Unimplemented Features - Comprehensive Analysis

Priority Matrix

```
High Priority + Low Complexity  → Implement First ⭐⭐⭐
High Priority + High Complexity → Plan Carefully ⭐⭐
Low Priority + Low Complexity   → Nice to Have ⭐
Low Priority + High Complexity  → Future Maybe ○
```

Category 1: Hardware Devices (ASCOM Interfaces)

⭐⭐⭐ FilterWheel (IFilterWheelV2)

Status: Placeholder exists, ready to implement

Complexity: Low-Medium
Priority: High (essential for color imaging)
Use Case: Multi-band imaging (LRGB, narrowband)

Common Hardware:

- ZWO EFW (Electronic Filter Wheel)
- QHYCFW2/3
- Atik EFW2
- Manual filter wheels via serial

Implementation Needs:

- [ ] Hardware SDK integration (ZWO has Python SDK)
- [ ] Position control (move to filter 0-7)
- [ ] Filter naming system
- [ ] Focus offsets per filter
- [ ] Movement detection/completion
- [ ] Calibration routine

Effort: 2-3 days
Files to modify: `filterwheel.py`, `main.py`, `config.py`

Example SDK:

```python
import zwoasi_efw as efw
efw.get_num_devices()  # Detect wheels
wheel = efw.EFW(0)
wheel.set_position(2)  # Move to filter 2
```

⭐⭐⭐ Focuser (IFocuserV3)

Status: Placeholder exists, ready to implement

Complexity: Low-Medium
Priority: High (essential for maintaining focus)
Use Case: Auto-focus, temperature compensation

Common Hardware:

- Moonlite (USB/Serial)
- Pegasus Astro (USB)
- ZWO EAF (Electronic Auto Focuser)
- MyFocuserPro2
- Lakeside focusers

Implementation Needs:

- [ ] Absolute positioning (move to step N)
- [ ] Relative moves (+/- N steps)
- [ ] Movement speed control
- [ ] Backlash compensation
- [ ] Temperature compensation
- [ ] Temperature sensor reading
- [ ] Position limits/safety

Effort: 3-4 days
Files to modify: `focuser.py`, `main.py`, `config.py`

Key Methods:

- `move_to_position(steps)` - Absolute move
- `move_relative(steps)` - Relative move
- `halt()` - Emergency stop
- `get_temperature()` - Read temp sensor
- `set_temp_compensation(enabled)` - Enable temp comp

⭐⭐ Rotator (IRotatorV3)

Status: Not started

Complexity: Medium
Priority: Medium (needed for image composition)
Use Case: Field rotation, image framing

Common Hardware:

- Pegasus Astro Falcon Rotator
- Optec Pyxis
- PrimaLuceLab SESTO SENSO 2

Implementation Needs:

- [ ] Position control (degrees)
- [ ] Mechanical position (degrees)
- [ ] Sky position angle
- [ ] Reverse direction support
- [ ] Step size configuration
- [ ] Sync operation

Effort: 2-3 days
New files: `rotator.py` + routes in `main.py`

⭐ Switch (ISwitchV2)

Status: Not started

Complexity: Low-Medium
Priority: Low-Medium (convenience feature)
Use Case: Power control, dew heaters, flat panels

Common Hardware:

- Pegasus Astro Pocket Powerbox
- PrimaLuceLab EAGLE
- Custom Arduino-based controllers
- Lunatico AAG CloudWatcher

Implementation Needs:

- [ ] Multiple switch support (typically 4-8 switches)
- [ ] Switch naming
- [ ] Get/Set state (on/off)
- [ ] Analog value reading (voltage, current)
- [ ] PWM control (for dimmers)

Effort: 2-3 days
New files: `switch.py` + routes in `main.py`

Use Cases:

- Dew heater control
- Flat panel on/off
- Camera power cycling
- Mount power control
- Dust cover motor

⭐ ObservingConditions (IObservingConditionsV1)

Status: Not started

Complexity: Low-Medium
Priority: Low-Medium (safety/automation)
Use Case: Weather monitoring, safety checks

Common Hardware:

