Sputter Control System - Software Manual

Repository Structure

The sputter control software is organized into logical modules to handle different aspects of the vacuum sputtering system:

```
auto_control/
├── python/                          # Main application code
│   ├── app.py                       # Main PyQt5 GUI application
│   ├── main.py                      # Entry point script
│   ├── config.py                    # Configuration loader
│   ├── arduino_controller.py         # Hardware communication (serial)
│   ├── auto_procedures.py           # Automated vacuum procedures
│   ├── safety/                      # Safety interlock system
│   │   ├── safety_controller.py     # Safety logic evaluator
│   │   └── safety_conditions.yml    # YAML-based safety rules
│   ├── security/                    # User authentication
│   │   ├── password_manager.py      # Password hashing & verification
│   │   └── reset_passwords.py       # Admin password reset utility
│   ├── gas_control/                 # Mass flow controller integration
│   │   ├── controller.py            # Main MFC driver
│   │   ├── subprocess_controller.py # Subprocess-based MFC variant
│   │   ├── recipes.py               # Gas flow recipes/presets
│   │   ├── config.yml               # MFC configuration
│   │   └── safety_integration.py    # MFC safety checks
│   ├── widgets/                     # PyQt5 UI components
│   │   ├── indicators.py            # Status indicator widgets
│   │   ├── mfc_dialog.py            # Mass flow controller dialog
│   │   ├── mode_dialog.py           # System mode selector
│   │   ├── plotter_widget.py        # Real-time data plotting
│   │   ├── password_setup_dialog.py # Password initialization
│   │   └── other_dialogs.py         # Additional dialog boxes
│   ├── tests/                       # Unit and integration tests
│   └── __init__.py
├── relay_controller/                # Arduino firmware
│   └── relay_controller.ino         # Arduino Mega 2560 sketch
├── docs/                            # Documentation
│   ├── TECHNICAL_MANUAL.md          # Hardware pin assignments
│   ├── software_manual.md           # This file
│   ├── README_desktop_launcher.md   # System launch guide
│   ├── security_codes.txt           # Emergency codes
│   └── pics/                        # Hardware photos
├── sput.yml                         # Runtime configuration (pins, thresholds)
├── vacuum_system_gui.ui             # Qt Designer UI file
└── __init__.py
```

Module Overview

Core Application

`app.py` - Main PyQt5 Application

- **Purpose**: Primary GUI application for system control
- **Key Components**:
  - Main window with relay/procedure controls
  - Timer-based status monitoring (refresh_status, refresh_inputs)
  - Safety state evaluation (update_safety_state)
  - Background procedure execution via QThreadPool
  - Real-time pressure and sensor display
  - User authentication and role-based access control
- **Responsibilities**:
  - Loads UI from `vacuum_system_gui.ui`
  - Manages Arduino serial connection
  - Coordinates safety system with sensor inputs
  - Handles all user interactions and GUI updates
  - Runs automated procedures safely in background threads

`main.py` - Application Entry Point

- **Purpose**: Launcher script for the GUI application
- **Functionality**: Initializes QApplication and runs the main window

`config.py` - Configuration System

- **Purpose**: Loads and manages system configuration
- **Loads From**: `sput.yml` runtime configuration file
- **Provides**: Relay mappings, pin assignments, pressure thresholds, scaling factors

Hardware Communication

`arduino_controller.py` - Arduino Relay Interface

- **Purpose**: Serial communication with Arduino Mega 2560 microcontroller
- **Key Features**:
  - Thread-safe relay control via serial protocol
  - Automatic port detection and connection management
  - Relay state tracking (23 relays total)
  - Digital input reading (4 sensors: Door, Water, Rod, Spare)
  - Analog input reading (4 pressure gauges and turbo speed)
  - Automatic reconnection and connection persistence
- **Hardware Target**: Arduino Mega 2560 R3 with relay driver boards

