Focuser160MQ

Desenvolvimento do controlador do focalizador para o Telescópio Perkin-Elmer 1.60m

Ramon Carlos Gargalhone

LNA | OPD

Julho 2024

Sumário

1 Introdução

O Telescópio Perkin-Elmer 1.60m, o maior em solo brasileiro, é principal telescópio do Observatório do Pico dos Dias e tem servido à astronomia brasileira desde 1980, permitindo a captura de dados precisos e detalhados. Para maximizar a eficiência e a precisão das observações, é fundamental garantir que todos os componentes do telescópio operem de maneira coordenada e eficaz. Um desses componentes críticos é o focalizador, que ajusta a posição do espelho secundário para manter a imagem do objeto observacional em foco.

Com a intenção de modernizar o sistema de foco, foi elaborado um projeto pelas coordenadorias de Astrofísica (COAST), Engenharia e Desenvolvimento de Projetos (COEDP) e do Observatório do Pico dos Dias (COOPD), no qual era previsto o desenvolvimento de um controlador de *host* para o focalizador. Esse controlador gerencia a comunicação entre *softwares* de controle e aquisição do telescópio (TCSPD, S4GUI -*Acquisition system of the SPARC4 instrument* e OPDAcquisition) e o *hardware* do focalizador, assegurando que os comandos de foco sejam executados de maneira rápida e precisa. Para alcançar um desempenho otimizado e uma comunicação robusta, a implementação desse controlador utilizando ZeroMQ como uma plataforma de mensagens é uma abordagem moderna e eficaz.

Ao implementar o controlador de *host* do focalizador como um microsserviço utilizando ZeroMQ, vários benefícios são obtidos: modularidade, facilidade de manutenção, escalabilidade e a capacidade de integração com outros sistemas distribuídos do telescópio, dividindo em componentes menores e independentes, cada um responsável por uma funcionalidade específica, comunicando-se entre si através de interfaces bem definidas. Esta abordagem não apenas melhora a eficiência operacional do telescópio, mas também permite uma flexibilidade maior para futuras atualizações e expansões.

2 ZeroMQ

Nos sistemas de mensageria assíncrona e escalável como o ZeroMQ, a comunicação eficiente entre os componentes é crucial para garantir a flexibilidade, escalabilidade e desempenho. Existem vários padrões de comunicação que podem ser utilizados, cada um com suas próprias características e adequações para diferentes cenários. A seguir, uma breve descrição dos principais padrões: *Pub/Sub, Push/Pull, Req/Reply* e *Dealer/Router*[1].

ZeroMQ é uma biblioteca de mensagens avançada que facilita a implementação de comunicação entre processos e entre máquinas, sem a necessidade de configurar complexos servidores de mensagens. Esta tecnologia não requer dependências externas como um servidor de mensagens ou um *broker*, de forma que os soquetes ZMQ podem se comunicar diretamente uns com os outros.

2.1 Padrões de comunicação ZeroMQ

O ZeroMQ oferece diferentes tipos de soquetes que correspondem a diferentes padrões de comunicação. Soquetes (ou *sockets*, em inglês) são uma interface de comunicação que permite a troca de dados entre dois programas, seja em um mesmo dispositivo ou através de uma rede. Nesta seção serão apresentados quatro padrões e comunicação em ZeroMQ juntamente com exemplos de utilização em python.

2.1.1 Padrão *Pub/Sub*

É um padrão de um-para-muitos. Cenário onde múltiplos assinantes estão interessados em receber tipos específicos de mensagens ou eventos, como na distribuição de dados de sensores em uma rede IoT onde diferentes dispositivos se inscrevem para fluxos de dados relevantes.

2.1.2 Padrão *Push/Pull*

Utiliza um padrão de comunicação unidirecional e sem estado entre os nós que executam a tarefa e aqueles que processam essa tarefa. É um padrão de um-para-um ou um-para-muitos. Desta forma, a distribuição de tarefas entre diferentes os nós sem exigir respostas específicas ou confirmações, como em um cenário de computação distribuída.

