Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Mecatrónica

Trabalho de Projeto

Armazém industrial de contentores automático

André Filipe dos Santos Simões

Orientador(es) | Fernando Manuel Janeiro

Évora 2024

Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Mecatrónica

Trabalho de Projeto

Armazém industrial de contentores automático

André Filipe dos Santos Simões

Orientador(es) | Fernando Manuel Janeiro

Évora 2024

O trabalho de projeto foi objeto de apreciação e discussão pública pelo seguinte júri nomeado pelo Diretor da Escola de Ciências e Tecnologia:

Presidente	|	João Manuel Figueiredo (Universidade de Évora)
Vogais	|	Fernando Manuel Janeiro (Universidade de Évora) (Orientador) Frederico José Grilo (Universidade de Évora) (Arguente)

Évora 2024

Agradecimentos

O autor deseja expressar um agradecimento ao Professor Fernando Janeiro por se ter disponibilizado para ser o orientador deste trabalho. O autor agradece ainda a toda a sua família e amigos por todo o suporte e acompanhamento prestado, nesta importante etapa da vida pessoal e académica.

Resumo

Armazém industrial de contentores automáticos

Nas operações industriais modernas, sistemas de recolha e armazenamento eficientes são fundamentais para garantir um fluxo de trabalho sem falhas, uma utilização ótima dos recursos e uma produtividade melhorada. Com o crescimento exponencial do comércio global e do fabrico, a procura por soluções de armazenamento inovadoras intensificou-se. Entre estas soluções, os armazéns automáticos de contentores surgiram como um paradigma transformador, redefinindo as referências de eficiência e precisão na gestão do armazenamento.

Este trabalho centra-se na conceção, design e implementação de um armazém industrial automático de contentores através de uma maquete - com o objetivo da receção de contentores transportados por comboio numa via-férrea sem intervenção humana, e armazenamento dos mesmos.

Palavras-chave: *Automação Industrial; Armazém de contentores; Impressão 3D; ESP32; Raspberry Pi 4 Model B;*

Abstract

Automatic Industrial Container Warehouse

In modern industrial operations, efficient collection and storage systems are essential to ensure seamless workflow, optimal resource utilization, and enhanced productivity. With the exponential growth of global trade and manufacturing, the demand for innovative storage solutions has intensified. Among these solutions, automatic container warehouses have emerged as a transformative paradigm, redefining the benchmarks of efficiency and precision in storage management.

This work focuses on the conception, design, and implementation of an automatic industrial container warehouse through a prototype - with the aim of receiving containers transported by train on a railway without human intervention and storing them.

Keywords: *Industrial automation; Container warehouse; 3D printing; ESP32; Raspberry Pi 4 Model B;*

Índice

Lista de Figuras

Capítulo 1: Introdução

1.1 Estado da arte

As crescentes complexidades inerentes às cadeias de abastecimento modernas exigem um arranque das metodologias tradicionais de armazém para sistemas mais ágeis e responsivos. Os armazéns tradicionais, caracterizados pelo trabalho manual e por configurações de armazenamento estáticas, são cada vez mais inadequados para satisfazer a procura do mercado atual, que evolui rapidamente e de forma dinâmica. Fatores como a escassez de mão-de-obra, as diferentes preferências dos consumidores e os prazos de entrega rigorosos realçam a necessidade de inovação na infraestrutura de armazenamento. Os armazéns automáticos de contentores representam uma iniciação da fase revolucionária dos paradigmas de armazenamento tradicionais. Ao utilizar tecnologias de ponta como braços robóticos, veículos guiados automaticamente (AGVs) e algoritmos de aprendizagem automática, estas instalações oferecem uma eficiência, fiabilidade e escalabilidade. A integração perfeita de componentes de *hardware* e *software* permite a monitorização em tempo real, a tomada de decisões adaptativas e a coordenação perfeita das atividades do armazém, capacitando as organizações para alcançar níveis de excelência operacional.

1.2 Motivação e objetivos

A escolha do tema deste trabalho, deveu-se à possibilidade de realizar uma ideia pessoal antiga com as matérias lecionadas no mestrado de Engenharia Mecatrónica. A minha carreira académica em Engenharia Mecânica e Engenharia Mecatrónica ilustram o meu particular interesse nas tecnologias e na indústria mecânica, revelando-se decisivas na escolha do tema deste trabalho.

A motivação para este trabalho provém de utilizar o que foi aprendido ao longo do percurso académico, para melhor aprofundar os conhecimentos em várias matérias, e comprovar que é possível realizar qualquer ideia, tendo os conhecimentos para tal.

O objetivo deste trabalho é a idealização e criação de uma maquete simulando um armazém industrial, com sistema de recolha e armazenamento automático de contentores.

1.3 Programas utilizados

1.3.1 *Autodesk Inventor Professional* 2023

Este *software* de Desenho Assistido por Computador (CAD) da *Autodesk* é amplamente utilizado na indústria para criar modelos 3D detalhados e esquemas de design. Desde o seu lançamento, o *Autodesk Inventor* evoluiu para oferecer recursos avançados de modelagem e simulação, tornando-se uma escolha popular para projetos de engenharia mecânica e design de produtos.

1.3.2 *Cura*

O *Cura* é um *software* de código aberto amplamente utilizado para fatiar modelos 3D e gerar instruções de impressão para impressoras 3D. Desenvolvido pela *Ultimaker*, o *Cura* tem um historial de desenvolvimento colaborativo e melhoria contínua. Com funcionalidades avançadas de fatiamento e suporte para uma variedade de formatos de ficheiros, é uma escolha popular entre os entusiastas e profissionais da impressão 3D.

1.3.3 *EasyEDA*

A *EasyEDA* é uma plataforma online que oferece ferramentas de design de circuitos eletrónicos e PCB (Placa de Circuito Impresso). Desde o seu lançamento, o *EasyEDA* tem sido elogiado pela sua interface intuitiva e capacidades de colaboração, tornando-se uma escolha popular entre os engenheiros e entusiastas da eletrónica.

1.3.4 *Python*

Python é uma linguagem de programação de alto nível conhecida pela sua simplicidade e legibilidade de código. Desde a sua criação nos anos 90, o *Python* tem crescido em popularidade e tornou-se uma das linguagens de programação mais utilizadas

numa variedade de domínios, incluindo o desenvolvimento *web*, análise de dados, automação de sistemas e inteligência artificial.

1.3.5 *Thonny*

Thonny é um ambiente de desenvolvimento integrado (IDE) para *Python*, concebido tanto para principiantes como para educadores. Lançado em 2016, o *Thonny* oferece uma interface simplificada e capacidades de depuração interativa que facilitam a aprendizagem da programação em *Python*. Com uma abordagem amigável para principiantes, o *Thonny* tem sido uma ferramenta valiosa para estudantes e professores que estão a aprender e a ensinar *Python*.

1.4 Tecnologias de comunicação

1.4.1 Comunicação Serial UART

A comunicação série é uma parte essencial de muitos sistemas eletrónicos modernos, permitindo a transferência fiável de dados entre dispositivos. Um dos protocolos mais comuns utilizados para este efeito é a UART, ou Transmissor/Recetor Universal Assíncrono.

A UART é uma interface série que permite a transmissão e receção de dados entre dispositivos de forma assíncrona, sem necessidade de um relógio partilhado. Em vez disso, os dispositivos UART utilizam uma taxa de baud predefinida para sincronizar a comunicação. Esta flexibilidade faz da UART uma escolha popular numa variedade de aplicações.

Na comunicação UART, cada dispositivo possui uma linha de transmissão (TX) e uma linha de receção (RX). O dispositivo transmissor envia dados através da linha TX, enquanto o dispositivo recetor os recebe através da linha RX. Os dados são transmitidos em pacotes de bits, com um formato que pode incluir bits de dados, bits de paragem e, opcionalmente, bits de paridade para deteção de erros.

A sua versatilidade faz com que seja utilizada numa variedade de aplicações, incluindo a comunicação entre microcontroladores e periféricos, a transferência de dados entre dispositivos eletrónicos, interfaces de comunicação e IoT (*Internet of Things*).

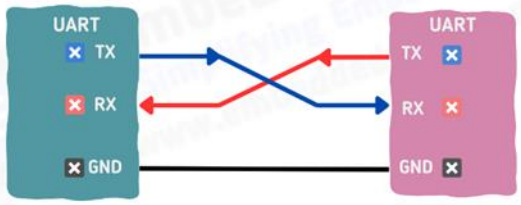

Figura 1: UART Protocol [1]

Capítulo 2: Idealização da maquete

2.1 Introdução

Neste capítulo, iremos debruçar-nos sobre a criação da maquete e os seus constituintes, começando pelo material utilizado, passando depois ao desenho dos mecanismos no *software* (CAD) *Inventor* e à sua impressão em 3D.

2.2 *Hardware*

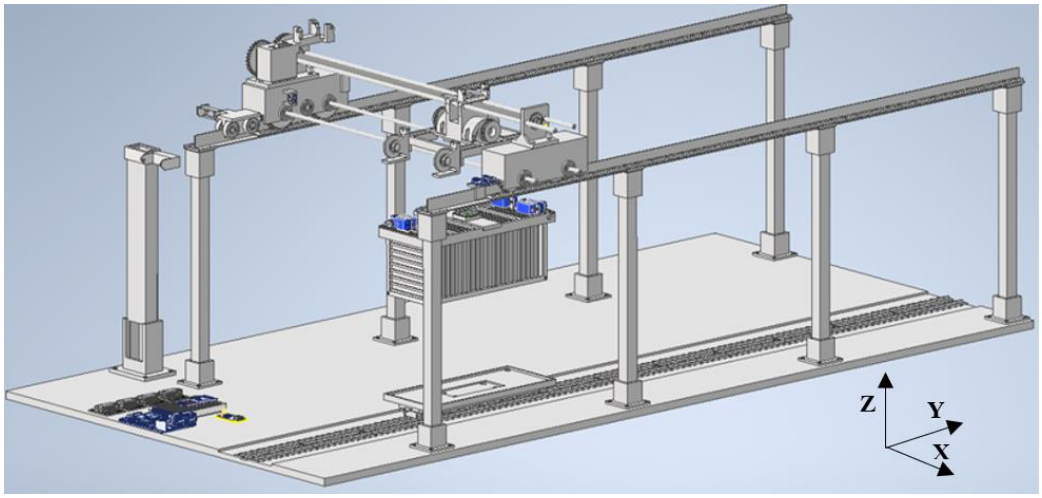

Figura 2: Maquete em CAD

2.2.1 ESP32 *Dev_board*

O ESP32 é um microcontrolador de baixo custo e alta performance, com conectividade *Wi-Fi* e *Bluetooth* integrada, a razão da utilização deste processador em vez de outros no mercado tem a ver com as suas características sendo as mais relevantes, a memoria disponível tendo 520KB de memória SRAM, 4MB de armazenamento *flash* e 448KB memória ROM, frequência de operação sendo esta variável entre 80MHz e 240MHz, 34 pinos I/O 16 deles com PWM, 18 ADC com uma resolução de 12-bit, 2 DAC de 8-bit e

as seguintes interfaces SPI, I2C, I2S, CAN, UART. No projeto, é utilizado como o cérebro do sistema para o controlo dos dispositivos na maquete, recebendo e executando ordens.

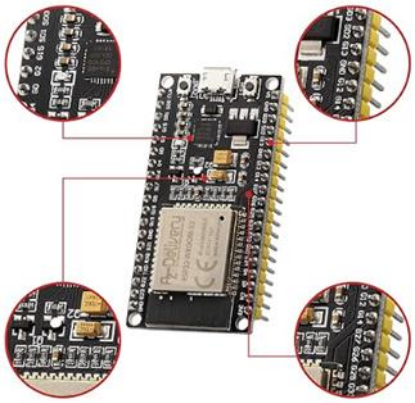

Figura 3: ESP32 Dev board [2]

2.2.2 FTDI USB to UART Serial Adapter

O adaptador FTDI USB para UART Serial é utilizado para estabelecer uma conexão serial entre o ESP32 e o Raspberry Pi. O adaptador serve para converter sinais USB em sinais UART e vice-versa, este funciona da seguinte forma o chip FTDI (normalmente FT232R) contém: Um protocolo USB, Interface UART, EEPROM para armazenar os parâmetros do dispositivo e buffers para dados. A maioria dos adaptadores FTDI incluem deslocadores de nível para converter entre os níveis lógicos internos do chip (normalmente 3,3V) e os níveis exigidos pelo dispositivo conectado sendo esta a razão para a sua utilização no projeto.

Figura 4: FTDI USB to UART Serial Adapter [3]

2.2.3 *Raspberry* Pi

O *Raspberry* Pi utilizado neste trabalho é o *Raspberry* Pi 4 *Model* B, um computador de placa única (SBC) usando um CPU ARM Cortex-A72 quad-core de 1,5 GHz para processamento e 8 GB de RAM que são suficientes para suportar uma variedade de aplicações. Ao observar a conectividade este tem *Wi-Fi* 2,4 GHz e 5,0 GHz , Bluetooth 5.0, BLE e Ethernet Gigabit, no que toca ao vídeo e som esta equipado com duas portas micro HDMI (que suportam até 4Kp60), porta de ecrã MIPI DSI de 2 vias, porta de câmara MIPI CSI de 2 vias, 3.5mm *audio-video jack* que também pode produzir vídeo composto (embora seja menos utilizado atualmente, dadas as opções HDMI), 40 pins para GPIO o que o torna muito bom para projetos eletrónicos, para a alimentação o *Raspberry* Pi pode utilizar 5V DC através do conetor USB-C (mínimo 3A), 5V DC através do conetor GPIO (mínimo 3A), Power over Ethernet (PoE) ativado (requer um HAT PoE separado).Este dispositivo opera num ambiente *Linux*.

Neste trabalho, é utilizado como centro de comando e processamento de imagem.

