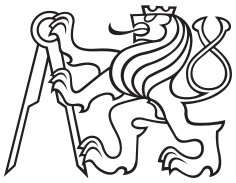
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra radioelektroniky

Řízení malého robotického dalekohledu

Tomáš Heinl

Vedoucí: doc. Ing. Stanislav Vítek, Ph.D.
Studijní program: Elektronika a komunikace
Květen 2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Heinl** Jméno: **Tomáš** Osobní číslo: **498827**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra radioelektroniky**

Studijní program: **Elektronika a komunikace**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Řízení malého robotického dalekohledu

Název bakalářské práce anglicky:

Small Robotic Telescope Control

Pokyny pro vypracování:

1) Prostudujte existující řešení řízení malých robotických dalekohledů
2) Seznamte se s rozhraními, která se pro řízení robotických dalekohledů používají, jako jsou např. INDI, ASCOM, Alpaca nebo Indigo.
3) Na základě studie navrhněte a implementujte vlastního klienta s využitím možností konceptu IoT
4) Ověřte funkčnost vlastního řešení a porovnejte s komerčně i nekomerčně dostupnými řešeními

Seznam doporučené literatury:

[1] HUSSER, Tim-Oliver, et al. pyobs--An observatory control system forrobotic telescopes. arXiv preprint arXiv:2203.12642, 2022.
[2] KUBÁNEK, Petr, et al. RTS2: a powerful robotic observatory manager. In: Advanced Software and Control for Astronomy. SPIE, 2006. p. 562-571.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Stanislav Vítek, Ph.D. katedra radioelektroniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **15.02.2023** Termín odevzdání bakalářské práce: **24.05.2024**

Platnost zadání bakalářské práce: **22.09.2024**

doc. Ing. Stanislav Vítek, Ph.D. podpis vedoucí(ho) práce	doc. Ing. Stanislav Vítek, Ph.D. podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Poděkování

Chtěl bych poděkovat svému vedoucímu doc. Ing. Stanislavu Vítkovi, Ph.D za vedení této práce a umožnění její realizace.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 23. 5. 2024

Abstrakt

Tato práce se zabývá rešerší existujících možností řízení malých robotických dalekohledů, seznámení s rozhraními, která se v této oblasti využívají a návrhem vlastního řešení s využitím možností konceptu IoT. Práce seznamuje s komerčně i nekomerčně dostupnými možnostmi řízení. Návrh řešení se skládá z vlastního INDI [1] klienta a Raspberry Pi 4 jako INDI serveru, které přes USB ovládá zrcadlovou kameru Nikon D7000 a equatoriální montáž.

Klíčová slova: Řídící systém dalekohledu, vzdálené dalekohledy, robotická astronomie, IoT kamery, DSLR astronomie, flexible image transport system, zpracování obrazu

Vedoucí: doc. Ing. Stanislav Vítek, Ph.D.

Abstract

This work deals with the research of existing solutions for controlling small robotic telescopes, familiarization with the interfaces that are used in this area and the implementation of my own solution using the possibilities of the IoT concept. The thesis introduces commercially and non-commercially available control solutions. The proposed solution consists of a custom INDI [1] client and a Raspberry Pi 4 as an INDI server, which controls the Nikon D7000 mirrorless camera and equatorial mount via USB.

Keywords: telescope control system, remote telescopes, robotic astronomy, IoT cameras, DSLR astronomy, flexible image transport system, image processing

Title translation: Remote control of small robotic telescope

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Tato práce se zabývá návrhem možného řešení vzdáleného řízení malého robotického dalekohledu pomocí vlastního klienta a Raspberry Pi jako INDI serveru. Navrhované řešení umožňuje automatizaci a zjednodušení práce amatérským astronomům a je také cenově příznivé.

V teoretickém úvodu jsem rozebral existující řešení a potřebné softwarové prostředky. Praktickou částí této práce bylo vytvořit vlastní klient, který bude ovládat zrcadlovou kameru a montáž připojenou k Raspberry Pi a umožní stáhnutí a zpracování obrazových dat.

Samotný klient umožňuje připojit se k jakémukoliv INDI serveru a ovládat jemu podružná zařízení. Chtěl jsem vytvořit co nejjednoduší program, který umožní nastavit všechny potřebné parametry kamery a umožní snímání s následným zpracováním.

Důležitým prvkem při realizaci byla kompatibilita jednotlivých klientů, kterou umožňuje INDI knihovna. Při běhu mého klienta je možné na stejná zařízení nahlížet i z jiných klientů, kteří mohou poskytovat další potřebná data.

Při realizaci jsem funkce testoval na virtuální CCD kameře, kterou poskytuje INDI knihovna. Následně jsem provedl i pozorování v terénu pomocí zrcadlové kamery Nikon D7000 a equatoriální montáže.

Kapitola 2
Teoretický rozbor

V této části práce jsem rozebral aktuální řešení řízení a využitá rozhraní komunikace mezi prostředky.

2.1 Komerční řešení

Díky rychlému rozvoji internetu a elektroniky se složité metody pozorování oblohy zjednodušily natolik, že umožňují i amatérským astronomům provádět velmi efektivní pozorování bez nákladných přístrojů dostupných pouze odborníkům. Na principu IoT vzniklo hned několik kompaktních řešení řízení domácích teleskopů za přijatelnou cenu.

2.1.1 StellarMate

Zařízení StellarMate na obrázku 2.1 založené na Raspberry Pi umožňuje propojení astronomického vybavení jako DSLR (Digital single-lens reflex) kamery, montáží, kopulí a dalších s internetem a následnou kontrolu nad jejich vlastnostmi a prostředky viz. [2].

Obrázek 2.1: Zařízení StellarMate [3]

StellarMate lze využívat bez dalšího zařízení připojením klávesnice a monitoru. Pro vzdálenou kontrolu je dostupné ovládání přes Ekos, observační a automatizační nástroj se zaměřením na astronomii pro Windows, MacOS

i Linux. Na mobilní telefony je dostupná aplikace. Podpora vybavení je zajištěna přes INDI knihovnu 2.3.2 . Velké množství připojitelných zařízení je již podporováno a další jsou přidávány komunitou. V základním nastavení StellarMate funguje jako hotspot, na který se dá připojit přes mobil nebo počítač. Lze ho připojit i na vlastní síť pomocí Wi-Fi nebo LAN a ovládat ho odkudkoliv.

KStars

Stellarmate je poskytován se softwarem KStars, který poskytuje přesné grafické simulace noční oblohy v jakémkoliv místě na Zemi, jakýkoliv den a čas. Katalog vesmírných těles obsahuje přes 100 milionů hvězd, tisíce objektů hlubokého vesmíru, všechny planety naší sluneční soustavy, komety, asteroidy, supernovy a satelity.

Další katalogy jako elektromagnetické spektrum můžou být zobrazeny přímo na mapu oblohy. Simulace podporuje ovládání rychlosti pro zobrazení jevů, které se odehrávají v dlouhých časových intervalech. Obsažený astrokalkulátor je možné využít k predikci konjunkcí.

KStars obsahuje také plánovač pozorování, kalendář a FOV (Field of View) editor, který umožňuje vypočítat viditelnou část oblohy se zadaným zařízením a zobrazit ji.

FITS Viewer tool může být otevřený separátně pro prohlížení pořízených FITS souborů. Podporuje monochromní, barevné a bayerované obrázky, které můžou být automaticky debayerované. Dále umožňuje zobrazení ekvatoriální sítě a při vybrání jakéhokoliv bodu na obrázku vyslat příkaz montáži, aby se k němu nasměrovala. Úprava obrázku je sice omezená, ale podporuje manipulaci s histogramem a identifikaci objektů na obrázku.

Plánovač pozorování slouží k vyhledání objektů, jsou aktuálně viditelné, následné simulaci pozorování a příkazu montáži k navádění na hledaný objekt.

Pořizování snímků lze s veškerým nastavením provádět přes Capture Tool, snímky můžou být plánovány do fronty a poté ukládány na Stellarmate nebo odeslány do počítače.

Ekos

Ekos je observační nástroj určený specificky pro astronomii a její automatizaci. Obsahuje mnoho užitečných nástrojů jako:

Smart Dark Librabry - ulehčuje práci s tmavými snímky. Pokaždé co je pořízen tmavý snímek, je označen a uložen do SDL, odkud může být využit později. Snímky můžou být nastaveny s expiračním datem a teplotní hranicí, kdy jsou ještě validní.

Sharp AutoFocus - s využitím half flux diameter kalkulací může StellarMate automaticky vybrat hvězdu na kterou bude ostřit, vyříznout ji, aby zrychlil vzdálenou komunikaci, a zaostřit co nejlépe. Automatické ostření lze opakovat před každou sekvencí snímků pro ideální ostrost nehledě na teplotu a pozici teleskopu.

Guider - umožňuje zaměření na specifickou hvězdu a její sledování po celou dobu pozorování. Je možné i použití externích sledovacích aplikací. Výsledné příkazy k pohybu jsou odeslány montáži.

Astrometry - oproti obvyklému star alignment pomocí několika hvězd využívá služby analýzy přes astrometry.net a výsledek využívá ke korekci pozice montáže. Existuje i možnost nahrát jakýkoliv obrázek oblohy a po analýze se montáž otočí přesně tak, aby zobrazovala lokaci, kde byl snímek pořízen.

Polar Assistant Tool - pomáhá s ustavením montáže. Ekos automaticky vyfotí a vyřeší množství snímků a vypočítá odchylku od ideálního stavu. Poté navede uživatele, jaké korekce aplikovat pro lepší výsledky.

Job Creator - vytváří požadavky na automatické sledování bez lidského zásahu. Pozorovatel nastaví požadovaný objekt, podmínky sledování a čas. Plánovač může automaticky otevřít střechu observatoře, navigovat k cíli, provést automatické ostření a pořídit snímky.

2.1.2 ASIAIR PLUS

ASIAIR na obrázku 2.2 je integrované řešení astronomického fotografování od ZWO. Oproti StellarMate ale oficiálně podporuje jen vlastní mobilní aplikaci. Mezi podporované nástroje patří: AutoFocus, PolarAlign, AutoGuiding, AutoRun, plánování a další [4].

Obrázek 2.2: Zařízení ASIAIR PLUS [4]

Přístroj podporuje většinu ZWO kamer a menší počet zrcadlovek. Umožňuje propojení s montáží a velkou výhodou jsou výstupní DC porty pro napájení vybavení. Ve výbavě je také integrované úložiště a anténa s podporou 5GHz Wi-Fi.

2.2 Vědecká řešení

Profesionální observatoře často vyvíjejí své vlastní řídicí systémy dalekohledů, aby splňovaly specifické provozní požadavky, které nejsou adekvátně řešeny komerčním softwarem. Systémy jako RTS2 [5] a Pyobs [6]. jsou příklady řešení vyvinutých se zaměřením na podporu široké škály zařízení a zajištění robustního provozu v různorodých a náročných podmínkách profesionálního astronomického výzkumu. Tyto systémy potřebují spravovat komplexní pole hardwaru včetně několika teleskopů, ovladačů kopulí, meteorologických stanic a kamer s vysokým rozlišením, přičemž všechny vyžadují přesnou synchronizaci a odolnost proti chybám.

Pyobs využívá Institut astrofyziky a geofyziky v Gottingenu k ovládání dvou MONET teleskopů a kamery SBIG STL-6303E, což je kamera v hodnotě 300 tis. Kč.

Vývoj těchto systémů je řízen potřebou vysoké spolehlivosti, flexibility při integraci nových technologií a schopnosti přizpůsobit operace pro specializované vědecké účely. Například RTS2 se používá k ovládání zařízení, která nejsou typicky součástí amatérských sestav, jako jsou pokročilé fotometry a složité montáže, což vysvětluje širší rozsah profesionálních astronomických projektů.

Tyto systémy jsou navrženy tak, aby prováděly automatická pozorování založená na složitých plánovacích algoritmech, zpracovávaly data v reálném čase a adaptivně reagovaly na měnící se pozorovací podmínky, které jsou klíčové pro maximalizaci vědeckého výstupu profesionálních observatoří.

Navíc jsou tyto systémy často vyvíjeny jako open-source projekty, které podporují komunitu uživatelů a vývojářů, kteří přispívají k neustálému zlepšování a aktualizacím. Tento přístup založený na spolupráci nejen pomáhá vylepšovat vlastnosti a stabilitu systémů, ale také podporuje standardizaci napříč různými observatořemi, což usnadňuje sdílené používání a výměnu dat mezi globální astronomickou komunitou.