- Lunatico AAG CloudWatcher
- Boltwood Cloud Sensor
- Davis Weather Station
- PrimaLuceLab EAGLE weather

Implementation Needs:

- [ ] Temperature reading
- [ ] Humidity reading
- [ ] Dew point calculation
- [ ] Cloud coverage
- [ ] Wind speed/direction
- [ ] Rain detection
- [ ] Sky brightness
- [ ] Pressure reading

Effort: 2-3 days
New files: `observingconditions.py` + routes

○ SafetyMonitor (ISafetyMonitorV2)

Status: Not started

Complexity: Low
Priority: Low (can use ObservingConditions)
Use Case: Simple safe/unsafe signal

Implementation Needs:

- [ ] Safe/Unsafe property
- [ ] Connect/Disconnect

Effort: 1 day
Note: Often combined with ObservingConditions

○ Dome (IDomeV2)

Status: Not started

Complexity: High
Priority: Low (niche use case)
Use Case: Observatory dome control

Common Hardware:

- NexDome
- Sirius Observatory Dome
- Custom dome controllers

Implementation Needs:

- [ ] Open/Close shutter
- [ ] Rotate to azimuth
- [ ] Slaving to telescope
- [ ] Park position
- [ ] Home detection
- [ ] Safety interlocks

Effort: 5-7 days
New files: `dome.py` + routes

Category 2: Network & Discovery

⭐⭐⭐ Alpaca Discovery Protocol (UDP)

Status: Not implemented

Complexity: Medium
Priority: High (improves user experience)
Use Case: Auto-discovery by clients (N.I.N.A., PHD2)

Current Limitation: Users must manually enter IP address

Implementation Needs:

- [ ] UDP broadcast listener on port 32227
- [ ] Respond with JSON discovery packet
- [ ] Include all device information
- [ ] Handle IPv4 and IPv6

Effort: 1-2 days
Files to modify: `main.py`

Discovery Packet Format:

```json
{
  "AlpacaPort": 5555,
  "AlpacaDevices": [
    {"DeviceName": "OnStepX", "DeviceType": "Telescope", "DeviceNumber": 0},
    {"DeviceName": "ZWO Camera", "DeviceType": "Camera", "DeviceNumber": 0}
  ]
}
```

User Experience Impact: ⭐⭐⭐
N.I.N.A. will find your server automatically!

⭐⭐ mDNS/Bonjour Support

Status: Not implemented

Complexity: Low
Priority: Medium
Use Case: Service advertising on local network

Implementation:

```python
from zeroconf import ServiceInfo, Zeroconf
# Advertise as _alpaca._tcp.local.
```

Effort: 1 day

Category 3: Imaging Workflows

⭐⭐⭐ Auto-Focus Routine

Status: Not implemented

Complexity: Medium-High
Priority: High (critical for quality)
Use Case: Maintain perfect focus during session

Implementation Needs:

- [ ] V-curve focus algorithm
- [ ] Star detection (HFR calculation)
- [ ] Temperature-based triggering
- [ ] Filter-specific offsets
- [ ] Backlash compensation
- [ ] Coarse + fine focusing

Requires: Focuser device implementation
Effort: 3-5 days
New files: `autofocus.py`

Algorithm:

1. Take exposure
2. Measure star HFR (Half-Flux Radius)
3. Move focuser
4. Repeat to find minimum HFR
5. Fit curve and move to optimal position

⭐⭐⭐ Plate Solving Integration

Status: Not implemented

Complexity: Medium
Priority: High (accurate positioning)
Use Case: Precise goto, blind solving

Options:

- **ASTAP** (free, local)
- **Astrometry.net** (cloud-based)
- **PinPoint** (commercial)
- **ANSVR** (local server)

Implementation Needs:

- [ ] Image capture
- [ ] Call solver API/binary
- [ ] Parse RA/Dec result
- [ ] Sync telescope
- [ ] Iterative improvement

Effort: 3-4 days
New files: `platesolve.py`

Workflow:

1. Take image
2. Solve for RA/Dec
3. Compare to target
4. Sync or slew to correct
5. Repeat until within tolerance

⭐⭐ Dithering Support

Status: Not implemented

Complexity: Low-Medium
Priority: Medium (improves stacking)
Use Case: Reduce pattern noise in stacks

