Product Requirements Document (PRD)

TOSCA Laser Control System

Document Number: TOSCA-PRD-001 **Version:** 0.9.11-alpha **Date:** 2025-10-30 **Status:** Development - Describes Implemented Functionality

---

Document Purpose

This Product Requirements Document describes WHAT the TOSCA Laser Control System does from a user perspective. Every requirement listed here is implemented in version 0.9.11-alpha and verifiable in the source code.

For technical details about HOW each feature is implemented, see TECHNICAL_SPECIFICATION.md.

---

1. System Overview

1.1 What TOSCA Does

TOSCA is a laser control system that allows operators to:

- Connect to and control laser hardware, actuators, and cameras
- Create and manage subject records
- Start and track treatment sessions
- Build and execute automated treatment protocols
- Monitor safety interlocks in real-time
- Capture images and record video during treatments
- Log all system events for audit purposes

Source: Observed functionality across src/ui/ and src/core/ modules

1.2 User Interface Structure

The system provides a tabbed interface with four main sections:

1. **Hardware & Diagnostics Tab** - Hardware connection and testing
2. **Treatment Setup Tab** - Subject selection and session initiation
3. **Treatment Control Tab** - Protocol execution and monitoring
4. **System Diagnostics Tab** - Safety status and event logs

Source: src/ui/main_window.py:120-206

1.3 Global Controls

Available on all screens:

- **Emergency Stop (E-Stop) Button** - Red button in toolbar, stops all operations
- **Safety Status Indicator** - Shows SAFE/UNSAFE/EMERGENCY_STOP state
- **Connection Status** - Shows which hardware devices are connected
- **Developer Mode Toggle** - Accessible via menu (when configured)

Source: src/ui/main_window.py:262-331 (toolbar implementation)

---

2. Hardware Management

2.1 Camera System

What the user can do:

- **Connect/Disconnect** camera via button
- **View live video** at 30 FPS
- **Adjust exposure time** using slider (50μs to 100ms range)
- **Adjust gain** using slider (0dB to 24dB range)
- **Capture still images** - saves PNG with timestamp
- **Record video** - toggle recording, saves MP4 with timestamp
- **View frame rate** - displays current FPS
- **See connection status** - visual indicator (green=connected, gray=disconnected)

Source: src/ui/widgets/camera_widget.py, src/ui/widgets/camera_hardware_panel.py

2.2 Laser System

What the user can do:

- **Connect/Disconnect** laser controller
- **Set laser power** - slider or text input (0.0 to 10.0W in 0.1W increments)
- **Enable/Disable laser output** - checkbox control
- **View actual laser power** - real-time display in mW
- **View laser status** - connection indicator and output state
- **Control aiming laser** - separate on/off toggle for alignment

Source: src/ui/widgets/laser_widget.py, src/hardware/laser_controller.py:59-64

2.3 TEC Temperature Controller

What the user can do:

- **Connect/Disconnect** TEC controller
- **Set target temperature** - input field and slider
- **Enable/Disable TEC** - checkbox control
- **View current temperature** - real-time display
- **Monitor TEC status** - connection and operational state

Source: src/ui/widgets/tec_widget.py, src/hardware/tec_controller.py

2.4 Linear Actuator

What the user can do:

- **Connect/Disconnect** actuator
- **Perform homing sequence** - button to establish zero position
- **Move to position** - text input for target position (0.0 to 20.0mm)
- **Move relative** - buttons for incremental movements
- **View current position** - real-time position display
- **Check homing status** - indicator shows if homed

Source: src/ui/widgets/actuator_connection_widget.py, src/hardware/actuator_controller.py

2.5 GPIO Safety System

What the user can do:

- **Connect/Disconnect** Arduino GPIO controller
- **Control smoothing motor** - on/off toggle
- **Adjust motor speed** - slider (0-153 PWM, 0-3.0V)
- **View vibration level** - real-time g-force display with color coding
- **View photodiode voltage** - analog reading (0-5V)