2.1.3 Padrão *Req/Reply*

Quando é necessária uma comunicação síncrona onde um solicitante envia uma mensagem e espera por uma resposta específica. Bastante utilizada em sistemas de consulta/resposta, como consultas a um banco de dados ou requisições HTTP onde o cliente faz uma solicitação e aguarda a resposta do servidor. É tipicamente um-para-um e é ideal para cenários que requerem confirmação e resposta.

2.1.4 Padrão *Dealer/Router*

Quando é necessário implementar uma distribuição e orquestrar tarefas entre diferentes *workers*. Pode ser de muitos-para-muitos ou um-para-um. Assim, um nó central (roteador) distribui tarefas para *workers* disponíveis (*dealers*), como filas de trabalho ou balanceamento de carga entre vários *workers*.

3 Desenvolvimento

Para o desenvolvimento do controlador de *host* do focalizador do Telescópio Perkin-Elmer 1.60m, foi realizado um estudo sobre a estrutura da API do ASCOM Alpaca como referência[2]. O ASCOM Alpaca fornece uma base comum para desenvolvedores de software e fabricantes de equipamentos astronômicos[3] através de uma interface HTTP RESTful e por isso foi utilizado como referência. A partir desta referência, elaboramos uma lista de comandos específicos para o Focuser160 utilizando ZeroMQ, pois ele tende a ser mais eficiente em termos de consumo de banda, uma vez que ele usa um protocolo

binário otimizado e mantém conexões persistentes, o que minimiza a quantidade de dados de controle.

Os clientes permitidos de acessar esse *host* podem enviar os comandos, listados abaixo, utilizando o padrão REQ.

1. **HOME**: Move para o fim de curso de inicialização e zera *encoder*.
2. **CONNECT**: Basicamente faz uma requisição de status ao se conectar.
3. **STATUS**: Publica o status atual.
4. **MOVE**: Move o valor do foco para uma posição especificada (microns).
5. **FOCUSIN**: Move o valor do foco para dentro com velocidade (microns/s) como parâmetro.
6. **FOCUSOUT**: Move o valor do foco para fora com velocidade (microns/s) como parâmetro.
7. **HALT**: Interrompe o movimento.

O controlador *host* publica pelo padrão PUB o *status* do dispositivo no formato JSON, e cada cliente monitora de maneira independente pelo padrão SUB. A figura 1 exemplifica como o sistema opera. O *host* foi elaborado de forma a permitir uma configuração de parâmetros rápida e ser escalável, e também produzir um *log* detalhado das operações.

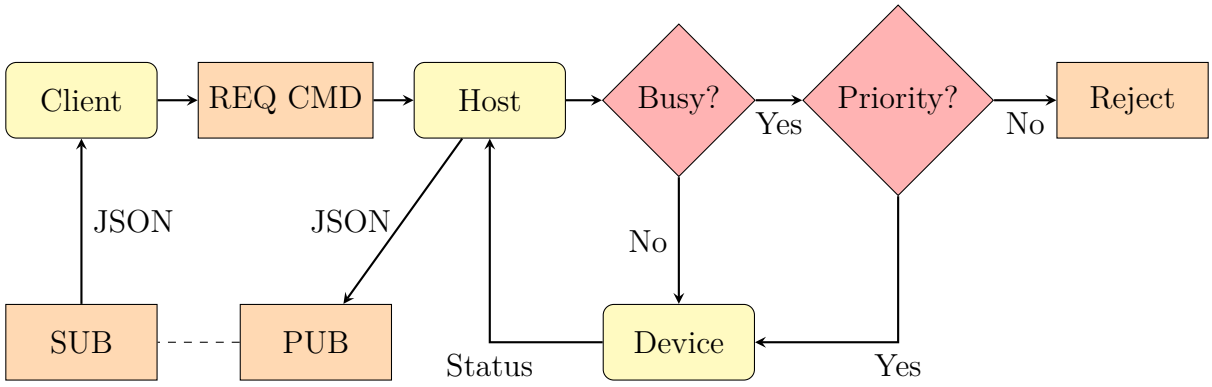

Figura 1: Fluxograma da comunicação Cliente-*Host*-Dispositivo

4 Conteinerização

Os contêineres encapsulam seu aplicativo, suas dependências e ambiente de tempo de execução. Esse isolamento garante que seu aplicativo se comporte de forma consistente, independentemente do sistema *host* subjacente.