Implementation Needs:

- [ ] Random offset generation
- [ ] Pulse guide in RA/Dec
- [ ] Settling time after dither
- [ ] Pattern tracking (spiral, random)
- [ ] Maximum dither distance

Effort: 2 days
New files: `dithering.py`

Typical Pattern:

- After each exposure
- Move ±5-15 pixels randomly
- Wait for settling (3-5 seconds)
- Continue imaging

⭐⭐ Meridian Flip Automation

Status: Not implemented

Complexity: Medium
Priority: Medium (long sessions)
Use Case: Continue imaging past meridian

Implementation Needs:

- [ ] Detect approaching meridian
- [ ] Pause imaging
- [ ] Flip mount (slew to other side)
- [ ] Recalibrate guiding
- [ ] Refocus if needed
- [ ] Resume imaging

Effort: 2-3 days
Requires: Integration with imaging sequence

⭐⭐ Image Calibration Pipeline

Status: Not implemented

Complexity: Medium-High
Priority: Medium (quality improvement)
Use Case: Dark/Flat/Bias frame management

Implementation Needs:

- [ ] Automatic dark frame capture
- [ ] Flat frame sequences
- [ ] Bias frame library
- [ ] Temperature-matched darks
- [ ] Frame storage/organization
- [ ] Apply calibration to lights

Effort: 4-6 days
New files: `calibration.py`

⭐ Sequence Manager

Status: Not implemented

Complexity: High
Priority: Low (N.I.N.A. does this)
Use Case: Automated imaging sessions

Implementation Needs:

- [ ] Target list management
- [ ] Filter/exposure sequences
- [ ] Time-based scheduling
- [ ] Conditions monitoring
- [ ] Recovery from failures
- [ ] Session planning

Effort: 7-10 days
Note: N.I.N.A. already provides this excellently

Category 4: Advanced Camera Features

⭐⭐ Multiple Simultaneous Exposures

Status: Not implemented

Complexity: Medium
Priority: Medium (efficiency)
Use Case: Guide while imaging, dual-camera rigs

Current Limitation: Cameras take turns exposing

Implementation Needs:

- [ ] Thread-safe camera operations
- [ ] Independent state machines per camera
- [ ] Synchronized start option
- [ ] Resource locking

Effort: 2-3 days
Files to modify: `camera_*.py`, `main.py`

⭐ Fast Download Mode

Status: Partial (Base64 implemented)

Complexity: Low-Medium
Priority: Low (current is acceptable)
Use Case: Faster image transfer

Current Performance:

- Full frame: ~2-3 seconds download
- Base64: ~1.33x overhead

Potential Improvements:

- [ ] Compression (JPEG/PNG for preview)
- [ ] Progressive download
- [ ] Chunked transfer
- [ ] WebSocket streaming

Effort: 2-3 days

⭐ Sub-Frame Download

Status: Not implemented

Complexity: Low
Priority: Low
Use Case: Guiding with small ROI

Implementation:

- [ ] Download only ROI from sensor
- [ ] Reduce network traffic
- [ ] Faster for small guide stars

Effort: 1-2 days

○ Video/Streaming Mode

Status: Not implemented

Complexity: High
Priority: Low (not typical use case)
Use Case: Focusing, polar alignment

Implementation:

- [ ] Continuous exposure loop
- [ ] H.264 encoding
- [ ] RTSP/WebRTC streaming
- [ ] Low latency (<500ms)

Effort: 7-10 days
Note: Planetary imaging users need this

Category 5: Configuration & Management

⭐⭐⭐ Web Configuration UI

Status: Not implemented

Complexity: Medium-High
Priority: High (user experience)
Use Case: Easy configuration without SSH

Implementation Needs:

- [ ] Web interface (React/Vue/vanilla JS)
- [ ] Device status dashboard
- [ ] Configuration editor
- [ ] Live camera preview
- [ ] Mount control panel
- [ ] Log viewer
- [ ] System info (CPU, temp, disk)

