



Relatório Técnico de Projeto

Simulador de Pequenos Satélites: Hardware e Software

Informações do Projeto:

Autores	Flavio Amaral e Silva Lukas Lorenz de Andrade Matheus Ribeiro
Universidade	Universidade de Brasília (UnB), Distrito Federal, Brasília
Laboratório	Laboratory of Simulation and Control of Aerospace Systems (LSCSA)
Objetivo	Estabelecimento de metodologia de testes e atualização de software e hardware na <i>air bearing table</i> do Simulador de Pequenos Satélites.
Linguagens de programação e Softwares	C, C++, Python 3, Matlab, Eagle 9.5.2 e Fusion 3d.

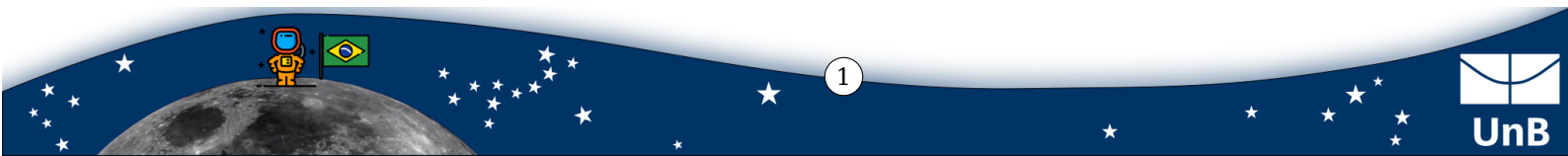
Contatos:

Gmail	Github/Gitlab do Projeto	Github/Gitlab	Autor
flavio.asilva@gmail.com	ADCV-Rpi Repositorio	@flavioasilva2	Flavio
lukaslorenzdeandrade@gmail.com	ADCV-Rpi Repositorio	@lorenz-lukas	Lukas
matheusrbv99@gmail.com	ADCV-Rpi Repositorio	@matheusr1321	Matheus

Conteúdo

Relatório Técnico de Projeto

Simulador de Pequenos Satélites: Hardware e Software	1
I - Metodologia de Testes	2
Motivação	2
Estrutura do Simulador - Mark I	2
Estrutura do Simulador - Mark III	3
Padronização de Arquivos	4
II - Hardware	5
Motivação	5
Atualização da Eletrônica	6
(a) Motores de Passo	6
(b) Motores EC	8
(c) Sensor de Orientação	11
(d) Integração Mockup 2U	11
(e) Placa de Circuito Impresso	12



III - Software	14
Motivação	14
Fluxo dos Programas	14
(a) Fluxo no Rpi	15
(b) Fluxo no computador - ADCV	16
Dependência do Sistema	17
V - Trabalhos Futuros	18
Apêndice	19
Códigos	19
Pinout's, Conexões e Datasheets	19
Diagramas	23
Bibliografia	24

I - Metodologia de Testes

Motivação

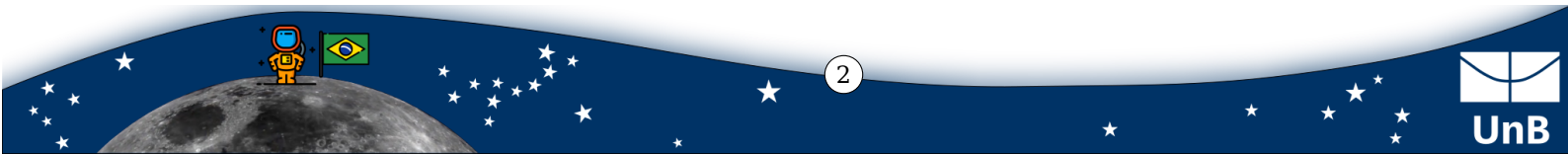
A antiga plataforma do simulador de pequenos satélites não era modular, ou seja, a inserção de novos módulos, como atuadores ou sensores, era de extrema dificuldade devido a escassez de documentação técnica, sub utilização de componentes e dificuldade de pareamento do Xbee. Dessa forma, a atualização do hardware/estrutura mecânica e do software visa desacoplar módulos, de forma facilitar a inserção ou atualização de novos módulos. Além disso, a padronização de uma metodologia para a realização de testes no simulador será estabelecida devido a necessidade.

Estrutura do Simulador - Mark I

O sistema do simulador [1, 2, 3, 4] era composto por 2 computadores e 2 Arduinos Mega (Figura 1) de forma que o PC1 era responsável pela Propagação Orbital e o sistema do ADCV (Attitude Determination by Computer Vision), e o PC2 por gerar a ação de controle que seria executada pelo Mockup. Todo o sistema se comunicava utilizando o Xbee como meio.

O ciclo de missão da arquitetura pode ser exemplificado de forma macro da seguinte forma:

- No Matlab, o PC1 gera o valor de corrente nas bobinas da Gaiola de HelmHoltz a partir da leitura do magnetômetro HMR-2300 posicionado no centro da gaiola;
- O valor de corrente correspondente é enviado para as fontes que estão conectadas aos enrolamentos das bobinas, produzindo um campo magnético resultante;
- Paralelamente, no Visual Studio®, há a comunicação com um servidor TCP-IP que processa o frame da câmera no eixo Z e envia os dados de atitude estimados para a IDE. Estes valores são enviados pelo Xbee para a mesa (*air bearing table*);
- Após o recebimento dos dados, o Arduino mega coleta os dados da IMU e calcula um valor de erro de atitude, considerando a atitude determinada pelo ADCV como a verdadeira (*ground truth*).
- Em seguida, é enviado, por Xbee, para o PC2, onde roda o controlador e a ação de controle é gerada, simulando a atuação pela estação solo. A ação é enviada para o Arduino Mega presente na parte superior da mesa, o qual comunicará com o Mockup 2U e atuará os magnetorques.



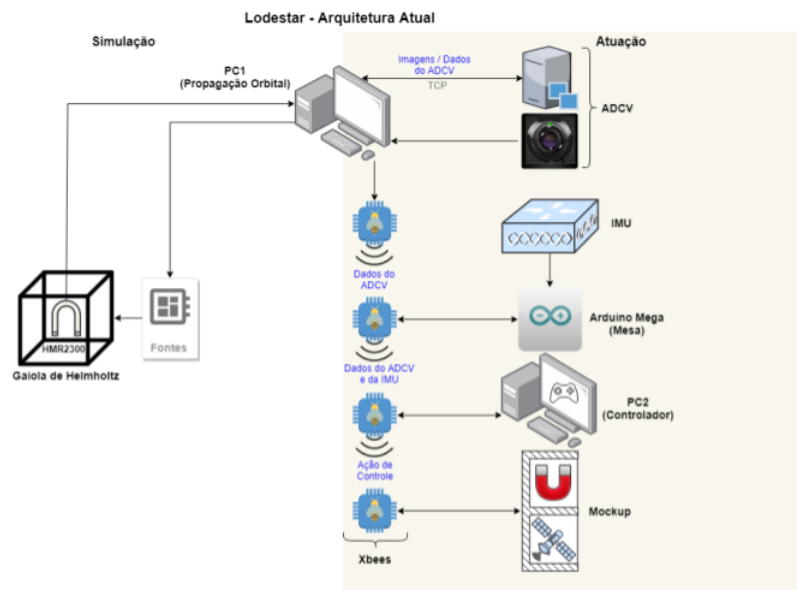


Figura 1: Esquemático da antiga arquitetura do Simulador de Pequenos Satélites

Um ponto a ser destacado é que nesse sistema não havia a implementação do atuador por roda de reação na mesa nem no satélite. A implementação desse atuador, por exemplo, ou a troca dos módulos Xbee se dava de forma custosa, ou seja, sendo necessário mudar muito na estrutura dos códigos para fazer funcionar e propiciando o uso de soluções não convencionais.

Estrutura do Simulador - Mark III

A nova estrutura consiste na otimização dos componentes utilizados de forma a explorar recursos dos sistemas operacionais, como Threads e Semáforos, e a integração de módulos por meio de bibliotecas, ao invés da incorporação direta no código, como era realizado anteriormente, seguindo padrões de engrenharia de software. A complexidade do sistema aumentou, apesar da modularidade e escalabilidade do sistema ser possível.

Dessa forma, o novo sistema (Figura 2) consiste na separação de módulos: simulação e atuação. Além disso, a sistematização permite o uso de recursos em diferentes tempos, podendo ser extraído a máxima capacidade dos componentes na hora dos testes, ou seja, dividiu-se o teste em três fundamentais etapas: calibração do sistema, setup e missão. Este processo pode ser exemplificado da seguinte forma:

- A etapa de calibração não tem requisitos de sistema em tempo real uma vez que o ambiente é ajustado para a realização da missão. Assim, a mesa é calibrada com o auxílio dos motores de passo e massa móvel ou manualmente, a IMU é calibrada e, futuramente, o tensor de inércia da mesa com e sem o Mockup da missão será estimado e a função de transferência da gaiola também. Assim, esta etapa será programada na linguagem Python;
- Na etapa de Setup, com esses parâmetros, os arquivos com os parâmetros dos controladores projetados em Matlab para o Mockup serão enviados para a mesa, caso o controlador seja embarcado, se não este ficará no computador que terá enlace de comunicação com a mesa durante a missão. Além disso, um ambiente é configurado no Raspberry Pi presente na mesa com todos os arquivos necessários para a missão via SSH (Python);
- Durante a missão, etapa com requisitos de tempo real, o processamento do computador fica exclusivo para o sistema de ADCV (C++) que comunica, via TCP, com o Raspberry Pi 3B (Rpi) na mesa. Assim, os dados de atitude gerado pelas câmeras em 3

eixos é comparado com os dados da IMU capturados pelo Rpi (C) para a ação do controlador embarcado. No caso em que a ação de controle é gerada pela estação solo, o computador recebe os dados de telemetria do Mockup e gera a ação de controle correspondente, levando em consideração os atrasos medidos na fase de calibração, que será enviada para os atuadores conectados ao Rpi da mesa.