- **Communication**: 9600 baud serial protocol with command queue

Safety System

`safety/safety_controller.py` - Safety Interlock Logic

- **Purpose**: Central safety condition evaluator and system state manager
- **Key Features**:
  - Evaluates complex YAML-based safety rules with OR/AND logic
  - Automatic system state detection (e.g., "vented", "rough pump", "turbo pump", "sputter")
  - Button enable/disable checking before user actions
  - Pressure threshold monitoring
  - Relay state validation
  - Confirmation dialogs for risky operations
- **Responsibilities**:
  - Prevents unsafe operations (e.g., can't vent while sputtering)
  - Determines next valid operations based on current system state
  - Tracks active procedures to allow safe overrides
  - Synchronizes with relay controller state

`safety/safety_conditions.yml` - Safety Rules Configuration

- **Purpose**: Defines all safety interlocks and button conditions
- **Contains**:
  - Emergency stop conditions
  - Button enable/disable rules
  - Pressure thresholds for system states
  - Relay dependency checks
  - Confirmation requirements for operations
- **Format**: YAML with conditional logic (pressure > X, relay_state, digital_input checks)

Automated Procedures

`auto_procedures.py` - Vacuum Procedure Library

- **Purpose**: Implements automated sequences for vacuum system operations
- **Main Procedures**:
  - `pump_down()` - Multi-stage pump sequence (rough → medium → high vacuum)
  - `vent_system()` - Safe venting with interlock checks
  - `load_unload()` - Load-lock chamber management
  - `sputter()` - Sputtering mode with ion gauge and gas flow
- **Features**:
  - Real-time pressure monitoring and feedback
  - Automatic stage transitions

    - Safety interlock verification at each step
    - User cancellation support
    - Waits for physical sensors (door closure, etc.)
    - Unicode icons for terminal feedback
- **Responsible For**:
    - Coordinating complex multi-relay sequences
    - Polling sensor states during operations
    - Waiting for user actions (e.g., close door, load sample)
    - Error recovery and safety violations

Security System

`security/password_manager.py` - Authentication

- **Purpose**: Secure user authentication and role management
- **Features**:
    - Password hashing (bcrypt)
    - Multi-level access control (Admin, Operator, Technician)
    - Session management
    - Failed login attempt tracking

`security/reset_passwords.py` - Admin Tools

- **Purpose**: Password reset and account management utilities
- **Functionality**: Emergency admin password reset for system recovery

Gas Flow Control

`gas_control/controller.py` - Mass Flow Controller (MFC) Driver

- **Purpose**: Interface with Alicat APEX mass flow controllers
- **Features**:
    - Multi-device support (independent gas lines)
    - Real-time gas flow monitoring
    - Setpoint control and ramping
    - Thread-safe command queue
    - Data logging and plotting
    - Integration with safety system
- **Communications**: Serial interface (typically RS-232)

`gas_control/subprocess_controller.py` - Alternative MFC Implementation

- **Purpose**: Subprocess-based MFC control variant
- **Use Case**: When Python threading conflicts with serial communication

`gas_control/recipes.py` - Gas Flow Presets

- **Purpose**: Pre-configured gas flow recipes for different materials/processes
- **Contains**: Named recipes with setpoints for each gas line

`gas_control/config.yml` - MFC Configuration

- **Purpose**: MFC serial ports, gas types, and scaling parameters
- **Defines**: Device addresses, port mappings, gas identities

`gas_control/safety_integration.py` - MFC Safety Interlocks

- **Purpose**: Gas safety checks and procedure integration
- **Ensures**: Gas flows only during appropriate system states

User Interface Widgets

`widgets/indicators.py` - Status Indicators

- **Purpose**: Visual feedback elements for system state
- **Displays**: Relay status, sensor states, alarm conditions

`widgets/mfc_dialog.py` - Gas Flow Control Dialog

- **Purpose**: UI for setting mass flow controller setpoints
- **Features**: Graphical setpoint input, real-time feedback display