Zatímco profesionální observatoře využívají robustních schopností vlastních systémů, komerční řešení jako StellarMate a ASIAIR PLUS nabízejí amatérským astronomům přístup k pokročilým astronomickým technologiím za přijatelnější cenu. Tyto komerční produkty využívají sílu technologií IoT a mini počítačů jako Raspberry Pi k poskytování uživatelsky přívětivých integrovaných řídicích systémů, které jsou výrazně levnější než profesionální řešení.

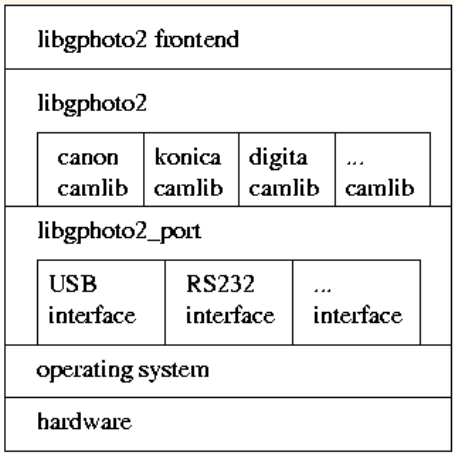
Komerční systémy, i když nejsou tak komplexní ve schopnostech jako jejich nekomerční protějšky, poskytují základní funkce, které pokrývají potřeby většiny amatérských pozorovatelů. Vývoj těchto systémů je podporován živou komunitou vývojářů a amatérských astronomů. Tato demokratizace technologie umožňuje amatérům užít si více sofistikovaný zážitek z pozorování, který byl kdysi možný pouze pro profesionální astronomy.

2.3 Softwarové prostředky

Podstatnou částí všech astronomických systémů je komunikace mezi počítačem a hardwarovým vybavením. Každý systém má své výhody a nedostatky. V této sekci jsem porovnal jednotlivá rozhraní využívána v komerčních i vědeckých řešeních.

2.3.1 Knihovna libgphoto2

libgphoto2 je knihovna umožňující ovládání mnoha digitálních kamer. Také podporuje získání a úpravu konfigurace, pokud to daná kamera umožňuje. Architektura této knihovny je znázorněna na obrázku 2.3.



Obrázek 2.3: Architektura libgphoto [7]

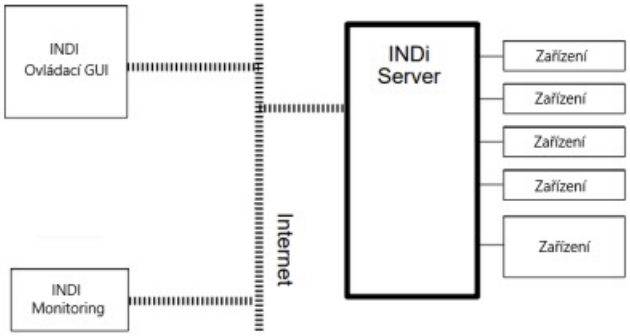
Knihovna abstrahuje komunikační porty a protokol kamery, aby umožnila kompletní modularitu. V základu podporuje většinu dnes dostupných DSLR a uživatelům je umožněno jednoduše naprogramovat podporu dalších. Pro ovládání astronomického vybavení je dostupný jednoduchý INDI server, který je založen na libgphoto2 a umožňuje ovládání kamery pomocí astronomických software jako Ekos a další.

2.3.2 INDI

INDI (Instrument Neutral Distributed Interface) [1] je protokol, který umožňuje kontrolu, získávání a výměnu dat mezi hardwarem a softwarovým frontendem. Základním konceptem v INDI je, že zařízení má schopnost se popsat kompatibilním klientům. Jakmile má uživatel klient, může ovládat kolik zařízení chce beze změn na straně klienta. Jelikož je INDI protokol založen na XML, může být jednoduše implementován. Diagram 2.4 popisuje nejjednodušší INDI konfiguraci – jeden klient připojený k jednomu zařízení.

Obrázek 2.4: Základní INDI konfigurace

INDI klient je proces, který se připojí k INDI zařízení, požádá zařízení o jeho sadu Properties (vlastností) a může vyžádat změnu daných vlastností. Finální kontrolu nad instrumenty má ovladač zařízení. Klient může reprezentovat GUI s widgety a příkazy, které umožňují měnit hodnoty. Také to může být proces, který nikdy nevyšle žádné změny a pouze zařízení monitoruje.



Obrázek 2.5: INDI konfigurace se zprostředkovávacím serverem

INDI klient a zařízení nemusí být v přímém kontaktu. Protokol je navržen pro vysílání mezi více klienty a zařízeními. Diagram 2.5 ukazuje využití serveru na propojení více zařízení s klienty. Každému klientu se server jeví jako zařízení a každému zařízení jako klient. Servery mohou mít implementované podmínky pro bezpečnost, prioritní přístup a další situace, které můžou nastat u více klientů.

2.3.3 ASCOM

ASCOM (Astronomy Common Object Model) se liší od INDI ve svém přístupu tím, že poskytuje standardizované rozhraní prostřednictvím architektury založené na COM primárně pro systémy Windows. Tento model nabízí robustní sadu předdefinovaných rozhraní pro různé typy zařízení, což usnadňuje vývoj interoperabilních komponent. Jeho kompatibilita pouze s Windows však omezuje jeho použitelnost v prostředích, která fungují na Unixových systémech běžně vyskytujících se v astronomických observatořích.

2.3.4 ALPACA

Alpaca rozšiřuje využití ASCOM tím, že převádí jeho rozhraní COM do koncových bodů RESTful HTTP, ke kterým lze přistupovat z jakéhokoli operačního systému. Tato adaptace umožňuje multiplatformní použití rozsáhlé knihovny zařízení ASCOM bez nutnosti přímé podpory COM, čímž překlenuje významnou mezeru v původním přístupu ASCOM.

Navzdory těmto pokrokům může potřeba síťového připojení pro emulaci chování COM způsobit latenci a problémy se sítí.

2.3.5 INDIGO

INDIGO vylepšuje protokol INDI zaměřením na modulární architekturu a pokročilé síťové možnosti. Podporuje distribuovanou architekturu zařízení, která umožňuje jedinému serveru bezproblémově ovládat více zařízení na různých platformách. INDIGO si zachovává filozofii INDI být jednoduchý a síťově transparentní, ale zároveň zlepšuje modularitu a údržbu systému.

2.3.6 RTS2

RTS2 (Remote Telescope System 2) nabízí komplexní řešení speciálně navržené pro autonomní observatoře. Integruje ovládání zařízení se správou dat a plánováním, to vše pod robustním rámcem C++ optimalizovaným pro výkon a spolehlivost. RTS2 se vyvinul z počáteční verze založené na Pythonu na stabilnější a výkonnější verzi v C++. Tento přechod řešil problémy související s prací v reálném čase a zpracováním chyb, které jsou zásadní pro operace bezobslužné observatoře.[5]

RTS2 umožňuje širokou kompatibilitu zařízení. Řeší problémy s vícevláknovými procesy a s opakovaným vstupem zařízení, které jsou časté u robotických dalekohledů. Použití abstraktní vrstvy zařízení umožňuje RTS2 adaptivně řídit různé hardwarové konfigurace, což je významný vývoj v jeho designu s cílem zlepšit odolnost proti chybám a provozní stabilitu.

2.3.7 Pyobs

Pyobs je moderní framework vyvinutý v Pythonu, navržený tak, aby využíval rozsáhlé softwarové knihovny, jako jsou NumPy, SciPy a Astropy. Jeho architektura je vysoce modulární, což uživatelům umožňuje rozsáhle přizpůsobit systém tak, aby vyhovoval specifickým potřebám observatoře [6].

Pyobs dobře spolupracuje s moderním softwarem a nabízí řešení, jako je asynchronní provoz a funkce dálkového ovládání, které jsou užitečné pro zpracování složitých pozorovacích sekvencí. Podporuje rozsáhlé úpravy a skriptování, které jsou podstatné pro vývoj adaptivních pozorovacích strategií a zvládání dynamických pozorovacích podmínek.

2.3.8 INDI Python wrapper

INDI protokol je implementován v INDI Core Library, která obsahuje INDI server, drivery, definice všech zařízení a knihovnu pro tvorbu klientů. Knihovna je napsána v jazyce C++, ale svou implementaci řídícího klienta jsem psal v Pythonu. Proto využívám knihovnu pyindi-client, která je z C++ knihovny generována automaticky pomocí SWIGu.

Nejjednodušší IndiClient v PyIndi [8] může vypadat následovně:

```
import PyIndi

class IndiClient(PyIndi.BaseClient):
    def __init__(self):
        super(IndiClient, self).__init__()

    def newDevice(self, dev):
        pass

    def removeDevice(self, dev):
        pass

    def newProperty(self, genericProperty):
        pass

    def updateProperty(self, genericProperty):
        pass

    def removeProperty(self, genericProperty):
        pass

    def newMessage(self, device, messageId):
        pass

    def serverConnected(self):
        pass

    def serverDisconnected(self, exitCode):
        pass

indiclient=IndiClient()
indiclient.setServer("localhost",7624)

indiclient.connectServer()
while (1):
    pass
```

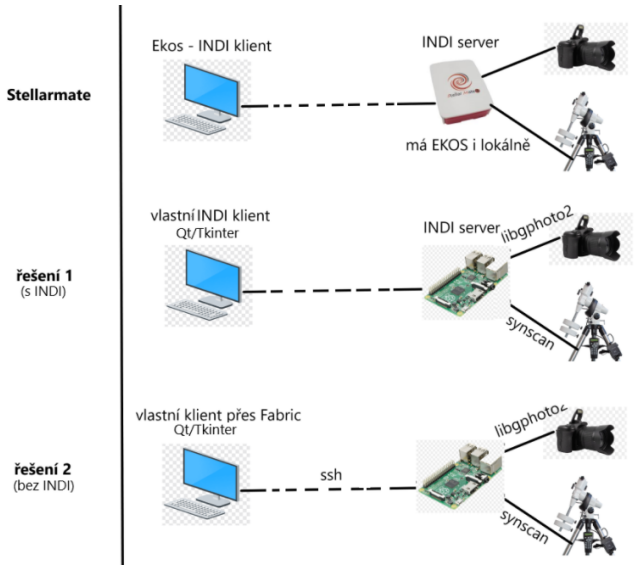
Kód 2.1: Základní IndiClient

V tomto kódu je definována třída IndiClient, která dědí z PyIndi.BaseClient. Jsou zde implementovány všechny virtuální funkce. Je zde možné nastavit chování klienta při všech situacích na serveru. Například `updateProperty()` definuje chování při získání nových informací z INDI serveru. Z této funkce lze poté získat i vyfocený snímek ve formě Blobu. INDI Blob je v PyIndi namapovaný jako bytearray.

Kapitola 3

Návrh vlastního řešení

V oblasti řízení robotických dalekohledů existuje mnoho možností pro řízení a automatizaci. Tyto řešení se mohou lišit v míře složitosti a kompatibility s jinými systémy. Na obrázku 3.1 je znázorněno zapojení řešení, která připadala v úvahu pro vlastní implementaci.



Obrázek 3.1: Uvažovaná řešení

Po pečlivém zvážení a hodnocení existujících řešení jsem usoudil, že nejlepším řešením je použití knihovny INDI, která nabízí jednoduchou implementaci a vysokou flexibilitu při řízení různých typů zařízení. Hlavní výhodu vidím v kompatibilitě s již existujícími klienty a zařízeními. Vlastní klient bude schopen ovládat jakýkoliv INDI server a taktéž lze tento server ovládat i jinými klienty.

Jak jsem zmínil v teoretickém úvodu, knihovna INDI je napsána v jazyce C++. Přesto, pokud chci vytvořit aplikaci pro řízení kamery a montáže, preferuji použití Pythonu s frameworkem PyQt. Tuto volbu jsem učinil z důvodu rozšířenosti a jednoduchosti Pythonu a také kvůli možnosti snadného využití existujících knihoven v oblasti astronomie.

3.1 PyQt rozhraní

Celý klient je implementován pomocí rámce PyQt, který poskytuje sadu vazeb Pythonu pro aplikační rámec Qt, což umožňuje vytvářet robustní a interaktivní grafická uživatelská rozhraní. Program je uspořádán do několika záložek, z nichž každá představuje jiný aspekt řídicího a monitorovacího systému dalekohledu. Každá karta obsahuje specifické widgety a rozvržení relevantní pro její funkce, jako je nastavení kamery, informace o montáži a údaje o prostředí.