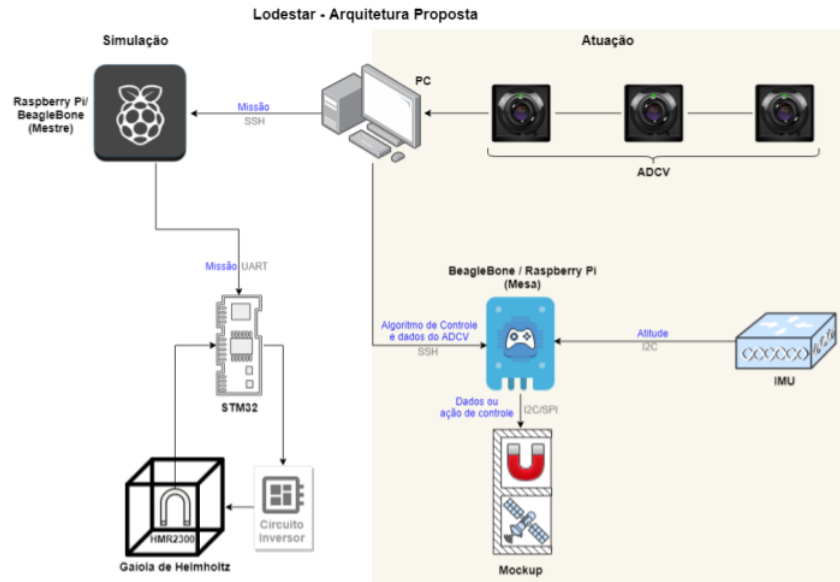


Figura 2: Esquemático da nova arquitetura do Simulador de Pequenos Satélites

Padronização de Arquivos

A fim de padronizar os meios de compartilhamento de dados, formalizou-se o JSON como padrão de arquivos já que este é independente de linguagem e arquitetura de hardware. Assim, foi escrito códigos exemplos nas linguagens C, C++, Python 3 e Matlab a fim de se ter um formato fixo, definindo-se campos e um cabeçalho, que deverá ser usado para Log de missões e testes em laboratório. Dessa forma, especifica-se um padrão em que os campos correspondentes devem ser preenchidos conforme a situação em questão.

Dessa forma, o código a seguir representa um arquivo Json modelo aplicado para o caso do ADCV, onde os campos *Author*, *Local* e *Project* devem ser devidamente preenchidos pelo usuário e os dados representados pela "CAMX", "CAMY", "CAMZ" de acordo com a funcionalidade atribuída ao Json. Ou seja, esses objetos podem ser facilmente trocados por valores de Temperatura, pressão, tempo de transmissão ou ciclo de código, atitude da IMU, entre outros, apenas adaptando os códigos exemplos e bibliotecas implementadas nas linguagens referidas.

```

1 {
2   "AttitudeADCV": {
3     "CAMX": [
4       {
5         "absTime": 1585883813055,
6         "x": 3.299999952316284,
7         "y": 2.200000047683716,
8         "z": 1.100000023841858
9       },
10      {
11        "absTime": 1585883813075,
12        "x": 3.299999952316284,
13        "y": 2.200000047683716,
14        "z": 1.100000023841858

```

```

15     },
16   ],
17   "CAMY": [
18     {
19       "absTime": 1585883813055,
20       "x": 3.299999952316284,
21       "y": 2.200000047683716,
22       "z": 1.100000023841858
23     },
24     {
25       "absTime": 1585883813075,
26       "x": 3.299999952316284,
27       "y": 2.200000047683716,
28       "z": 1.100000023841858
29     },
30   ],
31   "CAMZ": [
32     {
33       "absTime": 1585883813055,
34       "x": 3.299999952316284,
35       "y": 2.200000047683716,
36       "z": 1.100000023841858
37     },
38     {
39       "absTime": 1585883813075,
40       "x": 3.299999952316284,
41       "y": 2.200000047683716,
42       "z": 1.100000023841858
43     },
44   ],
45 ],
46 "Author": "LODESTAR STAFF",
47 "Local": "Brasilia, UnB, SG11 - LODESTAR",
48 "LocalTime": "Fri Apr 3 00:16:41 2020\n",
49 "Project": "Adcv Testbed integration",
50 "UTCTime": "Fri Apr 3 03:16:41 2020\n"
51 }

```

II - Hardware

Motivação

A implementação da atualização do Hardware surgiu da necessidade de um sistema de atuação do controle da mesa simuladora, visto que ela apenas possuía um sistema de atuação para balanceamento, realizado pelos motores de passo . Para isso, motores EC foram utilizados, acionados por um Raspberry Pi. Além disso, a nova proposta também visa resolver problemas como a sub-utilização de componentes, atraso de comunicação entre módulos do sistema, por meio do uso de apenas um micro-processador (RPi) e um computador externo, ao passo que anteriormente se utilizava dois arduinos e dois computadores. Por fim, visa-se também, por meio deste relatório, resolver o problema da falta de documentação da implementação anterior, tornando acessível e de fácil entendimento o funcionamento da mesa de simulação atual, assim como os desafios, problemáticas per-



sistentes e recomendações do que ainda precisa ser melhorado.

Atualização da Eletrônica

À princípio, a mesa de simulação contava com motores de passo distribuídos nos 3 eixos para o balanceamento, sensor de orientação e um sistema eletrônico no qual se conectava o mockup ao computador externo por meio de dois arduinos e comunicação XBee, conforme simplifica a imagem da metodologia(linkar imagem). Já na nova eletrônica do sistema (Figura 3, serão adicionados três motores EC para realizar o controle do CubeSat nos três eixos por torque. Eles serão controlados pelo micro-processador RPi, que também controlará o sensor de orientação e os motores de passo. Frisa-se que esta atualização do Hardware é apenas para atuação da mesa simuladora, o controle e simulação não foram feitas nesta versão, permanecendo para uma futura implementação.

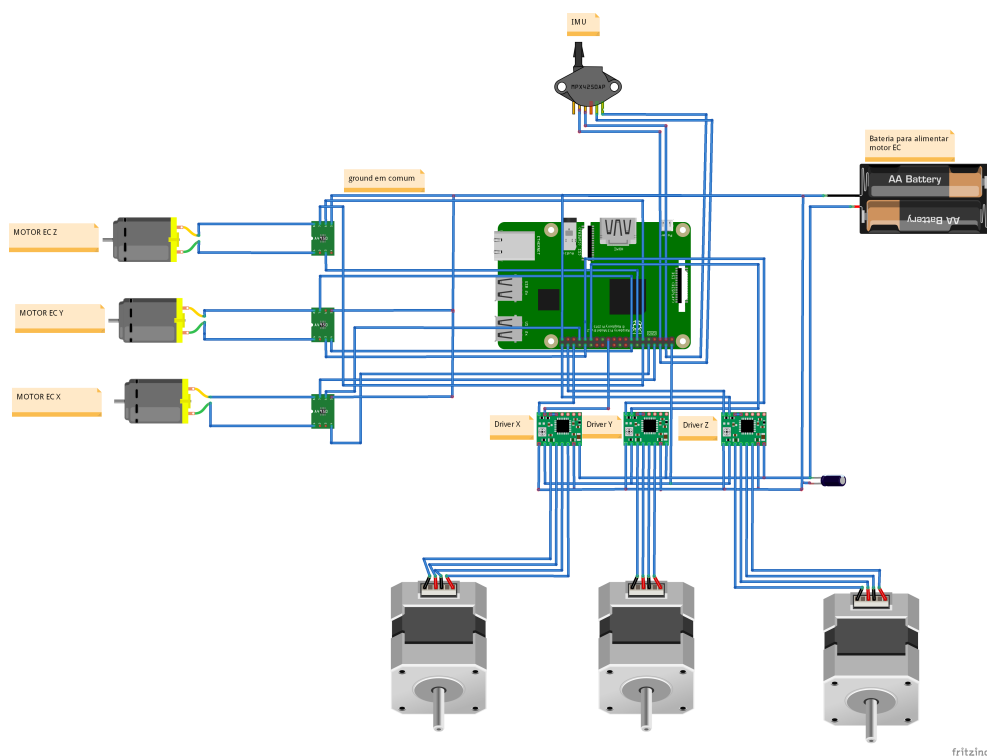


Figura 3: Diagrama elétrico da atualização do Hardware

Para melhor explicação, dividiu-se em quatro subseções: Motores de Passo, Motores EC, sensor de orientação (IMU) e PCB.

