Medical Laser Control System - Architecture Overview

Document Version: 1.0 **Date:** 2025-10-15 **Status:** Planning Phase

Executive Summary

This document outlines the architecture for a medical laser control system designed for clinical use. The system integrates laser control, linear actuator positioning, GPIO-based safety interlocks, camera-based alignment, and comprehensive patient/session tracking.

System Purpose

Control and monitor medical laser treatments with:

- Precise power and timing control
- Adjustable ring size via linear actuator
- Real-time safety monitoring via photodiode and hotspot smoothing device
- Camera-based alignment and focus verification
- Complete treatment recording and audit trail
- Longitudinal patient tracking across multiple sessions

Technology Stack

Core Technologies

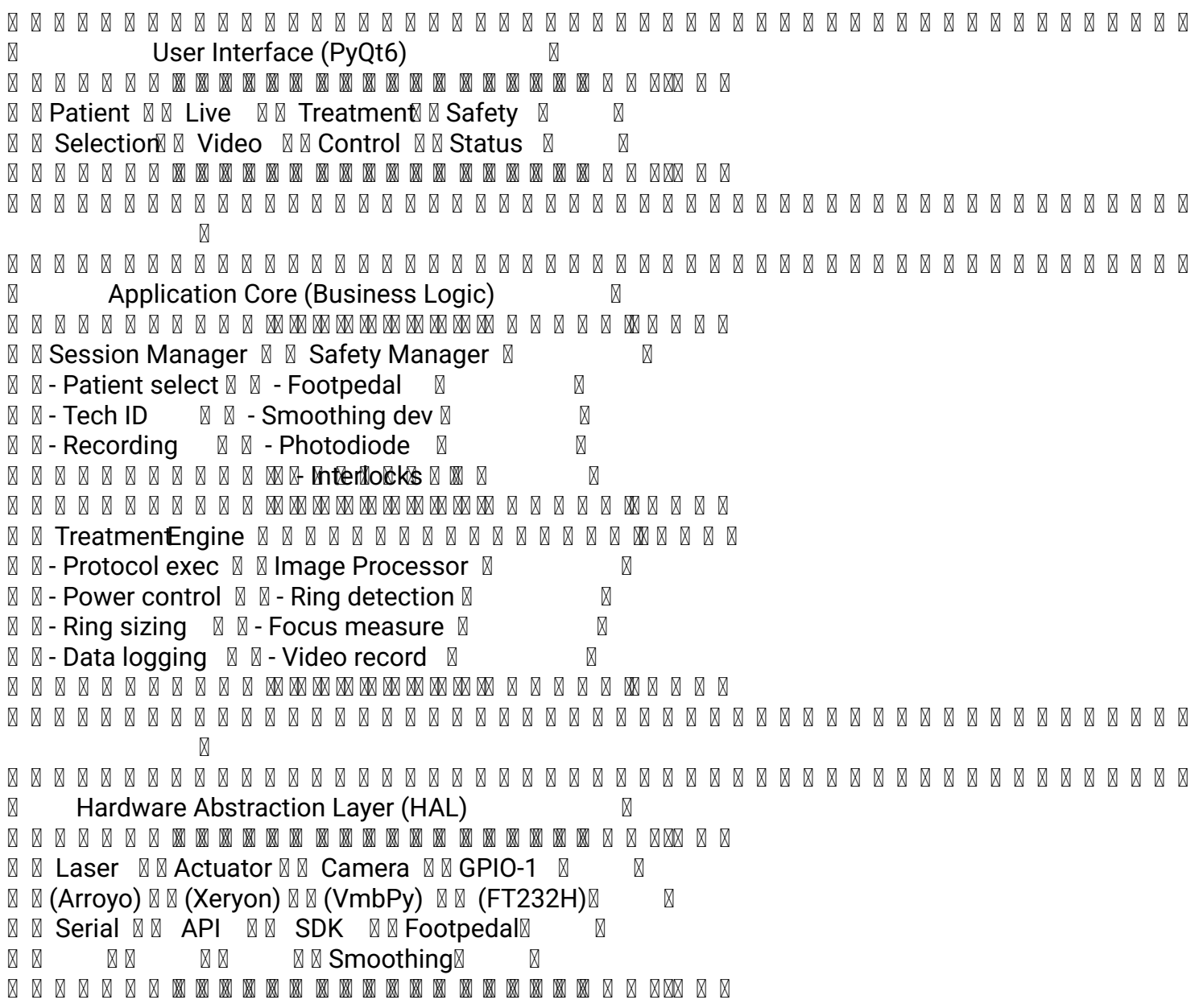
- **Language:** Python 3.10+
- **GUI Framework:** PyQt6 (modern, cross-platform, feature-rich)
- **OS Platform:** Windows 10 (Mini PC)
- **Database:** SQLite (local, single-user)