Významnou vlastností programu je synchronizovaný stav připojení napříč všemi záložkami. Toho je dosaženo implementací sekce řízení sdíleného připojení. Každá karta obsahuje vstupní pole pro IP adresu a port, tlačítko pro připojení/odpojení a stavovou ikonu. Stav připojení je řízen prostřednictvím signálových slotů, kde se při každé změně stavu připojení vyšle vlastní signál `connection_status_changed`. Kód 3.1 ukazuje vytvoření těchto ovládacích prvků:

```
def create_connection_controls(self):
    connection_layout = QHBoxLayout()
    ip_input = QLineEdit(self)
    ip_input.setFixedWidth(100)
    port_input = QLineEdit(self)
    port_input.setFixedWidth(50)
    connect_button = QPushButton('Connect', self)
    status_label = QLabel('Disconnected', self)
    status_label.setStyleSheet("QLabel { color : red; }")

    connection_layout.addWidget(QLabel("IP:"))
    connection_layout.addWidget(ip_input)
    connection_layout.addWidget(QLabel("Port:"))
    connection_layout.addWidget(port_input)
    connection_layout.addWidget(connect_button)
    connection_layout.addWidget(status_label)

    connect_button.clicked.connect(lambda: self.
    toggle_connection(ip_input.text(), port_input.text(),
    connect_button, status_label))
    return connection_layout
```

Kód 3.1: Ukázka implementace rozhraní v PyQt

Stav připojení se aktualizuje na všech kartách navrženého rozhraní vysláním signálu `connection_status_changed`, který je připojen k funkci, která iteruje každou záložku a aktualizuje příslušné widgety. To zajišťuje, že jakákoli změna stavu připojení se projeví v celém rozhraní, zachová konzistenci a poskytne uživateli zpětnou vazbu v reálném čase.

Všechny funkce v klientu jsou určené pro Linux, protože jen na něm je PyIndi podporováno. Funkce mimo ovládání kamery a montáže jsou ale dostupné i na jiných operačních systémech pro pohodlnou analýzu dat.

3.2 Implementace funkcí

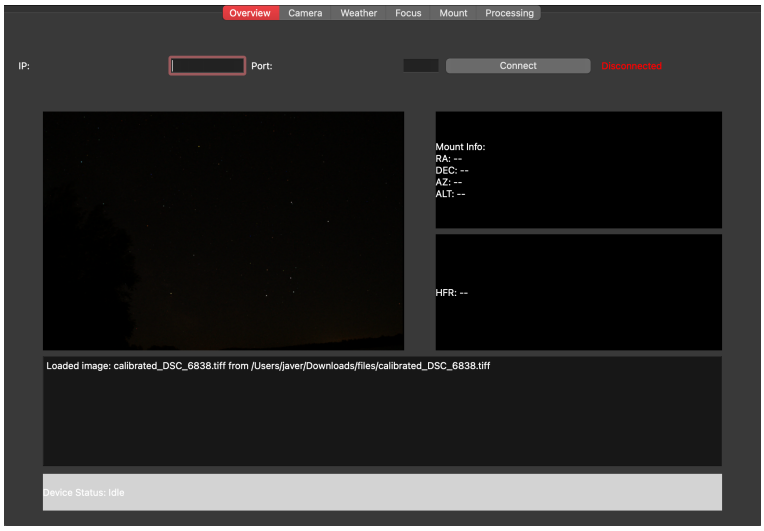
Vlastní klient pro řízení montáže a kamerou obsahuje několik panelů, které mají specifickou funkci při astrofotografii a získávání dat:

- **3.2.1** Celkový přehled
- **3.2.2** Pořizování snímků
- **3.2.3** Pozorovací podmínky
- **3.2.4** Zaostření
- **3.2.5** Ovládání montáže
- **3.2.6** Zpracování dat

Tyto funkce jsou dostupné v separátních panelech a uživatel může volit, kterou funkci chce v danou chvíli používat. Panely jsou navrženy tak, aby bylo možné co nejjednodušší a nejpřehlednější získávání dat a astrofotografování.

3.2.1 Celkový přehled

Karta Overview v klientu slouží jako centrum pro zobrazení základních informací a záznamu událostí. Je navržena tak, aby poskytovala komplexní přehled o stavu systému a posledních aktivitách. Jednou z výrazných funkcí karty Overview je její schopnost automaticky zobrazit nejnovější snímek ze složky programu, což zajišťuje, že uživatelé mají vždy rychlý přístup k nejnovějšímu snímku pořízenému kamerou.



Obrázek 3.2: Návrh rozhraní

K zobrazení tohoto nejnovějšího obrázku se používá widget zobrazení kamery, implementovaný jako QLabel. Automaticky se aktualizuje skenováním složky programu pro nejnovější soubor, načtením a zobrazením při prvním otevření karty nebo vždy po vytvoření nových souborů tímto programem. Kód 3.2 ukazuje, jak je aktualizován poslední snímek kamery:

```
def update_camera_view(self):
    program_folder = os.getcwd()  # Pouzije aktualni slozku
    image_files = [f for f in os.listdir(program_folder) if f.endswith(('.NEF', '.tiff', '.fits'))]

    if image_files:
        latest_image = max(image_files, key=lambda x: os.path.getctime(os.path.join(program_folder, x)))
        image_path = os.path.join(program_folder, latest_image)

        if image_path.lower().endswith('.nef'):
            with rawpy.imread(image_path) as raw:
                rgb_image = raw.postprocess(no_auto_bright=True, use_auto_wb=False, gamma=None)
            image = Image.fromarray(rgb_image)
            qt_image = ImageQt(image)
            pixmap = QPixmap.fromImage(qt_image)
        elif image_path.lower().endswith('.fits'):
            with fits.open(image_path) as hdul:
                data = hdul[0].data
                if data is not None:
                    data = np.clip(data, 1000, np.percentile(data,99.9))
                    data = (data - np.min(data)) / (np.max(data) - np.min(data)) * 255
                    data = data.astype(np.uint8)

                    image = Image.fromarray(data)
                    qt_image = ImageQt(image)
                    pixmap = QPixmap.fromImage(qt_image)
                else:
                    self.camera_view.setText("Failed to load image.")
                    return
        else:
            pixmap = QPixmap(image_path)

        if not pixmap.isNull():
            self.camera_view.setPixmap(pixmap.scaled(self.camera_view.size(), Qt.KeepAspectRatio))
        else:
            self.camera_view.setText("Failed to load image.")
    else:
        self.camera_view.setText("No image files found.")
```

Kód 3.2: Implementace zobrazení posledního souboru

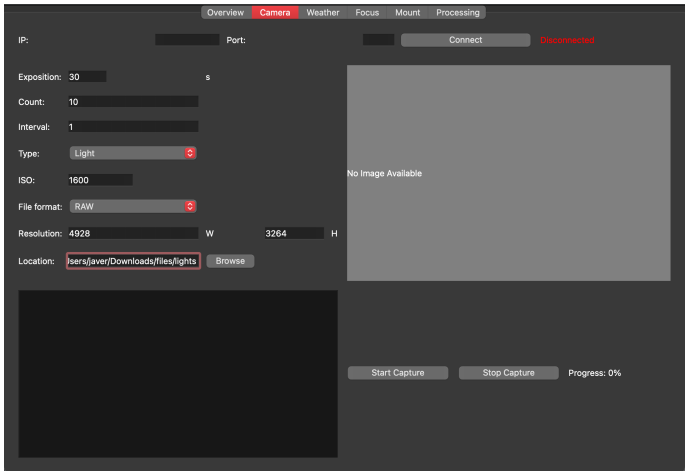
Klient může produkovat soubory různých formátů, proto je také v této sekci musí umožnit otevřít. Soubory typu .NEF a .fits potřebují před zobrazením další zpracování jelikož jsou ve formátu RAW. Více informací o zpracování dat je v kapitole 3.2.6

Mount info a HFR (Half Flux Radius) poskytují aktuální informace o poloze dalekohledu a stavu zaostření v reálném čase. Jsou taktéž implementovány jako widgety QLabel, inicializovány s výchozími hodnotami a aktualizovány, jakmile jsou k dispozici data. Informace o montáži ukazují rektascensi (RA), deklinaci (DEC), azimut (AZ) a výšku (ALT), zatímco HFR displej ukazuje ostrost hvězd na pořízených snímcích. Pro zobrazení HFR se nejdříve musí tato hodnota nechat vypočítat na panelu 3.2.4.

Karta Overview shromažďuje důležité informace do jediného uživatelsky přívětivého rozhraní. Sloučením záznamů událostí a zobrazením nejnovějšího obrázku poskytuje efektivní a komplexní přehled o provozním stavu systému.

3.2.2 Pořizování snímků

Tento panel slouží jako hlavní kontrolní centrum k pořizování obrazu. Umožňuje pořízení obrázku nebo sekvence obrázků.



Obrázek 3.3: Návrh rozhraní pro snímání

Nastavitelné parametry snímání jsou následující:

- Exposition: Nastavení doby expozice v sekundách
- Count: Počet snímků, které budou pořízeny
- Interval: Interval mezi pořízením jednotlivých snímků
- Type: Typ pořizovaného snímku (Light, Dark, Bias, Flat)
- ISO: Nastavení hodnoty ISO
- File format: Výběr formátu uložení: TIFF, JPEG, FITS
- Rosolution: Rozlišení kamery
- Location: Výběr kam soubory ukládat

Jakmile jsou všechny parametry nastaveny, uživatel může pořídit obrázek kliknutím na tlačítko Start Capture. Snímání jde vždy přerušit tlačítkem Stop Capture. Výsledné snímky se postupně zobrazí v klientu nebo je může uživatel otevřít jako soubor v externí aplikaci.

INDI ovládá připojená zařízení pomocí Properties. Každá Property znázorňuje jinou vlastnost připojeného zařízení. Properties můžou být Text, Number, Switch, Light nebo Blob. Jsou vázané přímo k zařízení a ovládá je ovladač zařízení. Standardní properties je možné vidět v [9].

INDI server získává od připojených zařízení jejich properties, aby je mohl využít klient. Klient může hodnoty properties získávat a měnit. INDI Properties jsou tvořené jako vektory, které jsou implementované jako pole v C knihovnách. PyINDI wrapper mapuje tyto pole jako Python iterables.

Pokud chci například získat hodnotu Number Property, udělám to pomocí `value = numberProperty[0].getValue()`. Obdobně pokud chci hodnotu nastavit, využiji `setValue()`. V řízení kamery jsem omezen hardwarem a některé Properties musím nastavit vždy, aby bylo možné focení. V kódu 3.3 je vidět nastavení základních proměnných pro mé požadavky.

```
ccd_info = device_ccd.getNumber("CCD_EXPOSURE")
while not (ccd_info):
    time.sleep(0.5)
    ccd_info = device_ccd.getNumber("CCD_EXPOSURE")
ccd_info[0].setValue(4945) #sirka senzoru
ccd_info[1].setValue(3275) #vyska senzoru
ccd_info[2].setValue(4.77) #velikost pixelu (um)
ccd_info[3].setValue(4.77) #velikost pixelu x
ccd_info[4].setValue(4.77) #velikost pixelu y
ccd_info[5].setValue(8) #pocet bitu na pixel
indiclient.sendNewProperty(ccd_info)

ccd_compression = device_ccd.getSwitch("CCD_COMPRESSION")
while not ccd_compression:
    time.sleep(0.5)
    ccd_compression = device_ccd.getSwitch("CCD_COMPRESSION")
ccd_compression[1].setState(PyIndi.ISS_ON)  # nastavit na RAW (
    bez komprese)
indiclient.sendNewProperty(ccd_compression)

upload_mode = device_ccd.getSwitch("UPLOAD_MODE")
while not upload_mode:
    time.sleep(0.5)
    upload_mode = device_ccd.getSwitch("UPLOAD_MODE")
upload_mode[0].setState(PyIndi.ISS_ON)  # Upload do klientu
indiclient.sendNewProperty(upload_mode)

capture_format = device_ccd.getSwitch("CAPTURE_FORMAT")
while not capture_format:
    time.sleep(0.5)
    capture_format = device_ccd.getSwitch("CAPTURE_FORMAT")
capture_format[3].setState(PyIndi.ISS_ON)  # Nastavit na RAW
indiclient.sendNewProperty(capture_format)

ccd_transfer_format = device_ccd.getSwitch("CCD_TRANSFER_FORMAT"
    )
```

```
while not ccd_transfer_format:
    time.sleep(0.5)
    ccd_transfer_format = device_ccd.getSwitch("
    CCD_TRANSFER_FORMAT")
ccd_transfer_format[1].setState(PyIndi.ISS_ON)  # NEF soubory,
    ccd_transfer_format[0] by byly FITS
indiclient.sendNewProperty(ccd_transfer_format)

ccd_capture_target = device_ccd.getSwitch("CCD_CAPTURE_TARGET")
while not ccd_capture_target:
    time.sleep(0.5)
    ccd_capture_target = device_ccd.getSwitch("
    CCD_CAPTURE_TARGET")
ccd_capture_target[1].setState(PyIndi.ISS_ON)  # SD karta
indiclient.sendNewProperty(ccd_capture_target)
```