(a) Motores de Passo

Primeiramente, os motores de passo utilizados são os mesmos do modelo antigo do simulador: Kysan 1124090 NEMA 17 Stepper Motor ([Datasheet Motor de passo](#)). O driver utilizado é o A4988 Stepper Motor Driver Carrier ([Datasheet Driver Motor de passo](#))(Figura 4). Antes de usá-lo, deve-se ajustar a corrente máxima do driver. Para isso, calcula-se a corrente máxima do driver através da seguinte fórmula:

$$I_{max} = V_{ref} / (8 \times 0,1)$$

Logo, V_{ref} (Tensão no multímetro) deve estar no intervalo 0,75V-0,797V. Para ajustá-la em tal valor, alimenta-se o driver (Ver datasheet), conecta-se o Ground do multímetro no



Ground lógico do Driver e o Vcc do multímetro no parafuso do driver e girá-lo até permanecer no intervalo do Vref. Acrescenta-se que se utilizou uma chave condutora de eletricidade adequada para girar o parafuso e uma cabo-licate no conector Vcc do multímetro.

É importante salientar que os motores de passo servem para o balanceamento da mesa nos 3 eixos, o que requer uma alta precisão de passos do motor, logo, recomenda-se utilizar os microsteps. Todavia, como o RPi não possui GPIO's suficiente para o uso dos motores de passo com microsteps (requer mais três entradas), não os utilizou nesta atualização. Porém, corrobora-se que, se possível, em uma futura implementação da mesa, seja utilizado a configuração de microsteps.

Ao longo da implementação, percebeu-se que os drivers dos motores de passo danificam-se facilmente por falta de circuitos de proteção adequado, tornando-os um dos modelos mais baratos no mercado. Então, torna-se imprescindível o uso de capacitores para dividir a corrente e evitar flutuações na tensão de alimentação do driver e de alimentação dos motores de passo. Nesta versão, utilizou-se 3 capacitores 33nF 50V em paralelo. Porém, salienta-se que, se possível, em uma nova implementação da mesa, seja utilizado drivers mais sofisticados para evitar qualquer tipo de problema nos motores de passo.

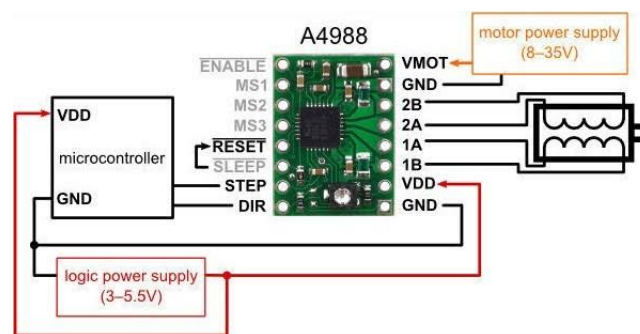


Figura 4: Esquemático Driver A4988

Ademais, no protótipo do projeto, percebeu-se que os motores faziam um ruído relativamente alto ao se conectar à alimentação, embora imóveis. Este problema deve-se aos jumpers utilizados, sujeitos a movimentos diversos, que causavam interferência no motor. O problema cessou ao se utilizar jumpers curtos imóveis e, por fim, a PCB.

Além disso, um movimento não-proposital, descontrolado e barulhento do motor era notado em algumas situações. Acredita-se que este problema era devido tanto aos jumpers utilizado como um defeito no header do RPi, visto que o problema cessava quando se o reiniciava.

Já com relação ao código para acionamento do motor de passo ([código-base Motor de passo](#)), fê-lo em Python, visto que o acionamento dos motores de passo para balanceamento não requer um controle em tempo real. Com efeito, utilizou-se a biblioteca WiringPi (já inclusa no RPi, mas se deve certificar se está atualizada) para relacionar os pinos do raspberry PI com a biblioteca utilizada. Para obter tal relação (Figura 17), deve-se digitar no terminal "gpio readall". Nesta figura, a coluna BCM mostra os GPIO's do pinout do raspberry (Figura 18) e a coluna wPi mostra o valor do GPIO do pinout na biblioteca WiringPi, que deve ser usado no código para estabelecer as conexões. Não obstante, este não é o código final, que foi incluído em uma biblioteca, assim como o código do motor de passo e do sensor de orientação.

Então, estabeleceu-se as conexões para cada um dos motores de passo denominados X, Y e Z, referentes, respectivamente, ao eixo que atuam (Tabela 1).

Tabela 1: Conexões Motor de passo - Raspberry PI

DIR STEPPER MOTOR X	Pin 20
STEP STEPPER MOTOR X	Pin 21
DIR STEPPER MOTOR Y	Pin 25
STEP STEPPER MOTOR Y	Pin 8
DIR STEPPER MOTOR Z	Pin 23
STEP STEPPER MOTOR Z	Pin 24

Por fim, levantou-se uma tabela (Tabela 2) com as características elétricas do Driver motor de passo:

Tabela 2: Características Elétricas do Driver Motor de passo

Load Supply Voltage Range	8V-35V
Logic Supply Voltage Range	3V-5.5V
Motor Supply Current Operating	0-2mA
Logic Supply Current	0-5mA

(b) Motores EC

Os motores EC e respectivos drivers utilizados foram os disponíveis no laboratório LO-DESTAR, adquiridos por integrantes passados, mas não utilizados. O modelo do motor utilizado nesta atualização foi o Maxon 397172 EC (Figura 19, [Datasheet Motor EC](#)) com o driver Escon 36/3 EC ([Datasheet Driver Motor EC](#)), onde o pinout's podem ser encontrados nas figuras 20, 21 e 22.

É importante ressaltar que o motor escolhido exige uma tensão nominal de 24V! Essa tensão é extremamente alta, exigindo baterias de várias células em série para suprir tal potência, aumentando consideravelmente o peso do simulador como um todo. Visto que os motores devem gerar um torque para o simulador rotacionar em torno do próprio eixo, a máxima redução da massa do sistema é crucial, pois afeta diretamente no torque. Logo, se possível, em uma futura implementação deve-se comprar motores (de preferência da mesma fabricante do driver Maxon) com uma tensão nominal baixa compatíveis com os drivers disponíveis em laboratório.

Antes de configurar o driver, deve-se ser cauteloso ao manuseá-lo pois o driver é eletroestático, podendo danificá-lo ao tocar em seus componentes internos. Se possível, deve-se utilizar uma luva.

Dito isso, o primeiro passo é instalar o software [Escon Studio](#), em seguida, conectar o driver ao PC por meio de um cabo USB. Ao abrir o programa, inicializar-se-á, automaticamente, o startup wizard no qual reconhecerá o driver conectado. Então, deve-se configurá-lo conforme a figura 5.

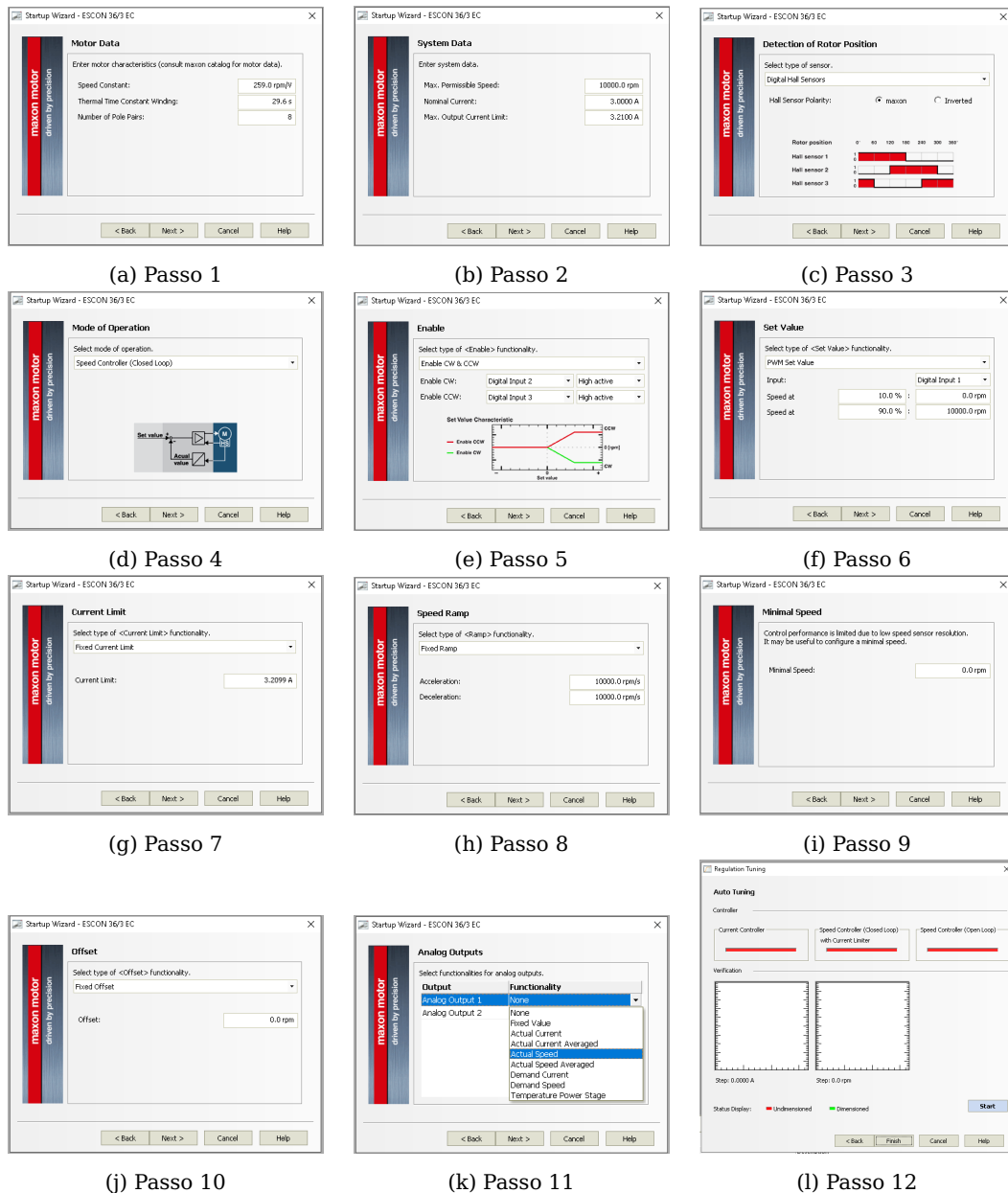


Figura 5: Passo a passo de configuração do driver do motor EC.