4.1 Docker

Para conteinerização, foi utilizado o Docker, um software de código aberto usado para criação de aplicativos dentro de contêineres virtuais. A criação de um Dockerfile, com instruções de como a imagem Docker será montada é exemplificada no *script* 1.

```
FROM python:3.10.12
# stdout and stderr streams directly
ENV PYTHONUNBUFFERED=1
# Do not write .pyc bytecode files to disk
ENV PYTHONDONTWRITEBYTECODE=1
# Set the working directory in the container
WORKDIR /focuser160
# Install dependencies
COPY requirements.txt /focuser160/
RUN pip install --no-cache-dir -r requirements.txt
# Copy the rest of the application code to the container
COPY . /focuser160/
# Set the command to run application
CMD ["python", "main.py"]
```

Script 1: Exemplo de Dockerfile

Para executar um aplicativo GUI (*graphical user interface*) em um contêiner Docker, você precisará configurar o Docker para utilizar um servidor X para que o aplicativo possa exibir a interface gráfica no *host*. Um servidor X (ou servidor X11) é um componente central do sistema de janelas X, que fornece a base para ambientes gráficos em sistemas Unix e Unix-*like* (como Linux e BSD). É ele o responsável por desenhar e gerenciar janelas na tela, além de lidar com a entrada do usuário, como cliques do mouse e pressionamentos de teclas. Para isso, algumas dependências devem ser instaladas na imagem Docker, mostradas pelo *Script* 8.

```
RUN apt-get update && \
    apt-get install -y --no-install-recommends \
        libopencv-dev \
        libgl1-mesa-glx \
        libglib2.0-0 \
    && rm -rf /var/lib/apt/lists/*
```

Script 2: Dependências a serem instaladas

Em um sistema Linux, você pode usar o servidor X padrão (Xorg) que geralmente já está instalado. No Windows, você pode usar um servidor X como o Xming ou o VcXsrv.

4.2 Criando uma Docker *Image*

Para criar a imagem da aplicação, é necessário usar o comando *build*. o Script 3 apresenta a linha de comando necessária.

```
docker build --no-cache -t focuser160:tag .
```

Script 3: Exemplo de Comando Docker *build*

4.3 Rodando um Contêiner

Já o *Script* 4 exemplifica o comando *docker run*, que é usado para criar e iniciar um contêiner a partir de uma imagem Docker especificada.

```
docker run -it --rm
--env DISPLAY=host.docker.internal:0
--volume focuser_log:/focuser160/logs
--volume focuser_config:/focuser160/src/config
-p YOUR_HOST_IP:PUB_PORT:PUB_PORT
-p YOUR_HOST_IP:PULL_PORT:PULL_PORT
focuser160:tag
```

Script 4: Exemplo de Comando Docker *RUN*

Basicamente, ele executa a imagem do Docker, que pode conter uma aplicação, serviço ou ambiente de desenvolvimento, e cria um contêiner, que é uma instância em execução dessa imagem. Também são criados **volumes**, que são usados para persistir dados fora do ciclo de vida do contêiner. Eles permitem que dados sejam compartilhados entre contêineres ou mantidos mesmo após o contêiner ser excluído.

- Argumento –**env (ou -e)**: Define variáveis de ambiente dentro do contêiner;
- Argumento –**volume (ou -v)**: associa o diretório /focuser160/logs ao volume focuser_log;
- Argumento **-p**: Usado para mapear portas do contêiner para o *host*;
- Argumento **-it**: Combina dois outros argumentos: -i (interativo) e -t (terminal).

5 Utilização

5.1 Interface Geral

O Controlador, quando iniciado, já estabelece as comunicações com o motor e vincula o IP e as portas.