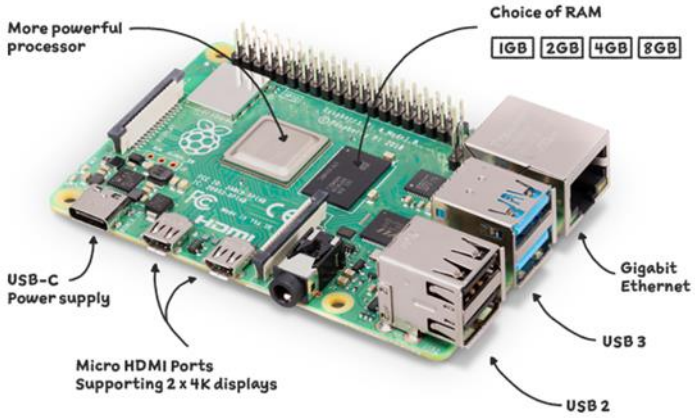

Figura 5: Raspberry Pi [4]

2.2.4 Pi-camera

A câmara Pi é uma câmara de alta resolução projetada especificamente para uso com o *Raspberry* Pi. Esta oferece funcionalidades avançadas de captura de imagem até 12 megapixéis para imagens fixas e pode gravar vídeo 4K a 50 *frames* por segundo, utilizando

um sensor Sony IMX708, com focagem automática, controlo de exposição e suporta imagens HDR para uma melhor qualidade em condições de iluminação difíceis.

Nesta aplicação, foi utilizada para a identificação de códigos QR e a localização de contentores.

Figura 6: Pi-camera [5]

2.2.5 Nema 17 Stepper Motor

O motor de passo Nema 17 é um motor de alto binário e precisão, amplamente utilizado em aplicações de controlo de movimento. Ele oferece passos de rotação precisos sendo estes programáveis usando um *driver* de motor de passo e é capaz de realizar movimentos suaves e controlados. Neste projeto, o motor de passo Nema 17 é utilizado para controlar o movimento do guindaste no eixo X.

Figura 7: Nema 17 Stepper Motor [6]

2.2.6 SG-90 Micro Servo

O micro servo SG-90 é um pequeno motor de alta precisão, comumente utilizado em aplicações que exigem movimentos angulares precisos e controlo de posição. Ele oferece uma ampla faixa de rotação sendo esta cerca de 180 graus (90 em cada direção) e é capaz de realizar movimentos rápidos aproximadamente 60 graus em 0,1 segundos sem carga, tendo cerca de 1,8 kgf.cm de binário a 4,8V binário sendo este ligeiramente superior a 6V.

Neste projeto, o micro servo SG-90 controla as patilhas usadas para agarrar os contentores.

Figura 8: SG-90 Micro Servo [7]

2.2.7 HC-SR04 Ultrasonic Sensor

O sensor ultrassónico HC-SR04 é um dispositivo de deteção de proximidade que utiliza ondas sonoras para medir a distância entre o sensor e um objeto, este funciona da seguinte maneira pino de disparo é colocado num nível elevado durante 10µs para iniciar a medição de seguida o sensor envia um impulso ultrassónico, ao mesmo tempo que é enviado o impulso é ativado o pino de eco este permanece ativo até receber o eco do impulso utilizando o tempo de ativação do pino para realizar o calculo da distancia sendo este dado por (tempo de ativação do pino eco × velocidade do som) / 2 . Este oferece uma faixa de deteção de 2 cm até 4 metros e é amplamente utilizado em aplicações de robótica, automação e segurança. Neste projeto, é utilizado para verificar a distância entre o mecanismo que agarra os contentores e o chão.

Figura 9: HC-SR04 Ultrasonic Sensor [8]

2.2.8 28BYJ-48 Stepper Motor

O motor de passo 28BYJ-48 12 V costuma ser utilizado em projetos de automação e controlo de movimento. Ele é capaz de realizar movimentos suaves e controlados devido a uma engrenagem interna (relação 1:64), o que faz o motor ter um ângulo de passo 5,625°, com precisão de ângulo de passo de 3%. O motor tem cerca de 0.34 N.cm de torque sendo

um motor Unipolar de 5 fios, controlado por uma placa driver ULN2003. Neste projeto, o 28BYJ-48 *Stepper Motor* é utilizado para controlar o movimento do guindaste nos eixos Y e Z sendo utilizado um motor para o eixo Y e dois para o eixo Z.

Figura 10: 28BYJ-48 Stepper Motor [9]

2.3 Desenhos CAD

2.3.1 Maquete

2.3.1.1 Idealização

O primeiro passo para a realização da maquete foi escolher a base, para o que foram verificadas as dimensões disponíveis numa loja de materiais. Foi escolhida uma base de contraplacado com dimensões de 1200 x 600 x 10 mm, juntamente com o material para os pilares e vigas sendo estes feitos de PVC branco. A maquete é constituída por 8 vigas de 400 mm e duas vigas de 1000 mm, uma viga oca em alumínio usado para passagem de cabos, um ESP32, um *Raspberry* Pi, tendo depois as seguintes peças realizadas na impressora 3D, um caminho de ferro, dois carris, dois contentores e estrutura com guindaste para a recolha dos contentores.

2.3.1.2 Desenho base Maquete CAD

Com o material inicial disponível, podemos então começar a desenhar a maquete. Após desenharmos os componentes adquiridos, podemos prosseguir com o desenho das peças para fixar os pilares e as ligações às vigas.

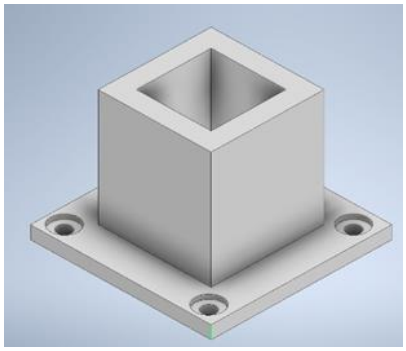

Figura 12: Peças de fixação inferiores

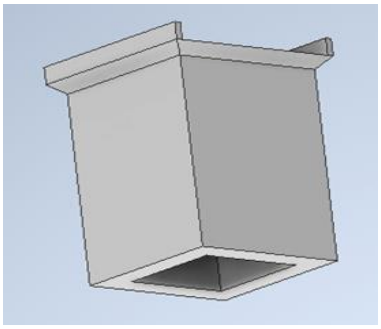

Figura 11: Peças de fixação superiores

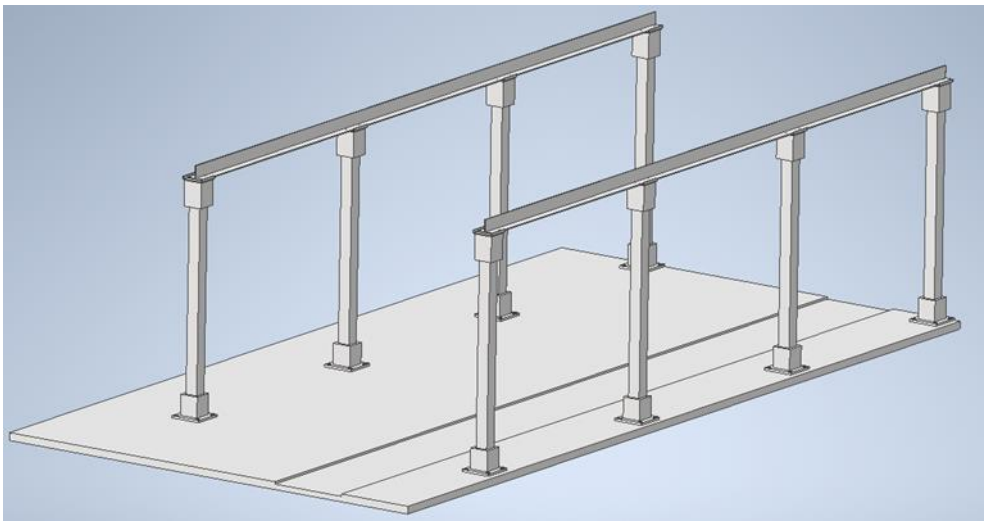

Figura 13: Estrutura completa das vigas

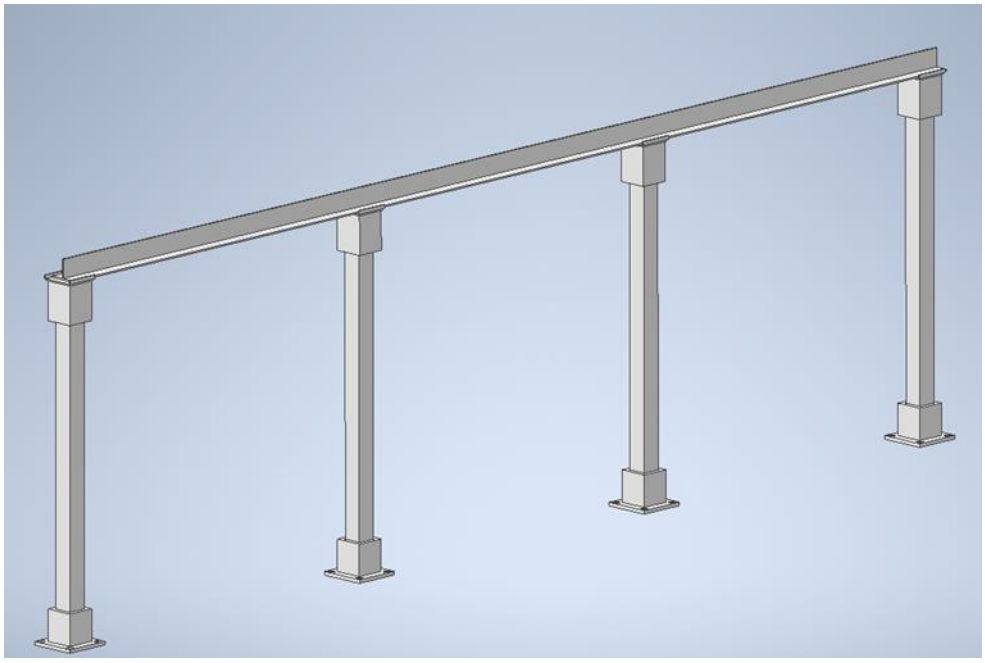

Figura 14: Estrutura das vigas

2.3.1.3 Desenho mecanismo para agarrar contentor (*Rack*_parte_baixo)

Com a base concluída, foi elaborado o mecanismo para recolher o contentor. Para isso, começou-se por procurar um modelo de contentor numa base de desenhos 3D, que serviu de referência para o projeto.

Figura 15: Contentor

Este modelo foi posteriormente alterado para acomodar um código QR no centro do contentor, como pode ser visualizado na imagem. Com o contentor disponível, foi então iniciado o desenvolvimento do acoplamento para o agarrar.

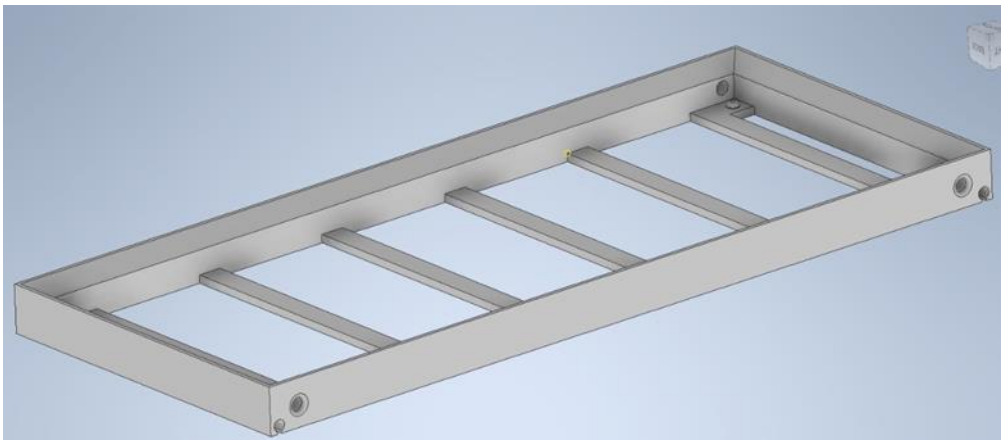

Figura 16: Início da estrutura Rack

Como podemos observar, este acoplamento, denominado "*Rack*", foi desenhado com atenção especial à geometria, para facilitar o alinhamento com o contentor. Após vários desenhos e testes com peças, foi definido o desenho final.