Fatos a se considerar:

- No passo 5, basicamente, tem-se que a configuração do motor de girar no sentido horário ou anti-horário deve ser feita por meio do acionamento HIGH do sinal. Torna-se impossível fazer acionamento LOW pelo fato de utilizar-se um raspberry PI.
- No passo 6, tem-se que o acionamento para o motor girar será feito por meio de PWM, porém, recomenda-se que, se possível, a próxima versão seja feita por meio de "Analog Set Value".
- No passo 11, uma das funcionalidades possíveis nas saídas analógicas é medir a velocidade atual dos motores por meio dos sensores Hall. Saber a velocidade da roda de reação é imprescindível para a simulação do sistema. Porém, visto que o RPi não possui GPIO's analógicos, há necessidade de um conversor A/D que, nesta versão, não foi utilizada. Então, se possível, em uma versão posterior, deve-se utilizar estes sensores de velocidade.

Observação: Não esquecer de conectar o GROUND digital do driver ao RPi.

É importante frisar que, como se está alimentando o motor com alta tensão (24V-30V), caso haja um curto nos pinos de alimentação do driver, este queimará permanentemente, visto o frágil circuito de proteção.

Ademais, deve-se notar que o RPi 3b+ só possui 4 GPIO's PWM (GPIO's 12, 13, 18 e 19). Porém, não se sabe a efetividade caso haja necessidade dos 3 motores EC atuarem ao mesmo tempo com velocidades diferentes, visto que, há uma probabilidade de apenas 2 PWM serem compartilhados para os 4 GPIO's. Porém, realizou-se com sucesso um teste das 3 rodas girando ao mesmo tempo com a mesma velocidade.

Então, estabeleceu-se as conexões para cada um dos motores EC denominados X, Y e Z, referentes, respectivamente, ao eixo que atuam (Tabela 3).

Tabela 3: Conexões Motores EC - RPi

MOTOR EC X PWM	Pin 12
MOTOR EC X CW	Pin 16
MOTOR EC X CCW	Pin 7
MOTOR EC Y PWM	Pin 18
MOTOR EC Y CW	Pin 14
MOTOR EC Y CCW	Pin 15
MOTOR EC Z PWM	Pin 13
MOTOR EC Z CW	Pin 19
MOTOR EC Z CCW	Pin 26

Por fim, levantou-se uma tabela (Tabela 4) com as características elétricas do Driver motor EC:

Tabela 4: Características Elétricas do Driver Motor EC

Nominal operating voltage Vcc	10V-36V
Output voltage (max.)	0.98 x Vcc
Output current Icont / Imax (<4 s)	2.7A / 9A

Já com relação ao [Código-base do Motor EC](#), tem-se que ele foi feito em C visto o desempenho (rapidez) desta linguagem de programação para um sistema de controle. Não obstante, este não é o código final, que foi incluído em uma biblioteca, assim como o código do motor de passo e do sensor de orientação.

A fim de proteger o driver contra movimentos indesejados, fez-se um case de proteção em impressão 3D (Figura 6) no software Fusion 360 da Autodesk.([CADs files case](#))

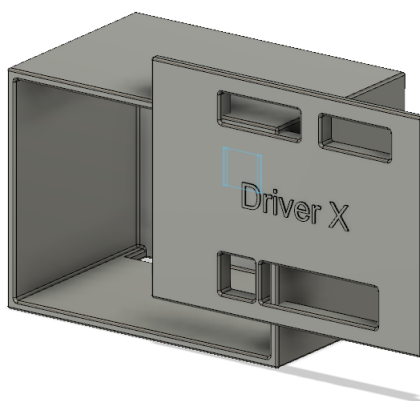


Figura 6: Driver case CAD

(c) Sensor de Orientação

Primeiramente, o sensor de orientação utilizado foi o adafruit BNO055 absolute orientation ([Datasheet Sensor de Orientação](#)). Para conectá-lo ao RPi, deve-se conectar os pinos SDA e SCL do sensor aos respectivos do Raspberry. Pode-se alimentá-lo tanto com o 3.3V quanto com o 5V. Reitera-se que o código final do sensor de orientação encontra-se em uma biblioteca, assim como o código do motor de passo e do motor EC.

Por fim, levantou-se uma tabela (Tabela 5) com as características elétricas do Sensor de Orientação:

Tabela 5: Características Elétricas do Sensor de Orientação

Supply Voltage (only Sensors) Vdd	2.4V-3.6V
Supply Voltage (μ C and I/O Domain) VddIO	1.7V-3.6V
Total supply current (normal mode at TA)	0-12.3mA

(d) Integração Mockup 2U

O mockup 2U [3] não possui microcontrolador embarcado, de forma que em seu interior apenas possui fios para as conexões, dois drivers TB6612FNG (Figura 8), as três bobinas e dois conectores do tipo DB9 fêmea nas laterais (Figura 7).

Dessa forma, o conector DB9 fêmea conecta o meio externo aos drivers de motor de passo TB6612, que, por meio de sinais PWM, enviam o valor de corrente para as bobinas, gerando o campo magnético correspondente.

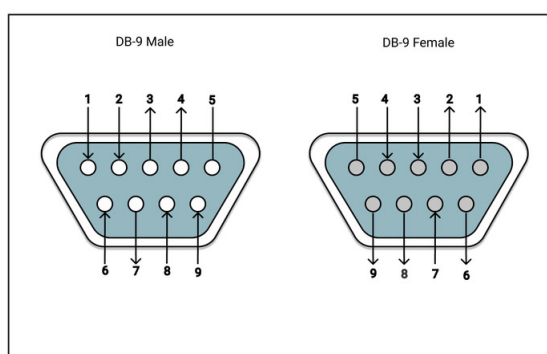
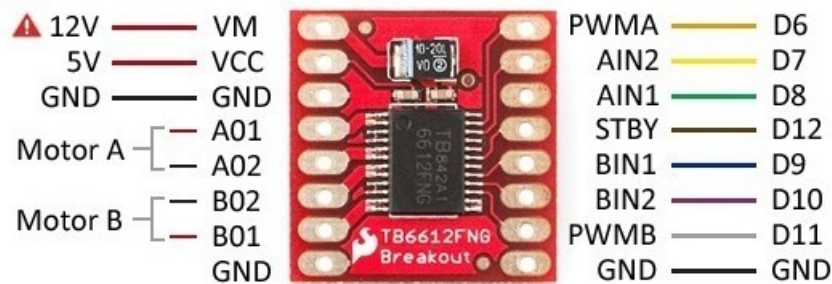


Figura 7: Conector DB9. As pinagens 1 a 5 são: . Já as de 6 a 9 são:

No conector macho, possui um chicote elétrico (conjunto de fios organizados) que liga o mockup a central de processamento. Essa central, no Mark I era composta por um Arduino Mega, um Xbee e um shield para Xbee (Figura 9). Já no mark III, será diretamente conectada a placa, de forma que o Rpi controlará as bobinas ou comunicará com um possível placa de controle dentro do 2U.





Vin entre 5 y 15V. Al usar alimentación externa SIEMPRE poner con GND común.

Figura 8: Driver TB6612.

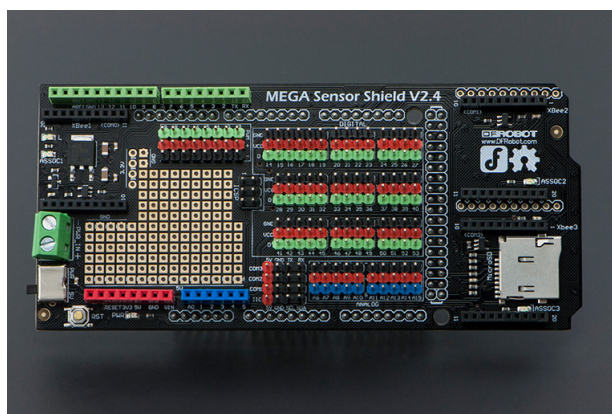


Figura 9: Shield de interface do arduino Mega, figura retirada do site [Dfrobot](https://www.dfrobot.com/).