Kód 3.3: Nastavení základních Properties kamery

Pokud jsou nastaveny tyto Properties, INDI driver nám umožňuje provést expozici. V klientu je tato část prováděna ve vlastní QThread, aby se zamezilo zamrznutí programu. Získaný snímek se objeví v RAW formátu jako Property typu Blob. V kódu 3.4 je vidět jak se tento příchozí soubor zpracovává. Předchází mu připojení k INDI serveru a kameře, což je umožněno také změnou hodnoty proměnné. Aby vše fungovalo musí být definovaná třída IndiClient s funkcí `updateProperty`, jak je znázorněno v kódu 2.1

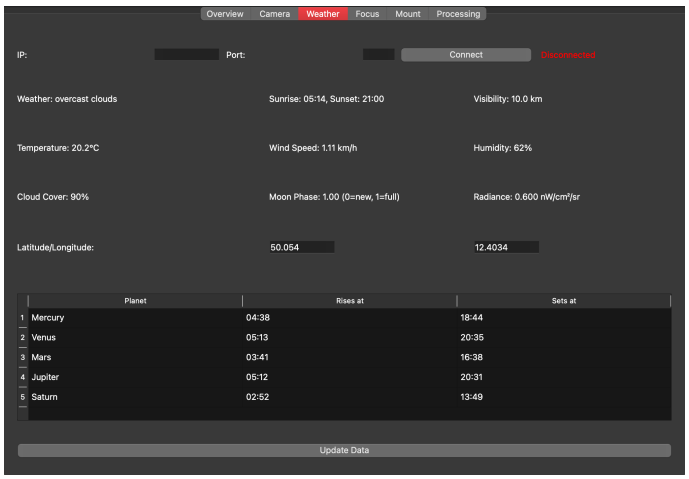
```
....
self.indiclient.setBLOBMode(PyIndi.B_ALSO, ccd, "CCD1")
ccd_ccd1 = device_ccd.getBLOB("CCD1")
while not ccd_ccd1:
    time.sleep(0.5)
    ccd_ccd1 = device_ccd.getBLOB("CCD1")

exposures = [self.exposure] * self.count
blobEvent = threading.Event()
blobEvent.clear()

for i in range(len(exposures)):
    ccd_exposure[0].setValue(exposures[i])
    self.indiclient.sendNewProperty(ccd_exposure)
    blobEvent.wait()
    for blob in ccd_ccd1:
        raw = blob.getblobdata()
        image = Image.open(io.BytesIO(raw))
        rgb = image.postprocess(gamma=(1,1), no_auto_bright=True
    , output_bps=16)
        imageio.imsave(f'output_{i}.tiff', rgb)
        qimage = ImageQt(image)
        pixmap = QPixmap.fromImage(qimage)
        self.image_signal.emit(pixmap)
        self.progress_signal.emit(f"Image {i + 1}/{self.count}
    captured and saved as output_{i}.nef")
    blobEvent.clear()
...
```

Kód 3.4: Ukázka zpracování příchozího souboru

Kód přichozí snímky rovnou převede do formátu .tiff, případné .NEF soubory se dají získat z SD karty kamery po ukončení snímání. Tlačítko Stop Capture umožňuje zastavit QThread vysláním signálu pro ukončení.

3.2.3 Pozorovací podmínky

Pozorování hvězd vyžaduje přesná data o počasí v reálném čase k zajištění optimálních pozorovacích podmínek. Tato aplikace integruje různé datové sady, aby poskytovala komplexní informace o povětrnostních podmínkách, viditelnosti planet, fázi měsíce a světelném znečištění, které jsou pro astronomy a hvězdáře zásadní. Přesná data o počasí pomáhají předpovídat oblačnost a viditelnost, zatímco časy západu a východu planet umožňují nadšencům plánovat svá pozorování. Údaje o světelném znečištění jsou zásadní pro identifikaci oblastí s minimálním umělým osvětlením, což zlepšuje datový výstup z pozorování hvězd.



Obrázek 3.4: Rozhraní zobrazení pozorovacích podmínek

Aplikace využívá několik API k načítání a zobrazování potřebných dat. OpenWeatherMap API se využívá k načítání dat o počasí. To zahrnuje parametry, jako jsou aktuální povětrnostní podmínky, teplota, viditelnost, oblačnost a vlhkost. Koncový bod používaný pro načítání dat o počasí je https://api.openweathermap.org/data/2.5/weather. Kód 3.5 ukazuje způsob načítání dat z této API.

```
def fetch_weather(self, api_key, city):
    url = f"https://api.openweathermap.org/data/2.5/weather?q={
    city}&appid={api_key}&units=metric"
    try:
        response = requests.get(url)
        if response.status_code == 200:
            return response.json()
...
```

Kód 3.5: Implementace volání OpenWratherMap API

K výpočtu videtelnosti a časů východů/západů planet a určení fáze měsíce se využívá knihovna PyEphem. Tato knihovna vypočítává polohy nebeských objektů na základě polohy pozorovatele a aktuálního data a času. Fragment kódu 3.6 ukazuje, jak vypočítat časy vzestupu a západu planet:

```
#ziskani dalsiho zapadu/vychodu, posunuto o 2h pro UTC+2
next_rise = (observer.next_rising(planet_body).datetime() +
    timedelta(hours=2)).strftime('%H:%M')
next_set = (observer.next_setting(planet_body).datetime() +
    timedelta(hours=2)).strftime('%H:%M')
visible_planets_info.append({'planet': planet_name, 'rise':
    next_rise, 'set': next_set})
#ziskani faze mesice
moon = ephem.Moon()
observer = ephem.Observer()
observer.date = ephem.now()
moon.compute(observer)
phase = moon.phase / 100
```

Kód 3.6: Výpočet polohy planet a měsíce

Light Pollution Map API se používá k získávání dat světelného znečištění pro konkrétní souřadnice pomocí datové sady `VIIRS_2023`, která poskytuje hodnoty záření v $nW/cm^2/sr$. Koncovým bodem pro načítání těchto dat je https://www.lightpollutionmap.info/QueryRaster/. Tato API není veřejná a pro získání klíče je potřeba kontaktovat vývojáře dané stránky. Kód 3.7 zobrazuje způsob získání dat o světelném zněčistění.

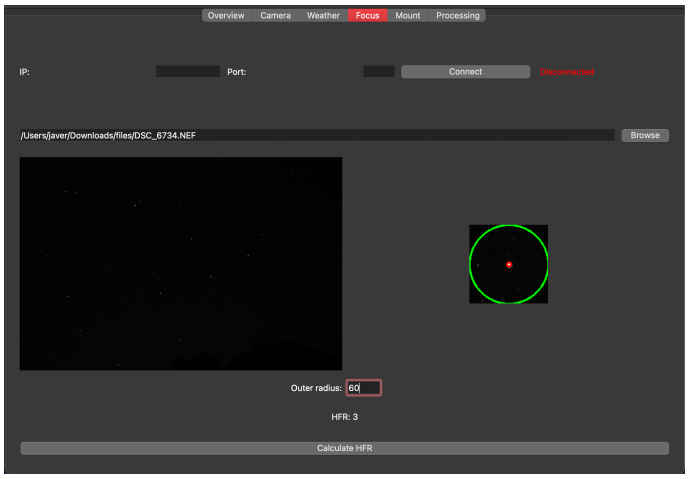
```
def fetch_light_pollution(self, lon, lat, dataset="viirs_2023"):
    base_url = "https://www.lightpollutionmap.info/QueryRaster/"
    params = {
        'ql': dataset,  # verze VIIRS dat
        'qt': 'point',  # "point" pro jednotnou zadost
        'qd': f"{lon},{lat}",  # longitude and latitude
        'key': "f6CflAAO8SvYq4rP"  # API klic
    }
    try:
        response = requests.get(base_url, params=params)
        if response.status_code == 200:
            radiance_value = response.text.strip()
            return radiance_value
        else:
            print("Failed to fetch light pollution data:",
   response.status_code)
            return None
    except Exception as e:
        print("Error during light pollution data request:", e)
        return None
```

Kód 3.7: Získání dat o světelném znečištění

Tato karta pro zobrazení venkovních podmínek poskytuje astronomům a pozorovatelům hvězd komplexní nástroj pro efektivní plánování jejich pozorování. Použití různých API, jako je OpenWeatherMap pro data o počasí, PyEphem pro astronomické výpočty a Light Pollution Map pro data světelného záření, demonstruje sílu kombinace více zdrojů dat k vytvoření takového systému.

3.2.4 Zaostření

Pro lidského pozorovatele je jednoduché usoudit, jestli je obrázek ostrý nebo ne. Automatický systém má několik možností, jak toto hodnocení provést. První z nich je spočítat FWHM (Full Width at Half Maximum) hvězdy v obrázku a poté nastavit zaostření tak, aby bylo dosaženo optimální (užší) FWHM. Problém FWHM je v předpokladu, že počáteční focus je blízko toho ideálního. Alternativní metodou je Half-Flux-Radius, což je vzdálenost v pixelech od centra hvězdy až do vzdálenosti, kde je světelný tok hvězdy poloviční. Čím nižší je HFR, tím lepší je seeing (degradace obrázku z důvodu turbulence v atmosféře Země) a ostrost obrázku.



Obrázek 3.5: Rozhraní výpočtu zaostření

Rozhraní na obrázku 3.5 umožňuje vybrat snímek a najde v něm nejjasnější hvězdu u které vypočítá HFR.

HFR je definován jako poloměr kruhu, který je vycentrován na nezaostřeném snímku hvězdy, ve kterém polovina celkového světelného toku hvězdy je uvnitř kruhu a polovina je vně [10].

Matematicky vypadá definice takto:

$$\sum_{i=0}^{N} V_i \cdot (d_i - HFR) = 0 \Leftrightarrow HFR = \frac{\sum_{i=0}^{N} V_i \cdot d_i}{\sum_{i=0}^{N} V_i}, \tag{3.1}$$

kde:

V_i je hodnota pixelu bez průměrné hodnoty pozadí

d_i je vzdálenost středu hvězdy ke každému pixelu

N je počet pixelů ve vnějším kruhu

HFR je Half Flux Radius

Funkce zodpovědná za výpočet HFR v kódu je `perform_hfr_calculation()`. Tato funkce převezme cestu k obrazovému souboru jako vstup, zpracuje obraz a vypočítá HFR pro nejjasnější hvězdu na snímku. Funkce začíná načtením obrázku ze souboru RAW nebo jiného standardního souboru pomocí knihoven rawpy a cv2. Pokud je soubor ve formátu RAW (.nef), použije se knihovna rawpy k následnému zpracování obrázku do formátu RGB, který se následně převede do formátu BGR vhodného pro zpracování OpenCV. Pokud je soubor již ve standardním formátu, načte se přímo pomocí funkce imread OpenCV.

```
def perform_hfr_calculation(self, file_path):
    if file_path.lower().endswith('.nef'):
        with rawpy.imread(file_path) as raw:
            rgb_image = raw.postprocess(gamma=(1, 1),
    no_auto_bright=True, output_bps=16)
        image = cv2.cvtColor(np.array(rgb_image), cv2.
    COLOR_RGB2BGR)
    else:
        image = cv2.imread(file_path, cv2.IMREAD_COLOR)
```

Kód 3.8: Načtení souboru k výpočtu zaostření

Jakmile je obrázek načten, je převeden na stupně šedi a normalizován, aby se zvýšil kontrast pro další zpracování. Vnější poloměr pro výpočet HFR je získán z uživatelského vstupu.

```
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
image_gray = cv2.normalize(image_gray, None, 0, 255, cv2.
    NORM_MINMAX)
image_gray = np.uint8(image_gray)
...
    outer_radius = int(self.radius_input.text())
```

Kód 3.9: Převod do stupňů šedi

Další krok zahrnuje identifikaci nejjasnější hvězdy na snímku. Toho je dosaženo prahováním obrazu ve stupních šedi a vytvořením binárního obrazu, kde jsou zvýrazněny nejjasnější oblasti. V binárním snímku jsou pak detekovány obrysy a obrys s největší plochou je považován za nejjasnější hvězdu. Těžiště tohoto obrysu je vypočítáno pro určení polohy hvězdy.

```
_, thresh = cv2.threshold(image_gray, np.max(image_gray) - 10,
    255, cv2.THRESH_BINARY)
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.
    CHAIN_APPROX_SIMPLE)
...
brightest = max(contours, key=cv2.contourArea)
M = cv2.moments(brightest)
centroid = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
```

Kód 3.10: Identifikace nejjasnější hvězdy

Pro výpočet HFR se kolem těžiště vytvoří kruhová maska se zadaným vnějším poloměrem. Střední intenzita pozadí se vypočítá pomocí pixelů mimo tuto masku. Celkový světelný tok hvězdy se vypočítá sečtením intenzit pixelů v masce po odečtení střední intenzity pozadí. HFR je pak určeno opakovaným zvětšováním poloměru kruhové masky, dokud kumulativní tok v masce nedosáhne poloviny celkového toku.