- **Monitor interlock status** - visual indicators for all safety interlocks

Source: src/ui/widgets/gpio_widget.py, src/hardware/gpio_controller.py

2.6 Hardware Diagnostics

What the user can do:

- **Test all hardware** - single button runs diagnostics on all systems
- **View test results** - dialog shows pass/fail for each component: - Camera (connection, streaming, FPS, model identification) - Actuator (connection, homing, positioning, range verification) - Laser (aiming and treatment laser operation) - GPIO (controller, motor, photodiode, interlock signals)
- **See summary statistics** - "X/4 systems PASSED" overall result

Source: src/ui/main_window.py:516-639 (hardware test implementation)

---

3. Subject and Session Management

3.1 Subject Records

What the user can do:

- **Search for subjects** - enter subject ID (format: P-YYYY-NNNN)
- **Create new subjects** - generates new subject ID automatically
- **View subject information** - displays subject details in text area
- **View session history** - button opens dialog with past sessions for subject

Source: src/ui/widgets/subject_widget.py:66-95, src/database/models.py

3.2 Treatment Sessions

What the user can do:

- **Start new session** - requires: - Subject selected - Technician ID entered - "Start Session" button press
- **View active session info** - displays session ID and start time
- **End session** - "End Session" button stops recording and closes session
- **View session list** - dialog shows all sessions with: - Session ID - Subject ID - Technician ID

- Start time - End time - Status

Source: src/ui/widgets/subject_widget.py:97-146, src/ui/widgets/view_sessions_dialog.py

3.3 Technician Identification

What the user can do:

- **Enter technician ID** - required field before starting session
- **System tracks operator** - all actions logged with technician ID

Note: Current version has no authentication - any ID can be entered

Source: src/ui/widgets/subject_widget.py:108, PROJECT_STATUS.md:202-207

---

4. Treatment Protocol Management

4.1 Protocol Builder

What the user can do:

- **Create new protocols** - build protocols from scratch
- **Define protocol metadata:** - Protocol name - Description - Creator name
- **Add actions to protocol:** - Set Laser Power (fixed power level) - Ramp Laser Power (gradual change over time) - Move Actuator (position change) - Wait (pause for duration) - Loop (repeat action sequence)
- **Configure action parameters:** - Power levels (watts) - Duration (seconds) - Position (millimeters) - Ramp types (linear, exponential, logarithmic) - Loop count (iterations)
- **Reorder actions** - move up/down in sequence
- **Remove actions** - delete from protocol
- **Set safety limits:** - Maximum laser power - Maximum position
- **Save protocol** - exports to JSON file
- **Load protocol** - imports from JSON file
- **Validate protocol** - checks for errors before saving

Source: src/ui/widgets/protocol_builder_widget.py, src/core/protocol.py:1-150

4.2 Protocol Selection

What the user can do:

- **Browse protocol library** - visual list of available protocols
- **View protocol preview** - see actions, safety limits, descriptions
- **Select protocol** - choose protocol for execution
- **See protocol details:** - Number of actions - Estimated duration - Safety limits - Creator information

Source: src/ui/widgets/protocol_selector_widget.py

4.3 Protocol Execution

What the user can do:

- **Load protocol** - select protocol to run
- **Start treatment** - begins automated protocol execution
- **Monitor progress** - real-time display shows: - Current action being executed - Elapsed time - Remaining time - Current power level - Current position
- **Pause treatment** - (future feature - not yet implemented)
- **Stop treatment** - terminates protocol execution
- **View execution log** - see completed actions and timestamps

Source: src/core/protocol_engine.py, src/ui/widgets/active_treatment_widget.py

---

5. Safety Monitoring

5.1 Safety Interlocks

What the user sees:

