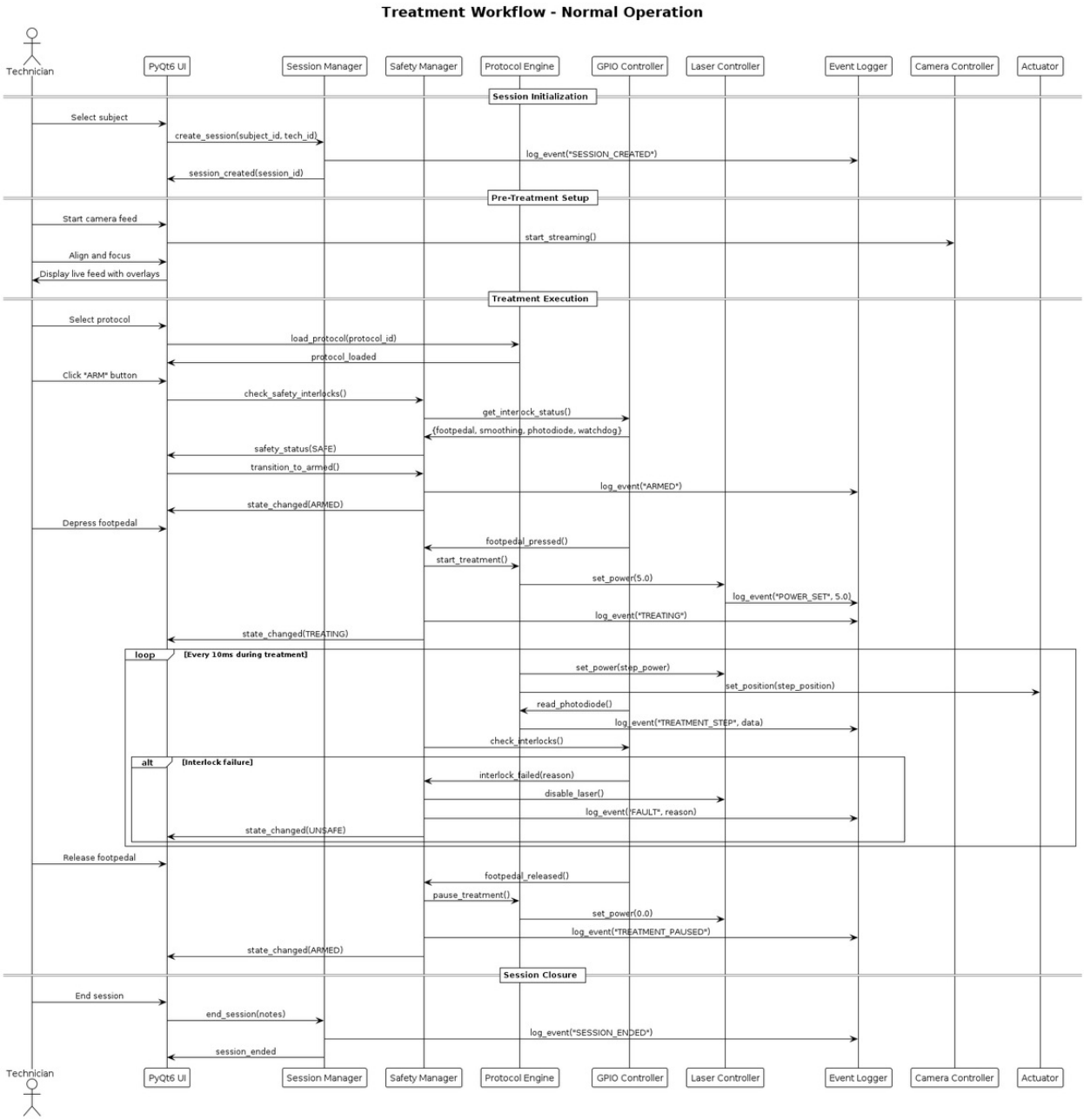
TOSCA Protocol Builder - Enhanced Action-Based Design

- Architecture Diagrams
  - Figure 1: TOSCA Treatment Workflow Sequence
- Overview
- Enhanced Protocol Data Model
  - Action Types
  - JSON Schema
  - Action Parameter Schemas
- Example Protocols
  - Simple Treatment
  - Complex with Loops
  - Ramp with Actuator Movement
- GUI Design
  - Protocol Builder Tab Layout
  - Action Editor Dialog
- Protocol Execution
  - Execution Engine (src/core/protocol_engine.py)
  - Execution Recording
- Safety Checks
- File Storage
- Implementation Phases

Architecture Diagrams

Figure 1: TOSCA Treatment Workflow Sequence



TOSCA Treatment Workflow Sequence

Last Updated: 2025-11-04

Document Version: 2.0 **Date:** 2025-10-22 **Supersedes:** Section of 04_treatment_protocols.md

Overview

The Protocol Builder provides a visual interface for creating action-based treatment protocols. Unlike the step-based model, this uses granular actions that can be sequenced, looped, and recorded.

