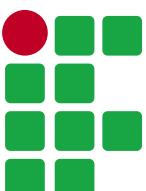


Trabalho de Conclusão de Curso

Amanda Jorge Mendes, Kauã Ortiz Silveira

RISO: ROBÔ INTELIGENTE SEPARADOR E ORGANIZADOR

Rio Grande(RS)
2021



INSTITUTO FEDERAL
Rio Grande do Sul
Campus Rio Grande

Técnico em Automação Industrial
Instituto Federal de Educação, Ciência e Tecnologia
do Rio Grande do Sul
Campus Rio Grande

RISO: Robô Inteligente Separador e Organizador

Amanda Jorge Mendes, Kauã Ortiz Silveira

Orientador: Carlos Rodrigues Rocha

Coorientador: Betânia Vargas Oliveira

Resumo

Este projeto trata-se da construção de um sistema robótico *pick-and-place* para identificação, classificação e organização de peças através de um sistema de visão computacional. Esse protótipo é composto por uma superfície delimitada sobre a qual as peças devem ser colocadas, uma câmera que captura as imagens que serão processadas com os algoritmos de visão computacional e aprendizado de máquina, motores de passo para movimentação das juntas do robô e uma simulação da cadeia cinemática do manipulador, uma vez que não foi possível construir a estrutura mecânica. O projeto explora a interseção entre aspectos da robótica como operações cinemáticas, geração de trajetória e acionamento de motores, e técnicas de inteligência artificial, como a aplicação de aprendizado de máquina à visão computacional. O sistema implementado pode servir tanto para fins didáticos quanto para pesquisa de técnicas de identificação, classificação e separação de peças.

Palavras Chave: Visão computacional; aprendizado de máquina; inteligência artificial; *pick-and-place*; sistemas robóticos; classificação de objetos; Python.

Abstract

This project proposes the construction of a robotic system aiming object recognition, classification and separation using computer vision. The system's components consist of a delimited surface, where the objects are going to be placed, a camera to capture the images, which are going to be processed by the computer vision and machine learning algorithms, stepper motors, to move the robot's joints, and a simulation of the manipulator's kinematic chain, since it wasn't possible to build its mechanical structure. This project explores the intersection of robotics aspects such as kinematic operations, trajectory generation and motor controlling, and artificial intelligence techniques, such as the use of machine learning in computer vision. The implemented system can be used for didactic purposes as well as for research on methods for object recognition, classification and separation.

Keywords: Computer vision; machine learning; artificial intelligence; pick-and-place; robotic systems; object classification; Python.

1 Introdução

A relação entre ser humano e ferramentas sempre esteve presente na História. Por exemplo, nos primórdios da humanidade, o uso de utensílios estava diretamente relacionado às necessidades de sobrevivência. Gradativamente, conforme novas invenções e descobertas surgiam, as máquinas foram se fazendo cada vez mais presentes na vida dos seres humanos. Com a aplicação sistemática da ciência dentro da indústria, consolidou-se a sociedade industrial, que passou a impulsionar a utilização de máquinas para fins produtivos. Desde a invenção da máquina a vapor, em 1769, essa tendência foi acentuada e a automação dos processos produtivos tomou força (ROMANO; DUTRA, 2002).

Hoje, o modelo de automação rígida aplicado na Revolução Industrial, que preconizava a produção em série de grandes volumes de produtos idênticos, vem sendo substituída por uma automação mais flexível. É nesse sentido que se fazem relevantes os robôs industriais, pois são máquinas capazes de realizar as mais diversas tarefas e de se adaptar às necessidades operacionais dos processos. A utilização extensiva de robôs no contexto industrial transcende o aumento de produtividade e lucro, já que também atinge positivamente a esfera humana, por exemplo, ao poupar os operadores da realização de tarefas perigosas e/ou insalubres (ROMANO; DUTRA, 2002).

Esse vínculo entre operadores humanos e sistemas robóticos pode beneficiar as mais diversas aplicações, entre elas, tarefas de *pick-and-place*. A automação dessas operações elementares da indústria já é uma realidade. No entanto, em cenários mais complexos e dinâmicos, surge a relevância do uso de câmeras e a aplicação de inteligência artificial para alcançar uma automação flexível, capaz de cumprir com as demandas específicas de cada processo e ambiente (PICCININI et al., 2009).