`widgets/mode_dialog.py` - System Mode Selector

- **Purpose**: Switch between Normal, Manual, and Override modes
- **Safety**: Enforces mode selection interlocks

`widgets/plotter_widget.py` - Real-time Data Visualization

- **Purpose**: Live pressure and parameter trending
- **Displays**: Time-series graphs of all analog inputs

`widgets/password_setup_dialog.py` - Initial Password Configuration

- **Purpose**: First-run password setup for system security

`widgets/other_dialogs.py` - Additional UI Components

- **Contains**: Miscellaneous dialog boxes and utility dialogs

Testing

`tests/` - Unit and Integration Tests

- **Contents**:
  - `test_arduino_relay.py` - Arduino communication tests
  - `serial_tests.py` - Serial port utilities
  - `test_mode_dialog.py` - UI component tests
- **Purpose**: Validation of critical components during development

Configuration Files

`sput.yml` - Runtime Configuration

- **Purpose**: Centralized configuration for all hardware mappings
- **Defines**:
  - Arduino pin assignments for relays, digital inputs, analog inputs
  - Pressure threshold values
  - Scaling factors for analog sensors
  - Baud rates and timeout values
- **Format**: YAML key-value pairs

`vacuum_system_gui.ui` - Qt Designer UI Layout

- **Purpose**: Visual layout of the main application window
- **Contains**: Button positions, indicator widgets, dialogs
- **Format**: Qt Designer XML (auto-generated, edit in Qt Creator for reliability)

Arduino Firmware

`relay_controller/relay_controller.ino` - Microcontroller Firmware

- **Target**: Arduino Mega 2560 R3
- **Purpose**: Hardware-level relay and sensor control
- **Responsibilities**:
  - Relay driver control (23 relays on pins 22-41)
  - Digital input reading (Door, Water, Rod, Spare)
  - Analog input reading (4 pressure gauges, turbo speed)
  - Serial communication protocol implementation
  - Hardware safety interlocks (e.g., relay interlock logic)

System Architecture

Timer-Based Functions for System State and Safety

The sputter control system uses multiple timer-based functions to continuously monitor hardware status and maintain safety conditions. These functions ensure real-time system state tracking and safety interlocks.

1. `refresh_status()` - Runs every 1000ms (1 second)

- **Purpose**: Relay status synchronization
- **Timer setup**: `self.status_timer.setInterval(1000)`
- **What it does**:
  - Gets current relay states from Arduino
  - Updates button visual states to match hardware
  - Handles special logic for ion gauge (based on analog voltage)
  - Detects connection loss

2. `refresh_inputs()` - Runs every 700ms (0.7 seconds)

- **Purpose**: Digital/Analog input polling and safety state updates
- **Timer setup**: `self.input_timer.setInterval(700)`
- **What it does**:
  - Reads digital inputs (Door, Water, Rod, Spare sensors)
  - Reads analog inputs (pressure gauges, ion gauge, turbo spin)
  - Updates visual indicators
  - **Calls `update_safety_state()` at the end**

3. `update_safety_state()` - Called by `refresh_inputs()` every 700ms

- **Purpose**: Core safety evaluation and system state determination
- **What it does**:
  - Passes current readings to SafetyController
  - Updates relay states
  - Calls `SafetyController.determine_system_state()` to auto-detect system state
  - Updates app's system status if state changed
  - Keeps SafetyController in sync with UI state

Key Points:

- **Most frequent**: `refresh_inputs()` at 700ms intervals is the primary safety monitoring function
- **Safety evaluation**: Happens every 700ms through `update_safety_state()`
- **System state detection**: The SafetyController automatically determines the best-matching system state based on current sensor readings and relay positions
- **State transitions**: When `determine_system_state()` detects a different state, it automatically updates the system status

Summary: `refresh_inputs()` (every 700ms) is the main function that triggers safety condition evaluation and system state determination through its call to `update_safety_state()`.

This manual is a work in progress. Additional sections will be added before production release.