- **Visual interlock status** - each interlock shows GREEN (OK) or RED (FAILED): - GPIO Interlock (smoothing motor + vibration + photodiode) - Session Valid (active session exists) - Power Limit OK (within configured maximum)
- **Overall safety state** - large indicator shows: - SAFE (green) - all interlocks satisfied - UNSAFE (yellow) - one or more interlocks failed - EMERGENCY_STOP (red) - E-stop activated

Source: src/ui/widgets/interlocks_widget.py, src/core/safety.py:16-22

5.2 Emergency Stop

What the user can do:

- **Press E-Stop button** - large red button in toolbar
- **System response:** - Laser immediately disabled - System transitions to EMERGENCY_STOP state - All operations halt - System locked until manual reset
- **Reset E-Stop:** - Requires all safety interlocks to be satisfied - Requires explicit "Reset" button press - Confirmation dialog before resetting

Source: src/ui/main_window.py:272-295, src/core/safety.py:100-111

5.3 Safety Event Log

What the user can view:

- **Recent safety events** - scrolling list shows last 100 events
- **Event information:** - Timestamp - Event type - Severity (INFO, WARNING, ERROR, CRITICAL) - Description
- **Event filtering** - can filter by severity level
- **Event export** - (future feature)

Source: src/ui/widgets/safety_widget.py, src/core/event_logger.py

5.4 Safety Watchdog

What happens automatically:

- **Heartbeat monitoring** - system sends heartbeat every 500ms to Arduino
- **Timeout detection** - Arduino watchdog triggers if no heartbeat for 1000ms
- **Automatic shutdown** - laser disabled if watchdog timeout occurs
- **Visual feedback** - watchdog status displayed in safety widget

User cannot directly control watchdog - it operates automatically

Source: src/core/safety_watchdog.py, firmware/arduino_watchdog/arduino_watchdog.ino

---

6. Data Recording and Logging

6.1 Event Logging

What the system records automatically:

- **Hardware events:** - Device connections/disconnections - Hardware errors - Status changes
- **Safety events:** - Interlock state changes - Emergency stop activations - Watchdog alerts
- **Session events:** - Session start/end - Protocol execution start/pause/stop - Power changes - Position changes
- **User actions:** - Button presses - Setting changes - Manual overrides (if in developer mode)

Where logs are stored:

- SQLite database (`data/tosca.db` - `events` table)
- JSONL files (`data/logs/events_YYYYMMDD.jsonl`)

Source: src/core/event_logger.py

6.2 Image Capture

What the user can do:

- **Capture still image** - button press saves current camera frame
- **Automatic naming** - files named `capture_YYYYMMDD_HHMMSS.png`
- **Storage location** - saved to `data/images/` directory
- **Full resolution** - saves full camera resolution (1456×1088 pixels)
- **Event logging** - capture events logged automatically

Source: src/hardware/camera_controller.py (capture functionality), PROJECT_STATUS.md:85-90

6.3 Video Recording

What the user can do:

- **Start recording** - "Start Recording" button begins video capture
- **Stop recording** - button changes to "Stop Recording" to end capture
- **Automatic naming** - files named `recording_YYYYMMDD_HHMMSS.mp4`
- **Storage location** - saved to `data/videos/` directory
- **Video format** - MP4 with H.264 codec at 30 FPS
- **Full resolution** - records at full camera resolution (1456×1088 pixels)
- **Event logging** - recording start/stop events logged automatically

Note: Video recording causes frame rate drop (30 17 8 5 2 FPS) due to encoding overhead

Source: src/hardware/camera_controller.py (recording functionality),

PROJECT_STATUS.md:91-99

6.4 Session Data

What the system records per session:

- Session start/end times
- Subject ID
- Technician ID
- Protocol used (if any)
- All events during session
- Captured images (associated with session)
- Recorded videos (associated with session)
- Final session status (completed, aborted, error)