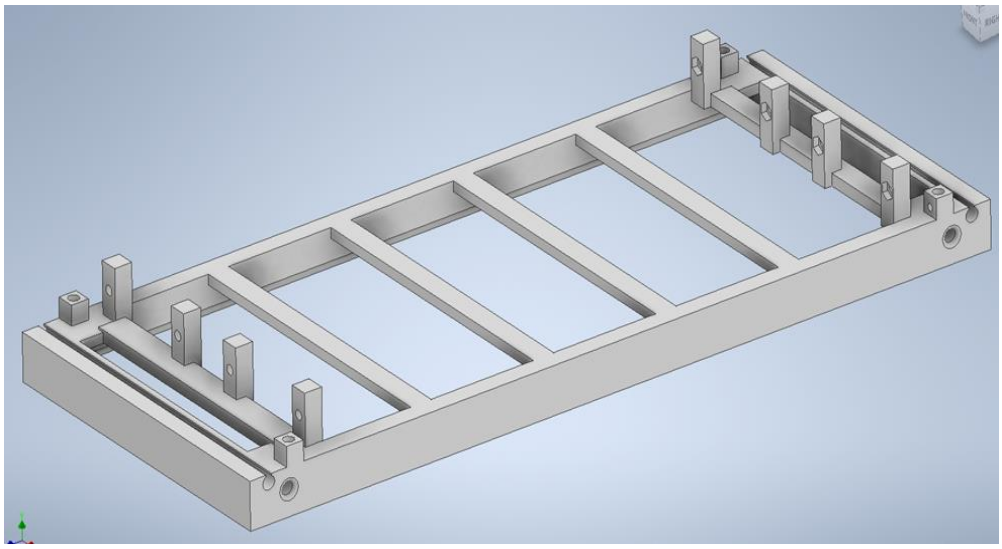

Figura 17: Estrutura final Rack baixo

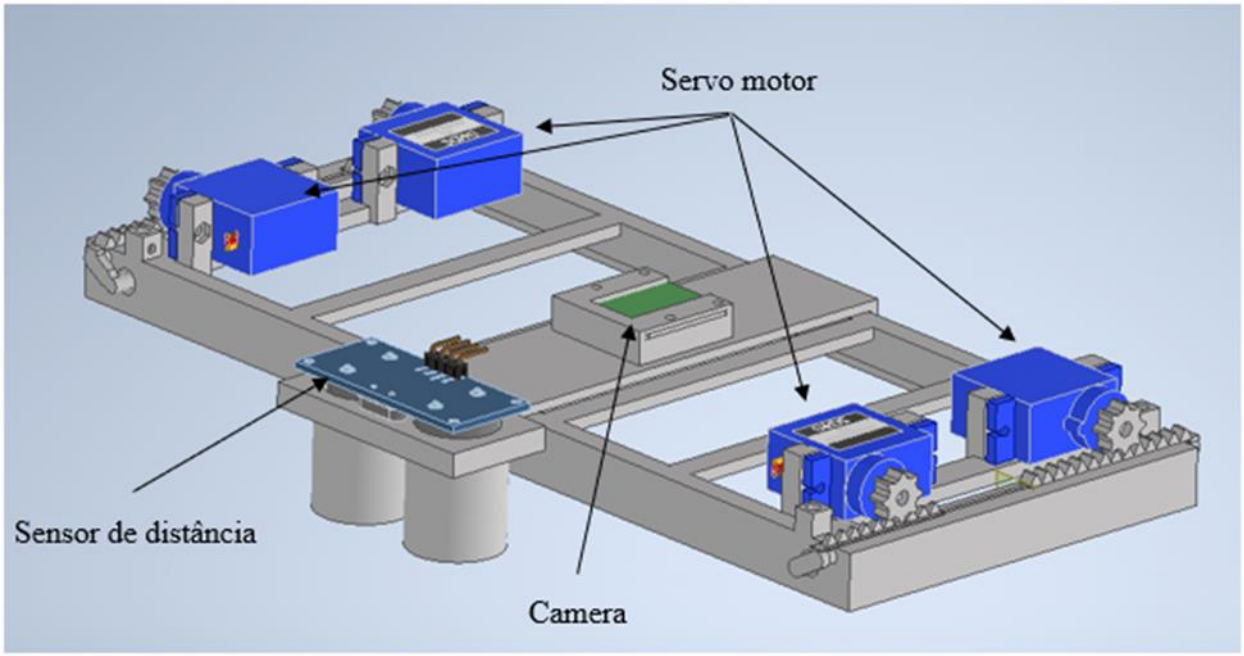

Figura 18: Rack baixo com periféricos

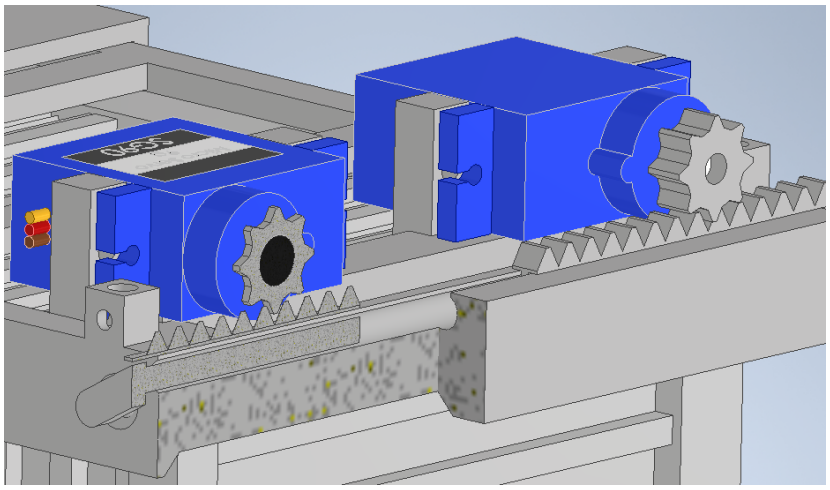

Figura 19 Sistema de fixação do contentor

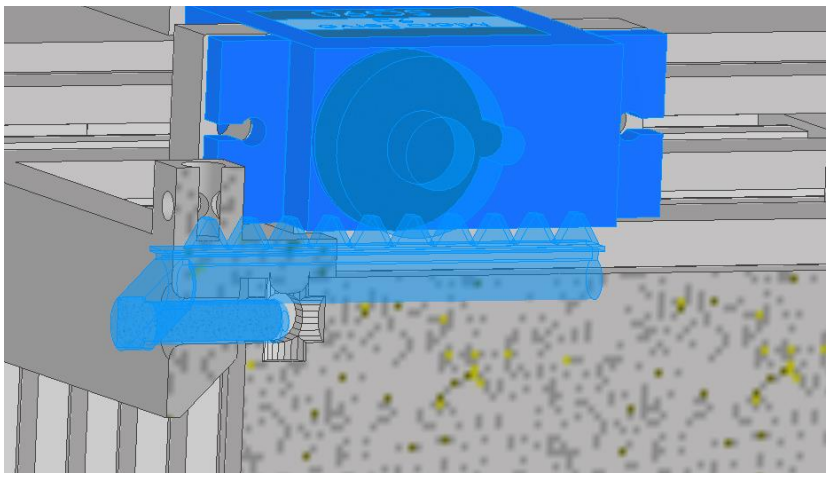

Figura 20 Vista de corte do sistema de fixação (trancas fechadas)

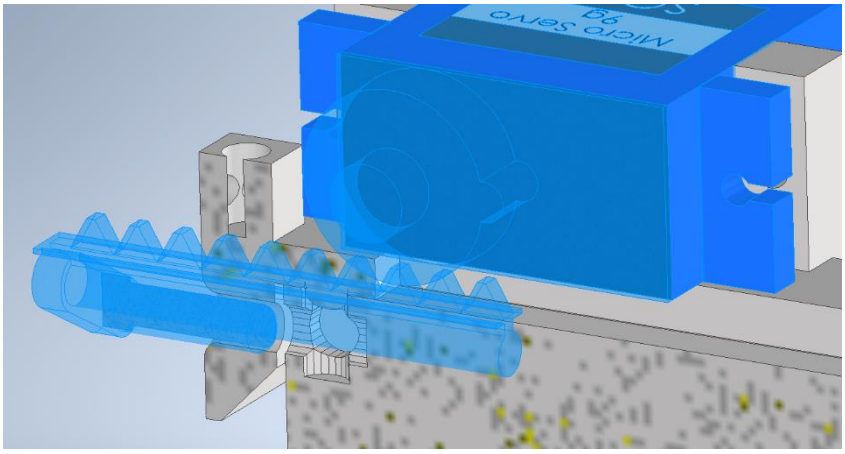

Figura 21 Vista de corte do sistema de fixação (trancas abertas)

2.3.1.4 Desenho mecanismo para elevação do contentor (*Rack*_parte_cima)

Com o acoplamento do contentor concluído, passámos à construção da parte superior que irá elevar a estrutura. Inicialmente, foi pesquisada a melhor forma de elevar a estrutura, considerando sistemas hidráulicos e por cabo.

Os sistemas hidráulicos, embora eficazes, exigiriam mais componentes (como bomba, filtros e electroválvula) e poderiam sofrer de problemas como fugas devido ao desgaste. Além disso, a utilização de um sistema hidráulico implicaria a necessidade de modificar o *Rack*_baixo para acomodar este sistema.

Foi então decidido optar por cabos para a elevação. Para puxar os cabos, serão utilizados dois motores de 12 V, previamente mencionados no ponto 2.2.8 identificados visualmente a azul na figura seguinte.

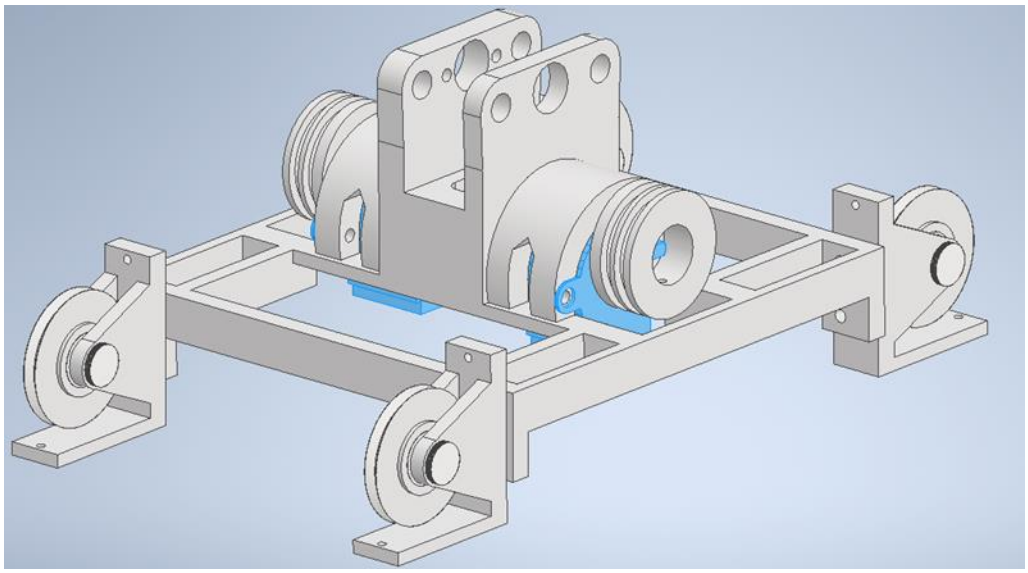

Figura 22: Rack parte de cima

Este design teve em consideração a passagem dos fios elétricos e a ligação à estrutura de apoio.

2.3.1.5 Desenho mecanismo para movimentação no eixo X

Para a movimentação do *Rack* no eixo X, foi estudado o funcionamento das impressoras 3D para movimentar a cabeça de impressão. Com base nesse conhecimento, optou-se por utilizar um motor Nema 17 e um fuso sem fim para a deslocação.

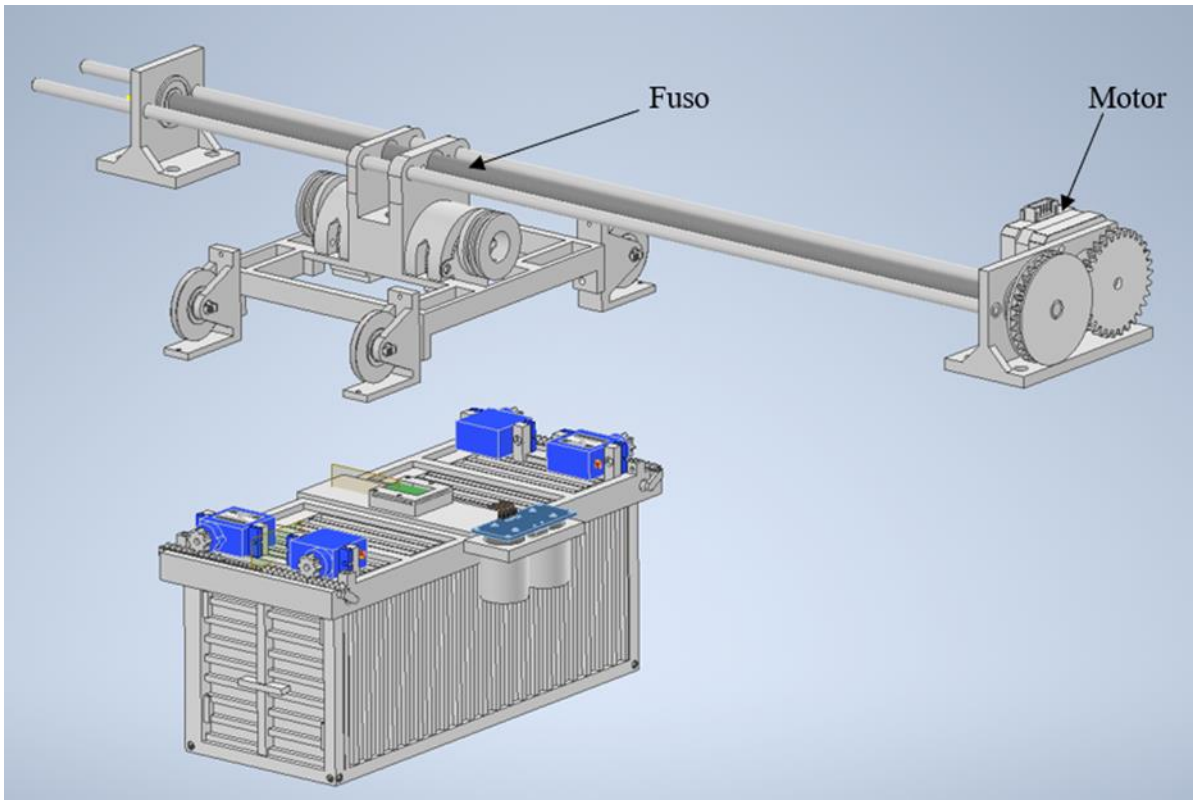

Figura 23: Mecanismo de movimento eixo X

2.3.1.6 Desenho mecanismo para movimentação no eixo Y

Com a movimentação no eixo X resolvida, a atenção voltou-se para o eixo Y, utilizando as vigas previamente criadas como suporte para a locomoção. A idealização do mecanismo de locomoção no eixo Y foi inspirada no funcionamento das lagartas de tanque, adaptando o mecanismo às necessidades do projeto.