Effort: 5-7 days
New files: `static/`, `templates/`

Pages:

- Dashboard (all device status)
- Telescope control
- Camera settings
- Filter/Focuser control
- System logs
- Configuration editor

⭐⭐ Configuration Persistence

Status: Not implemented (uses config.py)

Complexity: Low
Priority: Medium
Use Case: Save settings between sessions

Current Limitation: All settings reset on restart

Implementation:

- [ ] JSON/YAML config file
- [ ] Save on change
- [ ] Load on startup
- [ ] Default values
- [ ] Validation

Effort: 1-2 days
New files: `config.json`, config loader in `main.py`

⭐⭐ Enhanced Logging

Status: Basic (print statements)

Complexity: Low
Priority: Medium
Use Case: Debugging, session history

Implementation:

- [ ] Structured logging (JSON)
- [ ] Log levels (DEBUG, INFO, WARN, ERROR)
- [ ] Rotation policy
- [ ] Per-device logs
- [ ] Remote log viewing
- [ ] Log analysis tools

Effort: 2 days
Files to modify: All modules

⭐ Backup/Restore System

Status: Not implemented

Complexity: Low
Priority: Low
Use Case: Configuration backup

Implementation:

- [ ] Backup config files
- [ ] Export device settings
- [ ] Restore from backup
- [ ] Scheduled backups

Effort: 1-2 days

⭐ Performance Monitoring

Status: Not implemented

Complexity: Low-Medium
Priority: Low
Use Case: Optimization, diagnostics

Metrics to Track:

- [ ] Exposure timing
- [ ] Download speed
- [ ] CPU usage
- [ ] Memory usage
- [ ] Temperature
- [ ] Network throughput
- [ ] API response times

Effort: 2-3 days
Tools: Prometheus + Grafana

Category 6: Security & Access

⭐⭐ Authentication

Status: Not implemented (open access)

Complexity: Medium
Priority: Medium (if internet-facing)
Use Case: Secure remote access

Current Risk: Anyone on network can control equipment

Implementation Options:

- [ ] API key authentication
- [ ] Username/password (OAuth2)
- [ ] SSL/TLS encryption
- [ ] IP whitelist
- [ ] Rate limiting

Effort: 2-3 days
Libraries: Flask-Login, PyJWT

⭐ HTTPS/SSL Support

Status: HTTP only

Complexity: Low-Medium
Priority: Medium (if internet-facing)
Use Case: Encrypted communication

Implementation:

- [ ] Self-signed certificate
- [ ] Let's Encrypt integration
- [ ] Certificate management
- [ ] HTTPS redirect

Effort: 1-2 days

○ VPN/Reverse Tunnel

Status: Not implemented

Complexity: Medium
Priority: Low (network configuration)
Use Case: Remote observatory access

Options:

- WireGuard VPN
- ZeroTier
- Tailscale
- Cloudflare Tunnel

Effort: 1-2 days (setup)

Category 7: Reliability & Recovery

⭐⭐ Improved Slewing Detection

Status: Basic (OnStepX limitation)

Complexity: Medium
Priority: Medium
Use Case: Accurate slew completion

Current Issue: OnStepX doesn't provide reliable slewing status

Workarounds:

- [ ] Poll position repeatedly
- [ ] Detect when position stabilizes
- [ ] Timeout-based completion
- [ ] Use OnStepX advanced commands

Effort: 2-3 days
Files to modify: `telescope.py`

⭐⭐ Automatic Error Recovery

Status: Basic error handling only

Complexity: Medium-High
Priority: Medium
Use Case: Unattended operation

Implementation:

- [ ] Auto-reconnect on disconnect
- [ ] Retry failed operations
- [ ] Exposure retry on error
- [ ] Slew retry with offset
- [ ] Watchdog timer
- [ ] Error notifications

Effort: 3-4 days

⭐ Watchdog Service

Status: Not implemented

Complexity: Low
Priority: Low
Use Case: Automatic restart

Implementation:

```bash
# systemd already provides this
Restart=always
RestartSec=10
```