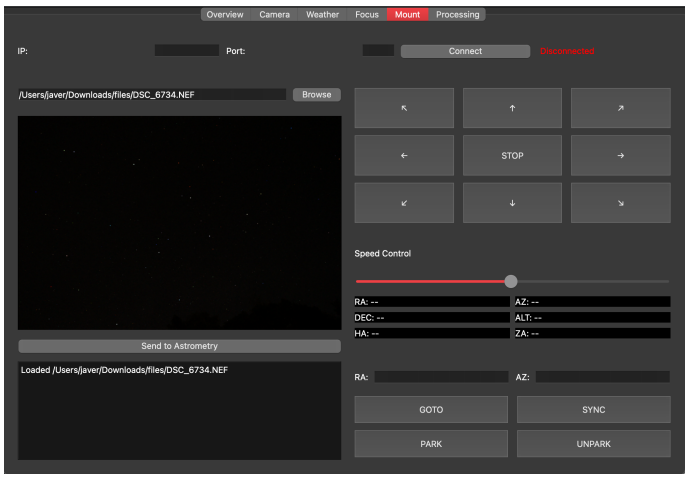
```
mask = np.zeros_like(image_gray, dtype=np.uint8)
cv2.circle(mask, centroid, outer_radius, 255, -1)
background_mask = np.bitwise_not(mask)
background_pixels = image_gray[background_mask == 255]
background_mean = np.mean(background_pixels)
star_pixels = image_gray[mask == 255]
adjusted_flux = np.sum(star_pixels - background_mean)

# Vypocet HFR
cumulative_flux = 0
radius = 0
for radius in range(1, outer_radius):
    mask = np.zeros_like(image_gray, dtype=np.uint8)
    cv2.circle(mask, centroid, radius, 255, -1)
    flux_in_circle = np.sum(image_gray[mask == 255] -
    background_mean)

    cumulative_flux += flux_in_circle
    if cumulative_flux >= adjusted_flux / 2:
        break

self.hfr_label.setText(f"HFR: {radius}")
```

Kód 3.11: Výpočet HFR

Nakonec je výsledek vizualizován vytvořením výřezu oblasti hvězdy, nakreslením vnějšího kruhu, HFR kruhu a zobrazením zpracovaného snímku.

3.2.5 Ovládání montáže

Na tomto panelu je možnost zapnout automatickou kompenzaci rotace Země. Aplikace umožňuje využít astronomy.net pro výpočet aktuální oblohy. Informace ze solveru jsou zobrazeny, takže si uživatel ověří, na které objekty míří.



Obrázek 3.6: Rozhraní ovládání montáže

Klient umožňuje navést montáž na určité souřadnice zadané uživatelem.

astrometry.net API

Astrometry.net je online služba, která přijímá astronomické snímky a vrací nebeské souřadnice hvězd a dalších objektů na snímku. Integrace s Astrometry.net zahrnuje několik klíčových kroků: přihlášení k získání klíče relace, nahrání obrázku, kontrola stavu úlohy a načtení výsledků.

Tento proces řídí funkce `send_to_astrometry`. Začíná ověřením cesty k souboru a přihlášením k Astrometry.net API pomocí poskytnutého klíče API. Pokud je soubor ve formátu RAW (.nef), je nejprve převeden do formátu TIFF. Obrázek je poté nahrán a samostatné vlákno zkontroluje stav úlohy a po dokončení načte anotovaný obrázek. Tato funkce zajišťuje, že uživatelské rozhraní bude během těchto operací reagovat.

```
def send_to_astrometry(self):
    file_path = self.file_path_input_mount.text()
    if not file_path:
        self.mount_log_area.append("No file selected.")
        return

    def log_func(message):
        self.mount_log_area.append(message)
        self.mount_log_area.repaint()  # Zajisteni ze se log updatuje okamzite

    try:
        # Zpracovani obrazku
        if file_path.lower().endswith('.nef'):
            log_func("Processing NEF file...")

            raw = rawpy.imread(file_path)
            rgb = raw.postprocess()
            processed_file_path = 'out.tiff'
            imageio.imsave(processed_file_path, rgb)
            log_func("Image processed successfully: out.tiff")
        else:
            processed_file_path = file_path
            log_func(f"Using provided file: {file_path}")

        # Odeslani zpravy o dokonceni zpracovani
        log_func("Image processing completed successfully.")

        # Spusteni sitovych operaci
        self.perform_network_operations_sequential(processed_file_path, log_func)

    except Exception as e:
        log_func(f"Error during image processing: {str(e)}")
```

Kód 3.12: Hlavní funkce načítání z Astrometry

Funkce pro přihlášení, nahrání obrázku a načtení výsledků jsou zapouzdřeny v pomocných funkcích. Každá funkce zaznamenává svůj průběh a výsledky do výstupního pole v uživatelském rozhraní.

```
def perform_network_operations_sequential(self,
    processed_file_path, log_func):
    try:
        # Prihlaseni do Astrometry.net
        log_func("Logging in to Astrometry.net...")
        session_key = self.login_to_astrometry(API_KEY, log_func
    )
        log_func(f"Session key received: {session_key}")

        # Upload obrazku
        log_func("Uploading image...")
        sub_id = self.upload_image(session_key,
    processed_file_path, log_func)
        log_func(f"Submission ID received: {sub_id}")

        # Start vlakna ke kontrole stavu zpracovani a ziskani
    anotovaneho obrazku
        self.astrometry_worker = AstrometryWorker(sub_id)
        self.astrometry_worker.log_signal.connect(log_func)
        self.astrometry_worker.completed_signal.connect(lambda
    message: log_func(message))
            self.astrometry_worker.start()

    except Exception as e:
        log_func(f"Error during network operations: {str(e)}")
```

Kód 3.13: Rozdělení síťových operací

Aby bylo uživatelské rozhraní stále responzivní, funkce `send_to_astrometry` spouští síťové operace v samostatném vlákně pomocí QThread. To zajišťuje, že hlavní aplikace zůstane interaktivní, zatímco volání API a zpracování obrázků probíhají na pozadí. Třída AstrometryWorker dědí z QThread a stará se o kontrolu stavu úlohy a načítání výsledků.

```
class AstrometryWorker(QThread):
    log_signal = pyqtSignal(str)
    completed_signal = pyqtSignal(str)

    def __init__(self, sub_id, parent=None):
        super().__init__(parent)
        self.sub_id = sub_id

    def run(self):
        try:
            job_id = self.check_job_status(self.sub_id)
            self.log_signal.emit(f"Job ID received: {job_id}")
            self.fetch_annotation_image(job_id)
            self.completed_signal.emit("Astrometry process
    completed.")
        except Exception as e:
            self.log_signal.emit(f"Error: {str(e)}")

    def log_func(self, message):
        self.log_signal.emit(message)
    ...
```

Kód 3.14: Implementace QThread

Ovládání montáže

Pravá polovina záložky Mount v klientu je věnována ovládání montáže dalekohledu. Tato část obsahuje ovládací prvky pro pohyb dalekohledu v různých směrech, úpravu rychlosti pohybu a provádění specifických akcí, jako je parkování, vyparkování a synchronizace dalekohledu s danými souřadnicemi. Řízení montáže je dosaženo pomocí protokolu INDI 2.3.2

K ovládání telescopu je využito mnoho funkcí, které mění stavy switchů. Připojení k montáží zajišťuje funkce `connect_telescope()`, jejíž definici můžeme vidět v kódu 3.15.

```
def connect_telescope(self):
    if not indi_available:
        self.mount_log_area.append("PyIndi not available.")
        return
    ...
    # Pripojeni k teleskopu
    self.device_telescope = self.indiclient.getDevice(self.
    telescope)
    while not self.device_telescope:
        time.sleep(0.5)
        self.device_telescope = self.indiclient.getDevice(self.
    telescope)

    self.telescope_connect = self.device_telescope.getSwitch("
    CONNECTION")
    while not self.telescope_connect:
        time.sleep(0.5)
        self.telescope_connect = self.device_telescope.getSwitch
    ("CONNECTION")

    if not self.device_telescope.isConnected():
        self.telescope_connect.reset()
        self.telescope_connect[0].setState(PyIndi.ISS_ON)  #
    CONNECT switch
        self.indiclient.sendNewProperty(self.telescope_connect)
```

Kód 3.15: Připojení k montáži

Důležitou součástí této sekce je možnost provést GOTO a následné sledování pomocí kompenzace rotace. Toto je implementováno také pomocí přepnutí několika switchů.

```
def goto_coordinates(self):
    if not indi_available:
        self.mount_log_area.append("PyIndi not available.")
        return

    try:
        ra = float(self.ra_input.text())
        dec = float(self.az_input.text())
    except ValueError:
        self.mount_log_area.append("Invalid RA/DEC values.")
        return
    if not self.device_telescope.isConnected():
        self.connect_telescope()

```

```
15      telescope_on_coord_set = self.device_telescope.getSwitch("
        ON_COORD_SET")
16      while not telescope_on_coord_set:
17          time.sleep(0.5)
18      telescope_on_coord_set = self.device_telescope.getSwitch("
        ON_COORD_SET")
19      telescope_on_coord_set.reset()
20      telescope_on_coord_set[0].setState(PyIndi.ISS_ON)  # TRACK
21      self.indiclient.sendNewProperty(telescope_on_coord_set)
22
23      telescope_radec = self.device_telescope.getNumber("
        EQUATORIAL_EOD_COORD")
24      while not telescope_radec:
25          time.sleep(0.5)
26          telescope_radec = self.device_telescope.getNumber("
        EQUATORIAL_EOD_COORD")
27      telescope_radec[0].setValue(ra)
28      telescope_radec[1].setValue(dec)
29      self.indiclient.sendNewProperty(telescope_radec)
```

Kód 3.16: Implementace GOTO

Aby bylo zajištěno, že uživatelské rozhraní bude reagovat při neustálé aktualizaci souřadnic připojení, používá se vlákno podobně jako při získávání dat z Astrometry v kódu 3.14. Třída CoordinateWorker, která dědí z QThread, načítá rovníkové a horizontální souřadnice dalekohledu každých 5 sekund a vysílá je prostřednictvím signálu. To zabraňuje zablokování hlavního vlákna uživatelského rozhraní, což uživateli umožňuje hladkou interakci s aplikací.

3.2.6 Postprocessing a export dat

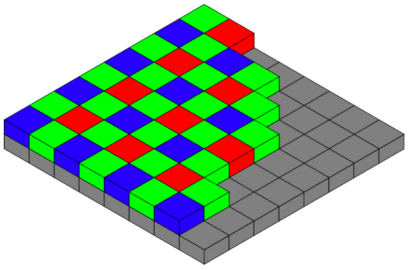
Pro účely astronomie chceme obrazová data získávat ze zrcadlovky ve formátu RAW. RAW je třída počítačových souborů, která typicky obsahuje nezpracovaný obraz neboli hodnoty jednotlivých pixelů senzoru a velké množství metadat o obrázku vytvořených kamerou (EXIF data). Samotné RAW soubory mají několik formátů (Nikon NEF, Canon CR2, atd.).