Enhanced Protocol Data Model

Action Types

1. **SetLaserPower** - Instantly set laser to specific power
2. **RampLaserPower** - Ramp power from A to B over time
3. **MoveActuator** - Move to position at specified speed
4. **Wait** - Delay for specified duration
5. **Loop** - Repeat a block of actions N times or infinitely

JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "required": ["protocol_name", "version", "actions"],
  "properties": {
```

```
    "protocol_name": {"type": "string"},
    "version": {"type": "string"},
    "description": {"type": "string"},
    "created_date": {"type": "string", "format": "date-time"},
    "author": {"type": "string"},
    "safety_limits": {
      "type": "object",
      "properties": {
        "max_power_watts": {"type": "number"},
        "max_duration_seconds": {"type": "number"},
        "min_actuator_position_um": {"type": "number"},
        "max_actuator_position_um": {"type": "number"}
      }
    },
    "actions": {
      "type": "array",
      "items": {
        "type": "object",
        "required": ["action_id", "action_type"],
        "properties": {
          "action_id": {"type": "integer"},
          "action_type": {
            "type": "string",
            "enum": ["SetLaserPower", "RampLaserPower", "MoveActuator", "Wait", "Loop"]
          },
          "parameters": {"type": "object"},
          "notes": {"type": "string"}
        }
      }
    }
  }
}
```

Action Parameter Schemas

SetLaserPower

```
{
  "action_type": "SetLaserPower",
  "parameters": {
    "power_watts": 5.0
  }
}
```

RampLaserPower

```
{
  "action_type": "RampLaserPower",
  "parameters": {
    "start_power_watts": 2.0,
    "end_power_watts": 8.0,
    "duration_seconds": 30.0,
    "ramp_type": "linear"  // "linear", "logarithmic", "exponential"
  }
}
```

MoveActuator

```
{
  "action_type": "MoveActuator",
  "parameters": {
    "target_position_um": 2500,
    "speed_um_per_sec": 100
  }
}
```

Wait

```
{
  "action_type": "Wait",
  "parameters": {
    "duration_seconds": 5.0
  }
}
```

Loop

```
{
  "action_type": "Loop",
  "parameters": {
    "repeat_count": 3,  // or -1 for infinite
    "actions": [
      // Nested actions go here
    ]
  }
}
```

Example Protocols

Simple Treatment

```
{
  "protocol_name": "Simple_5W_60s",
  "version": "1.0.0",
  "description": "Set 5W for 60 seconds",
  "created_date": "2025-10-22T00:00:00Z",
  "author": "User",
  "safety_limits": {
    "max_power_watts": 10.0,
```

```
    "max_duration_seconds": 300
  },
  "actions": [
    {
      "action_id": 1,
      "action_type": "SetLaserPower",
      "parameters": {"power_watts": 5.0},
      "notes": "Set to treatment power"
    },
    {
      "action_id": 2,
      "action_type": "Wait",
      "parameters": {"duration_seconds": 60.0},
      "notes": "Treatment duration"
    },
    {
      "action_id": 3,
      "action_type": "SetLaserPower",
      "parameters": {"power_watts": 0.0},
      "notes": "Turn off laser"
    }
  ]
}
```

Complex with Loops

```
{
  "protocol_name": "Pulsed_Treatment_3x",
  "version": "1.0.0",
  "description": "3 pulses of 5W, 10s each, with 5s gaps",
  "created_date": "2025-10-22T00:00:00Z",
  "actions": [
    {
      "action_id": 1,
      "action_type": "MoveActuator",
      "parameters": {
        "target_position_um": 2000,
        "speed_um_per_sec": 50
      }
    },
    {
      "action_id": 2,
      "action_type": "Loop",
      "parameters": {
        "repeat_count": 3,
        "actions": [
          {
            "action_id": 3,
            "action_type": "SetLaserPower",
            "parameters": {"power_watts": 5.0}
          },
          {
            "action_id": 4,
            "action_type": "Wait",
            "parameters": {"duration_seconds": 10.0}
          },
          {
            "action_id": 5,
            "action_type": "SetLaserPower",
            "parameters": {"power_watts": 0.0}
          },
          {
            "action_id": 6,
            "action_type": "Wait",
            "parameters": {"duration_seconds": 5.0}
          }
        ]
      }
    }
  ]
}
```

Ramp with Actuator Movement

