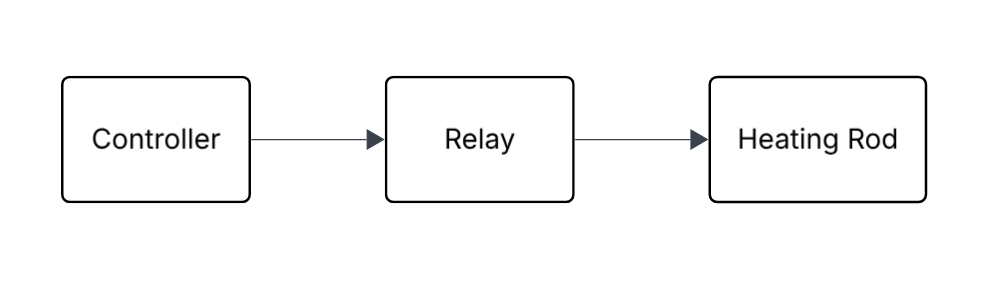
##### ****Power Supply****

* Ensure the thermocouple amplifier and microcontroller share a **common ground.**
* Typical supply: **5V or 3.3V**, depending on module and controller.



### Heating Rods

Electrical Heating Rods will heat the water which will be circulating inside the walls of vat to regulate the temperature as per desired value of the process. Controller will control it using an SSR.



### VFD

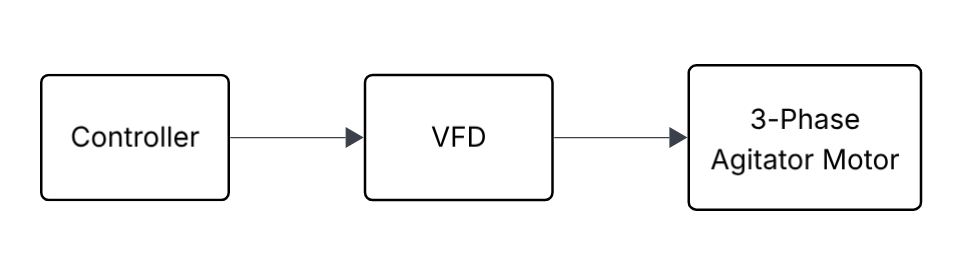
VFD will control the speed and direction of rotation of the agitator based on the output signal voltage mapping from the controller and STF and STR enabling.

#### **Working principle**

The FR-CS84-036-60 is a Variable Frequency Drive (VFD) used to control the speed, direction and torque of 3-phase induction motors by varying the frequency and voltage of the power supplied to the motor.

* **AC-DC-AC Conversion:**
  1. **Rectifier Stage:** Converts 3-phase AC input into DC.
  2. **Filter:** Smoothens the DC signal.
  3. **Inverter Stage:** Uses IGBT switches to convert DC back into a variable-frequency AC output.
* **Control Logic:**  
  The inverter adjusts the output frequency based on analog/digital inputs or serial communication commands, allowing for smooth motor acceleration, deceleration, and speed control.

#### Basic Wiring Diagram



### 3-Phase Agitator Motor

Agitator motor whose speed and direction will be adjusted by VFD, will be used for gentle or vigorous agitation depending on the process phase. Its specs will be following:

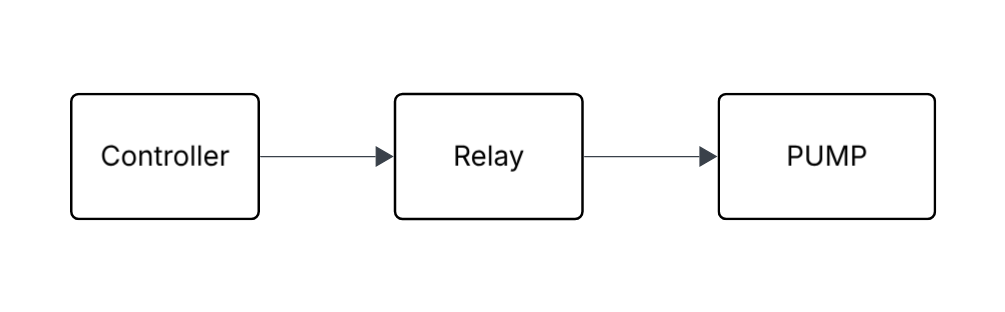
* Power: 0.33 hp
* Speed: 29.7 rpm
* Gearbox Ratio: 59:1

### Pump

The GRS 32/8 Taiffu pump with flow rate 140ltr/hr will circulate the water in the walls of vat.