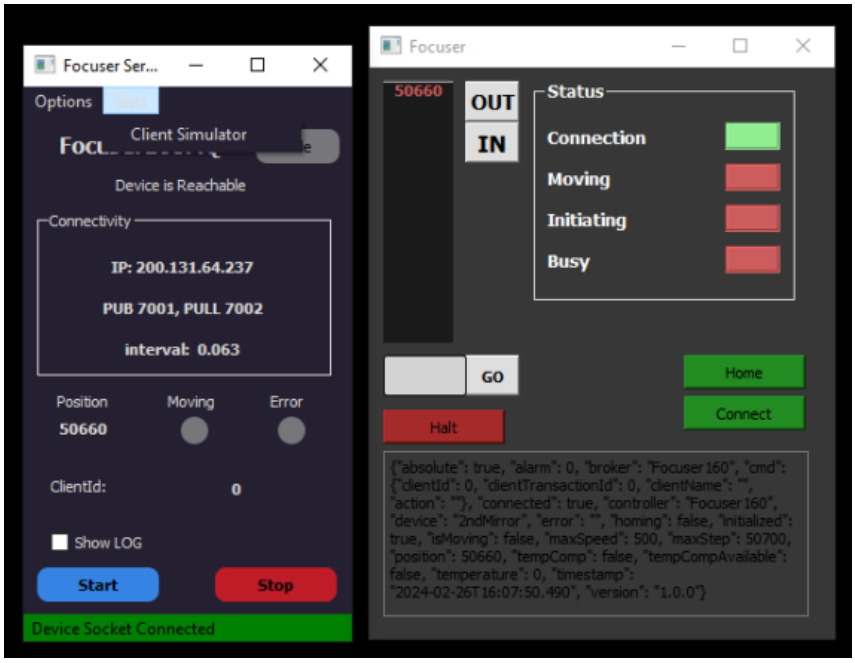

Figura 2: Gráfico Posição Foco x Largura à meia altura

- *Options* | *Config*: Contém as configurações do arquivo de configuração.
- *Tests* | *Simulator*: Executa uma versão local de um cliente para este controlador.
- *Status*: Localizado na parte superior, exibe se o dispositivo está "*reachable*"(alcançável) ou "*NOT reachable*"(não alcançável), indicando o status da comunicação com o dispositivo (motor do secundário).
- *Moving* LED: Quando verde, indica que dispositivo está em movimento.
- *Start Button*: Inicia o servidor. Utilize apenas se ocorrerem problemas.
- *Stop Button*: Desliga o servidor. Utilize apenas se ocorrerem problemas.
- Barra de Status: Verde quando a vinculação com o IP e portas foi estabelecida. Vermelha quando houve falha no vínculo.

5.2 *Config* e *Log*

O arquivo config.toml fornecido é um arquivo de configuração para a aplicação "Focuser160MQ", projetada especificamente para o Focalizador do Telescópio Perkin-Elmer. Este documento serve como um guia para entender e personalizar os parâmetros de configuração para a aplicação.

5.2.1 Configuração Geral [*General*]

A seção geral contém informações sobre a aplicação em si.

- ***title***: O título da aplicação, definido como "Focuser160MQ".

- ***name***: Um nome legível para o controlador do focador, definido como 'Focuser160'.
- ***version***: O número da versão da aplicação, definido como '1.0.0'.
- ***description***: Uma breve descrição da aplicação, definida como 'Interface para Focador Perkin-Elmer'.
- ***startup***: Flag booleana indicando se a aplicação deve iniciar na inicialização do sistema, definida como *true*.

5.2.2 Configuração do Dispositivo [*Device*]

A seção do dispositivo contém parâmetros de configuração relacionados ao Focador Perkin-Elmer.

- ***absolute***: *Flag* booleana indicando se o dispositivo usa posicionamento absoluto, definida como *true*.
- ***device_name***: Um nome atribuído ao dispositivo, definido como '2nsMirror'.
- ***deviceID***: Identificador único para o dispositivo, gerado usando GUID *Generator*.
- ***device_ip***: Endereço IP do Focalizador Perkin-Elmer, definido como '192.168.1.233'.
- ***device_port***: Número da porta para comunicação com o dispositivo, definido como 5001.
- ***max_step***: Valor máximo de passos para o focalizador, definido como 1000000.
- ***max_speed***: Velocidade máxima para o focador, definida como 0.
- ***maxincrement***: Incremento máximo para o focador, definido como 0.
- ***tempcompavailable***: *Flag* booleana indicando se a compensação de temperatura está disponível, definida como false.
- ***temp_comp***: *Flag* booleana indicando se a compensação de temperatura está ativada, definida como false.
- ***stepsize***: Tamanho do passo para o focador, definido como 0.
- ***encoder2microns***: Fator de conversão de unidades de codificador para micrômetros, definido como 42.2047244.