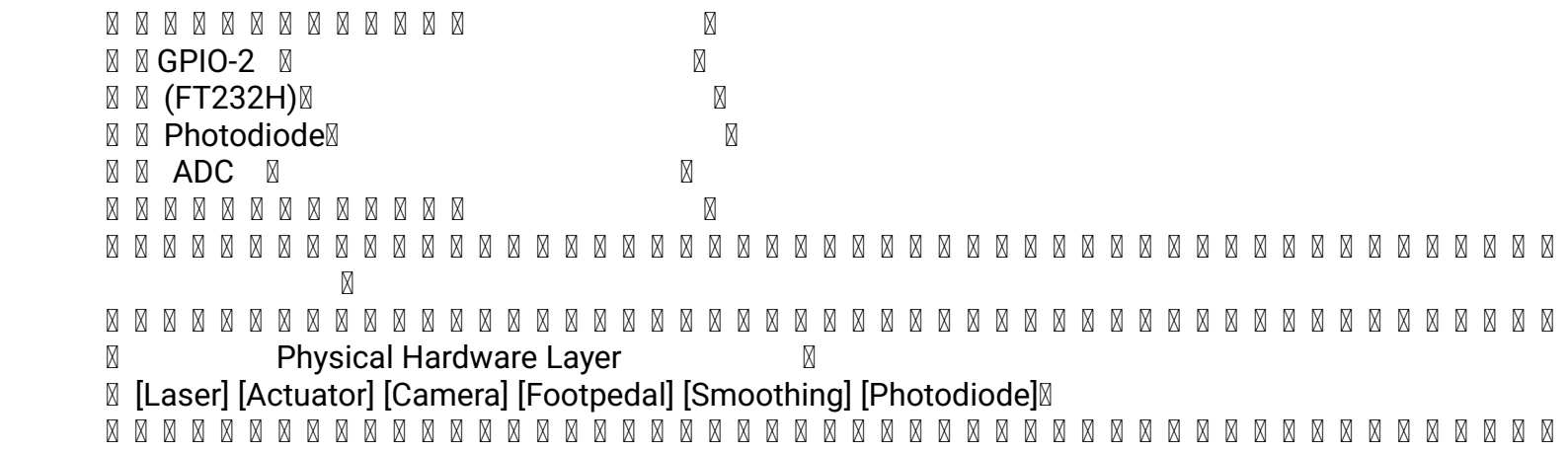
Key Libraries

```
# UI & Visualization
PyQt6                # Main GUI framework
pyqtgraph            # Real-time plotting (photodiode, power graphs)

# Image Processing & Computer Vision
opencv-python (cv2)     # Ring detection, focus measurement
numpy                # Image array operations
pillow               # Image saving/conversion
```

```
# Hardware Interfaces
pyserial            # Arroyo laser serial communication
adafruit-blinka     # FT232H GPIO/ADC support
board, busio        # Adafruit pin definitions
# Xeryon library      # Linear actuator control (existing)
# VmbPy SDK           # Allied Vision camera interface (existing)

# Database & Data Management
sqlite3             # Built-in database
sqlalchemy          # ORM for cleaner database code
alembic             # Database migrations

# Logging & Utilities
logging             # Event logging
python-dateutil     # Timestamp handling
pydantic            # Configuration validation
jsonschema          # Protocol validation
```

High-Level Architecture





Hardware Components

1. Laser Controller

- **Device:** Arroyo Instruments TEC Controller
- **Interface:** Serial communication (RS-232/USB)
- **Library:** Custom Python class for Arroyo serial protocol
- **Control:** Power settings, on/off, status queries

2. Linear Actuator

- **Device:** Xeryon linear stage
- **Interface:** External API library
- **Function:** Controls laser ring size by adjusting optical position
- **Control:** Position commands Ring diameter mapping