Depois de definir como a estrutura iria deslocar-se ao longo da viga, foi realizada a construção da estrutura para acomodar as rodas dentadas e a ligação ao sistema do eixo X.

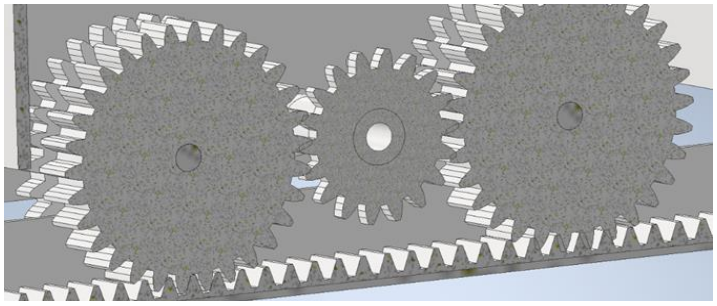

Figura 24: Rodas dentadas mecanismo de locomoção eixo Y

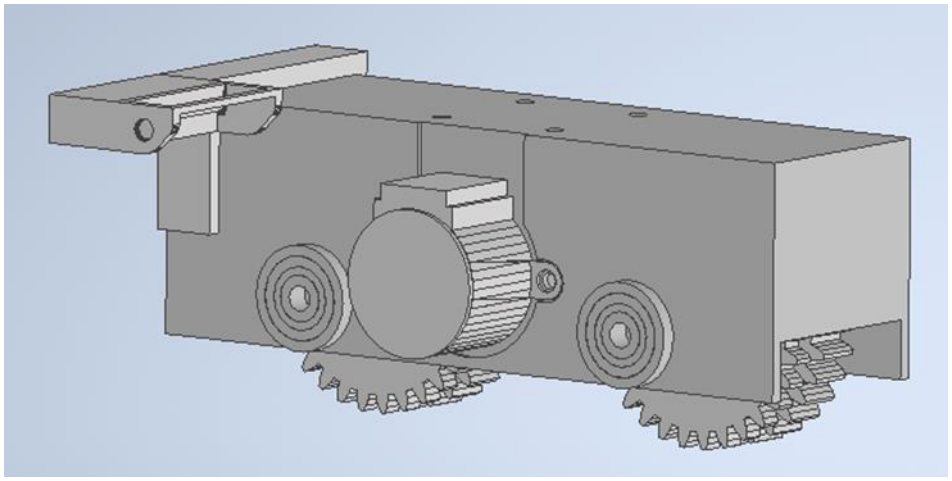

Figura 25: Mecanismo movimentação eixo Y

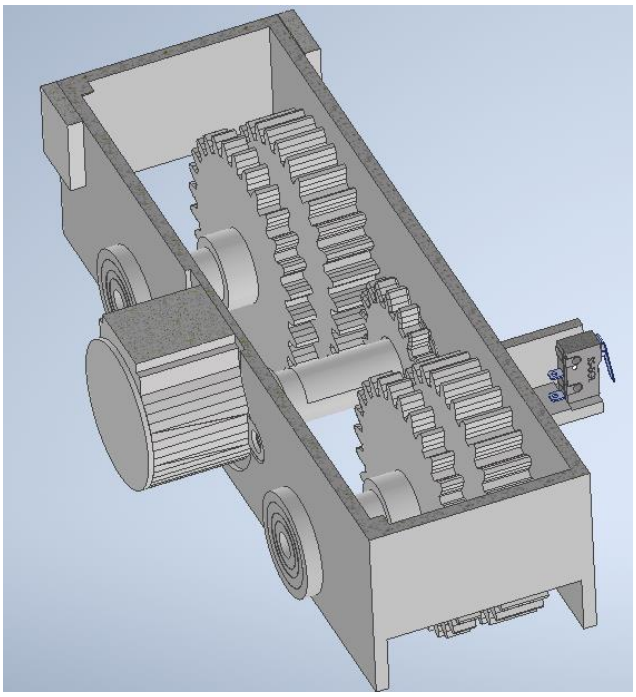

Figura 26: Vista de corte do mecanismo movimentação eixo Y

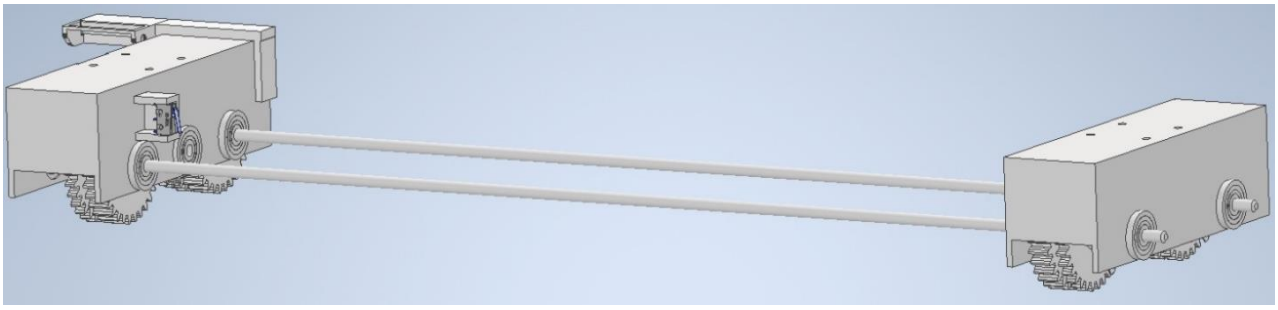

Figura 27: Estrutura completa do mecanismo movimentação eixo Y

2.3.1.7 Desenho mecanismo para passagem de fios elétricos

Devido às dimensões da maquete, as placas de comando foram colocadas na base, o que gerou a necessidade de passar vários cabos por toda a maquete para a ligação dos diversos componentes eletrónicos. Com base na gestão de cabos utilizada em máquinas CNC e impressoras 3D, foram feitos ajustes a uma "*cable chain*" encontrada na internet para atender às necessidades do projeto.

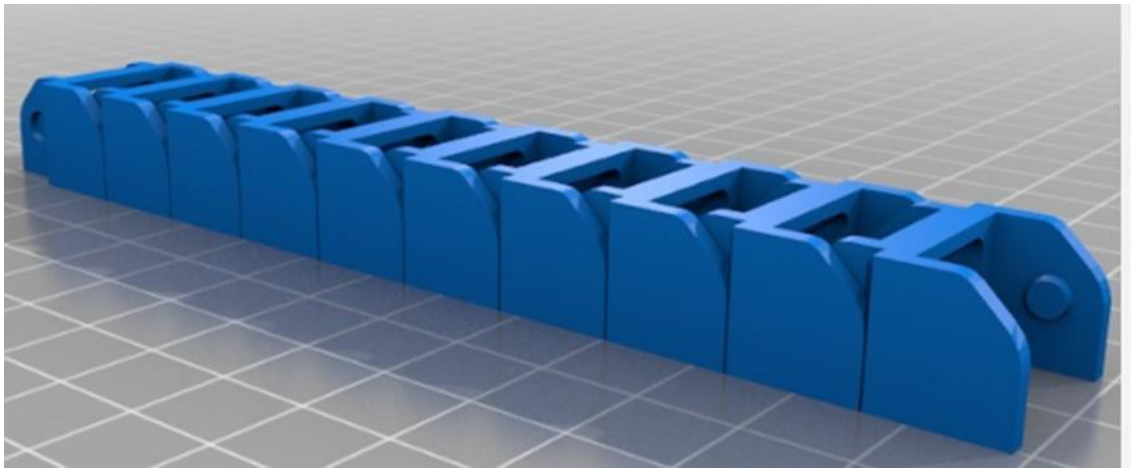

Figura 28: Cable chain [9]

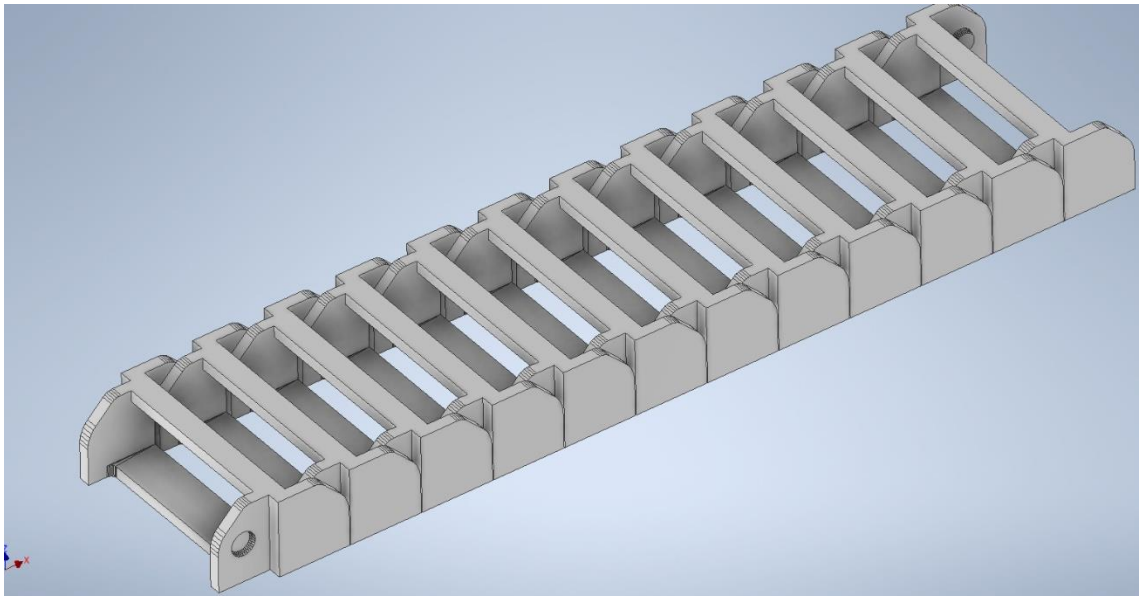

Figura 29: Cable chain modificada

Além disso, foi necessário criar os terminais de ligação.