5.2.3 Configuração de Rede [*Network*]

A seção de rede contém parâmetros de configuração para a comunicação em rede.

- ***ip_address***: Endereço IP para a aplicação, definido como '192.168.1.100'.
- ***port_pub***: Número da porta para publicação de dados, definido como 7001.

- ***port_rep***: Número da porta para comandos, definido como 7002

.

5.2.4 Configuração do Banco de Dados [Database]

A seção de banco de dados contém parâmetros de configuração para a conectividade com o banco de dados. Ainda não implementado, esperando definições.

- ***db_name***: Nome do banco de dados, definido como '*db_name*'.
- ***db_user***: Nome de usuário para autenticação no banco de dados, definido como '*username*'.
- ***db_password***: Senha para autenticação no banco de dados, definida como '*password*'.

5.2.5 Configuração de *Log* [*Logging*]

A seção de log contém parâmetros de configuração para o registro de eventos.

- ***log_level***: Nível de *log*, definido como 'INFO'.
- ***log_to_stdout***: *Flag* booleana indicando se os *logs* devem ser exibidos no console, definida como false.
- ***log_max_size_mb***: Tamanho máximo dos arquivos de *log* em megabytes, definido como 5.
- ***log_num_keep***: Número de arquivos de *log* a serem mantidos, definido como 10.

5.3 Comandos

Os clientes devem enviar comandos no seguinte formato JSON:

```
{
    "clientId": 1234,
    "clientTransactionId": 9876,
    "clientName": "TCSPD",
    "action": "NOME\_DO\_COMANDO"
}
```

Script 5: Estrutura de mensagem REQ

Substitua "NOME_DO_COMANDO"pelo comando desejado e forneça parâmetros adicionais conforme necessário. "NOME_DO_COMANDO"deve ser todo maiúsculo.

Do Alpaca:

- ***clientId***: *Client's unique ID. (1 to 4294967295). The client should choose a value at start-up, e.g. a random value between 1 and 65535, and send this on every*

transaction to associate entries in device logs with this particular client. Zero is a reserved value that clients should not use.

- ***clientTransactionId****: Client's transaction ID. (1 to 4294967295). The client should start this count at 1 and increment by one on each successive transaction. This will aid associating entries in device logs with corresponding entries in client side logs. Zero is a reserved value that clients should not use.*

```
{
    "clientId": 1234,
    "clientTransactionId": 9876,
    "clientName": "TCSPD",
    "action": "MOVE=10"
}
```

Script 6: Comando de exemplo para mover o *Focuser* para a posicao 10

O comando HALT é executado apenas se o ID do cliente corresponder ao ID do cliente armazenado no status da aplicação.

5.4 Resposta do servidor e *status*

Ao enviar uma solicitação ao controlador, a resposta contém uma *STRING*: "ACK"se o comando for aceito; "NAK", caso o comando seja rejeitado. O *status* também é uma *STRING*, que pode ser convertida para um objeto JSON com informações atualizadas sobre o estado do dispositivo. Aqui está a descrição de cada campo presente na resposta:

- absolute: **BOOL** Indica se o Focalizador está configurado para movimento absoluto (arquivo *Config*).
- alarm: **BOOL** *Flag* de alarme atual.
- cmd: **STRING** Comando atual ou última ação executada.
- connected: **BOOL** Indica se o Focalizador está conectado.
- controller: **STRING** Identifica o *HOST* (ex: S4GUI).
- device: **STRING** Identifica o dispositivo que ele aciona (“ex: Mirror2).
- error: **STRING** Mensagem de erro ou descrição vazia se não houver erro.
- homing: **BOOL** Indica se o focalizador está atualmente em processo de homing.
- initialized: **BOOL** Indica se rotina INIT foi realizada.
- isMoving: **BOOL** Indica se está atualmente em movimento.
- maxSpeed: **INT** Velocidade máxima do Focalizador em microns/s (arquivo *Config*).
- maxStep: **INT** Número máximo de passos permitidos (arquivo Config).