(e) Placa de Circuito Impresso

À princípio, tem-se que para integrar tais atuadores e sensores no RPi, fez-se necessário construir uma Placa de Circuito Impresso (PCB). Com efeito, utilizou-se o software Eagle 9.5.2 da Autodesk para fazer o esquemático elétrico (Figura 23) e o board (Figura 10).([Arquivos da PCB Eagle](#)).



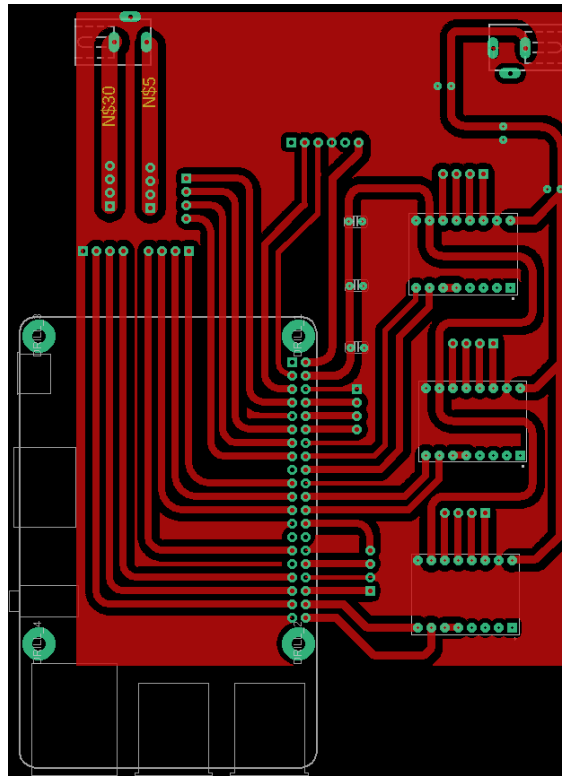


Figura 10: Board da PCB

Deve-se frisar que esta PCB foi feita para validar o sistema funcionando com todos os componentes eletrônicos simultaneamente, portanto, não há circuitos de proteção. Então, se possível, deve-se fazê-lo em uma nova versão. Recomenda-se adicionar um [Diodo Zener regulador de tensão](#).

Fatores levados em conta quando no design da PCB:

- O sensor de orientação deve ficar posicionado exatamente no ponto médio do eixo horizontal da placa, pois ele deve estar centralizado com a mesa de simulação;
- A PCB deve ser single-layer, já que a manufatura será caseira;
- A medida do eixo vertical não deve exceder 10cm, pois será encaixada em um suporte;

Por fim, criou-se um suporte (Figura 11) para acoplamento do Motor EC do eixo Z e a PCB, ambos centralizados com a mesa de simulação. O suporte foi feito em impressão 3D no software Fusion 360 da Autodesk. ([CADs files Suporte](#)).

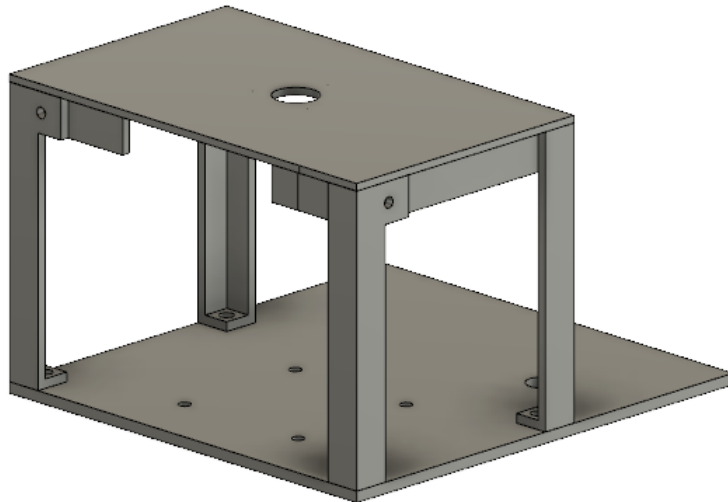


Figura 11: CAD do suporte de acoplamento

III - Software

Motivação

A parte de software e firmware foi implementada de forma a garantir maior consistência e modularidade entre as partes do sistema como um todo. Dessa forma, a implementação de novos módulos, como LoRa ou rádio para a transmissão de dados, usando ou não TCP, se dá de maneira prática, apenas substituindo os includes das bibliotecas e as respectivas funções de tratamento, recebimento e envio de dados. Dados que as linguagens usadas não dependem de determinado sistema operacional ou IDE, pode-se utilizar as bibliotecas construídas em qualquer sistema operacional, windows, linux ou raspbebian, por exemplo.

Além da baixa acopabilidade, o novo sistema implementado busca extrair o máximo da capacidade de processamento dos componentes, com o uso de threads e semaforos nas linguagens C e C++, de forma a tentar garantir as condições de tempo real do sistema em questão. Para o ADCV, utilizou-se o sistema operacional Ubuntu 16.04 LTS com os frameworks Paramiko (SSH Python 3 na etapa de Setup - seção (1) Metodologia), Opencv 4.1, biblioteca de TCP-IP implementada pelo integrante Flavio no repositório [TCP-Gitlab](#). No Rpi utilizou-se o SO Raspbean, baseado em linux, com a biblioteca da IMU BNO055 ([BNO055 Github](#)), a qual foi refatorada para atender melhor as necessidades do projeto.

Fluxo dos Programas

Os programas funcionam de forma encadeada, sendo um código chamando outro quando transpassa de uma etapa para outra. Isso facilita o uso, de forma que o usuário precisa apenas executar o programa **main_setup.py**. Como a calibração do sistema não é uma situação sempre desejada, dependendo da missão/teste necessita-se desbalanceamento da mesa para validação de algoritmos, por exemplo, esta função pode ser requisitada pelo usuário por linha de comando, e, posteriormente, pela interface gráfica. Dessa forma, a calibração pode acontecer de maneira sazonal programada ou pelo estado de uma flag passada pela linha de comando, sendo todos os comandos enviados para o Rpi via SSH em Python 3 por meio da biblioteca, que necessita ser instalada, Paramiko.

Em seguida, no mesmo programa, configura-se o ambiente de trabalho no Rpi por meio do envio dos arquivos que compoem o firmware e a sua organização em diretorios (a partir



da pasta **/home/pi/TestBed**), por meio do SSH. Os arquivos da missão - que contém os parâmetros e tipo de controlador ou se este será executado pelo computador (estação solo) - são enviados e checados se está no padrão JSON.

Após a configuração do ambiente, a `main_setup.py` compila o ADCV (**main.cpp**) e os códigos que serão utilizados no computador e no Rpi (**main_controller.c**). A compilação é realizada Makefile e executados em seguida de forma paralela (considerando a dinâmica da mesa). Os arquivos `main_setup.py` e `main.cpp` se encontram no diretório **src** do [ADCV-Rpi Repositorio](#).

Por fim, começa a fase de missão, que será explicado o loop no Rpi e no computador nas subseções (a) e (b), respectivamente. Nessa fase, o programa em python termina a sua execução após executar o executável da `main.cpp` e enviar o comando para executar o da `main_controller.c` no Rpi, via SSH. Dessa forma, os recursos alocados pela `main_setup.py` ficam disponíveis e o gerenciamento do SO e dos recursos do computador podem ser melhor controlados em C++.

(a) Fluxo no Rpi

No Raspberry Pi, o ciclo do programa é ligeiramente simples (Figura 12), de forma que no início do loop a função é travada em espera dos dados enviados por TCP pelo ADCV, coleta-se os dados da IMU, possibilitando a ação do controlador embarcado ou o envio dos dados de volta para o `adcv`. Para garantir um tempo de loop constante, há um `wait_time` que faz esperar até o tempo de ciclo definido no controlador (tempo de discretização).

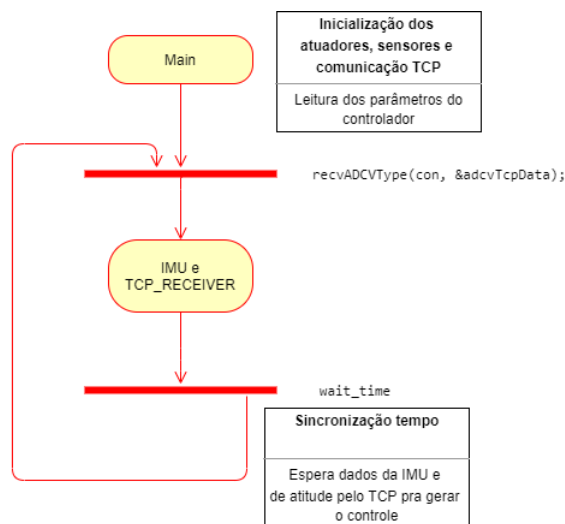


Figura 12: Fluxograma do código **main_controller.c** .

O ideal é o cenário encontrado na Figura 13, onde o controlador atua por atraso constante em uma thread separada da de aquisição de dados. Esta proposta será implementada em breve.