Source: src/core/session_manager.py, src/database/models.py

---

7. Developer Mode Features

7.1 Manual Hardware Control

What developer mode enables:

- **Bypass some safety checks** - for testing and calibration
- **Direct hardware access** - manual commands to devices
- **Override interlocks** - temporarily disable specific interlocks (with warnings)
- **View detailed diagnostics** - extended hardware information

How to enable:

- Set `enable_developer_mode: true` in `config.yaml`
- "Developer Mode" toggle appears in View menu
- Prominent warnings displayed when overrides are used

Important: All developer mode actions are logged with WARNING severity

Source: config.yaml:61

7.2 Manual Interlock Overrides

Available overrides (developer mode only):

- **GPIO Interlock Override** - force GPIO interlock status
- **Session Validity Override** - bypass session requirement
- **Power Limit Override** - bypass power limit check

Each override:

- Requires checkbox activation
- Shows danger warning banner
- Logs all override actions
- Automatically disabled when developer mode is off

Source: src/ui/widgets/manual_override_widget.py (referenced in PROJECT_STATUS.md:308-319)

7.3 Configuration Display

What the user can view:

- **All configuration values** from `config.yaml`: - COM ports and baud rates - Hardware timing settings - Safety parameters - GUI configuration
- **Read-only display** - cannot be edited through UI
- **Accessible from menu** - "View Configuration" option

Source: src/ui/widgets/config_display_widget.py, config.yaml

---

8. System Configuration

8.1 Configurable Settings

Settings in config.yaml:

Camera:

- GUI FPS target
- Hardware FPS
- FPS update interval

Actuator:

- COM port
- Baud rate
- Position update timer
- Homing check intervals

Laser:

- COM port
- Baud rate
- Timeout values
- Monitoring interval
- Power limits

GPIO:

- COM port
- Baud rate
- Motor PWM limits
- Vibration threshold
- Watchdog timeout

Safety:

- Watchdog enabled/disabled
- Heartbeat interval
- Emergency stop enabled
- Interlock requirements

GUI:

- Window title
- Default tab
- Auto-connect behavior
- Developer mode enable

Source: config.yaml:1-62

8.2 How Configuration Works

User workflow:

1. Edit `config.yaml` file with text editor
2. Save changes
3. Restart TOSCA application
4. New settings take effect

No GUI for configuration editing - must edit file manually

Source: Observed behavior, src/config/config_loader.py

---

9. System Status and Feedback

9.1 Status Bar

Always visible at bottom of window:

- **Safety Status** - Shows current safety state (SAFE/UNSAFE/EMERGENCY_STOP)
- **Session Status** - Shows if session is active
- **Hardware Status** - Shows connection status for: - Camera - Laser - Actuator - GPIO
- **Current Time** - System clock display

Source: src/ui/main_window.py:333-378 (status bar implementation)

9.2 Visual Indicators

Connection status:

- Section headers turn **GREEN** when device is connected
- Section headers are **GRAY** when device is disconnected
- Includes checkmark (✓) or X (✗) symbol

Safety indicators:

- **GREEN** - Safe to operate
- **YELLOW** - Warning state
- **RED** - Unsafe/fault condition

Button states:

- Buttons disable when actions are not available
- Visual feedback on button press
- Progress indicators during long operations

Source: src/ui/main_window.py (header styling), PROJECT_STATUS.md:368-376

9.3 Error Messages

When errors occur:

- **Pop-up dialogs** - for critical errors requiring user attention
- **Status bar messages** - for informational messages
- **Console logging** - detailed debug information (developer view)
- **Event log entries** - all errors recorded in database

Error information includes:

- Error description in plain language
- Affected component
- Suggested corrective action (when applicable)
- Timestamp

Source: Observed UI behavior, src/core/event_logger.py

---

10. Data Access and Export

10.1 Database Access

What data is stored:

- Subject records (demographics)
- Session records (treatment sessions)
- Event log (all system events)
- Technician records (operators)