- position: **DOUBLE** Posição atual do Focalizador em mícrons.
- tempComp: **BOOL** Compensação de temperatura (arquivo Config).
- tempCompAvailable: **BOOL** Indica se a compensação de temperatura está disponível (origem arquivo Config).
- temperature: **DOUBLE** temperatura do dispositivo (futura implementação).
- timestamp: **STRING** Data/hora da resposta em formato ISO.

```
{
    "absolute": true,
    "alarm": false,
    "cmd": {
        "clientId": 1234,
        "clientTransactionId": 9876,
        "clientName": "S4GUI",
        "action": "MOVE=10"
        },
    "connected": true,
    "controller": 'Focuser160',
    "device": '2ndMirror',
    "connected": true,
    "error": '',
    "homing": false,
    "initialized": true,
    "isMoving": false,
    "maxSpeed": 150,
    "maxStep": 1000000,
    "position": 1549,
    "tempComp": false,
    "tempCompAvailable": false,
    "temperatur": 22,
    "timestamp":'2000-10-31T01:30:00.00',
}
```

Script 7: JSON de *status*

6 Conclusão

Compreender os padrões de comunicação do ZeroMQ é essencial para desenvolver sistemas distribuídos eficientes e escaláveis. *Pub/Sub, Push/Pull, Req/Reply* e *Dealer/-Router* demonstram a flexibilidade e robustez do ZeroMQ, cada um com suas características e aplicações específicas.

Pub/Sub destaca-se pela simplicidade e capacidade de escalar bem, ideal para disseminar informações a múltiplos assinantes. *Push/Pull* facilita a distribuição de tarefas em sistemas de processamento paralelo com seu modelo unidirecional e sem estado. *Req/-Reply* é fundamental para comunicações que requerem confirmação e resposta, oferecendo uma abordagem estruturada e confiável. *Dealer/Router* permite estratégias complexas de

balanceamento de carga e roteamento, sendo poderoso para arquiteturas que demandam alta flexibilidade e controle.

Ao explorar esses padrões e suas implementações em Python, esta nota técnica evidenciou a capacidade do ZeroMQ de se adaptar a diversos requisitos de comunicação que pode ser muito importante no desenvolvimento de aplicações no âmbito do Laboratório Nacional de Astrofísica.

Referências

[1] ZMQ. *Get started*. Disponível em: <https://zeromq.org/get-started/?language=python#> Acessado em: 11/06/2023.

[2] Ascom alpaca device api. https://ascom-standards.org/api. Accessed: 11/06/2024.

[3] Motivation for ascom - why does it exist? https://ascom-standards.org/About/Index.htm. Accessed: 11/06/2024.

7 Apêndice A

Arquivo config.toml:

```
title = "Focuser160MQ"

[General]
name = "Focuser160"
version = "0.1.0"
description = "Interface for Perkin-Elmer Focuser"
startup = true

[Device]
absolute = true
device_name = '2ndMirror'
deviceID = '3285e9af-8d1d-4f9d-b368-d129d8e9a24b' # https://
    guidgenerator.com/online-guid-generator.aspx
device_ip = '192.168.1.233'
device_port = 5001
encoder2microns = 42.2047244
max_step = 50700 # microns
max_speed = 500 # microns/sec
max_increment = 0
tempcompavailable = false
temp_comp = false
speed_security = 215000
speedFactor = 428 # Converts units in microns to motor units
step_size = 0

[Network]
ip_address = "200.131.64.237"
port_pub = 7001
port_rep = 7002

[Logging]
log_level = "INFO"
log_to_stdout = false
log_max_size_mb = 10
log_num_keep = 10
```

Script 8: Config.toml