3. Camera System

- **SDK:** VmbPy (Allied Vision Vimba Python SDK)
- **Interface:** USB/GigE
- **Functions:** - Live video feed display - Manual focus/alignment by operator - Ring detection (circle finding) - Focus quality measurement - Treatment recording

4. GPIO Controller 1 - Safety Interlocks (FT232H)

- **Device:** Adafruit FT232H Breakout (USB-C)
- **Library:** adafruit-blinka, board, busio
- **Connections:** - **Footpedal Input:** Deadman switch - MUST be depressed to enable laser firing - **Hotspot Smoothing Device:** Signal monitoring - MUST be present and healthy for laser operation

5. GPIO Controller 2 - Photodiode Monitoring (FT232H)

- **Device:** Adafruit FT232H Breakout (USB-C)

- **Function:** ADC reading of photodiode voltage
- **Purpose:** Real-time measurement of laser output via optical pickoff
- **Monitoring:** Validates expected power output, detects anomalies

Safety Architecture

Critical Safety Interlocks (All Must Pass for Laser Operation)

1. **Footpedal Deadman Switch** (GPIO-1) - Type: Active-high requirement - Behavior: Laser can only fire while footpedal is DEPRESSED - Fail-safe: Releasing pedal immediately disables laser - Poll rate: 100Hz minimum
2. **Hotspot Smoothing Device** (GPIO-1) - Type: Signal health monitoring - Behavior: Device must output valid signal - Fail-safe: Loss of signal triggers immediate laser shutdown - Validation: Signal presence + value within acceptable range
3. **Photodiode Feedback** (GPIO-2 ADC) - Type: Output power verification - Behavior: Measured power must match commanded power - Fail-safe: Deviation beyond threshold triggers shutdown - Monitoring: Continuous during treatment
4. **Software E-stop** - Type: UI button + keyboard shortcut (e.g., ESC key) - Behavior: Immediate treatment halt - Priority: Highest - bypasses all queues
5. **Session Active** - Type: Logical interlock - Behavior: Laser cannot fire outside active treatment session - Purpose: Ensures all actions are logged and attributed
6. **Image Valid** - Type: Camera feed health check - Behavior: Valid image frame received within timeout - Purpose: Ensures alignment/monitoring capability

Safety State Machine

[SYSTEM_OFF] → [INITIALIZING] → [READY]
↓
[FAULT] ← ← ← ← ← ← ← [ARMED] → → (all interlocks pass)
↑ ↓
[SAFE_SHUTDOWN] [TREATING] → (footpedal depressed)
↓
[TREATMENT_COMPLETE]

Any interlock failure → Immediate transition to FAULT state → Safe shutdown

Session Workflow

Session Initialization

1. Application Launch
↓
2. Hardware Connection & Self-Test
↓
3. Tech ID Entry (required for all operations)
↓

4. Patient Selection Screen
   - Option A: Select Existing Patient (search by patient code)
     - Load patient history
   - Option B: Create New Patient
     - Generate patient code, enter demographics
5. Session Creation
   - Log: Patient ID, Tech ID, Start Time

Pre-Treatment Setup

1. Display Live Camera Feed
2. Operator Manual Actions (outside software control):
   - Adjust focus (physical optics)
   - Align laser ring to treatment site
   - Position patient
3. Software Assistance:
   - Real-time focus quality indicator
   - Ring detection overlay
   - Alignment guides
4. Operator confirms ready

Treatment Execution

1. Select Treatment Protocol
   - Load saved protocol, OR
   - Create/modify custom protocol
2. Safety Pre-checks
   - All hardware connected
   - Interlocks in valid state
   - Camera image valid
   - Session active
3. Operator initiates FIRE trigger
4. System transitions to ARMED state
5. Treatment Loop (while footpedal depressed):
   - Execute protocol step (power, ring size)
   - Monitor photodiode
   - Monitor smoothing device
   - Capture camera frames
   - Log all parameters (timestamp, power, position, voltage)
   - Check safety interlocks (every cycle)
6. Treatment completion or pedal release
7. Return to READY state

Session Recording

Continuous Recording During Treatment:

- Video: Full treatment video saved to session folder
- Event log: Every parameter change, every cycle
- Images: Periodic snapshots + key events
- Metadata: Timestamps, device states, operator actions

Data Storage Location:

```
data/
├── sessions/
│   ├── session_YYYYMMDD_HHMMSS_/
│   │   ├── video.avi
│   │   ├── events.json
│   │   ├── snapshots/
│   │   │   ├── frame_001.png
│   │   │   ├── frame_002.png
│   │   └── metadata.json
```