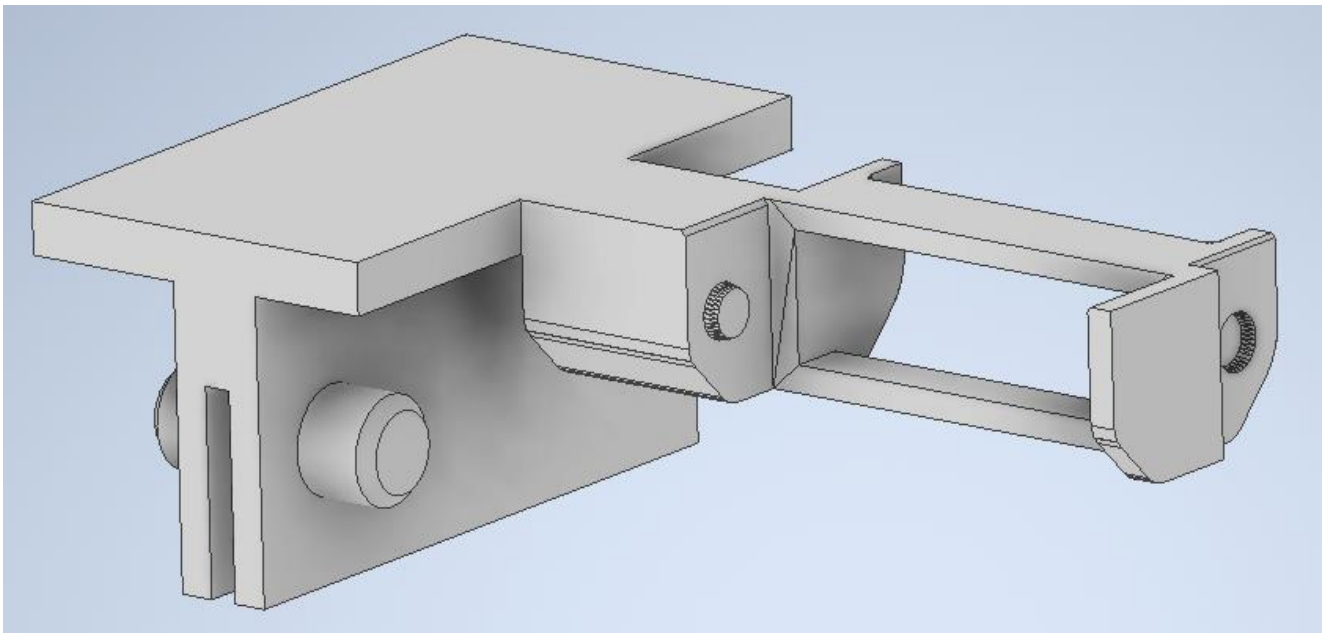

Figura 30: Elo de apoio

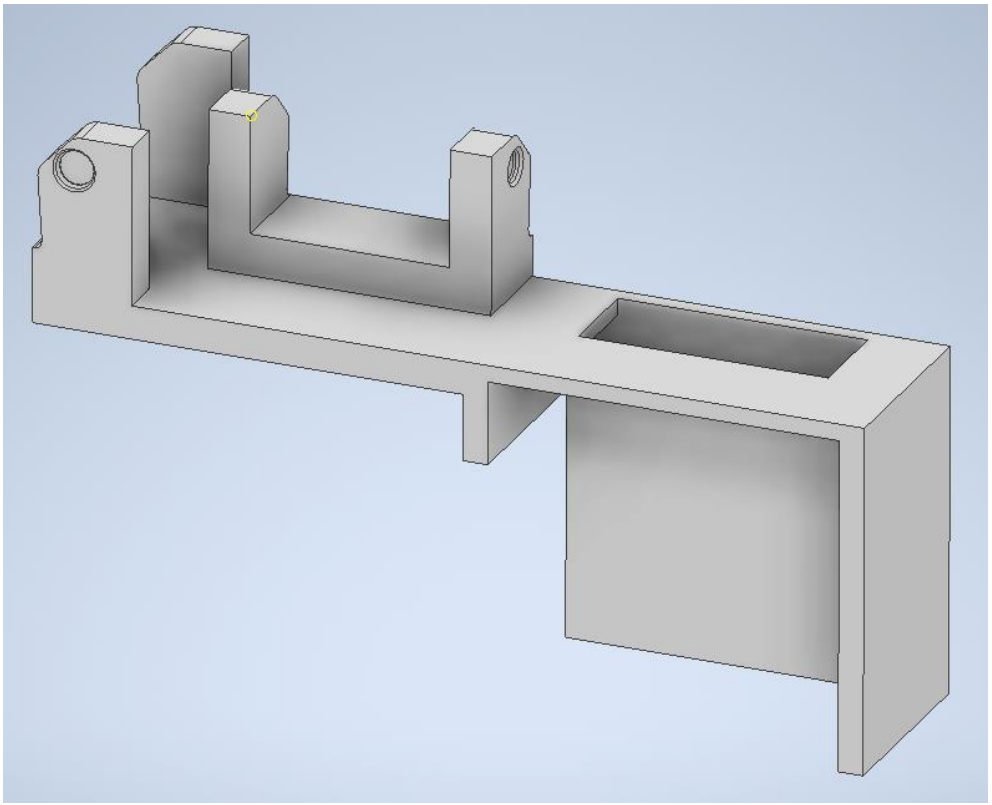

Figura 31: Elo inicial e final de cabos X e Y

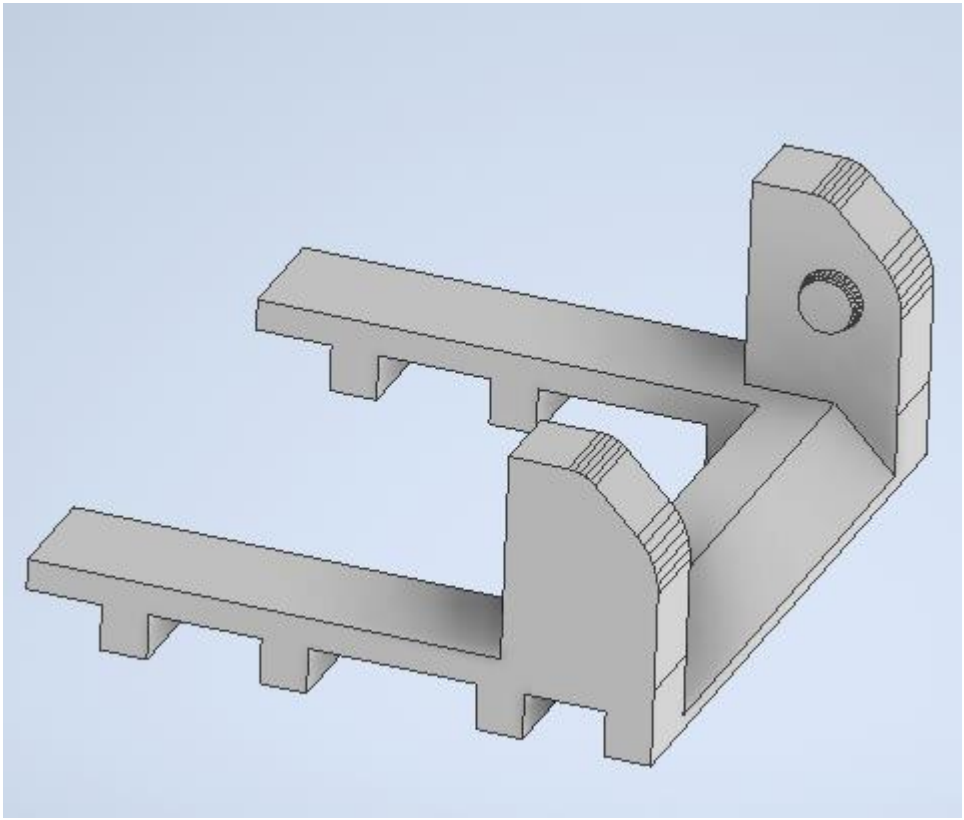

Figura 32: Elo inicial Rack

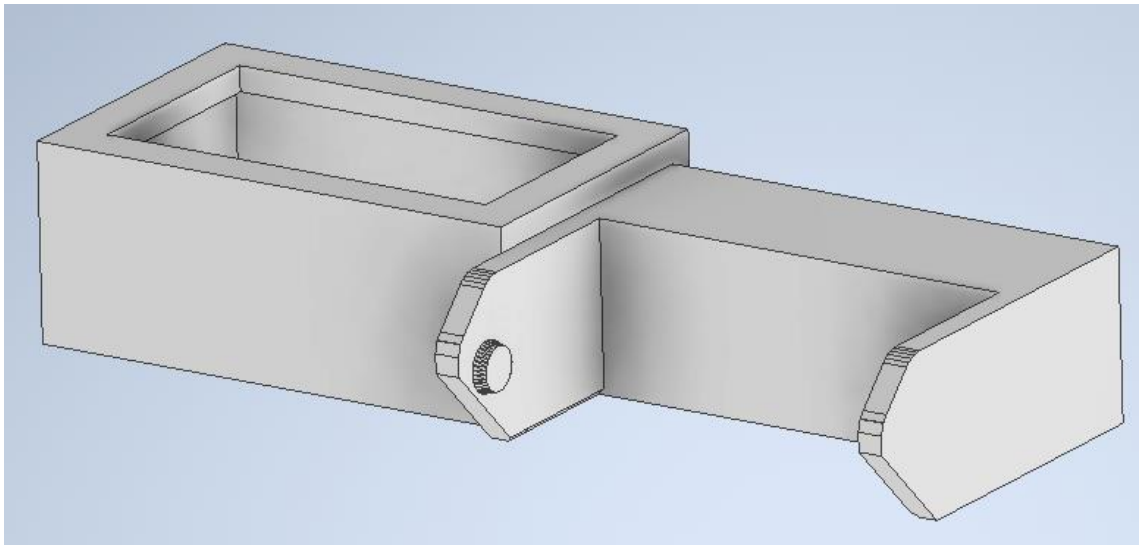

Figura 33: Elo final torre

Ao juntar os vários componentes previamente analisados, conseguimos ter uma visão clara da maquete final.

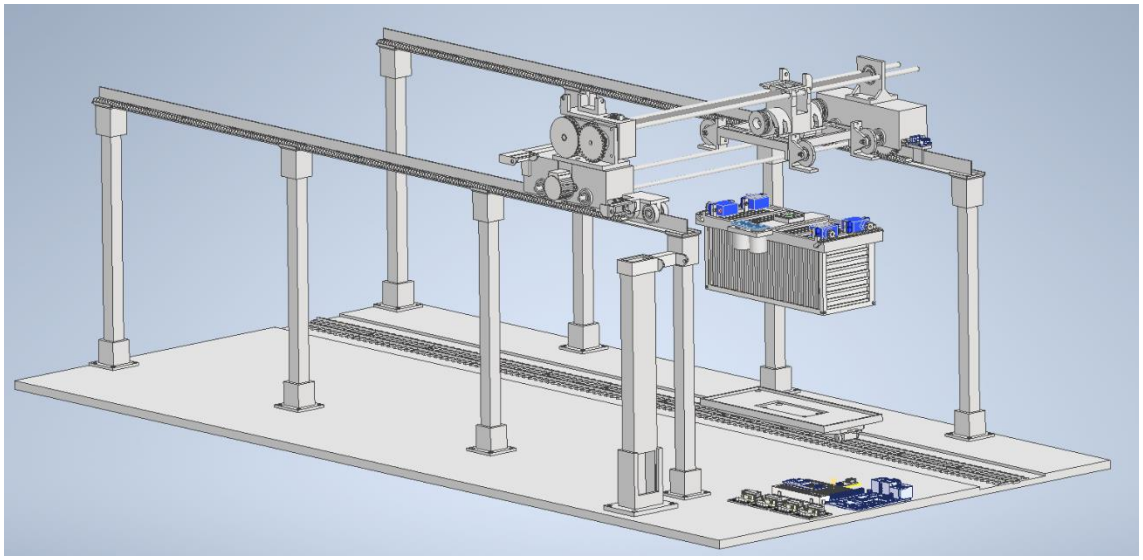

Figura 34: Maquete completa 3D

2.4 Maquete Final

Irá ser apresentada de seguida a maquete realizada fisicamente composta por todas as peças e componentes eletrónicos anteriormente descritos.

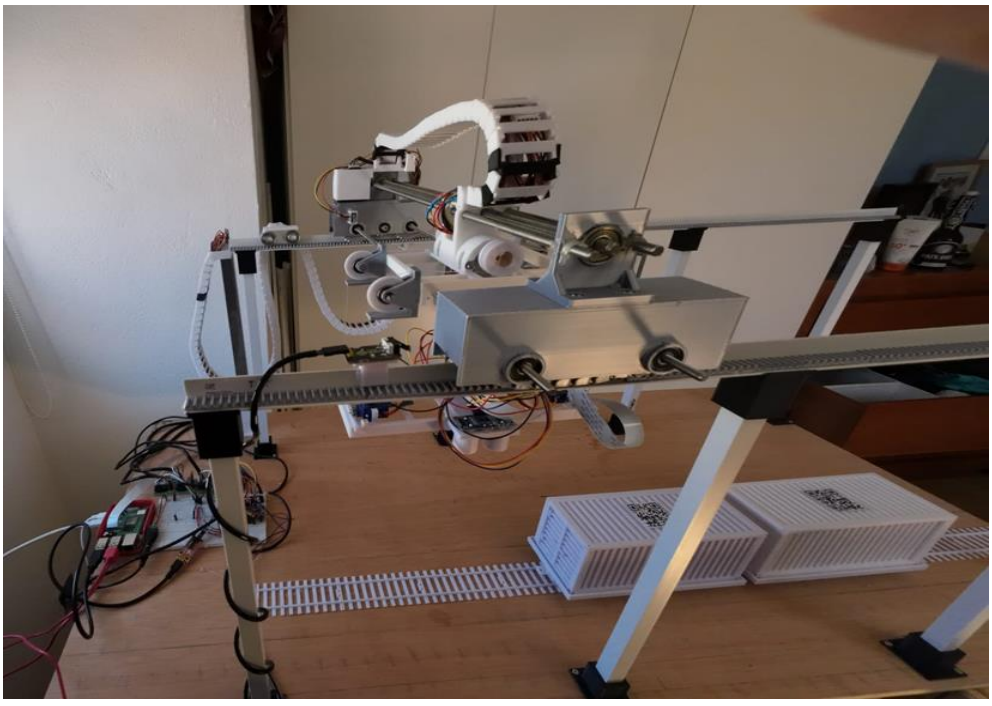

Figura 35: Imagem I real da maquete

Figura 36: Imagem II real da maquete

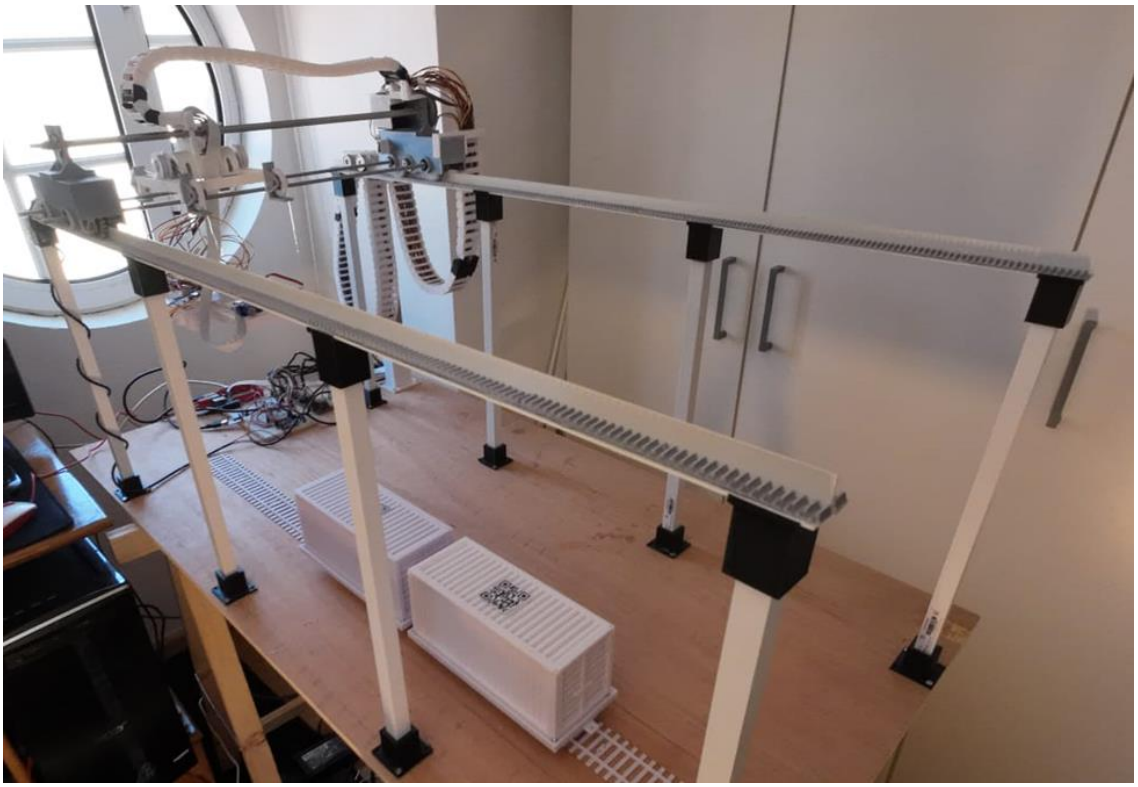

Figura 37: Imagem III real da maquete

Capítulo 3: Programação dos Controladores

3.1 Introdução

Neste capítulo, iremos abordar a programação dos controladores, focando em como o sistema funciona e nas várias classes criadas, estando o código disponível para consulta em anexo.