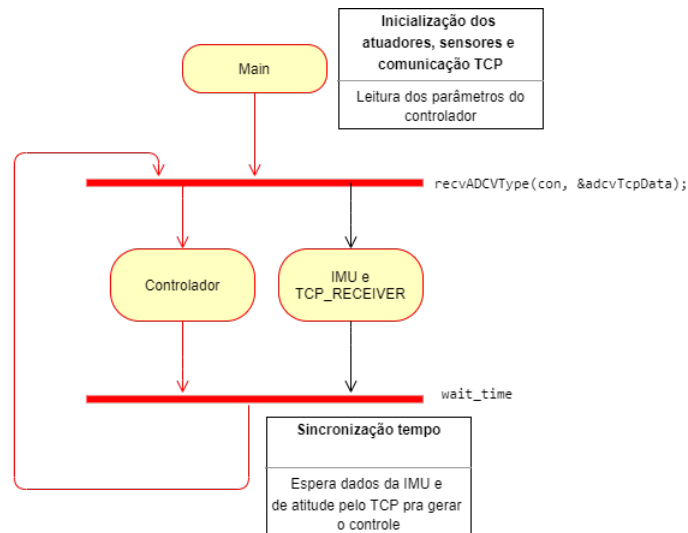


Figura 13: Fluxograma do código **main_controller.c** com relação a paralelização por threads e o controlador.

(b) Fluxo no computador - ADCV

O primeiro ponto a se destacar é que o TCP envia os dados e estes são gravados no arquivo Json após o processamento dos dados, dessa forma, eles são enviados na próxima interação do loop em um thread separada da captura de câmera. Todas as threads são sincronizadas pelo semáforo do ADCV, que controla a aquisição dos frames e, de forma indireta, o semáforo do TCP e do JSON.

Primeiramente, a main da **main.cpp** configura o sistema, declaração do construtor com as funcionalidades escolhidas por meio de flags na linha de execução da **main_setup.py** pelo usuário: todas as câmeras (-a), paralelização (-p), gravar os dados do experimento (-r), ou usar video ao invés de câmera (-v) seguido do *path* do video sem a indicação do repositório local, no caso de caminho relativo. Dessa forma, o construtor se adapta dependendo das flags passadas na linha de execução.

Após a conexão via TCP do Rpi (usuário) ao ADCV (server), as threads são alocadas com o uso da flag (-p), de forma a se ter a paralelização do sistema criando-se as threads da câmera X, Y, TCP e Json (Figura 14). Assim, quando se inicializa a função *adcVLoopP*, o semáforo do ADCV é incrementado de forma atômica (evitando condições de corrida), indicando que há uma thread para captura de câmera pelo método *adcV->getFrame(CAM_ID)*. Após o processamento do frame e a determinação de atitude pelo método *adcV->determAttitude(CAM_ID)*, o semáforo do ADCV é decrementado, indicando o término do processamento quando chega a zero e sinalizando para o semáforo do TCP o início da transmissão dos dados e os dados de atitude de cada câmera postos na fila que será consumida pelo Json pra gravar.

Dessa forma, as câmeras voltam para a aquisição dos dados e a thread do TCP sai do estado de espera e envia os dados para o RPI. Em seguida, o semáforo do TCP volta para o estado inicial, bloqueando o loop de envio. Em suma, o loop inicializa com o incremento do semáforo pelas threads das câmeras e finaliza com o decremento e desbloqueio da função pelo semáforo quando este está em zero, sendo os dados de atitude enviados na iteração seguinte da aquisição de dados. Ao término do programa, os dados são serializados e gravados no arquivo Json.

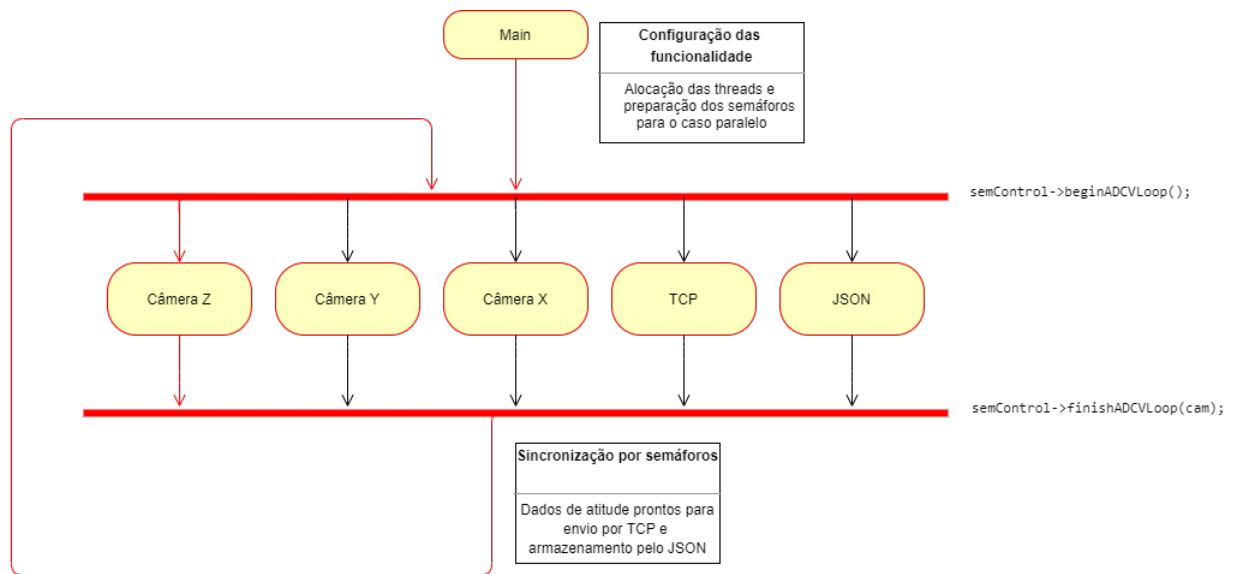


Figura 14: Fluxograma do código **main.cpp** com relação a paralelização por threads e sincronização por semáforos.

Dependência do Sistema

Esta seção trata das dependências de bibliotecas e classes na parte do ADCV-TCP e na parte do RPI-TCP. Dessa forma, a relação entre classes do ADCV pode ser melhor observada na Figura 15 e relação entre as bibliotecas dos sensores e atuadores no RPI na Figura 16.

A respeito da organização [ADCV-Rpi Repositorio](#), tem-se alguns arquivos de configuração, o **README.md** e as 5 principais pastas: Libs, adcv4, rpi, src e networking. Na pasta Libs, tem as dependências usadas na **main.cpp**, como uma biblioteca de semáforos para o sincronismo das threads das câmeras X, Y e Z do ADCV, JSON com bibliotecas e códigos exemplos em C, C++, Python 3 e Matlab, e a Commun com arquivos de definição de estruturas em commun entre os módulos do Json, ADCV e as adaptações da biblioteca TCP para o ADCV.

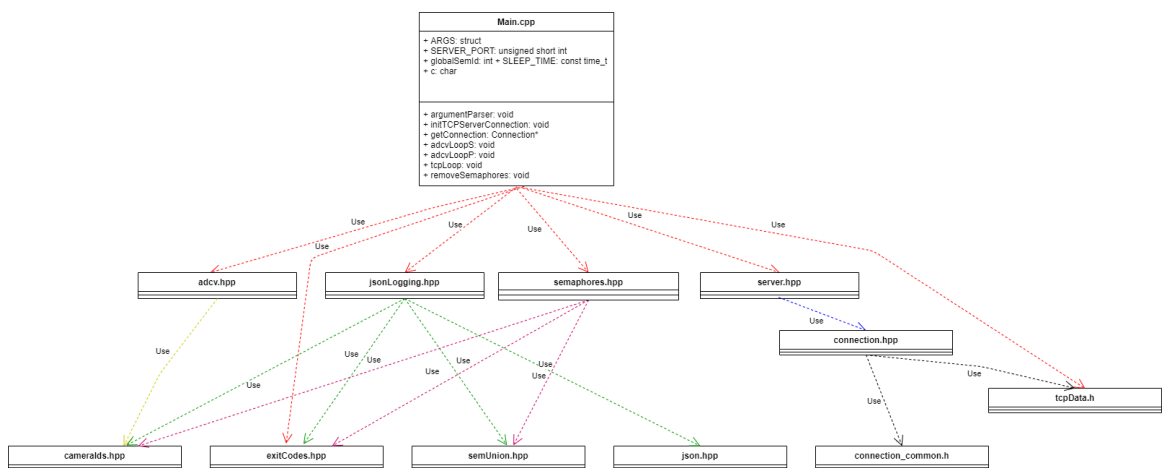


Figura 15: Relação de dependência entre bibliotecas e classes da main.cpp

Na pasta `adcv4`, possui o `adcv` na sua versão 4 e a pasta "data/files" com os arquivos que caracterizam o Aruco e os parâmetros de calibração das câmeras. A biblioteca do `adcv` possui estruturas para paralelização e a abstração do `adcv` em 4 simples funções `adcv->getFrame(CAM_ID)`, `adcv->determAttitude(CAM_ID)` e `adcv->showImage()` e `adcv->recVideo()`. Onde `CAM_ID` é o `id = 0,1,2` que indica as câmeras Z,Y,X. Assim, o `adcv`

possui a função de usar videos já gravados, as câmeras, gravar os dados e usar apenas a câmera Z ou todas., conforme explicado no arquivo README.md.