Session Closure

1. Operator ends treatment
   ↓
2. Save final recordings
   ↓
3. Add session notes
   ↓
4. Mark session as complete in database
   ↓
5. Update patient last_modified timestamp
   ↓
6. Return to Patient Selection (for next patient)

Treatment Protocol Engine

Protocol Structure

```
{
  "protocol_name": "Standard Treatment A",
  "description": "5W constant for 60s at 3mm ring",
  "steps": [
    {
      "step_number": 1,
      "duration_seconds": 60,
      "power_start_watts": 5.0,
      "power_end_watts": 5.0,  # Same as start = constant
      "ring_size_mm": 3.0,
      "ramp_type": "constant"  # or "linear", "logarithmic"
    }
```

```
  ]
}
```

Protocol Types

1. **Constant Power** - Fixed power for duration - Example: 5W for 60 seconds
2. **Linear Ramp** - Power increases/decreases linearly - Example: Ramp from 2W to 6W over 90 seconds
3. **Multi-Step** - Multiple sequential steps - Example: 3W for 30s, then 5W for 30s, then 3W for 30s
4. **Custom** - In-app protocol builder - Adjust any parameter on the fly

Real-Time Protocol Adjustment

- Operator can pause treatment
- Modify power/ring size during pause
- Changes logged as protocol deviation
- Resume with modified parameters

Image Processing Pipeline

Pipeline Overview

```
Camera Frame
    
[Preprocessing]
    Grayscale conversion
    Noise reduction
    Contrast enhancement
    
[Ring Detection]
    Hough Circle Transform
    Edge detection refinement
    Circle parameters (center, radius)
    
[Focus Measurement]
    Laplacian variance (sharpness)
    Gradient magnitude
    Focus score (0-100)
    
[Display Overlay]
    Detected ring outline
    Focus indicator
    Alignment guides
    
[Recording]
    Save annotated frames
```

Ring Detection Algorithm

```python
def detect_laser_ring(frame):
    """
    Detect circular laser ring in camera frame

    Returns:
        center (x, y), radius, confidence
    """
    # 1. Preprocess
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (9, 9), 2)

    # 2. Detect circles (Hough Transform)
    circles = cv2.HoughCircles(
        blurred,
        cv2.HOUGH_GRADIENT,
        dp=1,
        minDist=100,
        param1=50,
        param2=30,
        minRadius=20,
        maxRadius=200
    )

    # 3. Select best circle (brightest, most circular)
    # ... validation logic ...

    return center, radius, confidence
```

Focus Quality Measurement

```python
def calculate_focus_score(frame):
    """
    Calculate image sharpness/focus quality

    Returns:
        focus_score (0-100, higher = better focus)
    """
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Laplacian variance method
    laplacian = cv2.Laplacian(gray, cv2.CV_64F)
    variance = laplacian.var()

    # Normalize to 0-100 scale
    focus_score = min(100, variance / 10)  # Calibrate threshold

    return focus_score
```

Data Architecture

Database: SQLite

Location: data/laser_control.db

Key Tables:

1. `patients` - Patient records (anonymized)
2. `sessions` - Treatment sessions
3. `treatment_events` - Detailed event log (high frequency)
4. `protocols` - Saved treatment protocols
5. `calibrations` - Device calibration data
6. `safety_log` - Safety events and faults
7. `tech_users` - Technician/operator accounts

See: 02_database_schema.md for full schema

Event Logging Strategy

Two-tier logging:

1. **High-frequency data** (100Hz+): JSON files in session folder - Photodiode readings - Camera frame metadata - Real-time interlock states
2. **Event-based data**: SQLite database - Protocol steps - Power changes - Ring size adjustments - Safety triggers - User actions