3.2 ESP32

O microcontrolador ESP32 será responsável por comandar toda a parte mecânica da estrutura, que é composta por motores, limitadores, servomotores e sensores de distância.

3.2.1 Biblioteca hscr04

Esta biblioteca, da autoria de Roberto Sánchez [11], é responsável pelo controlo do Sensor de Distância Ultrassónico HC-SR04. A função utilizada permite a medição da distância em milímetros e inicializa com o programa principal para posicionar a estrutura *Rack* a uma distância de 305 mm da base, sendo esta a altura ideal, deixando uma distância de segurança entre o Rack parte cima e o Rack parte baixo.

3.2.2 Biblioteca *locks*

Esta biblioteca é responsável pelo sistema de abertura e fecho das trancas para prender o contentor, utilizando uma biblioteca para controlar os servos. A razão para a separação do código em várias bibliotecas, em vez de usar uma única folha de código, baseia-se no princípio do encapsulamento. Este método facilita a identificação de problemas e garante que alterações numa parte do código não provoquem problemas no restante.

A biblioteca inclui duas funções principais: uma para abrir o mecanismo, ‘abre()’, e outra para fechá-lo, ‘fecha()’.

3.2.3 Bibliotecas *stepmotor* e *stepmotor_no_stopper*

Estas bibliotecas são utilizadas para comandar o motor de passo 28BYJ-48 de 12 V. No caso do mecanismo *Rack*, que não utiliza um limitador mas sim um sensor de distância para determinar a sua posição no eixo Z, a função '*home()*' da biblioteca original '*stepmotor*' deixa de ser necessária. Por isso, foi criada a biblioteca '*stepmotor_no_stopper*', que se diferencia da original pela supressão de um input na classe, a remoção da função '*home()*' e a adição de uma nova função.

As funções existentes na biblioteca '*stepmotor*' são as seguintes:

- '*Move*(direcao, dist)': Esta função recebe como input a direção de rotação e a distância a percorrer em milímetros, ativando o motor para percorrer a distância requerida.
- '*home()*': Esta função move a estrutura até encontrar o sensor de pressão, colocando a estrutura no ponto de origem do referencial.
- 'Posi()': Esta função retorna a posição atual do mecanismo em relação ao ponto de origem.
- '*Get_to*(posicao)': Utiliza a função 'Posi()' para determinar a posição atual e, através de cálculos, move a estrutura para a posição desejada. A posição é fornecida como input.
- 'Calib(direcao)': Desenvolvida para ajustar o valor de "steps" do motor, esta função determina o número de passos necessários para mover o motor 1 mm. É especialmente útil quando várias rodas dentadas estão envolvidas, assegurando que a rotação/deslocação final corresponda a 1 mm.

Na biblioteca '*stepmotor_no_stopper*', foi adicionada a seguinte função extra:

- 'Alt(distancia)': Esta função altera a posição do mecanismo para o valor de input fornecido sendo utilizada no arranque do programa para informar a distancia ao chão do Rack, através da informação fornecida pelo sensor de distância.

3.2.4 Biblioteca *stepnema*

Esta biblioteca possui as mesmas funções que a biblioteca '*stepmotor'*, sendo utilizada para comandar o motor *Nema 17 Stepper Motor*. Esta inclui as funções essenciais para controlar a rotação e o posicionamento do motor, permitindo uma operação precisa e ajustável, conforme necessário para o projeto.

3.2.5 Biblioteca uart_serial

Esta biblioteca tem as seguintes funções:

- **'*Send(message)*':** Esta função envia a mensagem ('*message*') através da UART.
- **'*Receive()*':** Esta função aguarda a receção de uma mensagem enviada através da UART e, em seguida, retorna a mensagem, separando cada palavra e criando um tuplo.

3.2.6 *Main*

O programa principal ('*main*') incorpora todas as bibliotecas anteriormente descritas e é responsável por todo o processamento de informações e execução de comandos.

Estrutura do Programa '*main*':

- **Importação de Bibliotecas:** Primeiramente, são importadas todas as bibliotecas necessárias para o correto funcionamento do programa, incluindo as anteriormente descritas e a biblioteca `time`.
- ***Setup*:** Em seguida, os pinos são atribuídos a cada uma das classes para garantir o controlo adequado dos periféricos.

Inicialização:

Com as classes e bibliotecas configuradas, a estrutura é colocada na posição inicial. As seguintes funções são utilizadas:

- **'trancas.fecha()':** Fecha as trancas.
- **'*y_motor.home()*':** Move o motor no eixo Y para a posição inicial.
- **'*x_motor.home()*':** Move o motor no eixo X para a posição inicial.

Usando '*sensor.distance_mm()*', o *Rack* é posicionado aproximadamente a 305 mm do solo, utilizando '*z_motor.alt(distance)*' para ajustar a altura correta.

Após colocar a estrutura na posição inicial, é enviada uma mensagem pela UART para o *Raspberry* Pi, informando que o sistema está pronto para receber ordens.

Funções:

Com base nas bibliotecas, foram criadas funções para realizar ações específicas:

- **'agarra_contentor()':** Abre as trancas, desce o *Rack* até ao contentor, fecha as trancas e sobe o *Rack* até à altura inicial.
- **'deslarga_contentor()':** Desce o *Rack* até que o contentor toque no solo, abre as trancas, sobe o *Rack* até à altura inicial e fecha as trancas.
- **'guarda_contentor()':** Verifica qual foi a última posição utilizada para descarregar um contentor e move a estrutura para a próxima posição disponível. Neste projeto, existem duas posições para descarregamento, mas o número pode ser facilmente expandido.
- **'volta_inic()':** Retorna o *Rack* para a posição inicial sobre os carris.

Main - Programa Principal:

1. O programa começa aguardando a receção de uma mensagem através da UART-Serial.
2. Após receber a mensagem, verifica a posição da estrutura nos eixos X e Y e descodifica a mensagem.
3. Se a mensagem começar por '*move*', o programa entra no primeiro '*if*' para verificar se a deslocação solicitada está dentro dos limites da estrutura. Caso esteja, os motores são ordenados a mover-se a distância requerida.
4. Se a mensagem for '*go_to*', a função '*go_to*' das bibliotecas é utilizada para deslocar a estrutura para a posição desejada.
5. Se a mensagem for 'agarra', o programa procede ao armazenamento do contentor usando as funções descritas anteriormente.

6. Sempre que uma dessas ações é concluída, é enviada uma mensagem ao *Raspberry* Pi indicando o fim da ação solicitada.

3.3 *Raspberry* Pi

O controlador *Raspberry* Pi será responsável pela aquisição e processamento de imagens, identificando a forma dos contentores e os códigos QR. Após o processamento, o *Raspberry* Pi envia comandos ao ESP32 para o correto posicionamento do *Rack* antes da recolha do contentor.

3.3.1 Biblioteca cv2(*OpenCV*)

O *OpenCV* (*Open Source Computer Vision Library*) é uma biblioteca amplamente utilizada para visão computacional, oferecendo uma vasta gama de funcionalidades para o processamento de imagens e vídeos. Abaixo estão algumas das suas principais funcionalidades:

- **Processamento de Imagens:** Inclui a aplicação de filtros, transformações geométricas, deteção de contornos e outras técnicas avançadas para manipulação de imagens.
- **Análise de Vídeo:** Permite a captura de vídeo a partir de câmaras, processamento em tempo real e análise de movimento, facilitando o desenvolvimento de aplicações que requerem monitorização visual.
- **Visão Computacional:** Oferece ferramentas para a deteção e reconhecimento de faces, objetos, e suporte para recursos de realidade aumentada, tornando-o essencial para projetos que envolvem a compreensão e interpretação de dados visuais.

3.3.2 Biblioteca *picamera2*

A biblioteca '*picamera2*' foi desenvolvida para ser utilizada com as câmaras do *Raspberry* Pi. Esta oferece uma interface *Python* simplificada para adquirir imagens e vídeos, além de permitir a configuração avançada das câmaras. As principais funcionalidades são:

- **Captura de Imagem:** Permite a captura de fotografias com diferentes resoluções e formatos.
- **Gravação de Vídeo:** Facilita a gravação de vídeos com opções configuráveis, como taxa de quadros e qualidade.
- **Controlo Avançado:** Oferece ajuste de parâmetros como exposição, equilíbrio de cores e ganho, permitindo um controlo mais detalhado sobre as condições de captura e a qualidade das imagens e vídeos.

3.3.3 Biblioteca libcamera

A *libcamera* é uma biblioteca e *framework* de código aberto projetado para gerir a captura de imagens e vídeos em dispositivos *Linux*. O seu objetivo é substituir as interfaces de câmara específicas do fornecedor, fornecendo uma API comum e extensível. As principais funcionalidades são:

- **Abstração de *Hardware*:** Suporte a diversas câmaras e sensores sem necessidade de alterações específicas no código.
- **Flexibilidade:** APIs para desenvolvedores que permitem a criação de pipelines de processamento personalizados.
- **Integração:** Projetada para trabalhar em conjunto com outros sistemas e bibliotecas, como *GStreamer* e V4L2.

3.3.4 Biblioteca *pyzbar*

O '*pyzbar*' é uma biblioteca *Python* utilizada para decodificar códigos de barras e QR *codes* a partir de imagens. É uma ferramenta eficaz para aplicações que necessitam da leitura de códigos, como sistemas de inventário, ponto de venda e controle de acesso. As principais funcionalidades são:

- **Decodificação de Códigos de Barras:** Suporta diversos formatos de códigos de barras, incluindo EAN, UPC, *Code* 128, entre outros.
- **Decodificação de QR *Codes*:** Permite a leitura de QR *codes* em imagens com alta precisão.
- **Simplicidade:** É fácil de integrar com outras bibliotecas, como o *OpenCV*, para processamento de imagens antes da decodificação.

3.3.5 Biblioteca *qr_code_class*

Esta biblioteca foi criada com o intuito de encapsulamento de código, conforme descrito anteriormente, e utiliza todas as bibliotecas previamente mencionadas para o seu funcionamento. As funções existentes na biblioteca são:

- **'*Get_qr()*':** Esta função verifica a imagem à procura de QR *codes*. Se encontrar um QR *code*, descodifica-o e devolve a informação contida. Caso não encontre nenhum QR *code*, devolve o caractere `N`.
- **'*Get_xy()*':** Esta função retorna a distância do centro do QR *code* em relação ao centro da imagem.

3.3.6 Biblioteca *uart_class*

A biblioteca '*uart_class*' utilizada no Raspberry Pi contém as mesmas funções que a biblioteca '*uart_serial*' utilizada no ESP32.

3.3.7 *Main_rasp*

Este programa, tal como o *main* utilizado no ESP32, utiliza as funções anteriormente descritas e as suas respectivas inicializações.

Estrutura do Programa:

1. **Importação de Bibliotecas:** Primeiramente, são importadas todas as bibliotecas necessárias para o funcionamento do programa.
2. **Setup:** Seguidamente, inicializamos cada uma das classes, nomeadamente '*Uart_1*' e '*Qr_code*'.

Funções:

3. **Com base nas bibliotecas, foram criadas funções para realizar certas ações:**

Find_conteiner(): Esta função envia para o ESP32 o comando *go_to* 211 0 e aguarda a receção da mensagem de retorno. Esta posição é diretamente em cima da linha ferroviária. Em seguida, verifica se existe um código QR na imagem. Caso não seja detetado, a estrutura é comandada a avançar 3,5 cm e a verificação é repetida. O ciclo continua até que um contentor seja encontrado ou a estrutura atinja o limite físico.

Posi_conteiner(): Esta função tem como objetivo alinhar a estrutura com o contentor, usando o centro do código QR como referência. Utiliza a função get_xy da biblioteca *qr_code_class* para verificar a distância X e Y e comanda o ESP32 para deslocar a estrutura até que esta esteja alinhada com o contentor.

Guarda_conteiner(): Esta função envia o comando para agarrar e guardar o contentor.

Main – while loop:

Antes do início do *while loop*, há uma linha que aguarda uma mensagem através da UART, indicando que o ESP32 foi inicializado. Esta linha está fora do *while*, visto ser utilizada apenas quando o programa é iniciado pela primeira vez. Utilizando a *Shell* do *Thonny* como terminal, pressiona-se a tecla *Enter* do teclado quando um comboio se encontre dentro do armazém, esta tecla depois de pressionada aciona a função para encontrar os contentores. Se o programa não encontrar contentores, volta para a posição inicial e aguarda a receção do input que indica a presença de um comboio. Ao encontrar um contentor, este é recolhido e colocado no local de descarga, e a procura por mais contentores continua até que não haja mais nenhum na linha.

Capítulo 4: Conclusão e Futuros Desenvolvimentos

4.1 Conclusão

Ao utilizar os conhecimentos adquiridos ao longo da carreira académica, foi possível realizar o projeto, sendo atingidos os objetivos propostos.

O objetivo principal do projeto foi realizado correspondendo á recolha e descarregamento de contentores automaticamente, sendo o sistema de armazenamento e distribuição dos mesmos o próximo passo para a elevação do projeto a outros patamares. O mecanismo de movimentação dos contentores no armazém seria semelhante ao sistema utilizado para o descarregamento, sendo necessário apenas tratar da programação do *Raspberry* Pi e da realização da estrutura para acomodação dos camiões para carregamento.