Effort: Already configured in systemd service

Category 8: Integration & Interoperability

⭐⭐ INDI Protocol Support

Status: Alpaca only

Complexity: Very High
Priority: Low-Medium (Linux astronomy users)
Use Case: Integration with Linux astronomy tools

INDI = **Instrument Neutral Distributed Interface**

Effort: 10-15 days
Alternative: Use INDI-to-Alpaca bridge (exists)

○ ASCOM COM Bridge

Status: Not needed (Alpaca is replacement)

Complexity: High
Priority: Very Low
Use Case: Windows COM compatibility

Note: ASCOM Remote already does this

⭐ Action() Method Implementations

Status: Not implemented

Complexity: Low per action
Priority: Low (optional features)
Use Case: Device-specific commands

Examples:

- Mount: `:U#` (high precision toggle)
- Camera: "SetFanSpeed", "SetReadoutMode"
- OnStepX specific: PEC control, alignment

Effort: 1 day per device
Files to modify: All device drivers

Summary Tables

Quick Wins (High Impact, Low Effort)

Feature	Impact	Effort	Priority
UDP Discovery	⭐⭐⭐	1-2 days	⭐⭐⭐
Configuration Persistence	⭐⭐	1-2 days	⭐⭐
Enhanced Logging	⭐⭐	2 days	⭐⭐
Action() Methods	⭐	1 day	⭐

Essential Devices (Required for Full Imaging)

Device	Impact	Effort	Priority
FilterWheel	⭐⭐⭐	2-3 days	⭐⭐⭐
Focuser	⭐⭐⭐	3-4 days	⭐⭐⭐
Rotator	⭐⭐	2-3 days	⭐⭐
Switch	⭐	2-3 days	⭐

Advanced Workflows (Pro Features)

Feature	Impact	Effort	Priority
Auto-Focus	⭐⭐⭐	3-5 days	⭐⭐⭐
Plate Solving	⭐⭐⭐	3-4 days	⭐⭐⭐
Web UI	⭐⭐⭐	5-7 days	⭐⭐⭐
Dithering	⭐⭐	2 days	⭐⭐
Meridian Flip	⭐⭐	2-3 days	⭐⭐

Total Effort Estimates

- **High Priority Features:** 20-30 days
- **Medium Priority Features:** 25-35 days
- **Low Priority Features:** 40-60 days
- **Everything:** 85-125 days (3-4 months full-time)

Recommended Implementation Order

Phase 1: Core Devices (2-3 weeks)

1. FilterWheel (ZWO EFW)
2. Focuser (Moonlite or ZWO EAF)
3. UDP Discovery Protocol
4. Configuration Persistence

Phase 2: Essential Workflows (2-3 weeks)

5. Auto-Focus Routine
6. Plate Solving Integration
7. Enhanced Logging
8. Improved Slewing Detection

Phase 3: User Experience (1-2 weeks)

9. Web Configuration UI
10. Authentication
11. Performance Monitoring

Phase 4: Advanced Features (3-4 weeks)

12. Dithering
13. Meridian Flip
14. Simultaneous Exposures
15. Rotator Support
16. Switch Device

Phase 5: Nice-to-Have (ongoing)

17. Image Calibration Pipeline
18. ObservingConditions
19. Dome Control
20. INDI Support

What Makes Sense for YOU?

Answer these questions:

1. **Do you use filters?** → Implement FilterWheel ⭐⭐⭐
2. **Does focus drift?** → Implement Focuser + Auto-Focus ⭐⭐⭐
3. **Do you want auto-discovery in N.I.N.A.?** → UDP Discovery ⭐⭐⭐
4. **Need precise goto?** → Plate Solving ⭐⭐⭐
5. **Want easy configuration?** → Web UI ⭐⭐⭐
6. **Remote access needed?** → Authentication + HTTPS ⭐⭐
7. **Use dew heaters?** → Switch Device ⭐⭐
8. **Long exposures past meridian?** → Meridian Flip ⭐⭐
9. **Have rotator?** → Rotator Interface ⭐⭐
10. **Do deep sky imaging?** → Dithering ⭐⭐

Most users need: **FilterWheel** + **Focuser** + **UDP Discovery** + **Web UI** = ~2-3 weeks of development.