RAW data z obrazového senzoru obsahují hodnotu světelné intenzity focené scény, ale tyto data nemusí být rozpoznatelná pro lidské oko. Jsou to jednokanálová data intenzity obrazu, pravděpodobně s nenulovou minimální hodnotou reprezentující černou. Položky jsou integer hodnoty obsahující 10-14 bitů dat. Žádné hodnoty v obrázku nebudou vyšší než nějaké maximum, které reprezentuje saturační bod kamerového senzoru.

Data jsou v souboru většinou ve formě CFA(Color Filter Array). Je to m x n matice pixelů, kde každý pixel obsahuje informaci o jednom barevném kanálu: červená, zelená nebo modrá. Jelikož je světlo dopadající na jakýkoliv pixel CCD senzoru zaznamenáno jako počet elektronů v kapacitoru, může být uloženo jen jako skalární veličina.

Jeden pixel nemůže zaznamenat 3 dimenzionální povahu pozorovatelného světla. V CFA je informace o každém ze tří barevných kanálů zaznamenána v jiném místě pomocí spektrum-selektivníhoch filrů umístěných nad jednotlivými pixely.

Nejrozšířenější CFA vzor je Bayerovská maska, zobrazená na obrázku 3.7. Je zde dvakrát více zelených pixelů, protože lidské oko je více citlivé na různé odstíny zelené.



Obrázek 3.7: Bayerovská maska. Každý pixel reprezentuje buď červenou, zelenou nebo modrou hodnotu světelné intenzity na senzoru. [11]

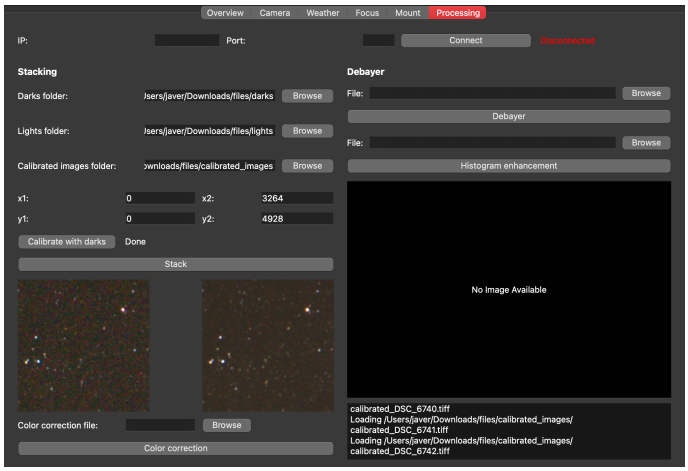
Pro získání všech 3 barev v každém pixelu je potřeba využít debayerizace. Debayerizace, také známá jako demosaikování je proces pro převod CFA obrázku (m x n) na true RGB obrázek (m x n x 3). Z CFA matice sice známe jen jednu hodnotu pro každý pixel, ale pomocí interpolace zbylých dvou hodnot z sousedních pixelů můžeme získat všechny 3 hodnoty.

Převod RAW obrázku do např. TIFF je pomocí knihoven velmi jednoduchý proces, lze to provést například jako v kódu 3.17.

```
raw = rawpy.imread("sample1.nef")
rgb = raw.postprocess(use_camera_wb=True)
imageio.imsave("sample.tiff", rgb)
```

Kód 3.17: Převod .NEF do .TIFF

Na obrázku 3.8 je vidět rozhraní pro zpracování. Je rozděleno na několik částí, které umožňují různé korekce.



Obrázek 3.8: Rozhraní zpracování souborů

Kalibrace tmavými snímky

Kalibrace tmavými (Dark) snímky je základním krokem předzpracování v astrofotografování. Tmavé snímky se používají k odečtení vlastního šumu a vad snímače fotoaparátu od světelných snímků (skutečných snímků nebeských objektů). Tento proces pomáhá dosáhnout čistších a přesnějších obrázků.

Vše v kamerovém senzoru neustále vibruje a poskakuje. Tuto agresivní vibraci označujeme teplota. Občas se náhodně stane, že elektron v senzoru odrazí tak silně, že to vypadá, jako by do senzoru zasáhl foton. Čím je snímač teplejší, tím je pravděpodobnější, že k tomu dojde. To je důvod, proč profesionální fotografové a vědci používají chlazené senzory.

Delší expozice generují vyšší úrovně tepelného šumu, zejména u DSLR fotoaparátů, protože ty nemají výhodu chladicího systému, který by je udržoval na teplotě mnoho stupňů pod okolní teplotou. Tepelný šum má dvě různé formy: celkový šum a zjevnější "hot pixely", které se zobrazují jako jasné světelné body.

Tmavé snímky se pořizují se stejným nastavením jako hlavní snímky, ale s nasazenou krytkou objektivu. Je důležité, aby tmavé snímky odpovídaly parametrům používaným pro zachycení hlavních snímků s ohledem na délku expozice, ISO a teplotu.

Proces kalibrace zahrnuje vytvoření hlavního tmavého snímku z více tmavých snímků, které vidíme v kódu 3.18

```
def create_master_dark(dark_folder):
    dark_files = [os.path.join(dark_folder, f) for f in os.
    listdir(dark_folder) if f.lower().endswith(('.nef', '.tiff')
    )]

    dark_stack_r = []
    dark_stack_g = []
    dark_stack_b = []

    for file in dark_files:
        dark_frame = load_image(file)
        dark_stack_r.append(dark_frame[:,:,0])
        dark_stack_g.append(dark_frame[:,:,1])
        dark_stack_b.append(dark_frame[:,:,2])

    master_dark_r = np.median(np.stack(dark_stack_r, axis=0),
    axis=0)
    master_dark_g = np.median(np.stack(dark_stack_g, axis=0),
    axis=0)
    master_dark_b = np.median(np.stack(dark_stack_b, axis=0),
    axis=0)

    master_dark = np.stack((master_dark_r, master_dark_g,
    master_dark_b), axis=-1)
    return master_dark
```

Kód 3.18: Vytvoření hlavního tmavého snímku

Po vytvoření hlavního tmavého snímku jej použiji ke kalibraci každého světlého snímku odečtením hlavního tmavého snímku od světlého snímku.

```python
def calibrate_image(light_image_path, master_dark, output_path):
    light_image = load_image(light_image_path)

    if light_image.shape != master_dark.shape:
        raise ValueError(f"Light image and master dark frame have different dimensions: {light_image.shape} vs {master_dark.shape}")

    calibrated_image = light_image - master_dark
    calibrated_image = np.clip(calibrated_image, 0, 65535)

    save_as_tiff(calibrated_image, output_path)
    return calibrated_image
```

Kód 3.19: Odečtení od světelných snímků

Pro úspěšné získaná kalibrovaných snímků v klientu stačí zvolit všechny tři složky pro tmavé, světelné a kalibrované snímky a spustit kalibraci pomocí tlačítka. Aby klient mohl během procesu kalibrace reagovat, používám ke spuštění kalibrace QThread. To umožňuje uživatelskému rozhraní zůstat aktivní, zatímco kalibrace běží na pozadí.

Skládání

Skládání snímků je klíčovou technikou v astrofotografování, která kombinuje vícenásobné expozice stejné scény pro zvýšení kvality výsledného snímku. Tento proces zvyšuje poměr signálu k šumu, odhaluje slabé detaily a snižuje šum, čímž vytváří jasnější a detailnější výsledný obraz.

Při pořízení jednotlivých snímků můžeme vidět hodně šumu a chybějící hladké přechody. Skládání snímků toto řeší a vytváří hezčí finální snímek. Celý proces skládání zahrnuje zarovnání více obrazů a jejich následné zkombinování. Zarovnání snímků je nezbytné pro přesné skládání, zejména proto, že se fotoaparát mohl mezi snímky mírně pohybovat. K zarovnání obrázků používáme porovnávání prvků a homografii z knihovny OpenCV. Jádro algoritmu je převzato z [12].

```python
def match(self, image1, image2):
    image1 = (self.brighten(image1) * 255).astype('uint8')
    image2 = (self.brighten(image2) * 255).astype('uint8')
    det = cv2.ORB_create(nfeatures=50000)
    kp1, desc1 = det.detectAndCompute(image1, None)
    kp2, desc2 = det.detectAndCompute(image2, None)
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(desc1, desc2)
    matches = sorted(matches, key=lambda x: x.distance)
    matches = matches[:len(matches) // 10]
    src_pts = np.float32([kp1[m.queryIdx].pt for m in matches]).reshape(-1, 1, 2)
    dst_pts = np.float32([kp2[m.trainIdx].pt for m in matches]).reshape(-1, 1, 2)
    M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
    return M
```

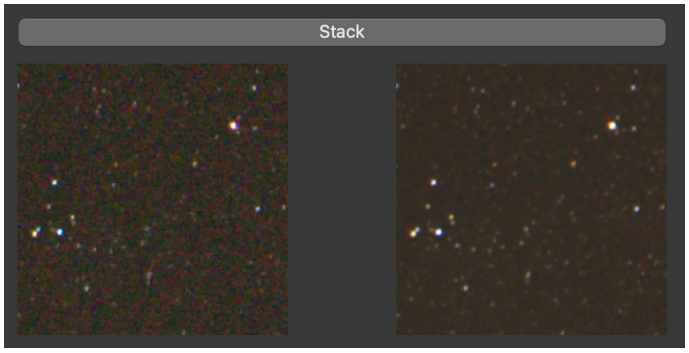
Kód 3.20: Zarovnání snímků

Jakmile jsou obrázky zarovnány, spojíme je a vytvoříme skládaný obrázek. Využijeme průměrování hodnot pixelů z více obrázků.

```
def add(self, im, loaders):
    h, w, *_ = im.shape
    out = im.copy()
    count = np.full((h, w), 1.0)
    for load_im2 in loaders:
        im2 = load_im2()
        M = self.match(im2, im)
        out += cv2.warpPerspective(im2, M, (w, h))
        counter = np.full(im2.shape[0:2], 1.0)
        count += cv2.warpPerspective(counter, M, (w, h))
    return (out / out.max(), count / count.max())
```

Kód 3.21: Sloučení snímků

Integrace skládání obrázků do této aplikace poskytuje uživatelům výkonný nástroj pro vytváření jasnějších a podrobnějších astrofotografií. Použití QThread zajišťuje, že uživatelské rozhraní zůstává citlivé, což uživatelům umožňuje interakci s aplikací, zatímco proces skládání běží na pozadí. Na obrázku 3.9 lze vidět, že výsledný obrázek má opravdu hladší přechody a nižší šum.



Obrázek 3.9: Ukázka výstupu skládacího algoritmu

Pro úspěšné skládání stačí vybrat složku s kalibrovanými snímky a stisknout tlačítko Stack. Klient také umožňuje vybrat výřez obrazu, který bude skládat. Tato funkce se může hodit, pokud na obrazu překáží nějaký objekt, který by dělal při skládání problémy.

Barevná korekce

Korekce barev je základním krokem při zpracování obrazu, aby se zajistilo, že barvy v konečném snímku přesně reprezentují scénu. Pomáhá upravit barvy obrazu tak, aby vypadaly přirozeně a vyváženě. Tento proces zahrnuje manipulaci s histogramem, korekci barevných nádechů a vylepšení celkové věrnosti barev obrazu. Obraz je nejdříve normalizován, poté jsou nalezeny středy histogramu jednotlivých kanálů a ty jsou synchronizovány. Vše běží v QThread pro zachování ostatních funkcí klienta. Pomocí funkce `find_histogram_bounds` lze najít střed histogramu daného snímku.

```python
def find_histogram_bounds(histogram: np.array, percentage: float = 0.4) -> tuple:
    peak = np.max(histogram)
    threshold = peak * percentage
    indices = np.where(histogram >= threshold)[0]
    if len(indices) == 0:
        return (0, len(histogram) - 1)
    lower_bound = indices[0]
    upper_bound = indices[-1]
    center = (lower_bound + upper_bound) // 2
    return center
```

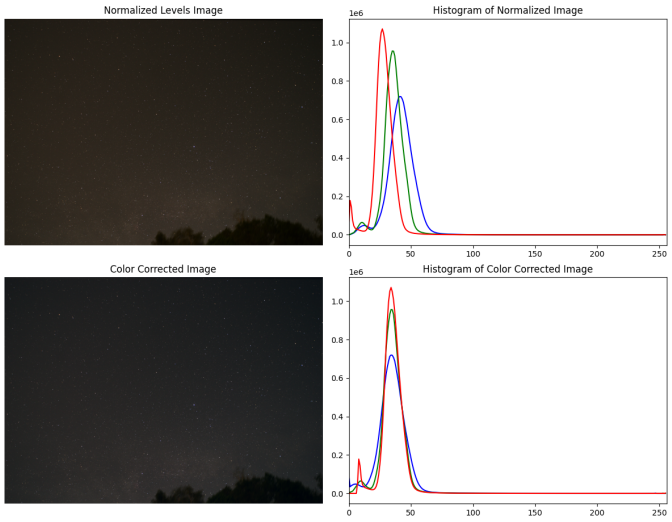
Kód 3.22: Určení středu histogramu

Poté je nutné všechny barevné kanály posunout do stejného bodu.

```python
def modify_colors(self, image: np.array, offsets: tuple) -> np.array:
        b_offset, g_offset, r_offset = offsets
        b, g, r = cv2.split(image)
        b = np.clip(b.astype(np.int16) + b_offset, 0, 255).astype(np.uint8)
        g = np.clip(g.astype(np.int16) + g_offset, 0, 255).astype(np.uint8)
        r = np.clip(r.astype(np.int16) + r_offset, 0, 255).astype(np.uint8)
        return cv2.merge((b, g, r))
```

Kód 3.23: Posunutí barevného kanálu

Celý proces zajišťuje, že konečný obraz bude mít vyvážené barvy a poskytne přirozenější a vizuálně přitažlivější výsledek. Ukázku výstupu lze vidět na obrázku 3.10. Lze vidět, že zmizelo žluté zabarvení scény.