Desafios Encontrados:

Durante a realização do projeto, foram encontrados diversos desafios que prolongaram a execução de certas tarefas, nomeadamente na criação da estrutura e sua produção, comunicação entre controladores e deteção de linhas. Estes desafios foram os seguintes:

- **Impressão 3D:** Os desafios associados à impressão 3D advêm das tolerâncias que esta tecnologia pode proporcionar. Dado que as impressoras utilizadas não conseguem uma alta precisão em termos de dimensionamento, foi necessário refazer várias peças para ajustar as tolerâncias da impressora 3D, o que resultou em problemas adicionais relacionados com o tempo necessário para a produção das peças. Embora a impressão 3D possa ser considerada rápida para a realização de peças de teste, o tempo de produção pode chegar a 12 horas para peças de maiores dimensões.
- **Comunicação entre Controladores:** Ao estabelecer a comunicação entre o ESP32 e o *Raspberry* Pi, surgiu um obstáculo inicial devido ao ESP32 ser o controlador responsável por todo o sistema mecânico e sensorial, exceto o sistema visual. Com apenas 2 pinos disponíveis para comunicação, foi necessário utilizar um protocolo

de comunicação que usasse apenas 2 linhas e fosse bidirecional. Foram considerados vários protocolos, como UART, I2C e SPI, destacando-se o protocolo Serial UART como o mais indicado. Durante o desenvolvimento do código para a comunicação *Serial* UART, surgiu um problema: a comunicação do ESP32 para o *Raspberry* Pi funcionava corretamente, mas o envio de informações do *Raspberry* Pi para o ESP32 não era realizado. Após várias tentativas de resolução, como alteração de pinos, código, processador ESP32 e verificação de ligações, foi especulado que o problema seria dos níveis de tensão dos sinais, resolvido utilizando o adaptador FTDI USB to UART *Serial* dando mais credibilidade a especulação considerando o facto do adaptador incluir deslocadores de nível para converter entre os níveis lógicos internos do chip (normalmente 3,3V) e os níveis externos.

- **Identificação de Objetos:** Outro desafio encontrado foi a identificação do contentor através do tratamento de imagem. Devido à madeira utilizada como base da maquete, a deteção de bordas com o *OpenCV* revelou linhas da madeira que interferiam na identificação do contentor. Este problema foi resolvido utilizando um código QR como identificador do contentor, visto que a sua forma se destaca mesmo em diferentes fundos, sendo este utilizado para o posicionamento da estrutura.

Futuros Desenvolvimentos:

Quando se fala em futuros desenvolvimentos, o foco é no melhoramento do projeto base e na adição de novas funcionalidades. Assim, os melhoramentos idealizados incluem:

- Alteração dos motores para motores com *encoders*, para melhorar a precisão.
- Implementação de mais um ESP32 para libertar pinos, com o objetivo de alterar o tipo de comunicação para *wireless*.
- Alteração da estrutura do *Rack* para permitir a alteração do comprimento consoante o tipo de contentor.
- Implementação do sistema de armazenamento e distribuição.

Bibliografia

[1] EmbeddedWala, (2023, Abril 26). *UART interview questions*. Embedded Wala. https://embeddedwala.com/EmbeddedSystems/interview-questions/uart-interview-questions#google_vignette

[2] Toplampe. (2023, Dezembro 20). *AZ-Delivery ESP32 NodeMCU Modul WLAN WiFi Dev Kit C*. https://www.toplampe.de/az-delivery-esp32-nodemcu-modul-wlan-wifi-dev-kit-c/

[3] Botnroll. (2024). *FTDI Basic Breakout 5V/3.3V (mini-USB)*. https://www.botnroll.com/en/rs232/984-ftdi-basic-breakout-5v33v-with-usb-cable-.html

[4] Qbee. (2023, Setembro 22). *Raspberry Pi Device Management: A Comprehensive guide*. Qbee. https://qbee.io/raspberry-pi-device-management-comprehensive-guide/

[5] Raspberry Pi Ltd. (2024, Agosto 9) The Picamera 2 Library A libcamera-based Python Library for Raspberry Pi cameras. https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf

[6] Bondtech. (2024, Agosto 22). *Nema17 25mm Stepper Motor 18mm O-Shaft - Bondtech*. https://www.bondtech.se/product/nema17-25mm-stepper-motor-18mm-o-shaft/?srsltid=AfmBOopB82r7RuYYJzxVfOd70lFcuMlDqOHSM1199bYHCgWyPe9ogRZH

[7] Thomsen, A. (2021, Outubro 19). *Controlando Micro Servo 9g usando Joystick*. MakerHero. https://www.makerhero.com/blog/controlando-micro-servo-9g-usando-joystick/

[8] Mouser. (2023, Julho 31). SparkFun | *HC-SR04 Ultrasonic Distance Sensor*. https://pt.mouser.com/new/sparkfun/sparkfun-hcsr04-distance-sensor/

[9] LME. (2023, Novembro 4). *Control 28BYJ-48 Stepper Motor with ULN2003 Driver & Arduino*. Last Minute Engineers. https://lastminuteengineers.com/28byj48-stepper-motor-arduino-tutorial/

[10] Thingiverse. (2016, Outubro 13). *Print in place cable chain! (MPCNC compatible) by wbarber69*. Thingiverse. https://www.thingiverse.com/thing:1824144

[11] Rsc. (2021, Dezembro 13). *GitHub - rsc1975/micropython-hcsr04: Micropython driver for ultrasonic sensor HC-SR04*. GitHub. https://github.com/rsc1975/micropython-hcsr04

Anexos

Main raspberry pi

```
from uart_class import Uart_1
from qr_code_class import Qr_code

"init"
uart=Uart_1('/dev/ttyUSB0',115200,1)

qr_code=Qr_code(35,4.74,2304,1296,0.5)

" funtions"
def find_conteiner():
    uart.send("go_to 211 10")
    uart.receive()
    qr=qr_code.get_qr()
    while qr=="N":
        uart.send("move 0 3500")
        mens=uart.receive()
        if mens=="max_range":
            return "max_range"
        qr=qr_code.get_qr()
    return "foud"

def posi_conteiner():
    while True:

        xy=qr_code.get_xy()
        if xy=="N":
            print("no qr-code")
            return "fail_to see_conteiner"
        x=0
        y=int(xy[1])
```

```
        mensagem="move "+str(x)+" "+str(y)
        uart.send(mensagem)
        uart.receive()
        if -20<y<20:
            return "ok"

def guarda_conteiner():
    uart.send("agarra")
    uart.receive()


" Main"
uart.receive()
while True:
    train=input("Train stop")
    while True:
        saida=find_conteiner()
        if saida=="max_range":
            print("no conteiners in track")
            uart.send("go_to 211 10")
            uart.receive()
            break
        else:
            posi=posi_conteiner()
            if posi== "ok":
                qr=qr_code.get_qr()
                guarda_conteiner()
                print("Contentor",qr,"guardado")
            else:

                uart.send("move 0 3500")
                uart.receive()
```

Qr_code_class

```
import cv2
from picamera2 import Picamera2
from pyzbar.pyzbar import decode
import math
from libcamera import controls
# Constants




class Qr_code:
    # Configure UART communication


    def __init__(self, Qr_size,focal,preview_w,preview_h,scale):
        self.qr_width_mm = Qr_size
        self.focal_length_mm = focal

        self.picam = Picamera2()
        self.picam.preview_configuration.main.size = (preview_w, preview_h)
        self.picam.preview_configuration.align()
        self.picam.configure("preview")
        self.picam.start()
        self.picam.set_controls({"AfMode": controls.AfModeEnum.Continuous})
        self.scale_factor = scale




    def get_qr(self):

        frame = self.picam.capture_array()
        resized_frame = cv2.resize(frame, None, fx=self.scale_factor, fy=self.scale_factor)

    # Convert frame to grayscale for better QR code detection
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Decode QR codes
        decoded_objects = decode(gray_frame)

        if not decoded_objects:

            return "N"
        else :
            for obj in decoded_objects:
                qr_data = obj.data.decode("utf-8")

            return qr_data



    def get_xy(self):

            frame = self.picam.capture_array()
            resized_frame = cv2.resize(frame, None, fx=self.scale_factor, fy=self.scale_factor)

        # Convert frame to grayscale for better QR code detection
            gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Decode QR codes
            decoded_objects = decode(gray_frame)

            if not decoded_objects:

                return "No qr code"
            else :
                for obj in decoded_objects:
                    (x, y, w, h) = obj.rect
```

```
    # Calculate distance from center
    center_x = (x + w / 2) * self.scale_factor
    center_y = (y + h / 2) * self.scale_factor
    distance_x = (resized_frame.shape[1] / 2 - center_x)
    distance_y = (resized_frame.shape[0] / 2 - center_y)

    # Calculate field of view angle
    fov_angle_half = math.atan(self.qr_width_mm / (2 * self.focal_length_mm))

    # Calculate distance from center to QR code in millimeters
    distance_to_center_mm_y =-( (distance_x / resized_frame.shape[1]) * 2 * self.focal_length_mm * math.tan(fov_angle_half))
    distance_to_center_mm_x = -((distance_y / resized_frame.shape[0]) * 2 * self.focal_length_mm * math.tan(fov_angle_half))
dist=[distance_to_center_mm_x*100,distance_to_center_mm_y*100]
return dist
```

Uart_class

```
import serial
import time


class Uart_1:
    # Configure UART communication


    def __init__(self, entrada,Baudrate,timeout_1):


        self.serial = serial.Serial(entrada, baudrate=Baudrate, timeout=timeout_1)



    def send(self,message):

        self.serial.write(message.encode())

    def receive(self):
        while self.serial.in_waiting == 0:
            time.sleep(1)

        mensagem=self.serial.readline().decode().strip()
        return mensagem
```

Hcsr04

```
import machine, time
from machine import Pin

__version__ = '0.2.0'
__author__ = 'Roberto Sánchez'
__license__ = "Apache License 2.0. https://www.apache.org/licenses/LICENSE-2.0"

class HCSR04:
    """
    Driver to use the untrasonic sensor HC-SR04.
    The sensor range is between 2cm and 4m.
    The timeouts received listening to echo pin are converted to OSError('Out of range')
    """
    # echo_timeout_us is based in chip range limit (400cm)
    def __init__(self, trigger_pin, echo_pin, echo_timeout_us=500*2*30):
        """
        trigger_pin: Output pin to send pulses
        echo_pin: Readonly pin to measure the distance. The pin should be protected with 1k resistor
        echo_timeout_us: Timeout in microseconds to listen to echo pin.
        By default is based in sensor limit range (4m)
        """
        self.echo_timeout_us = echo_timeout_us
        # Init trigger pin (out)
        self.trigger = Pin(trigger_pin, mode=Pin.OUT, pull=None)
        self.trigger.value(0)

        # Init echo pin (in)
        self.echo = Pin(echo_pin, mode=Pin.IN, pull=None)

    def _send_pulse_and_wait(self):
        """

        """
        Send the pulse to trigger and listen on echo pin.
        We use the method `machine.time_pulse_us()` to get the microseconds until the echo is received.
        """
        self.trigger.value(0) # Stabilize the sensor
        time.sleep_us(5)
        self.trigger.value(1)
        # Send a 10us pulse.
        time.sleep_us(10)
        self.trigger.value(0)
        try:
            pulse_time = machine.time_pulse_us(self.echo, 1, self.echo_timeout_us)
            return pulse_time
        except OSError as ex:
            if ex.args[0] == 110: # 110 = ETIMEDOUT
                raise OSError('Out of range')
            raise ex

    def distance_mm(self):
        """
        Get the distance in milimeters without floating point operations.
        """
        pulse_time = self._send_pulse_and_wait()

        # To calculate the distance we get the pulse_time and divide it by 2
        # (the pulse walk the distance twice) and by 29.1 becasue
        # the sound speed on air (343.2 m/s), that It's equivalent to
        # 0.34320 mm/us that is 1mm each 2.91us
        # pulse_time // 2 // 2.91 -> pulse_time // 5.82 -> pulse_time * 100 // 582
```

```
        mm = pulse_time * 100 // 582
        return mm

    def distance_cm(self):
        """
        Get the distance in centimeters with floating point operations.
        It returns a float
        """
        pulse_time = self._send_pulse_and_wait()

        # To calculate the distance we get the pulse_time and divide it by 2
        # (the pulse walk the distance twice) and by 29.1 becasue
        # the sound speed on air (343.2 m/s), that It's equivalent to
        # 0.034320 cm/us that is 1cm each 29.1us
        cms = (pulse_time / 2) / 29.1
        return cms
```

Locks

```
from servo import Servo
import time


class Locks:


    def __init__(self, pin_placa1,pin_placa2):
        """
        init:---------------------------------
        I1,I2,I3,I4: PINS PLACA DE CONTROLO DO MOTOR
        n_stepy: Numero de Stepps para o step motor 512 é uma rotação 360º
        stopper: botão de atuação que indica a posição inicial
        """
        self.motor1=Servo(pin=pin_placa1)
        self.motor2=Servo(pin=pin_placa2)
        self.motor1.move(180)
        self.motor2.move(0)



    def fecha(self):
        self.motor1.move(0)
        self.motor2.move(180)


    def abre(self):
        self.motor1.move(180)
        self.motor2.move(0)
```