Project Directory Structure

```
laser-control-system/
├── src/
│   ├── main.py                 # Application entry point
│   │
│   ├── config/
│   │   ├── settings.py         # User-configurable settings
│   │   ├── safety_limits.py    # Hard-coded safety parameters
│   │   └── hardware_config.py  # Hardware connection parameters
│   │
│   ├── ui/
│   │   ├── main_window.py      # Main application window
│   │   ├── patient_selection.py # Patient selection/creation dialog
│   │   ├── treatment_control.py # Treatment control panel
│   │   ├── video_display.py    # Live camera feed widget
│   │   ├── protocol_builder.py # Protocol creation/editing UI
│   │   ├── safety_panel.py     # Safety status indicators
│   │   └── widgets/            # Reusable UI components
│   │
│   ├── core/
│   │   ├── session_manager.py  # Session lifecycle management
│   │   ├── treatment_engine.py # Protocol execution engine
│   │   ├── safety_manager.py   # Safety interlock orchestration
│   │   └── recording_manager.py # Video/data recording
```

```
│   │   └── calibration_manager.py   # Calibration routines
│   │
│   ├── hardware/
│   │   ├── base.py                  # Abstract hardware device class
│   │   ├── laser_controller.py      # Arroyo laser interface
│   │   ├── actuator_controller.py   # Xeryon actuator interface
│   │   ├── camera_controller.py     # VmbPy camera interface
│   │   ├── gpio_interlocks.py       # GPIO-1: Footpedal + Smoothing
│   │   ├── gpio_photodiode.py       # GPIO-2: Photodiode ADC
│   │   └── hardware_manager.py      # Unified hardware coordination
│   │
│   ├── image_processing/
│   │   ├── ring_detector.py         # Laser ring circle detection
│   │   ├── focus_analyzer.py        # Focus quality measurement
│   │   ├── video_recorder.py        # Video file writing
│   │   └── frame_processor.py       # Image preprocessing pipeline
│   │
│   ├── database/
│   │   ├── models.py                # SQLAlchemy ORM models
│   │   ├── db_manager.py            # Database operations
│   │   ├── session_logger.py        # High-frequency session logging
│   │   └── migrations/              # Alembic migration scripts
│   │
│   └── utils/
│       ├── logger.py                # Application logging setup
│       ├── validators.py            # Input validation functions
│       ├── exceptions.py            # Custom exception classes
│       └── constants.py             # System-wide constants
│
├── data/
│   ├── laser_control.db             # SQLite database
│   └── sessions/                    # Per-session data folders
│       └── session_001/
│           ├── video.avi
│           ├── events.json
│           ├── photodiode_log.csv
│           ├── snapshots/
│           └── metadata.json
│   └── logs/                        # Application logs
│       ├── app_YYYYMMDD.log
│       └── errors_YYYYMMDD.log
│
├── docs/
│   └── architecture/
│       ├── 01_system_overview.md    # This file
│       ├── 02_database_schema.md
│       ├── 03_safety_system.md
│       ├── 04_treatment_protocols.md
│       └── 05_image_processing.md
│   ├── user_manual.md
│   └── installation.md
│
└── tests/
    └── test_hardware/
```

```
├── test_core/
├── test_safety/
└── test_integration/
│
├── requirements.txt
├── setup.py
└── README.md
```

Development Phases

Phase 1: Foundation (Hardware + Safety)

- [ ] Hardware abstraction layer for all devices
- [ ] Safety interlock system
- [ ] Basic GUI shell (PyQt6)
- [ ] Database schema and basic CRUD operations

Phase 2: Core Treatment Features

- [ ] Patient selection and session management
- [ ] Treatment protocol engine
- [ ] Manual treatment control (constant power)
- [ ] Basic event logging

Phase 3: Advanced Features

- [ ] Ring detection and focus measurement
- [ ] Video recording
- [ ] Protocol builder UI
- [ ] Advanced ramping protocols

Phase 4: Polish & Validation

- [ ] Comprehensive testing
- [ ] User manual
- [ ] Calibration procedures
- [ ] Performance optimization

Key Design Principles

1. **Safety First**: Multiple redundant interlocks, fail-safe design
2. **Audit Trail**: Every action logged, immutable records
3. **User Workflow**: Match clinical workflow, minimize clicks
4. **Hardware Abstraction**: Easy to swap/upgrade devices
5. **Modularity**: Loosely coupled components
6. **Testability**: Unit tests for critical paths
7. **Documentation**: Code + user docs maintained together

Next Documentation Files

1. `02_database_schema.md` - Complete SQL schema with indexes and constraints
2. `03_safety_system.md` - Detailed safety architecture and fault handling
3. `04_treatment_protocols.md` - Protocol format, execution engine, validation
4. `05_image_processing.md` - Computer vision algorithms and calibration

Document Owner: System Architect **Last Updated:** 2025-10-15 **Review Frequency:** Weekly during development