É a partir disso que emerge a idealização do protótipo de um sistema robótico *pick-and-place* para identificação, classificação e manipulação de peças. O desenvolvimento desse protótipo vai ao encontro da demanda por sistemas inteligentes para automação flexível de processos. O projeto trata de um manipulador robótico para organizar peças de acordo com suas classes. Ele é operado a partir de algoritmos de visão computacional responsáveis pela identificação e classificação de objetos. Assim, foram concebidos três módulos principais: o módulo de manipulação, o de imagem e de classificação.

O primeiro é composto por um robô do tipo SCARA, uma mesa onde os objetos se encontrarão inicialmente e contêineres para os quais eles serão sistematicamente movidos. O segundo abrange uma câmera e os algoritmos de processamento de imagem e detecção de objetos, enquanto o último é encarregado da classificação desses objetos através da aplicação de um modelo de aprendizado de máquina. Além de prototipar o equipamento, o projeto também visa ao estudo e teste de diferentes modelos e bibliotecas já existentes, a fim de

avaliar o funcionamento e resultados desses. Portanto, a proposta apresenta uma viés didática, sustentada pelo estudo de ferramentas *open-source* disponíveis nos ramos de visão computacional e aprendizado de máquina, e outra voltada para a prática, que compreende desde a construção das estruturas mecânicas e do desenvolvimento do *software* até a união e comunicação dos módulos constituintes do projeto.

1.1 Justificativa

O projeto RISO (Robô Inteligente Separador e Organizador) surge, primeiramente, da constatação das diversas vantagens decorrentes da cooperação entre seres humanos e robôs no ambiente industrial, tanto para os processos quanto para os trabalhadores envolvidos. Ainda, em um cenário em que a necessidade de sistemas de produção automatizados e dinâmicos imposta pelo mercado é crescente, os robôs industriais passam a ser elementos muito importantes, uma vez que seu desenvolvimento e sua adaptação a sistemas integrados de manufatura são muito flexíveis (ROMANO; DUTRA, 2002).

Assim, a aplicação da visão computacional vai justamente ao encontro da ideia de implementar sistemas robóticos capazes de atender demandas de operações de *pick-and-place* menos engessadas em que sensores convencionais são insuficientes (PICCININI et al., 2009). Por fim, o estudo de ferramentas de visão computacional e aprendizado de máquina é sustentado não somente pelo aproveitamento desse conhecimento na prototipação do sistema robótico proposto, mas também pela crescente utilização desses recursos em operações cada vez mais complexas e diversas, o que torna relevante seu aprendizado (IBM, 2021; SAS, 2021a).

1.2 Objetivos

O objetivo do projeto é construir um sistema robótico para separação e organização sistemática de peças através do uso de um modelo de aprendizado de máquina aplicado à visão computacional para detecção e classificação de objetos.

Para alcançar o propósito mencionado, é necessário efetivar os seguintes objetivos específicos:

- Estudar visão computacional e aprendizado de máquina;
- Projetar um robô manipulador do tipo SCARA;
- Implementar um sistema de detecção de peças;
- Desenvolver um modelo de aprendizado de máquina para classificação de imagens;
- Construir um equipamento composto por manipulador, mesa e contêineres;

- Implementar o software dos sistemas de detecção e classificação;
- Validar o protótipo quanto ao funcionamento do manipulador;
- Estabelecer comunicação entre os algoritmos de detecção e classificação de peças e a unidade manipuladora;
- Avaliar diferentes técnicas de classificação e separação de peças.
- Validar o sistema em um estudo de caso para detecção, classificação e organização de porcas e parafusos.

2 Fundamentação Teórica

O projeto de um robô inteligente separador e organizador de peças envolve conhecimentos de diversos segmentos da área de Automação Industrial. Sua implementação pressupõe o entendimento e a aplicação de vários conceitos e técnicas que vão desde a área de robótica industrial até o ramo da inteligência artificial. Assim, o desenvolvimento do RISO passa por explorar a interseção entre esses campos de conhecimento a fim de alcançar o objetivo de implementar um sistema robótico inteligente.