Na pasta rpi, possuem os códigos que serão enviados pro ADCV junto com os da networking, para cumprir as três etapas. Os códigos são enviados toda vez que a main_setup.py é executada, de forma que os arquivos podem ser alterados no computador e compilados/executados no Rpi. Por fim, a pasta src que contém as mains do projeto, main_setup.py e main.cpp.

Controller.h
<pre>int verbose = 0; int outflag = 0; int argflag = 0; // 1 dump, 2 reset, 3 load calib, 4 write calib char opr_mode[9] = {0}; char pwr_mode[9] = {0}; char datatype[256]; char senaddr[256] = "0x28"; char htmfile[256]; char calfile[256]; // Reaction Wheels void setReactionWheels(int axis); void actuate(int delay_value, int pwm); // IMU void readCalibrationFile(); void writeCalibrationFile(); void linAccel(); void readAcc(); void readGyro(); void readMag(); void euler(); void quaternion(); void gravity(); void getTemp();</pre>

Figura 16: Relação de variáveis e funções da biblioteca controller.h .

V - Trabalhos Futuros

- Implementação da main_controller.c conforme a figura 13;
- Ajustes na main_setup.py de funcionalidades (argparse);
- Ajustes (IP) na main.cpp;
- Testes de tempo e atraso do ADCV;
- Pinagem figura 7;
- Escrever biblioteca em C para os controladores;
- Adaptação do chicote do Mockup 2U para a pcb;
- Fazer PCB;

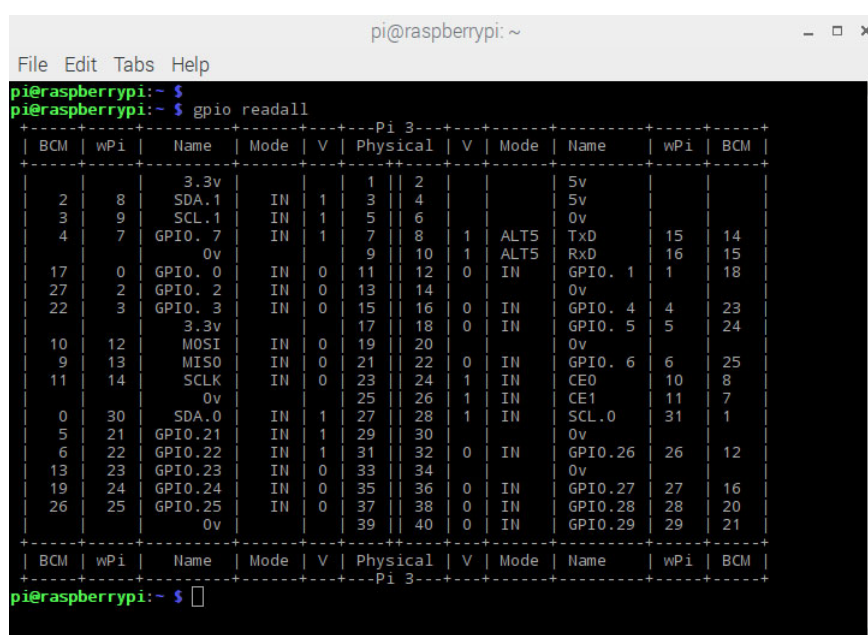


Apêndice

Códigos

Os códigos podem ser encontrados no repositório [HILTable](#), na plataforma GitLab, o qual faz-se necessário um pedido de acesso ao administrador.

Pinout's, Conexões e Datasheets



```
pi@raspberrypi:~$ gpio readall
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi | Name | Mode | V | Physical | V | Mode | Name | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 2 | 8 | 3.3v | IN | 1 | 3 | 4 | | | 5v | | | |
| 3 | 9 | SCL.1 | IN | 1 | 5 | 6 | | | 5v | | |
| 4 | 7 | GPIO. 7 | IN | 1 | 7 | 8 | 1 | ALT5 | TxD | 15 | 14 |
| 17 | 0 | GPIO. 0 | IN | 0 | 11 | 12 | 0 | IN | RxD | 16 | 15 |
| 27 | 2 | GPIO. 2 | IN | 0 | 13 | 14 | 0 | IN | GPIO. 1 | 1 | 18 |
| 22 | 3 | GPIO. 3 | IN | 0 | 15 | 16 | 0 | IN | GPIO. 4 | 4 | 23 |
| 10 | 12 | 3.3v | IN | 0 | 17 | 18 | 0 | IN | GPIO. 5 | 5 | 24 |
| 9 | 13 | MOSI | IN | 0 | 19 | 20 | 0 | IN | GPIO. 6 | 6 | 25 |
| 11 | 14 | SCLK | IN | 0 | 21 | 22 | 0 | IN | GPIO. 7 | 7 | 26 |
| 0 | 30 | SDA.0 | IN | 1 | 23 | 24 | 1 | IN | CE0 | 10 | 8 |
| 5 | 21 | GPIO.21 | IN | 1 | 25 | 26 | 1 | IN | CE1 | 11 | 7 |
| 6 | 22 | GPIO.22 | IN | 1 | 27 | 28 | 1 | IN | SCL.0 | 31 | 1 |
| 13 | 23 | GPIO.23 | IN | 0 | 29 | 30 | 0 | IN | GPIO.26 | 26 | 12 |
| 19 | 24 | GPIO.24 | IN | 0 | 31 | 32 | 0 | IN | GPIO.27 | 27 | 16 |
| 26 | 25 | GPIO.25 | IN | 0 | 33 | 34 | 0 | IN | GPIO.28 | 28 | 20 |
| | | | | | 35 | 36 | 0 | IN | GPIO.29 | 29 | 21 |
| | | | | | 37 | 38 | 0 | IN | | | | |
| | | | | | 39 | 40 | 0 | IN | | | | |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi | Name | Mode | V | Physical | V | Mode | Name | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 2 | 8 | 3.3v | IN | 1 | 3 | 4 | | | 5v | | | |
| 3 | 9 | SCL.1 | IN | 1 | 5 | 6 | | | 5v | | |
| 4 | 7 | GPIO. 7 | IN | 1 | 7 | 8 | 1 | ALT5 | TxD | 15 | 14 |
| 17 | 0 | GPIO. 0 | IN | 0 | 11 | 12 | 0 | IN | RxD | 16 | 15 |
| 27 | 2 | GPIO. 2 | IN | 0 | 13 | 14 | 0 | IN | GPIO. 1 | 1 | 18 |
| 22 | 3 | GPIO. 3 | IN | 0 | 15 | 16 | 0 | IN | GPIO. 4 | 4 | 23 |
| 10 | 12 | 3.3v | IN | 0 | 17 | 18 | 0 | IN | GPIO. 5 | 5 | 24 |
| 9 | 13 | MOSI | IN | 0 | 19 | 20 | 0 | IN | GPIO. 6 | 6 | 25 |
| 11 | 14 | SCLK | IN | 0 | 21 | 22 | 0 | IN | GPIO. 7 | 7 | 26 |
| 0 | 30 | SDA.0 | IN | 1 | 23 | 24 | 1 | IN | CE0 | 10 | 8 |
| 5 | 21 | GPIO.21 | IN | 1 | 25 | 26 | 1 | IN | CE1 | 11 | 7 |
| 6 | 22 | GPIO.22 | IN | 1 | 27 | 28 | 1 | IN | SCL.0 | 31 | 1 |
| 13 | 23 | GPIO.23 | IN | 0 | 29 | 30 | 0 | IN | GPIO.26 | 26 | 12 |
| 19 | 24 | GPIO.24 | IN | 0 | 31 | 32 | 0 | IN | GPIO.27 | 27 | 16 |
| 26 | 25 | GPIO.25 | IN | 0 | 33 | 34 | 0 | IN | GPIO.28 | 28 | 20 |
| | | | | | 35 | 36 | 0 | IN | GPIO.29 | 29 | 21 |
| | | | | | 37 | 38 | 0 | IN | | | | |
| | | | | | 39 | 40 | 0 | IN | | | | |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
pi@raspberrypi:~$
```

Figura 17: Relação pinos RPi - Biblioteca WiringPi

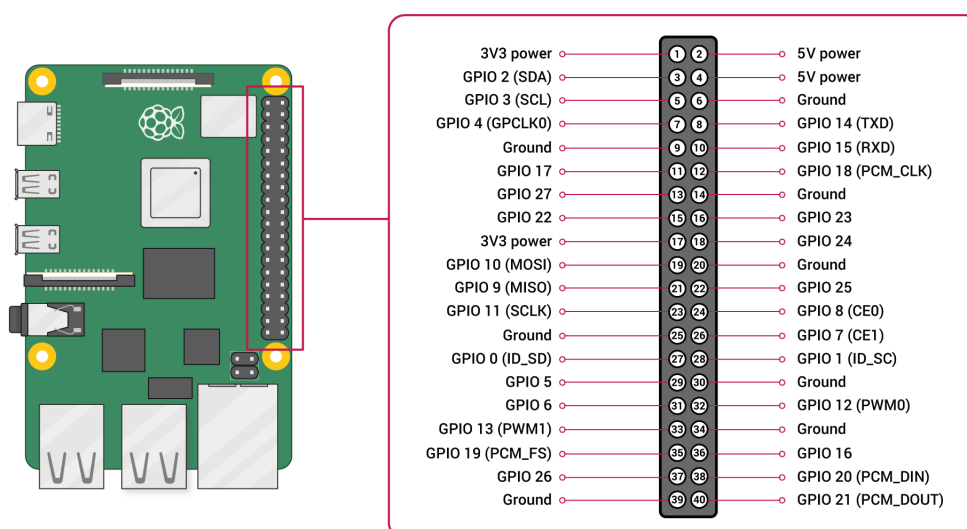
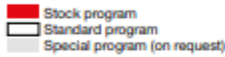