#### **Working principle**

It is connected to 230V AC power supply and an SSR will be controlled by the controller to turn it on/off as per specific requirement.



### GUI

The GUI serves as the central interface between the **user** and the **control system**, offering real-time feedback and control over processes such as **pasteurization, cheese production, cooling, water handling,** and **scheduling**.

#### ****System Integration and Communication****

##### ****Microcontroller ↔ GUI Communication:****

* + Via **UART**, **RS-485**, **MODBUS**, or **USB** depending on the platform
  + Data exchange for temperature, motor speed, direction, and mode status

**Service Interface:** Admin panel for diagnostics, manual control, and firmware updates

##### ****Main Functional Components****

##### ****Mode Selection Screen****

Touch buttons or dropdowns to choose:

* **Pasteurization**
* **Cheese Production**
* **Cooling**
* **Water Drain & Refill**
* **Scheduled Operations**

##### ****Mode Screens****

Each mode screen shows relevant data and controls:

##### ****Pasteurization Mode****

* Set target temperature
* Real-time temperature graph
* Start/Stop buttons
* Safety lock and status messages

##### ****Cheese Mode****

* Stirring speed and direction control
* Agitator on/off
* Target and current temperature
* Timer for cutting/stirring duration

##### ****Cooling Mode****

* Control water circulation pump
* Current vat temperature
* Cooling target settings

##### ****Water Drain & Refill Mode****

* Manual or automated control of solenoid valves
* Water level indicator (optional sensor-based)
* Timers for drain/fill cycles

##### ****Scheduling Mode****

* Calendar/time picker for process automation
* Define mode sequences and durations
* View/edit future task queue

#### ****UI Element Linking****

**Buttons:**

* Linked to control commands (Start, Stop, Drain, Stir, etc.)
* Trigger visual status changes

**Temperature Display:** Live data from thermocouple/ADC module

**Speed Display:**

* Live speed reading from VFD or encoder.
* Adjustable via slider or input field

**Direction Display:**

* Shows agitator motor direction
* Toggle or arrow icons for forward/reverse

## Pseudocode for Pasteurization

Initialize 🡪 Start motor and Heater 🡪 Heat to Target Temperature 🡪 Hold Temperature for assigned time 🡪 End of the process

**gui\_interface.h**

**gui\_update.h**

**config.h**

TempControl.h

MotorControl.h

PumpControl.h

### Initialize the parameters

targetTemperature ← 72°C

holdingTime ← 15 minutes

agitatorSpeed ← 50 RPM

agitatorDirection ← FORWARD

heatingStatus ← OFF

motorStatus ← OFF

holdingTimer ← 0

processStatus ← "Heating Phase"

### **2.** Start Pasteurization Process

//GUI.display("Starting Pasteurization.")

MotorControl.startMotor(agitatorSpeed, agitatorDirection)

motorStatus ← ON

TempControl.turnOnHeater()

heatingStatus ← ON

### 3. Heating Phase

While (currentTemperature < targetTemperature):

adcValue ← SensorInterface.readTemperatureSensor()

currentTemperature ← TempControl.convertADCtoTemperature(adcValue)

GUI.updateTemperature(currentTemperature)

If (currentTemperature < targetTemperature):

TempControl.turnOnHeater()

heatingStatus ← ON

Else:

TempControl.turnOffHeater()

heatingStatus ← OFF

GUI.updateHeatingStatus(heatingStatus)

//delay if needed

### 4. Holding Phase

holdingTimer.start()

While (holdingTimer.elapsedTime() < holdingTime):

adcValue ← SensorInterface.readTemperatureSensor()

currentTemperature ← TempControl.convertADCtoTemperature(adcValue)