```
{
  "protocol_name": "Ramp_With_Ring_Adjust",
  "version": "1.0.0",
  "description": "Ramp power while adjusting ring size",
  "created_date": "2025-10-22T00:00:00Z",
  "actions": [
    {
      "action_id": 1,
      "action_type": "MoveActuator",
      "parameters": {
        "target_position_um": 3000,
        "speed_um_per_sec": 100
      },
      "notes": "Start at 3mm ring"
    },
    {
      "action_id": 2,
      "action_type": "RampLaserPower",
      "parameters": {
        "start_power_watts": 2.0,
        "end_power_watts": 8.0,
        "duration_seconds": 60.0,
        "ramp_type": "linear"
      },
      "notes": "Ramp power over 60 seconds"
    },
    {
      "action_id": 3,
      "action_type": "MoveActuator",
      "parameters": {
        "target_position_um": 2000,
```

```
        "speed_um_per_sec": 50
      },
      "notes": "Reduce to 2mm ring"
    },
    {
      "action_id": 4,
      "action_type": "Wait",
      "parameters": {"duration_seconds": 30.0}
    },
    {
      "action_id": 5,
      "action_type": "SetLaserPower",
      "parameters": {"power_watts": 0.0}
    }
  ]
}
```

GUI Design

Protocol Builder Tab Layout

```
1. **Protocol Builder                                        [?] [⚙]**
2. **Protocol** - [Unnamed Protocol         ] [Open] [Save] [Save As]
3. **Description** - [_____________________________________________]
4. **Action Sequence**
5. **#   Type                Parameters          [][][✎][✗]**
6. **1   SetLaserPower      5.0 W              [][][✎][✗]**
7. **2   Wait               10.0 s             [][][✎][✗]**
8. **3   MoveActuator       2500 µm @ 100 µ/s  [][][✎][✗]**
9. **4   Loop (3x)          ▼ 4 actions        [][][✎][✗]**
10. **RampLaserPower     28W, 30s**
11. **Wait               5s**
12. **SetLaserPower      0W**
13. **Wait               2s**
14. **[+ Add Action ▼]**
15. **• Set Laser Power**
16. **• Ramp Laser Power**
17. **• Move Actuator**
18. **• Wait**
19. **• Loop**
20. **Total Duration** - ~120s     Max Power: 8.0W
21. **[Execute]**
22. **Status** - Ready            Safety: [DONE] OK         [Execute & Rec]
```

Action Editor Dialog

When user clicks [✎] or [+ Add Action], show dialog:

```
  Edit Action: RampLaserPower


   Start Power:  [2.0    ] W
   End Power:    [8.0    ] W
   Duration:     [30.0   ] seconds
   Ramp Type:    [Linear ▼]
                  • Linear
                  • Logarithmic
                  • Exponential

   Notes: [Warm-up phase___________]

          [Preview Curve]

               [Cancel]  [OK]
```

Protocol Execution

Execution Engine (src/core/protocol_engine.py)

```python
class ProtocolEngine:
    """Execute action-based protocols with real-time control"""

    def __init__(self, laser_controller, actuator_controller):
        self.laser = laser_controller
        self.actuator = actuator_controller
        self.is_running = False
        self.current_action_id = None

    async def execute_protocol(self, protocol: dict, record: bool = False):
        """Execute protocol actions sequentially"""

    async def execute_action(self, action: dict):
        """Execute single action"""

    async def execute_loop(self, loop_action: dict):
        """Execute looped actions"""

    def pause(self):
        """Pause execution"""

    def resume(self):
        """Resume execution"""

    def stop(self):
        """Stop and reset"""
```

Execution Recording

Record to database table `protocol_executions`: - execution_id - protocol_name - start_time - end_time - subject_id (optional) - session_id - execution_log (JSON: action timestamps, actual values) - video_file_path

Safety Checks

Before execution: 1. Validate all parameters against safety_limits 2. Check footpedal status 3. Verify subject selected 4. Confirm actuator homed and in range 5. Verify camera feed active

During execution: - Monitor safety interlocks continuously - Log every action with timestamp - Allow emergency stop at any time - Validate each action before execution

File Storage

Protocols saved to: `data/protocols/` - Format: JSON - Naming: `{protocol_name}_v{version}.json` - Include metadata: creation date, author, modifications

Implementation Phases

1. **Phase 1: Data Model & Core Engine**
   - Define action classes
   - Implement protocol_engine.py
   - Add JSON validation
2. **Phase 2: Basic GUI**
   - Create protocol_builder_widget.py
   - Action list table
   - Simple add/remove/edit
3. **Phase 3: Advanced Features**
   - Loop support with nesting
   - Drag-and-drop reordering
   - Protocol templates library
4. **Phase 4: Execution & Recording**
   - Real-time execution
   - Progress visualization
   - Execution recording to database
5. **Phase 5: Polish**
   - Parameter validation UI
   - Curve preview for ramps
   - Import/export protocols

Next Steps: 1. Review and approve this design 2. Implement protocol data classes in src/core/protocol.py 3. Create protocol_builder_widget.py UI skeleton 4. Begin Phase 1 implementation