2.1 Robótica Industrial

Robótica industrial é um conceito definido pela ISO (International Organization for Standardization) 10218, como "uma máquina manipuladora com vários graus de liberdade controlada automaticamente, reprogramável, multifuncional, que pode ter base fixa ou móvel para utilização em aplicações de automação industrial" ([ROMANO; DUTRA, 2002](#)). O uso de robôs industriais visa à automatização da produção que, por sua vez, gera diversos benefícios como ([ROMANO; DUTRA, 2002](#)):

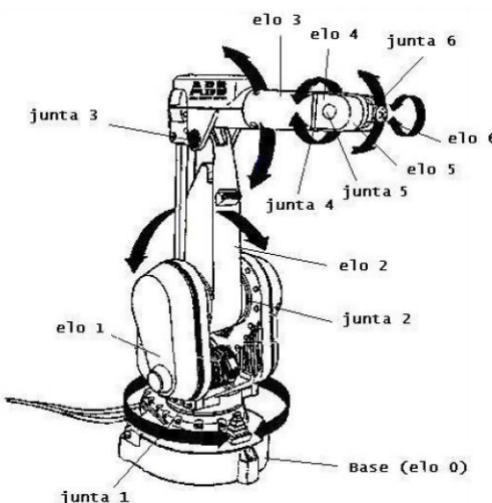
- Redução dos custos dos produtos, devido a aspectos como aumento de produtividade e diminuição do número de pessoas envolvidas;
- Melhora nas condições de trabalho dos operadores, uma vez que atividades perigosas, insalubres e repetitivas são destinadas aos robôs;
- Melhora na qualidade do produto, já que os parâmetros da produção podem ser melhor controlados.

Assim, é notória a importância de manipuladores mecânicos na realização de diversas tarefas no contexto industrial, dentre elas, operações de *pick-and-place*.

2.1.1 Manipuladores mecânicos

O conceito de manipulador refere-se à estrutura mecânica do robô, que consiste em elementos rígidos (elos) conectados entre si por articulações (juntas). O primeiro elemento do robô é chamado de base e o último é denominado extremidade terminal, é neste que será acoplado o efetuador que, por sua vez, será o ponto de interação do robô com o meio e serve, por exemplo, para a manipulação de objetos. O número de graus de liberdade de um manipulador é o número de variáveis de posição que precisam ser especificadas para se definir a localização de todas as partes do mecanismo, ou seja, é o número de juntas do robô (ROMANO; DUTRA, 2002). A Figura 1 mostra um exemplo de manipulador.

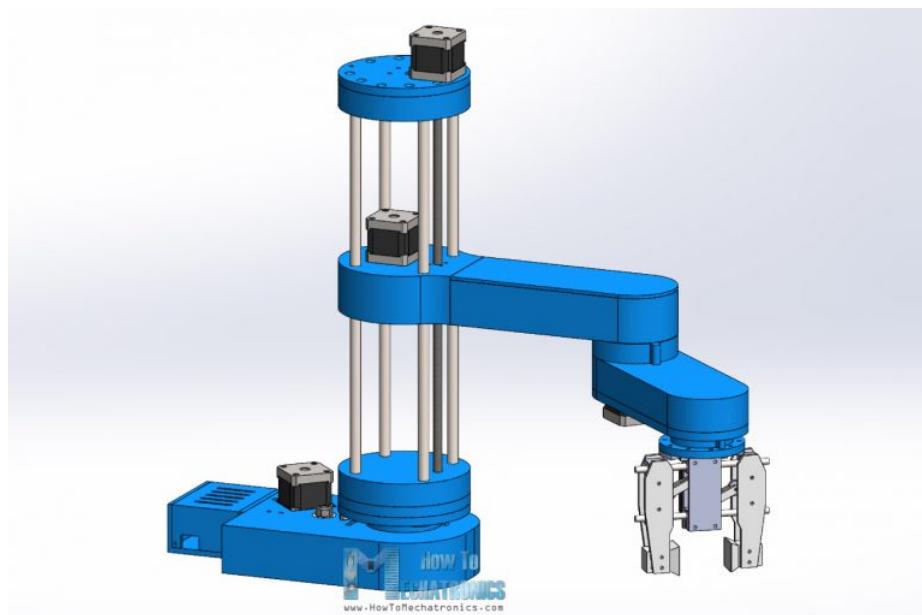
Figura 1 – Manipulador de seis graus de liberdade (ROMANO; DUTRA, 2002).



Os manipuladores mecânicos podem assumir diversas configurações de elos e juntas. Assim, existem vários tipos de robôs industriais como o de coordenadas cartesianas, o de coordenadas cilíndricas, o de coordenadas esféricas, o SCARA, o antropomórfico e o paralelo (ROMANO; DUTRA, 2002).