Main esp32

```
from stepmotor import Stepmotor
from stepmotor_no_stopper import Stepmotor_s
from stepnema import Stepnema
from hcsr04 import HCSR04
from locks import Locks
from uart_serial import Uart_1
from time import sleep
"""  Motor init """
x_max=250
y_max=735
y_motor=Stepmotor(19,18,5,21,6,4)
z_motor=Stepmotor_s(26,25,33,32,7)
x_motor=Stepnema(14,12,27,103,13)
"""  sensor init """
sensor = HCSR04(trigger_pin=2, echo_pin=15, echo_timeout_us=1000000)
"""  serial init """
serial=Uart_1(1,17,16,115200)
"""  lock conteiner """
trancas=Locks(22,23)
"""""""""
""""""" position for conteiner swap"
"[livre,x-posição,y-posiçao]"
guarda=[0,0,330],[0,0,655]



""" initiation"""

trancas.fecha()

y_motor.home()
```

```
x_motor.home()
distance = sensor.distance_mm()
while int(distance)<305:
    subir=305-int(distance)
    z_motor.move("+",subir)
    distance = sensor.distance_mm()
z_motor.alt(distance)

serial.send("done")

""   "Funtions"  ""
def agarra_contentor():
    distancia=z_motor.posi()
    trancas.abre()
    z_motor.move("-",(distance-90))
    sleep(1)
    trancas.fecha()
    z_motor.move("+",(305-90))

def deslarga_contentor():
    distancia=z_motor.posi()
    z_motor.move("-",(distance-67))
    trancas.abre()
    z_motor.move("+",(305-67))
    trancas.fecha()

def guarda_contentor():
    if guarda[0][0]==0:
```

```
        x_motor.get_to(guarda[0][1])
        y_motor.get_to(guarda[0][2])
        guarda[0][0]=1
        guarda[1][0]=0
    else :
        x_motor.get_to(guarda[1][1])
        y_motor.get_to(guarda[1][2])
        guarda[1][0]=1
        guarda[0][0]=0
def volta_inic():
    x_motor.get_to(209)
    y_motor.get_to(10)



""   "main"   ""
while True:
    resposta=serial.receive()
    xp=int(x_motor.posi())
    yp=int(y_motor.posi())

    if resposta[0] == "move":
        if (xp+(int(resposta[1])/100))>x_max:
            pass

        else:

            if int(resposta[1])<0:
                x_move="-"

            else :
                x_move="+"
            x_motor.move(x_move,abs(int(resposta[1])/100))


        if (yp+(int(resposta[2])/100))>y_max:
            serial.send("max_range")
        else:
            if int(resposta[2])<0:
                y_move="-"
            else :
                y_move="+"
            y_motor.move(y_move,abs(int(resposta[2])/100))
            serial.send("done")



    if resposta[0] == "go_to":

        x_motor.get_to(int(resposta[1]))
        y_motor.get_to(int(resposta[2]))
        serial.send("done")



    if resposta[0] == "agarra":
        agarra_contentor()
        guarda_contentor()
        deslarga_contentor()
        volta_inic()
        serial.send("done")
```

Servo

```
from machine import Pin, PWM

class Servo:
    # these defaults work for the standard TowerPro SG90
    __servo_pwm_freq = 50
    __min_u10_duty = 26 - 0 # offset for correction
    __max_u10_duty = 123- 0  # offset for correction
    min_angle = 0
    max_angle = 180
    current_angle = 0.001


    def __init__(self, pin):
        self.__initialise(pin)


    def update_settings(self, servo_pwm_freq, min_u10_duty, max_u10_duty, min_angle, max_angle, pin):
        self.__servo_pwm_freq = servo_pwm_freq
        self.__min_u10_duty = min_u10_duty
        self.__max_u10_duty = max_u10_duty
        self.min_angle = min_angle
        self.max_angle = max_angle
        self.__initialise(pin)


    def move(self, angle):
        # round to 2 decimal places, so we have a chance of reducing unwanted servo adjustments
        angle = round(angle, 2)
        # do we need to move?
        if angle == self.current_angle:
            return
```

```
        self.current_angle = angle
        # calculate the new duty cycle and move the motor
        duty_u10 = self.__angle_to_u10_duty(angle)
        self.__motor.duty(duty_u10)

    def __angle_to_u10_duty(self, angle):
        return int((angle - self.min_angle) * self.__angle_conversion_factor) + self.__min_u10_duty


    def __initialise(self, pin):
        self.current_angle = -0.001
        self.__angle_conversion_factor = (self.__max_u10_duty - self.__min_u10_duty) / (self.max_angle - self.min_angle)
        self.__motor = PWM(Pin(pin))
        self.__motor.freq(self.__servo_pwm_freq)
```

Stepmotor

```
from machine import Pin
from time import sleep

class Stepmotor:

    clockways = [[1,1,0,0],[0,1,1,0],[0,0,1,1],[1,0,0,1]]
    anticlockways= [[1,0,0,1],[0,0,1,1],[0,1,1,0],[1,1,0,0]]



    def __init__(self, I1, I2, I3, I4,n_stepy,stopper):
        """
        init:-----------------------------------
        I1,I2,I3,I4: PINS PLACA DE CONTROLO DO MOTOR
        n_stepy: Numero de Stepps para o step motor 512 é uma rotação 360º
        stopper: botão de atuação que indica a posição inicial
        """
        self.IN1 = Pin(I1,Pin.OUT)
        self.IN2 = Pin(I2,Pin.OUT)
        self.IN3 = Pin(I3,Pin.OUT)
        self.IN4 = Pin(I4,Pin.OUT)
        self.stopper = Pin(stopper,Pin.IN)
        self.pins = [self.IN1, self.IN2, self.IN3, self.IN4]
        self.n_step=n_stepy
        self.posicao= 0
```

```
def move(self, direcao,dist):

    """
    move:-----------------------------------
    direção: lado para o qual o motor ira girar
    dist: distancia em mm que queremos o motor girar

    """


    if direcao == "-":
        Dir=Stepmotor.clockways
    else :
        Dir=Stepmotor.anticlockways

    for y in range(dist):


        buton = self.stopper.value()
        if buton==False and direcao== "-":
            self.IN4.off()
            self.IN3.off()
            self.IN2.off()
            self.IN1.off()
            break



        for x in range(self.n_step): # 6 é igual a 1 mm movimento retilinio
            for step in Dir:
```

```
                self.IN1.off()
                self.posicao= 0
                break
            else:

                for x in range(1): # 6 é igual a 1 mm movimento retilinio
                    for step in Dir:
                        for i in range(len(self.pins)):
                            self.pins[i].value(step[i])
                            sleep(0.001)


    def posi(self):
        return self.posicao

    def get_to(self,posicao):

        dist=posicao-self.posicao
        if dist<0 :
            mexe="-"

        else :
            mexe="+"
        self.move(mexe,abs(dist))
```

```
            for i in range(len(self.pins)):
                self.pins[i].value(step[i])
                sleep(0.001)

    if direcao=="-":
        self.posicao-=1

    else:
        self.posicao+=1

self.IN4.off()
self.IN3.off()
self.IN2.off()
self.IN1.off()

def home(self):
    """
    home:---------------------------------
    coloca o motor a mexer até encontrar o stopper
    atenção motor a rodar na direção dos ponteiros dos relogios para mudar direção alterar Dir=Stepmotor.clockways para Dir=Stepmotor.anticlockways
    """


    Dir=Stepmotor.clockways

    while True:
        buton = self.stopper.value()
        if buton==False:
            self.IN4.off()
            self.IN3.off()
            self.IN2.off()
```

```
def calib(self,direcao):
    """
    calib:---------------------------------
    Para saber qual o numero de steps que devemos meter para o motor mexer 1 mm
    função feita para facilitar quando temos varias rodas dentadas e queremos que no final a rotação seja de 1 mm
    """
    if direcao == "C":
        Dir=self.clockways
    else :
        Dir=self.anticlockways

    for y in range(1):
        for x in range(self.n_step): # 6 é igual a 1 mm movimento retilinio
            for step in Dir:
                for i in range(len(self.pins)):
                    self.pins[i].value(step[i])
                    sleep(0.001)
    self.IN4.off()
    self.IN3.off()
    self.IN2.off()
    self.IN1.off()
    altera = int(input("Por favor colocar distancia percurrida em mm:\n"))
    numero_de_steps=round(self.n_step/altera)
    print("n_step para colocar na inicializaçao é =",numero_de_steps)
```

Stepmotor no stopper

```
from machine import Pin
from time import sleep

class Stepmotor_s:

    clockways = [[1,1,0,0],[0,1,1,0],[0,0,1,1],[1,0,0,1]]
    anticlockways= [[1,0,0,1],[0,0,1,1],[0,1,1,0],[1,1,0,0]]



    def __init__(self, I1, I2, I3, I4,n_stepy):
        """
        init:---------------------------------
        I1,I2,I3,I4: PINS PLACA DE CONTROLO DO MOTOR
        n_stepy: Numero de Stepps para o step motor 512 é uma rotação 360º
        stopper: botão de atuação que indica a posição inicial
        """
        self.IN1 = Pin(I1,Pin.OUT)
        self.IN2 = Pin(I2,Pin.OUT)
        self.IN3 = Pin(I3,Pin.OUT)
        self.IN4 = Pin(I4,Pin.OUT)
        self.pins = [self.IN1, self.IN2, self.IN3, self.IN4]
        self.n_step=n_stepy
        self.posicao= 0
```

```
def move(self, direcao,dist):

    """
    move:--------------------------------
    direção: lado para o qual o motor ira girar
    dist: distancia em mm que queremos o motor girar

    """


    if direcao == "-":
        Dir=Stepmotor_s.clockways
    else :
        Dir=Stepmotor_s.anticlockways

    for y in range(dist):


        for x in range(self.n_step): # 6 é igual a 1 mm movimento retilinio
            for step in Dir:

                for i in range(len(self.pins)):
                    self.pins[i].value(step[i])
                    sleep(0.001)


        if direcao=="-":
            self.posicao-=1

        else:
            self.posicao+=1
```

```
        self.IN4.off()
        self.IN3.off()
        self.IN2.off()
        self.IN1.off()


    def alt(self,distancia):
        self.posicao=distancia

    def posi(self):
        return self.posicao

    def get_to(self,posicao):


        dist=posicao-self.posicao
        if dist<0 :
            mexe="-"

        else :
            mexe="+"
        self.move(mexe,abs(dist))
```

```
    def calib(self,direcao):
        """
        calib:---------------------------------
        Para saber qual o numero de steps que devemos meter para o motor mexer 1 mm
        função feita para facilitar quando temos varias rodas dentadas e queremos que no final a rotação seja de 1 mm
        """
        if direcao == "-":
            Dir=self.clockways
        else :
            Dir=self.anticlockways

        for y in range(1):
            for x in range(self.n_step): # 6 é igual a 1 mm movimento retilinio
                for step in Dir:
                    for i in range(len(self.pins)):
                        self.pins[i].value(step[i])
                        sleep(0.001)
        self.IN4.off()
        self.IN3.off()
        self.IN2.off()
        self.IN1.off()
        altera = int(input("Por favor colocar distancia percurrida em mm:\n"))
        numero_de_steps=round(self.n_step/altera)
        print("n_step para colocar na inicializaçao é =",numero_de_steps)
```

Stepnema

```
from machine import Pin
from time import sleep,sleep_us

class Stepnema:



    def __init__(self,pin_stepp,pin_direc,pin_enable,steppy,pin_stopper):
        """
        init:---------------------------------

        stepp: pin para o motor
        direc: pin para controlar a direção
        enable: pin para deixar atuar o motor
        steppy: Numero de Stepps para o step motor nema, para uma rotação 360º utilizando uma placa DRV8825 temos de olhar para a configuração MS1-MS3
        ms1     ms2     ms3         steps para R 360º
        0       0       0           200
        0       0       1           400
        0       1       0           800
        0       1       1           1600
        1       0       0           3200
        1       0       1           6400
        stopper: pin botão de atuação que indica a posição inicial
        """
        self.stepp = Pin(pin_stepp,Pin.OUT)
        self.direc = Pin(pin_direc,Pin.OUT)
        self.enable = Pin(pin_enable,Pin.OUT)
        self.stopper = Pin(pin_stopper,Pin.IN)
        self.steppy=steppy
        self.posicao= 0
```

```
    self.enable.on()


def move(self, direcao,dist):

    """
    move:---------------------------------
    direção: lado para o qual o motor ira girar
    dist: distancia em mm que queremos o motor girar

    """


    if direcao == "+":
        self.direc.on()
    else :
        self.direc.off()

    self.enable.off()


    for x in range(dist):
        buton = self.stopper.value()
        if buton==False and self.direc.value()==0 :
            self.enable.on()
            break
        if self.direc.value()==False:
            self.posicao-=1

        else:

           self.posicao+=1


       for y in range(self.steppy):# 400 é igual a 1 mm movimento retilinio
           self.stepp.on()
           sleep_us(500)
           self.stepp.off()
           sleep_us(500)

    self.enable.on()

def get_to(self,posicao):

    dist=posicao-self.posicao
    if dist<0 :
        mexe="-"

    else :
        mexe="+"
    self.move(mexe,abs(dist))



def home(self):
    """
    home:---------------------------------
    coloca o motor a mexer até encontrar o stopper
    atenção motor a rodar na direção dos ponteiros dos relogios para mudar direção alterar Dir=Stepmotor.clockways para Dir=Stepmotor.anticlockways
    """
```

```
    self.enable.off()
    self.direc.off()

    while True:
        buton = self.stopper.value()
        if buton==False:

            self.enable.on()
            self.posicao=0
            break
        else:
            for y in range(50):# 400 é igual a 1 mm movimento retilinio
                self.stepp.on()
                sleep_us(500)
                self.stepp.off()
                sleep_us(500)


def posi(self):


    return self.posicao

def calib(self,direcao):
    """
    calib:---------------------------------
    Para saber qual o numero de steps que devemos meter para o motor mexer 1 mm
    função feita para facilitar quando temos varias rodas dentadas e queremos que no final a rotação seja de 1 mm
    """
    if direcao == "+":
```

```
        self.direc.on()
    else :
        self.direc.off()
    self.enable.off()

    for y in range(self.steppy):
        self.stepp.on()
        sleep_us(500)
        self.stepp.off()
        sleep_us(500)

    self.enable.on()


    altera = int(input("Por favor colocar distancia percurrida em mm:\n"))
    numero_de_steps=round(self.n_step/altera)
    print("n_step para colocar na inicializaçao é =",numero_de_steps)
```

Uart serial

```
from machine import UART
from time import sleep


class Uart_1:
    # Configure UART communication


    def __init__(self, numero, T_X, R_X,Baudrate):

        self.uart = UART(numero, tx=T_X, rx=R_X, baudrate=Baudrate)




    def send(self,message):
        self.uart.write(message.encode())

    def receive(self):
        while self.uart.any()==0:
            sleep(1)
        mensagem=self.uart.readline().decode().strip()

        print(mensagem)
        return mensagem.split()
```