GUI.updateTemperature(currentTemperature)

If (currentTemperature < (targetTemperature - 1°C)): // if the difference b/w current and target is 1°C

TempControl.turnOnHeater()

heatingStatus ← ON

Else:

TempControl.turnOffHeater()

heatingStatus ← OFF

GUI.updateHeatingStatus(heatingStatus)

GUI.updateHoldingTimer(holdingTime - holdingTimer.elapsedTime())

Delay(short\_interval)

### 5. End Pasteurization

TempControl.turnOffHeater()

MotorControl.stopMotor()

heatingStatus ← OFF

motorStatus ← OFF

GUI.display("Pasteurization Complete!")

//ProcessLogic.markProcessComplete("Pasteurization") //if want to update in this in logs

Cheese Making Process **Initialize → Start motor and Heater → Heat to Target Temperature → Hold Temperature for assigned time → End of the process**

**gui\_interface.h**

**gui\_update.h**

**config.h**

TempControl.h

MotorControl.h

PumpControl.h

### 1. Initialize the parameters

targetTemperature ← 38°C

holdingTime ← 30 minutes

agitatorSpeed ← 30 RPM

agitatorDirection ← FORWARD

heatingStatus ← OFF

motorStatus ← OFF

holdingTimer ← 0

processStatus ← "Heating Phase"

### 2. Start Cheese Making Process

// GUI.display("Starting Cheese Making.")

MotorControl.startMotor(agitatorSpeed, agitatorDirection)

motorStatus ← ON

TempControl.turnOnHeater()

heatingStatus ← ON

### 3. Heating Phase

While (currentTemperature < targetTemperature):

adcValue ← SensorInterface.readTemperatureSensor()

currentTemperature ← TempControl.convertADCtoTemperature(adcValue)

GUI.updateTemperature(currentTemperature)

If (currentTemperature < targetTemperature):

TempControl.turnOnHeater()

heatingStatus ← ON

Else:

TempControl.turnOffHeater()

heatingStatus ← OFF

GUI.updateHeatingStatus(heatingStatus)

// delay if needed

### 4. Holding Phase

holdingTimer.start()

While (holdingTimer.elapsedTime() < holdingTime):

adcValue ← SensorInterface.readTemperatureSensor()

currentTemperature ← TempControl.convertADCtoTemperature(adcValue)

GUI.updateTemperature(currentTemperature)

If (currentTemperature < (targetTemperature - 1°C)):

TempControl.turnOnHeater()

heatingStatus ← ON

Else:

TempControl.turnOffHeater()

heatingStatus ← OFF

GUI.updateHeatingStatus(heatingStatus)

GUI.updateHoldingTimer(holdingTime - holdingTimer.elapsedTime())

Delay(short\_interval)

### 5. End Cheese Making

TempControl.turnOffHeater()

MotorControl.stopMotor()

heatingStatus ← OFF

motorStatus ← OFF

GUI.display("Cheese Making Process Complete!")

// ProcessLogic.markProcessComplete("Cheese") // if want to update in logs

## **Cooling Process Pseudocode:**

**gui\_interface.h**

**gui\_update.h**

**config.h**

TempControl.h

MotorControl.h

PumpControl.h

#### **1. Initialize Parameters:**

// Set target temperature for cooling (user-defined)

targetTemperature ← 4°C // Cooling target temperature

// Initialize motor and pump status

// motorStatus ← OFF

// pumpStatus ← OFF

currentTemperature ← 0°C // Placeholder for current temperature

processStatus ← "Cooling Phase" // Track process status

#### **2. Start Cooling Process:**

// Start the cooling process

PumpControl.turnOnPump() // Turn on the pump for cooling

pumpStatus ← ON

// Optionally start the motor (if needed for agitation during cooling)

MotorControl.startMotor(agitatorSpeed, agitatorDirection)

motorStatus ← ON

#### **3. Monitoring Temperature:**

// continuously check the temperature

While (currentTemperature > targetTemperature):