O robô do tipo SCARA, como o da Figura 2, é um manipulador que combina juntas rotativas com uma junta prismática (movimento de translação). As juntas rotativas permitem a movimentação em um plano. Perpendicular a este plano, fica a junta prismática, que possibilita a movimentação do efetuador em um terceiro eixo. Esse tipo de robô é muito utilizado em tarefas de montagem de componentes pequenos, como placas de circuitos eletrônicos (ROMANO; DUTRA, 2002).

Figura 2 – Modelo de robô SCARA ([NEDELKOVSKI, c2019](#)).



2.1.2 Operações de *pick-and-place*

Operações de *pick-and-place* são utilizadas em processos elementares de manufatura. Elas consistem, fundamentalmente, na manipulação de objetos e, assim, são usadas em tarefas como montagens de componentes e retirada e colocação de peças em esteiras. Nessas operações, um manipulador robótico deve mover um objeto de uma posição inicial até uma posição final. Esses estados são definidos em termos das coordenadas e da orientação de um dos pontos do robô. Apesar de o movimento intermediário entre o estado inicial e o objetivo não ser definido, deve-se garantir uma trajetória suave e sem colisões ([ANGELES, 2007](#)).

Para mover um manipulador de um ponto a outro, são necessárias operações matemáticas que permitam descrever e controlar o movimento do robô. Isso inclui não só estabelecer as relações entre juntas e efetuador, mas também traçar um percurso para que o manipulador alcance seu objetivo. Assim, faz-se necessário implementar conceitos de cinemática direta e inversa, bem como realizar um planejamento geométrico e temporal da tarefa.

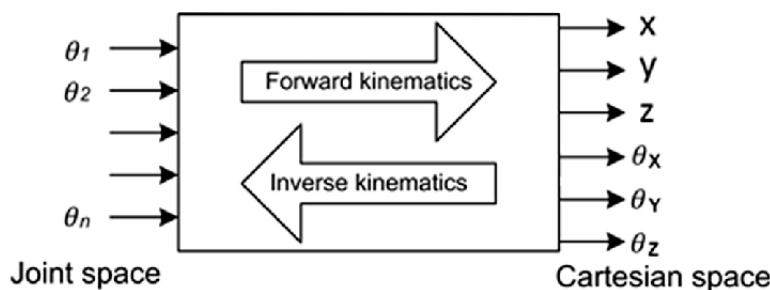
2.1.3 Cinemática direta e inversa

Cinemática é um ramo da Mecânica responsável pela descrição do movimento de corpos, sejam eles isolados ou em grupos, a partir da determinação da posição, velocidade e/ou aceleração de partes do sistema. Esse estudo, portanto, é fundamental para o desenvolvimento da robótica, destacando-se o detalhamento da relação entre o deslocamento das juntas e o do efetuador. Há modelos cinemáticos que se preocupam exclusivamente com a posição dos elementos, sendo categorizados como cinemática direta (*forward kinematics*) ou inversa (*inverse kinematics*) ([ROCHA; TONETTO; DIAS, 2011](#)).

O movimento de manipuladores para realização de tarefas ocorre no espaço cartesiano (*cartesian space*), que inclui vetores de posição e orientação. Já os atuadores, responsáveis por deslocar o robô, operam no espaço das juntas (*joint space*), definido pelos ângulos dessas (KUCUK; BINGUL, 2006). Assim, problemas de cinemática direta e inversa consistem na conversão da posição e orientação do efetuador do robô do espaço das juntas para o espaço cartesiano ou vice-versa (KUCUK; BINGUL, 2006).

Como mostra a Figura 3, a cinemática direta consiste em métodos analíticos para calcular a disposição do efetuador dada a configuração das juntas que compõem o sistema. Já a cinemática inversa descreve o arranjo das juntas dada a posição e a orientação do efetuador (ROCHA; TONETTO; DIAS, 2011).

Figura 3 – Relação entre cinemática direta e inversa (SHERBINY; EL-HOSSEINI; HAIKAL, 2017)



Modelos de cinemática direta são muito importantes para a simulação de cadeias cinemáticas. Um método amplamente utilizado para esse tipo de operação utiliza a notação de Denavit-Hartenberg. Esse método descreve a posição de um elo em relação ao seu antecessor considerando que esses elementos estão conectados entre si por uma junta de um grau de liberdade (rotativa ou prismática). Primeiramente, deve-se definir os quatro parâmetros de Denavit-Hartenberg para cada um dos elos do robô (ROCHA; TONETTO; DIAS, 2011). A Tabela 1 mostra os parâmetros de Denavit-Hartenberg para o robô SCARA da Figura 4.