Obrázek 3.10: Výstup barevné kalibrace

V klientu stačí vybrat soubor a spustit kalibraci tlačítkem. Jakmile bude proces dokončen zobrazí se výstup v dalším okně.

Debayerizace

Ve většině kódu je k převodu .NEF souborů použita knihovna rawpy. Knihovna rawpy provede veškerý processing sama, což je pro účely uživatele příjemné, ale pro edukační účely zde základní proces demonstruji manuálně. Zjednodušený proces [11] pro převod RAW obrázku do viditelné podoby vypadá takto:

- Pomocí knihovny rawpy extrahujeme data z NEF do CFA matice
- Příprava CFA obrázku, linearizace, normalizace do prostoru (0,1)
- Aplikace vyvážení bílé
- Implamentace debayerovacího algoritmu

Pro získání obrazových dat z RAW využijeme knihovnu rawpy.

```
with rawpy.imread(image_path) as raw:
    im_in = raw.raw_image
    wb_multipliers = raw.camera_whitebalance
```

Kód 3.24: Získání dat z RAW

Vytvořená 2D matice nemusí být lineární obrázek. Je možné, že kamera využila nelineární transformaci pro potřeby úložiště. Pokud tomu tak je, metadata obrázku budou obsahovat tabulku pro mapování daných hodnot CFA matice do plných 10-14 bitových hodnot. Pro naše účely předpokládám, že tomu tak není.

I přesto že není potřeba invertovat nelineární kompresi, RAW obrázek stále může mít offset a arbitrární škálování. Proto aplikuji affiní transformaci z rozsahu (černá, bílá) do rozsahu (o,1), abych normalizoval hodnoty pixelů. Hodnoty černé a bílé lze získat jako minimum a maximum CFA matice, případně pomocí rawpy.

```
linear_bayer = (im_in-np.min(im_in))/(np.max(im_in)-np.min(im_in
    ))
```

Kód 3.25: Normalizace RAW dat

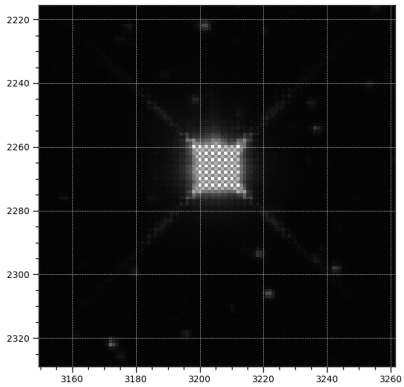
Objekt může mít jakoukoliv barvu v závislosti na světlu, které ho osvětluje. Pro zjištění barvy, kterou bychom viděli jako lidé, potřebuji referenční bod, něco co by mělo mít určitou barvu. Poté můžu přeškálovat RGB hodnoty pixelů dokud nemají správnou barvu. Jelikož je jednoduché identifikovat objekty, které by měly být bílé (stejné hodnoty r, g a b), tak najdu jeden bílý referenční pixel ze kterého získám škálovací faktory. Jakmile toto udělám pro jeden pixel, budu se domnívat, že celá scéna je osvětlena stejně a použiji toto škálování pro celý obrázek.

Problém je tedy redukován na nalezení dvou skalárů, které reprezentují relativní škálování 2 barevných kanálů oproti tomu třetímu. Typicky použiji zelený kanál jako ten, se kterým budu porovnávat ostatní. Tyto skaláry jsou uloženy v EXIF datech raw souboru. Implemantace v Pythonu je zobrazena v kódu 3.26

```
# vytvorime masku pomoci funkce popsane nize a pomoci numpy ji
    aplikujeme na nas linearni obrazek
mask = wbscalematrix(linear_bayer.shape[0], linear_bayer.shape
    [1], wb_multipliers, 'rggb')
balanced_bayer = np.multiply(linear_bayer, mask)
...
def wbscalematrix(self, m, n, wb_scales, align):
    # Vytvori skalovaci matici pro vyvazeni bile
    scalematrix = wb_scales[1] * np.ones((m, n))
    if (align == 'rggb'):
        scalematrix[2::2, 2::2] = wb_scales[0]
        scalematrix[1::2, 1::2] = wb_scales[2]
    elif (align == 'bggr'):
        scalematrix[1::2, 1::2] = wb_scales[0]
        scalematrix[0::2, 0::2] = wb_scales[2]
    elif (align == 'grbg'):
        scalematrix[0::2, 1::2] = wb_scales[0]
        scalematrix[0::2, 1::2] = wb_scales[2]
    elif (align == 'gbrg'):
        scalematrix[1::2, 0::2] = wb_scales[0]
        scalematrix[0::2, 1::2] = wb_scales[2]
    return scalematrix
```

Kód 3.26: Aplikace masky na obrázek

Pokud takto upravený obrázek zobrazím a přiblížím, můžu vidět, že je rozložen do čtverců a černobílý.



Obrázek 3.11: Příblížený obrázek

Nyní musíe implementovat debayerovací algoritmus, aby se toto vyřešilo. Existuje mnoho demozaikovací funkcí pro dokončení tohoto procesu. Místo využití těchto funkcí použiji základní algoritmus.

Nejjednodušší metoda je interpolace pomocí nejbližšího souseda zobrazená v kódu 3.27, kde červená hodnota nečerveného pixelu je počítána jako průměr dvou sousedních červených pixelů. Pro ostatní pixely obdobně.

```
def debayering(self, input):
    img = input.astype(np.double)
    m, n = img.shape

    red_m = np.tile([[1, 0], [0, 0]], (int(m / 2), int(n / 2)))
    green_m = np.tile([[0, 1], [1, 0]], (int(m / 2), int(n / 2))
    )
    blue_m = np.tile([[0, 0], [0, 1]], (int(m / 2), int(n / 2)))

    r = np.multiply(img, red_m)
    g = np.multiply(img, green_m)
    b = np.multiply(img, blue_m)

    filter_g = 0.25 * np.array([[0, 1, 0], [1, 0, 1], [0, 1,
    0]])
    missing_g = convolve2d(g, filter_g, 'same')
    g = g + missing_g

    filter1 = 0.25 * np.array([[1, 0, 1], [0, 0, 0], [1, 0, 1]])
    missing_b1 = convolve2d(b, filter1, 'same')
    filter2 = 0.25 * np.array([[0, 1, 0], [1, 0, 1], [0, 1, 0]])
    missing_b2 = convolve2d(b + missing_b1, filter2, 'same')
    b = b + missing_b1 + missing_b2

    missing_r1 = convolve2d(r, filter2, 'same')
    missing_r2 = convolve2d(r + missing_r1, filter1, 'same')
    r = r + missing_r1 + missing_r2

    output = np.stack((r, g, b), axis=2)
    return output
```

Kód 3.27: Interpolace pomocí nejbližšího souseda

Po zobrazení výstupního obrázku můžeme vidět, že je konečně barevný. V klientu proběhne všechno zpracování na pozadí a zobrazí se pouze debayerovaný obrázek.

Vyvážení histogramu

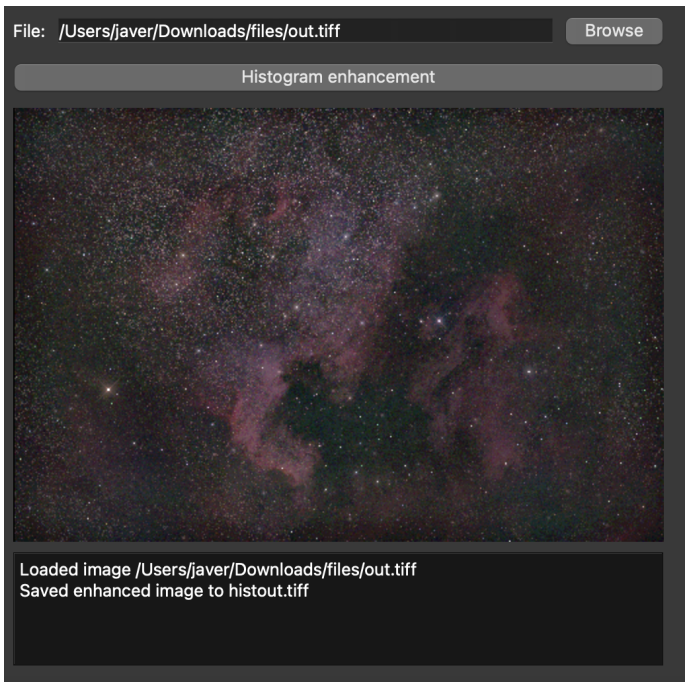
Výstupní obrázek je ale velmi tmavý, což je u astronomických snímků časté. Hodnoty pixelů se soustředí pouze na začátku histogramu. Toto můžeme napravit pomocí ekvalizace v OpenCV [13]. Nevyužijeme ale obyčejnou ekvalizaci, ale adaptivní ekvalizaci, která nám umožňuje nastavit dva důležité parametry: ClipLimit a oblast ekvalizace. Implementace vypadá takto:

```
def histogram_enhancement(self, image_path):
        colorimage = cv2.imread(image_path)
        self.progress_signal.emit(f"Loaded image {image_path}")

        clahe_model = cv2.createCLAHE(clipLimit=20.0,
    tileGridSize=(8,8))

        colorimage_b = clahe_model.apply(colorimage[:,:,0])
        colorimage_g = clahe_model.apply(colorimage[:,:,1])
        colorimage_r = clahe_model.apply(colorimage[:,:,2])

```

```
11          colorimage_clahe = np.stack((colorimage_b, colorimage_g,
    colorimage_r), axis=2)
12
13          output_path = "histout.tiff"
14          cv2.imwrite(output_path, colorimage_clahe)
15          self.progress_signal.emit(f"Saved enhanced image to {
    output_path}")
16
17          im = Image.fromarray(colorimage_clahe)
18
19          qimage = QImage(im.tobytes(), im.size[0], im.size[1],
    QImage.Format.Format_BGR888)
20          pixmap = QPixmap.fromImage(qimage)
21          self.image_signal.emit(pixmap)
```

Kód 3.28: Vyvážení histogramu

Na obrázku 3.12 můžeme vidět ukázku aplikace ekvalizace histogramu na astronomický snímek. Z tmavého zdrojového souboru můžeme jednoznačně vidět všechny zachycené prvky.