// Read the current temperature from the sensor

adcValue ← SensorInterface.readTemperatureSensor()

currentTemperature ← TempControl.convertADCtoTemperature(adcValue)

// Update GUI with current temperature

GUI.updateTemperature(currentTemperature)

// Check if cooling target temperature is achieved

If (currentTemperature <= targetTemperature):

// Cooling process is complete

PumpControl.turnOffPump() // Turn off the cooling pump

pumpStatus ← OFF

// Optionally stop the motor if not required after cooling is done

MotorControl.stopMotor()

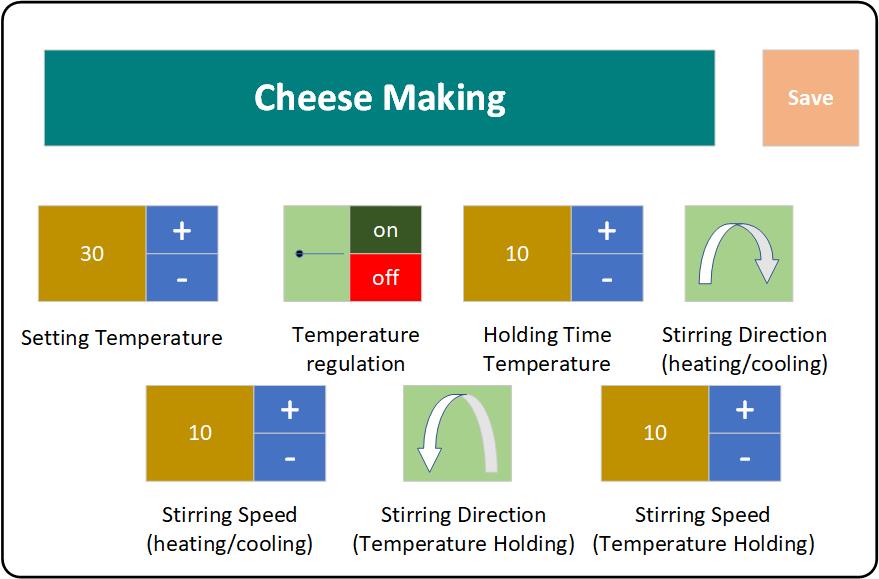
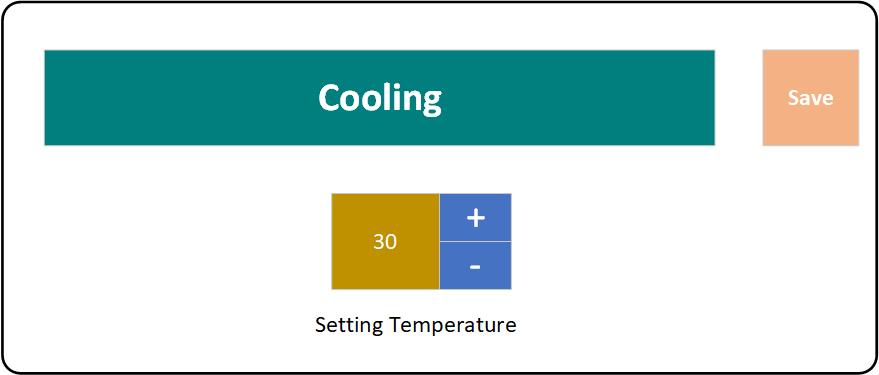
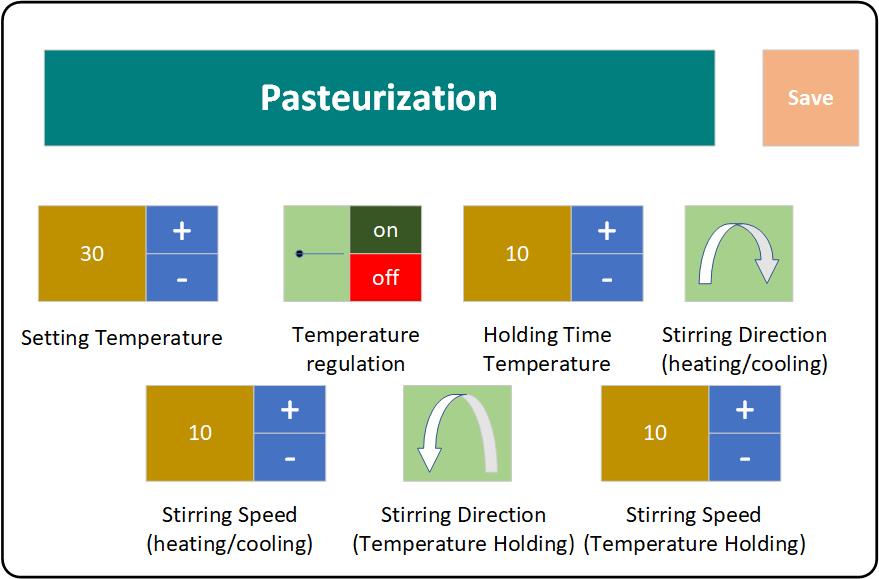
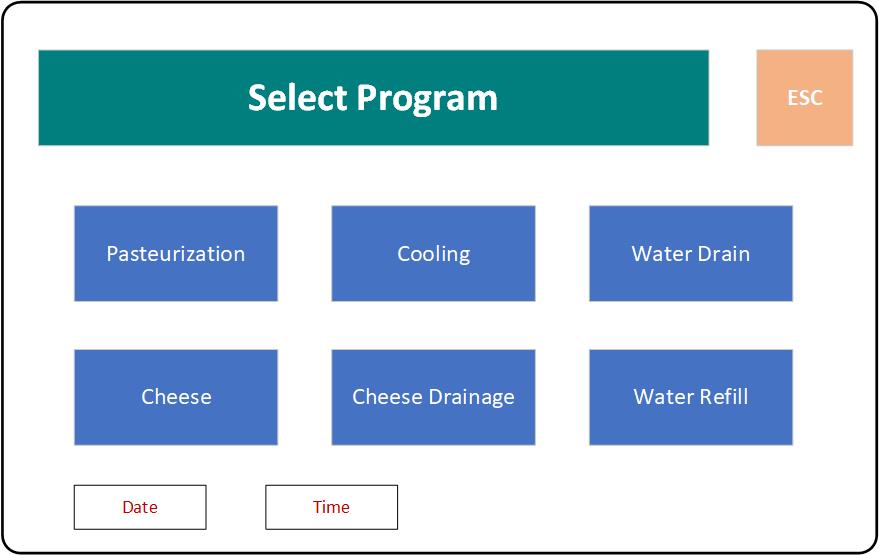
motorStatus ← OFF