Figura 4 – Manipulador SCARA.

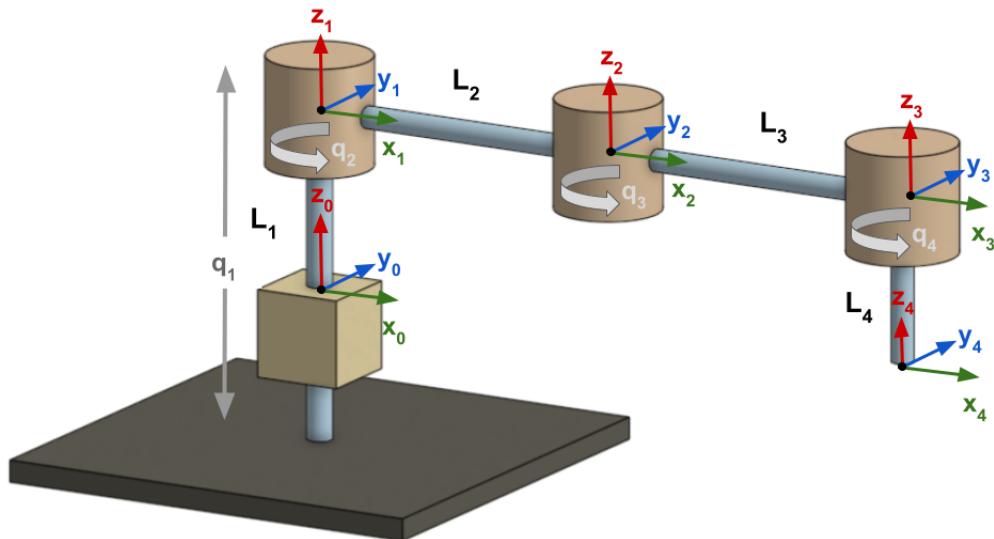


Tabela 1 – Identificação dos parâmetros de Denavit-Hartenberg para o SCARA da Figura 2.

Elo	a_i	α_i	d_i	θ_i
1	0	0	q_1	0
2	l_2	0	0	q_2
3	l_3	0	0	q_3
4	0	0	$-l_4$	q_4

Posteriormente, deve-se determinar, a partir dos parâmetros de Denavit-Hartenberg, as matrizes de transformação homogênea das juntas em relação às suas precedentes, como na Equação 1 (ROCHA; TONETTO; DIAS, 2011).

$${}_{i-1}A_i = \begin{bmatrix} \cos \theta_i & -\cos \alpha_i \sin \theta_i & \sin \alpha_i \sin \theta_i & a_i \cos \theta_i \\ \sin \theta_i & \cos \alpha_i \cos \theta_i & -\sin \alpha_i \cos \theta_i & a_i \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

Por fim, a matriz de transformação que determina a posição e a orientação do efetuador em relação à origem do sistema é definida através da multiplicação matricial sucessiva das matrizes de transformação de cada uma das n juntas como mostra a Equação 2 (ROCHA; TONETTO; DIAS, 2011).

$${}^0A_n = {}^0A_1 A_2 \dots {}^{n-1}A_n \quad (2)$$

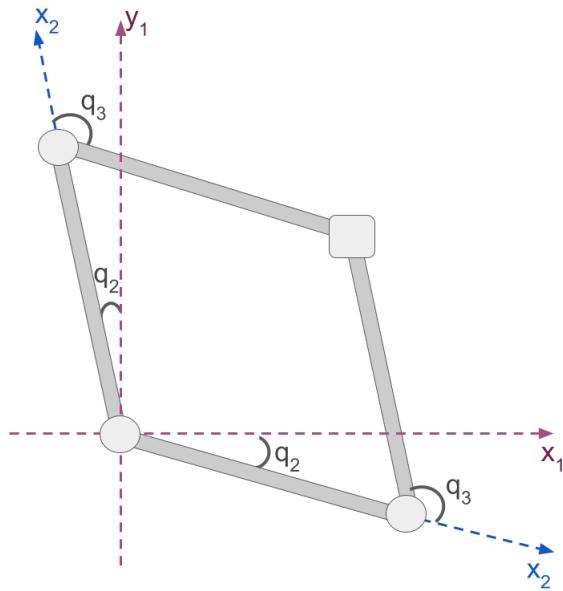
A cinemática inversa também é muito importante para o estudo do movimento de manipuladores. Diferentemente da direta, a cinemática inversa tem a peculiaridade de poder