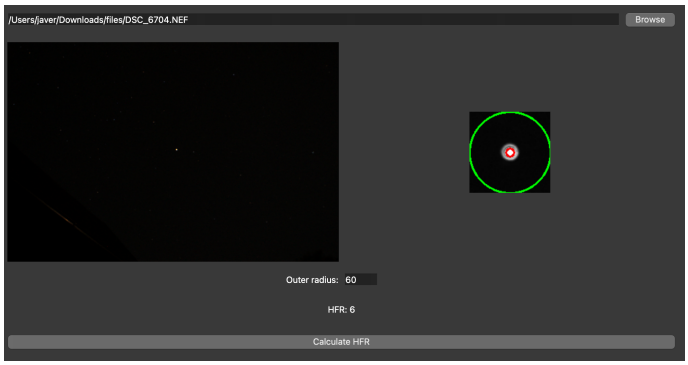


Obrázek 3.12: Ukázka vyváženého obrázku

Kapitola 4

Ověření funkčnosti

Pro ověření funkčnosti klienta jsem zvolil snímání mléčné dráhy, která vypadá zajímavě i bez použití teleobjektivu. Prvním krokem je ustavení montáže podle manuálu [14] a provedení zarovnání s polárkou, aby montáž mohla správně kompenzovat rotaci Země a navádět. Na montáž jsem namontoval DSLR kameru Nikon D7000 s objektivem 14mm a nastavil na ní režim Bulb, který je nutný pro propojení s PC. Kameru i montáž jsem propojil s Raspberry Pi na kterém běží INDI server. Na počítači v síti jsem spustil klient a připojil se k serveru. V sekci Camera jsem provedl první testovací snímek. Po stáhnutí snímku do počítače jsem ho zvolil v sekci Focus a vyzkoušel vypočítat HFR.

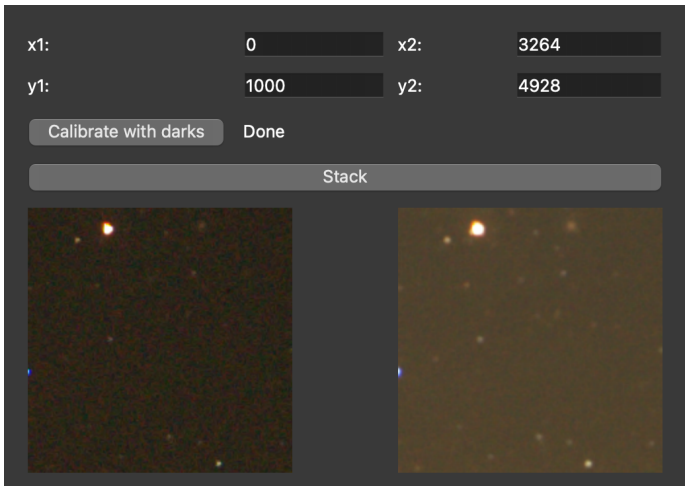


Obrázek 4.1: Výpočet HFR u nezaostřeného snímku

Aplikace potvrdila, že kamera není zaostřená, provedl jsem tedy manuální ostření a znova pořídil snímek. Tento snímek jsem také nahrál do okna Focus, kde mi bylo potvrzeno, že snímek je ostrý. Poté jsem přešel do sekce Mount, kde jsem otestoval funkci montáže. Zadal jsem RA a DEC mléčné dráhy a provedl GOTO. Následně jsem pomocí šipek posunul záběr tak, aby byla vidět co největší část oblohy. V sekci Camera jsem zadal snímání 20 Light snímků délky 30 sekund s ISO 1600. Delší snímání tato kamera přes USB nepodporuje. Po ukončení snímání jsem nasadil krytku na objektiv kamery a provedl další sekvenci se stejnými parametry. Na obrázku 4.2 je možné vidět jak vypadá jeden neupravený Light snímek zobrazený klientem pomocí rawpy.

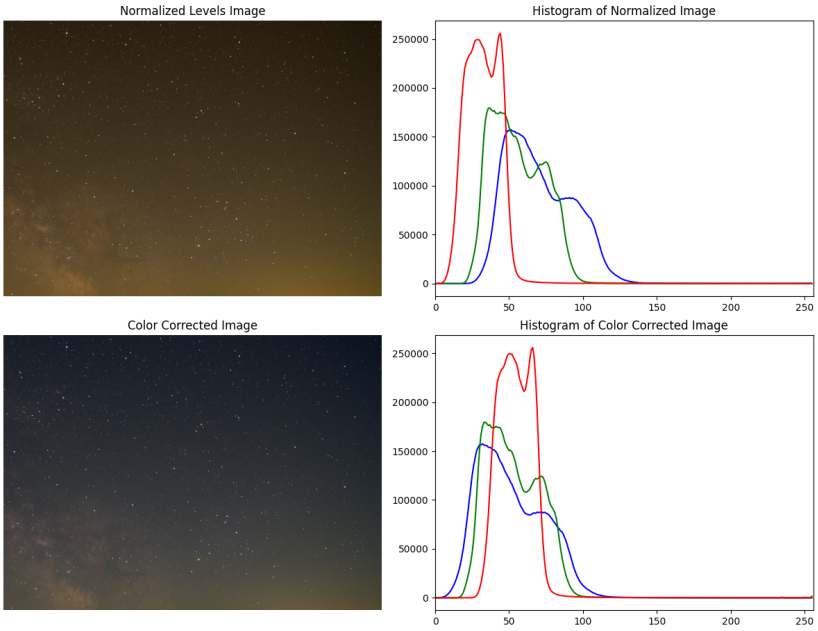
Obrázek 4.2: Nezpracovaný snímek

Následně jsem přešel do sekce Processing, kde jsem vybral tmavé a světelné snímky a cílovou složku pro kalibrované soubory. Po úspěšném vytvoření kalibrovaných souborů jsem využil možnost uříznout část snímku (strom) a provést skládání. Na obrázku 4.3 je možné vidět porovnání výřezu jednotlivého a kombinovaného snímku. Kombinovaný snímek obsahuje méně šumu a přechody jsou hladší.



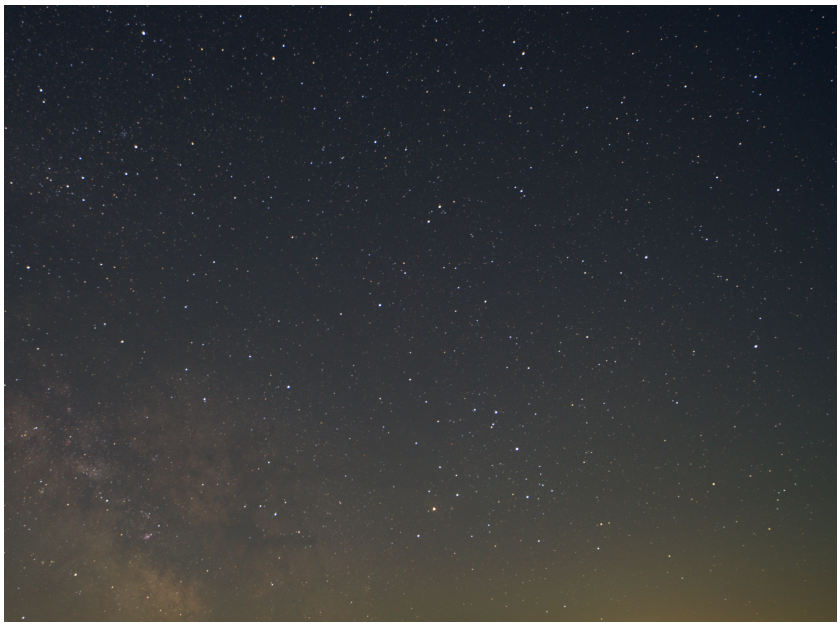
Obrázek 4.3: Sloučený snímek

Může být patrné, že snímek má žlutý nádech. Toto jsem opravil pomocí funkce Color correction. Vybral jsem výstupní složený snímek a spustil barevnou korekci. Výstup je možné vidět na obrázku 4.4.



Obrázek 4.4: Výstup baverné korekce

Výsledný obrázek odstranil barevný nádech. Vyzkoušení klienta považuji za úspěšné. Finální výstupní soubor je možné vidět na obrázku 4.5.



Obrázek 4.5: Výstupní obrázek

Kapitola 5

Závěr

V této práci jsem analyzoval aktuální řešení ovládání malých dalekohledů a navrhl vlastní klient pro ovládání DSLR a montáže. V porovnání s komerční i nekomerční konkurencí vlastní klient obsahuje méně specifických funkcí, které nejsou jednoduché na vývoj. Většina těchto funkcí ale není u základních pozorování nutná, a proto i tento základní klient stačí.

Na rozdíl od konkurence můj vlastní klient obsahuje i sekci pro zobrazení aktuálního počasí s předpovědí východu planet a světelného znečištění. Klient také umožňuje základní zpracování získaných souborů, což umožňuje vidět očekávaný finální snímek bez zdlouhavého ručního zpracování pomocí sofware jako PixInsight. Finální zpracování je u každého astronomického snímku lepší provádět ručně, takže nakonec je využití profesionálního softwaru stejně nutné. Pro prvotní představu o získaných datech se ale tato funkce v klientu hodí.

Klient jsem otestoval v terénu při snímání oblohy a požadovaná data jsem úspěšně získal. Považuji tento klient za využitelný při základním astrofotografování. Díky možnosti klienta využívat i pro zpracování dat a aktuální počasí ho využívám i při manuálním focení. Kód klientu můžu jednoduše editovat a přidávat nové funkce dle libosti, což mi jiný software neumožňuje.

Bibliografie

1. *INDI Technical documentation* [online]. [cit. 2024-04-31]. Dostupné z: https://docs.indilib.org.
2. STELLARMATE. *The StellarMate Plus Manual* [online]. [cit. 2024-04-31]. Dostupné z: https://www.stellarmate.com/help/.
3. PARKERSON, Stuart. *StellarMate Astrophotography Controller* [online]. solarastronomytoday.com [cit. 2024-01-23]. Dostupné z: https://solarastronomytoday.com/stellarmate-astrophotography-controller/.
4. ZWO. *ASIAIR Plus* [online]. [cit. 2024-04-31]. Dostupné z: https://astronomy-imaging-camera.com/product/asiair-plus.
5. KUBÁNEK, Petr. RTS2: a powerful robotic observatory manager. *Advanced Software and Control for Astronomy.SPIE*. 2006, č. 62741V, s. 562–571. Dostupné z DOI: 10.1117/12.672045.
6. HUSSER, Tim-Oliver. pyobs–An observatory control system for robotic telescopes. *arXiv preprint arXiv:2203.12642*. 2022. Dostupné také z: https://doi.org/10.48550/arXiv.2203.12642.
7. *libgphoto2* [online]. [cit. 2024-04-31]. Dostupné z: http://www.gphoto.org/proj/libgphoto2/.
8. *pyindi-client* [online]. [cit. 2024-04-31]. Dostupné z: https://github.com/indilib/pyindi-client/tree/master.
9. *Standart Properties* [online]. [cit. 2024-04-31]. Dostupné z: https://www.indilib.org/develop/developer-manual/101-standard-properties.html.
10. CARSTEN. *Night sky image processing – Part 6: Measuring the Half Flux Diameter (HFD) of a star – A Simple C++ implementation*. 2015. Dostupné také z: https://www.lost-infinity.com/night-sky-image-processing-part-6-measuring-the-half-flux-diameter-hfd-of-a-star-a-simple-c-implementation/.
11. SUMMER, Rob. Processing RAW Images in MATLAB. 2014. Dostupné také z: https://rcsumner.net/raw_guide/RAWguide.pdf.
12. YAGER, Will. *Intro to Computational Astrophotography* [online]. [cit. 2024-04-31]. Dostupné z: https://yager.io/Astro.html.

13. OPENCV. Histogram Equalization [online]. [N.d.] [cit. 2022-12-31]. Dostupné z: https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_histograms/py_histogram_equalization/py_histogram_equalization.html.

14. *INSTRUCTION MANUAL HEQ5/EQ6 MOUNT* [online]. [cit. 2024-04-31]. Dostupné z: https://inter-static.skywatcher.com/upfiles/en_download_caty01461887945.pdf.

Příloha A

Seznam zkratek

INDI	Instrument Neutral Distributed Interface
USB	Universal Serial Bus
IoT	Internet of Things
CCD	Charge Coupled Device
LAN	Local Area Network
FOV	Field Of View
FITS	Flexible Image Transport System
DC	Direct Current
RTS2	Remote Telescope System, 2nd Version
MONET	MOnitoring NEtwork of Telescopes
XML	eXtensible Markup Language
ASCOM	Astronomy Common Object Model
COM	Component Object Model
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
HFR	Half Flux Radius
RA	Right Ascension
DEC	Declination
AZ	Azimuth
ALT	Altitude
SD	Secure Digital (memory card)
API	Application Programming Interface
FWHM	Full Width at Half Maximum
RGB	Red, Green, Blue
NEF	Nikon Electronic Format (raw image file)
CFA	Color Filter Array

Příloha B

Seznam příloh

klient.py *Obsahuje celý kód klienta*