Database location: data/tosca.db (SQLite file)

How to access:

- Direct SQL queries (using SQLite tools)
- No built-in export functionality in UI (current version)
- Developers can query database programmatically

Source: src/database/models.py, src/database/db_manager.py

10.2 File Locations

All data stored in data/ directory:

```
data/
├── tosca.db                    # SQLite database
├── logs/
│   └── events_YYYYMMDD.jsonl   # Daily event logs
├── images/
│   └── capture_YYYYMMDD_HHMMSS.png  # Captured images
├── videos/
│   └── recording_YYYYMMDD_HHMMSS.mp4 # Recorded videos
└── protocols/
    └── [protocol_name].json    # Protocol definitions
```

Source: Observed directory structure, PROJECT_STATUS.md:82-99

10.3 Protocol Files

Protocol JSON format:

- Human-readable JSON structure
- Can be edited with text editor
- Can be shared between TOSCA installations
- Can be version-controlled (git, etc.)

Protocol file includes:

- Protocol metadata (name, description, creator)
- List of actions with parameters
- Safety limits
- Timestamps

Source: src/core/protocol.py, protocol JSON file examples

---

11. System Requirements and Limitations

11.1 What the System Requires

Hardware:

- Windows 10/11 PC (64-bit)
- USB ports for camera and Arduino
- Serial ports (COM ports) for laser, actuator, TEC
- Minimum 8GB RAM recommended
- 256GB storage minimum (for video recording)

Software:

- Python 3.10 or higher
- All dependencies from `requirements.txt`
- Allied Vision VmbPy SDK (for camera)
- Arduino firmware uploaded to Arduino Uno

Source: README.md:250-262

11.2 Known Limitations

Performance:

- Video recording reduces frame rate (encoding overhead)
- UI can freeze for 2 seconds during GPIO connection
- Large number of events can slow database queries

Security:

- **NO DATABASE ENCRYPTION** - all data stored in plaintext
- **NO USER AUTHENTICATION** - any technician ID accepted
- **NOT SUITABLE FOR CLINICAL DATA** in current state

Hardware:

- Footpedal not yet integrated (pin assigned, software ready)
- Only supports Allied Vision cameras (VmbPy SDK)
- Fixed to specific Arroyo laser/TEC models

Software:

- No protocol pause/resume functionality
- No built-in data export tools
- No automated backups
- No network/cloud features

Source: PROJECT_STATUS.md:499-521, docs/architecture/01_system_overview.md:6-8

11.3 Not Implemented Features

Image processing:

- Ring detection algorithm (stub exists)
- Focus measurement (stub exists)
- Automated targeting

User management:

- Authentication system
- Role-based access control
- User permissions

Data management:

- Automated export to CSV/Excel
- Report generation
- Data visualization/charts

Source: README.md:207-210, PROJECT_STATUS.md

---

12. Verification Map

Each requirement in this document is implemented and can be verified:

Section	Verification Method	Source File/Test
Camera Control	Run application, test camera buttons	src/ui/widgets/camera_widget.py
Laser Control	Run application, test laser controls	src/ui/widgets/laser_widget.py
Subject	Create/search subjects	src/ui/widgets/subject_widget.py

Management		
Protocol Builder	Build and save protocol	src/ui/widgets/ protocol_builder_widget.py
Protocol Execution	Load and run protocol	src/core/protocol_engine.py
Safety Interlocks	Test interlock failures	tests/ test_realtime_safety_monitoring.py
Emergency Stop	Press E-Stop button	src/ui/main_window.py:272-295
Event Logging	Check database and JSONL files	src/core/event_logger.py
Image Capture	Capture image, check data/ images/	src/hardware/camera_controller.py
Video Recording	Record video, check data/ videos/	src/hardware/camera_controller.py
Hardware Diagnostics	Click "Test All Hardware" button	src/ui/main_window.py:516-639