apresentar múltiplas ou até infinitas soluções (ROCHA; TONETTO; DIAS, 2011), já que é resolvida através de equações não-lineares (SANTOS; WATANABE; CARRARA, 2006).

Para solucionar, de forma analítica, um problema de cinemática inversa, é possível adotar duas abordagens: a geométrica ou a algébrica. O método geométrico consiste em partitionar o espaço do robô em vários problemas de geometria plana. Ele é utilizado para manipuladores mais simples que se movimentam em um único plano. Quando se trata de um robô com mais juntas e cujo movimento ocorre em espaço tridimensional, o método algébrico é o mais adequado (KUCUK; BINGUL, 2006).

No caso do manipulador SCARA da Figura 4, o valor da junta prismática (q_1) é a única variável que influencia a coordenada z do efetuador. Já a movimentação nos eixos x e y depende de duas variáveis: q_2 e q_3 . Assim, pode-se ter até duas maneiras de alcançar uma mesma posição, como mostra a Figura 5. A função da última junta (q_4) se restringe a determinar a orientação do efetuador e, portanto, pode ser desconsiderada quando o único interesse é chegar em determinado ponto, independente da orientação.

Figura 5 – Soluções de cinemática inversa para o manipulador SCARA da Figura 4



Para alcançar um ponto em (x, y, z) , o valor de q_1 pode ser definido através da Equação 3. Assim, tem-se definida a movimentação do manipulador no eixo z .

$$q_1 = z + l_4 \quad (3)$$

A partir de então, o movimento do robô que resta ser definido é planar, nos eixos x e y apenas. Portanto, é possível resolver a cinemática inversa do robô SCARA da Figura

4 através do método geométrico. Ao aplicar conceitos de trigonometria ao modelo desse manipulador é possível definir os possíveis valores dos ângulos q_2 e q_3 , como mostram as Equações 4 e 5 (SANTOS; WATANABE; CARRARA, 2006). A Equação 4 deve ser calculada para cada um dos dois valores encontrados na Equação 5.

$$q_2 = \arctan\left(\frac{(l_1 + l_2 \cos q_3)y - l_2 \sin q_3 x}{(l_1 + l_2 \cos q_3)x + l_2 \sin q_3 y}\right) \quad (4)$$

$$q_3 = \pm \arccos\left(\frac{x^2 + y^2 - l_2^2 - l_1^2}{2 \times l_1 \times l_2}\right) \quad (5)$$

Dessa forma, é possível calcular as duas possíveis configuração das juntas para que o efetuador chegue nas coordenadas desejadas. Esse procedimento de cinemática inversa, responsável por converter as posições do robô do espaço cartesiano para o espaço das juntas, é essencial para realizar o planejamento de trajetórias do robô.

2.1.4 Planejamento de tarefa

Além do modelo cinemático, também são procedimentos importantes para o deslocamento de um braço robótico, o planejamento geométrico do caminho a ser percorrido e a geração de uma trajetória no decorrer do tempo. Essas tarefas são fundamentais para que o robô possa se deslocar de sua posição inicial até o local desejado obedecendo restrições espaciais, mecânicas (KAVRAKI; LAVALLE, 2008) e temporais (LAGES, 2005).

O planejamento do caminho de um robô consiste em definir, geometricamente, uma rota para que seu efetuador, partindo de uma posição inicial, chegue efetivamente em uma posição final desejada. Esse caminho deve levar em consideração possíveis obstáculos no espaço de trabalho bem como restrições mecânicas do robô (KAVRAKI; LAVALLE, 2008).

A geração de trajetória adiciona ao planejamento do caminho uma nova variável: tempo. Assim, a descrição do movimento do robô passa a ter uma dependência temporal e, portanto, deve ser estabelecido um perfil de velocidade e aceleração das juntas do manipulador. A trajetória pode ser gerada tanto no espaço das juntas quanto no espaço cartesiano. Em ambos os métodos, os procedimentos adotados, na maioria das vezes, consistem em gerar uma função parametrizada com o tempo que contemple os pontos iniciais e finais do movimento (LAGES, 2005).