Figura 18: Pinout RPi

maxon flat motor



V1 with Hall sensors
all sensors and cables

Motor Data (provisional)

April 2019 edition / subject to change



3.4.4 Analog I/Os (J6)



Figure 3-13 Analog I/Os Socket J6

J6 & Head A Pin	Prefab Cable Color	Head B Pin	Signal	Description
1	white		AnIN1+	Analog input 1, positive signal
2	brown		AnIN1-	Analog input 1, negative signal
3	green		AnIN2+	Analog input 2, positive signal
4	yellow		AnIN2-	Analog input 2, negative signal
5	grey		AnOUT1	Analog output 1
6	pink		AnOUT2	Analog output 2
7	blue		GND	Signal ground

Table 3-16 Analog I/Os Socket J6 – Pin Assignment & Cabling

Figura 20: Analog pins do driver Motor EC

3.4.3 Digital I/Os (J5)



Figure 3-7 Digital I/Os Socket J5

J5 & Head A Pin	Prefab Cable Color	Head B Pin	Signal	Description
1	white		DigIN1	Digital input 1
2	brown		DigIN2	Digital input 2
3	green		DigIN/DigOUT3	Digital input/output 3
4	yellow		DigIN/DigOUT4	Digital input/output 4
5	grey		GND	Signal ground
6	pink		+5 VDC	Auxiliary output voltage (+5 VDC; ≤80 mA)

Table 3-14 Digital I/Os Socket J5 – Pin Assignment & Cabling

Figura 21: Digital pins do driver Motor EC

HEADER J2

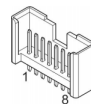


Figure 3-4 Motor / Hall Sensor Header J2



Best Practice

- For EC motors with built-in FPC (Flexprint cable), you might wish to use one of the ready-made adapters. For details → "Cable Selector" on page 3-13.

J2 & Head A Pin	Prefab Cable Color	Head B Pin	Signal	Description
1			Motor winding 1	EC motor: Winding 1
2			Motor winding 2	EC motor: Winding 2
3			Motor winding 3	EC motor: Winding 3
4			+5 VDC	Hall sensor supply voltage (+5 VDC; I _L ≤30 mA)
5			GND	Ground
6			Hall sensor 1	Hall sensor 1 input
7			Hall sensor 2	Hall sensor 2 input
8			Hall sensor 3	Hall sensor 3 input

Table 3-10 Motor / Hall Sensor Header J2 – Pin Assignment & Cabling

Figura 22: Header pins (Supply) do driver Motor EC

Tabela (Tabela 6) de características elétricas de todos os componentes:

Tabela 6: Características Elétricas de todos os componentes

Motor de Passo	
Load Supply Voltage Range	8V-35V
Logic Supply Voltage Range	3V-5.5V
Motor Supply Current Operating	0-2mA
Logic Supply Current	0-5mA
Motor EC	
Nominal operating voltage Vcc	10V-36V
Output voltage (max.)	0.98 x Vcc
Output current Icont / Imax (<4 s)	2.7A / 9A
Sensor de Orientação	
Supply Voltage (only Sensors) Vdd	2.4V-3.6V
Supply Voltage (μ C and I/O Domain) VddIO	1.7V-3.6V
Total supply current (normal mode at TA)	0-12.3mA



COMPONENTS:

- Raspberry Pi 4B:** Model B+, 8GB, 64-bit, 1.2GHz, 4x USB 3.0, 2x USB 2.0, 1x HDMI, 1x Ethernet, 1x DisplayPort, 1x Headset Jack, 1x MicroSD Card Slot, 1x USB Type-C Port.
- Stepper Motors:** NEMA 17, 1.8°/36°/7.2°/15°/18°/22.5°/36°/45°/90°/180°/360°/720°/1440°/2880°/5760°/11520°/23040°/46080°/92160°/184320°/368640°/737280°/1474560°/2949120°/5898240°/11796480°/23592960°/47185920°/94371840°/188743680°/377487360°/754974720°/1509949440°/3019898880°/6039797760°/12079595520°/24159191040°/48318382080°/96636764160°/193273528320°/386547056640°/773094113280°/1546188226560°/3092376453120°/6184752906240°/12369505812480°/24739011624960°/49478023249920°/98956046499840°/197912092999680°/395824185999360°/791648371998720°/1583296743997440°/3166593487994880°/6333186975989760°/12666373951979520°/25332747903959040°/50665495807918080°/101330991615836160°/202661983231672320°/405323966463344640°/810647932926689280°/1621295865853378560°/3242591731706757120°/6485183463413514240°/12970366926827028480°/25940733853654056960°/51881467707308113920°/103762935414616227840°/207525870829232455680°/415051741658464911360°/830103483316929822720°/1660206966633859645440°/3320413933267719290880°/6640827866535438581760°/13281655733070877163520°/26563311466141754327040°/53126622932283508654080°/106253245864567017308160°/212506491729134034616320°/425012983458268069232640°/850025966916536138465280°/1700051933833072276930560°/3400103867666144553861120°/6800207735332289107722240°/13600415470664578215444480°/27200830941329156430888960°/54401661882658312861777280°/108803323765316625723554560°/217606647530633251447109120°/435213295061266502894218240°/870426590122533005788436480°/1740853180245066011576872960°/3481706360490132023153745920°/6963412720980264046307491840°/13926825441960528092614983680°/27853650883921056185229967360°/55707301767842112370459934720°/111414603535684224740919869440°/222829207071368449481839738880°/445658414142736898963679477760°/891316828285473797927358955520°/1782633656570947595854717911040°/3565267313141895191709435822080°/7130534626283790383418871644160°/14261069252567580766837743288320°/28522138505135161533675486576640°/57044277010270323067350973153280°/114088554020540646134701946306560°/228177108041081292269403892613120°/456354216082162584538807785226240°/912708432164325169077615570452480°/1825416864328650338155231140904960°/3650833728657300676310462281809920°/7301667457314601352620924563619840°/14603334914629202705241849127239680°/29206669829258405410483698254479360°/58413339658516810820967396508958720°/116826679317033621641934793017917440°/233653358634067243283869586035834880°/467306717268134486567739172071669760°/934613434536268973135478344143339520°/1869226869072537946270956688286679040°/3738453738145075892541913376573358080°/7476907476290151785083826753146716160°/14953814952580303570167653506293432320°/29907629905160607140335307012586864640°/59815259810321214280670614025173729280°/119630519620642428561341228050347558560°/239261039241284857122682456100695117120°/478522078482569714245364912201390234240°/957044156965139428490729824402780468480°/1914088313930278856981459648805560936960°/3828176627860557713962919297611121873920°/7656353255721115427925838595222243747840°/15312706511442230855851677190444487495680°/30625413022884461711703354380888974991360°/61250826045768923423406708761777949927040°/122501652091537846846813417523555899854080°/24500330418307569369362683504711179910720°/49000660836615138738725367009422359821440°/98001321673230277477450734018844719642880°/19600264334646055495490148037689439285760°/39200528669292110990980296075378878571520°/78401057338584221981960592150757757143040°/156802114677168443963921184301515514286080°/313604229354336887927842368603031028572160°/627208458708673775855684737206062057144320°/12544169174173475517113694744121241142867840°/250883383483469510342273894882424822857360°/501766766966939020684547789764849645714720°/1003533533933878041369095579529698891428480°/2007067067867756082738191159059397782856960°/401413413573551216547638231811879556

Figura 23: Esquemático elétrico da PCB

Bibliografia

- [1] R. C. Silva, F. C. Guimaraes, J. V. L. Loiola, R. A. Borges, S. Battistini, and C. Cappelletti. Tabletop testbed for attitude determination and control of nanosatellites. *Journal of Aerospace Engineering*, 32:1–9, 2019.
- [2] J. V. L. Loiola, L. C. V. D. Ploeg, R. C. Silva, F. C. Guimaraes, R. A. Borges, G. A. Borges, S. Battistini, and C. Cappelletti. 3 axis simulator of the earth magnetic field. *Proceedings of the 39th IEEE Aerospace Conference*, pages 1–8, 2018.
- [3] I. S. K. Ishioka, L. M. B. Silva, R. A. Borges, S. Battistini, C. Cappelletti, D. ROLDUGIN, and M. Ovchinnikov. Hil testing of the b-dot attitude control law. *Proceedings of the III IAA Latin American Cubesat Workshop, Ubatuba*, 11:1–9, 2018.
- [4] R. C. Silva, I. S. K. Ishioka, C. Cappelletti, S. Battistini, and R. A. Borges. Helmholtz cage design and validation for nanosatellites hwil testing. *Journal of Aerospace Engineering*, 55:3050–3061, 2019.