// Display the process completion on the GUI

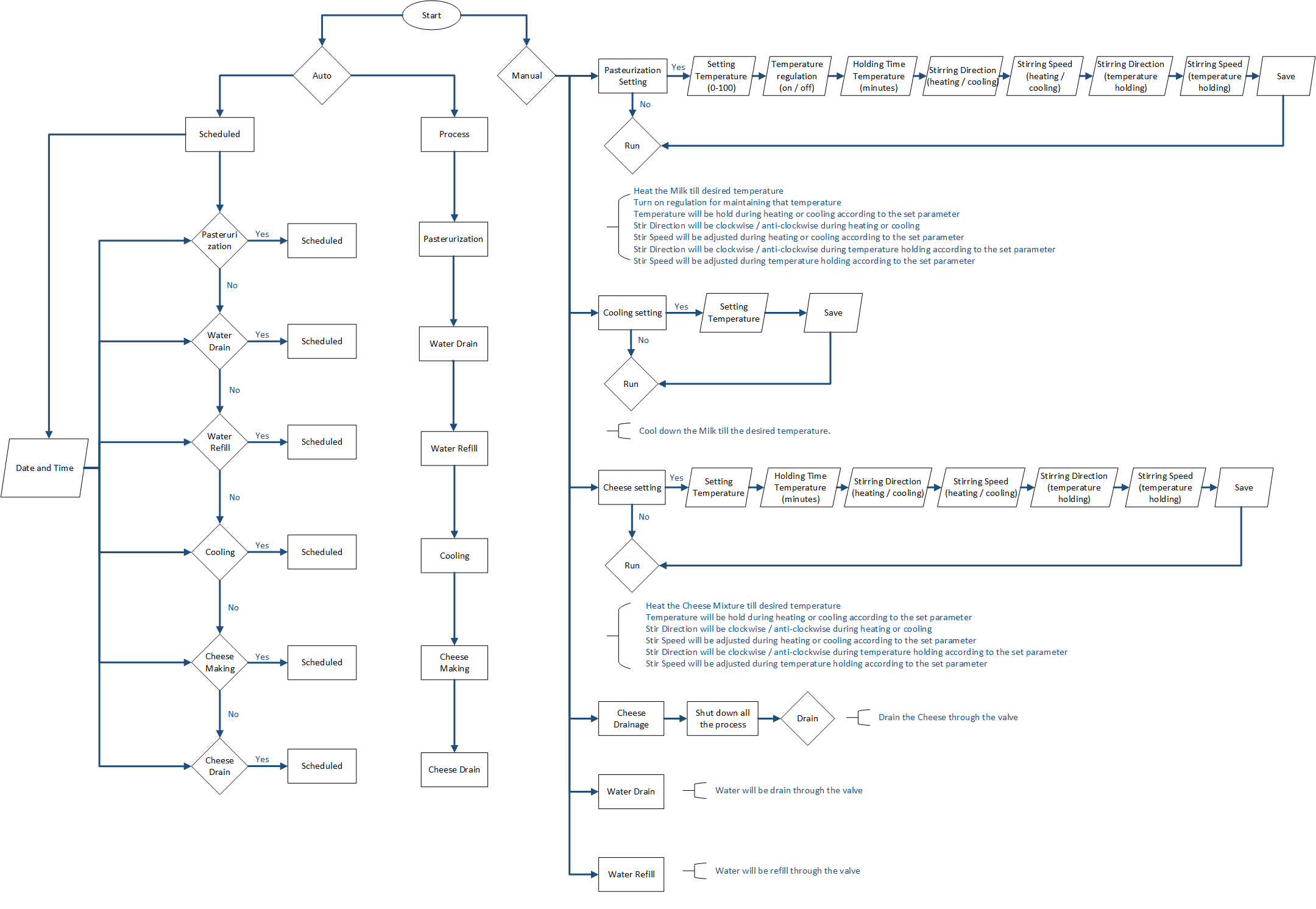
GUI.display("Cooling Complete!")

//delay

#### GUI Screens



### Flow chart



#### Flow Chart Description

The flow chart illustrates a cheese making control system with three operational modes:

##### 1. Auto Mode - Scheduled

This mode follows a predefined schedule:

* The system checks the current date and time
* It checks if any process (Pasteurization, Water Drain, Water Refill, Cooling, Cheese Making, or Cheese Drain) is scheduled
* When a scheduled time arrives, the corresponding process is automatically executed
* Each process follows its predefined parameters
* The system continues monitoring the schedule, executing processes when their scheduled time arrives

##### 2. Auto Mode - Process

This mode executes processes in sequence automatically:

* Processes run in a predetermined order: Pasteurization → Water Drain → Water Refill → Cooling → Cheese Making → Cheese Drain
* Each process starts immediately after the previous one completes
* Parameters for each process are predefined
* The system handles transitions between processes without user intervention
* This creates a continuous production flow

##### 3. Manual Mode

The manual mode gives operators complete control over each step of the cheese making process, while the auto modes provide either scheduled or continuous production capabilities

* The user selects which process to run (Pasteurization, Cooling, Cheese Making, etc.)
* For each process, the user can configure specific parameters:
  + For Pasteurization: temperature settings, regulation control, holding time, stirring direction and speed
  + For Cooling: desired temperature
  + For Cheese Making: temperature, holding time, stirring parameters
* The user initiates each process by selecting "Run"
* The system executes only the selected process with the specified parameters
* Settings can be saved for future use
* After process completion, the user can select the next process to run