No espaço das juntas, uma das formas de gerar uma trajetória consiste em sua representação através de um polinômio. Este pode ser um simples polinômio de primeiro grau, em que ocorre uma variação linear da posição em função do tempo. No entanto, esse método implica que a velocidade seja constante e, portanto, provoca aceleração e desaceleração

abruptas e descontínuas nos instantes iniciais e finais do movimento. Para contornar esse problema, é possível empregar polinômios de graus mais elevados ([LAGES, 2005](#)).

Um dos métodos utilizados consiste em gerar uma trajetória polinomial de quinto grau. Esse possibilita satisfazer as restrições de posição, velocidade e aceleração impostas às trajetórias e gerar um movimento contínuo e suave ([LAGES, 2005](#)), como mostram as Figuras [6](#), [7](#) e [8](#).

Figura 6 – Posição de junta em trajetória polinomial de quinto grau.

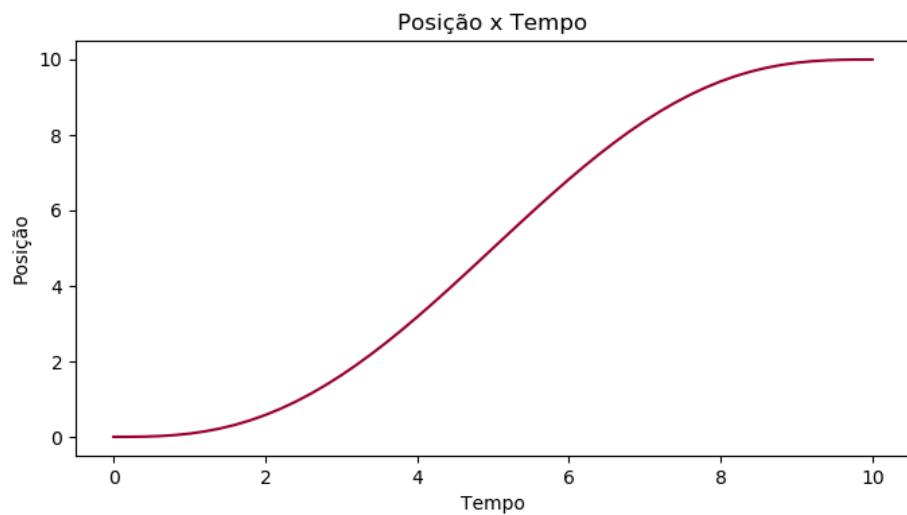


Figura 7 – Velocidade de junta em trajetória polinomial de quinto grau.

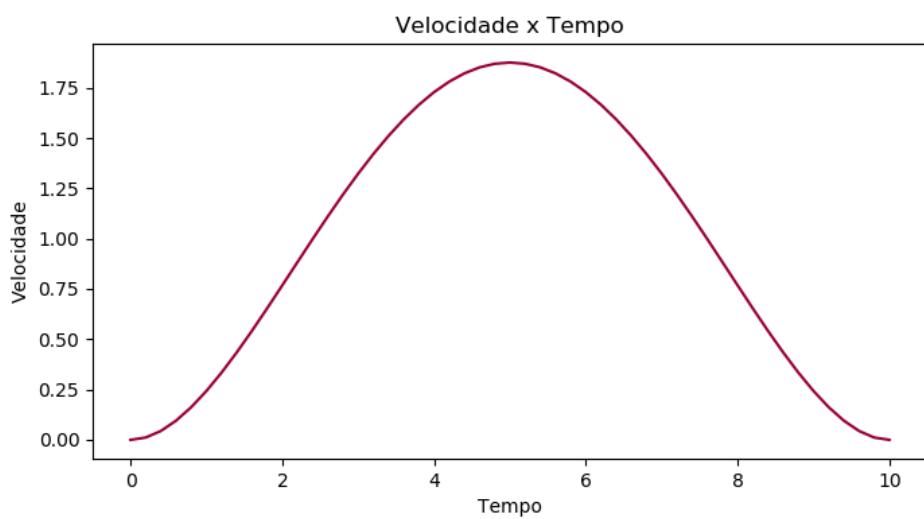
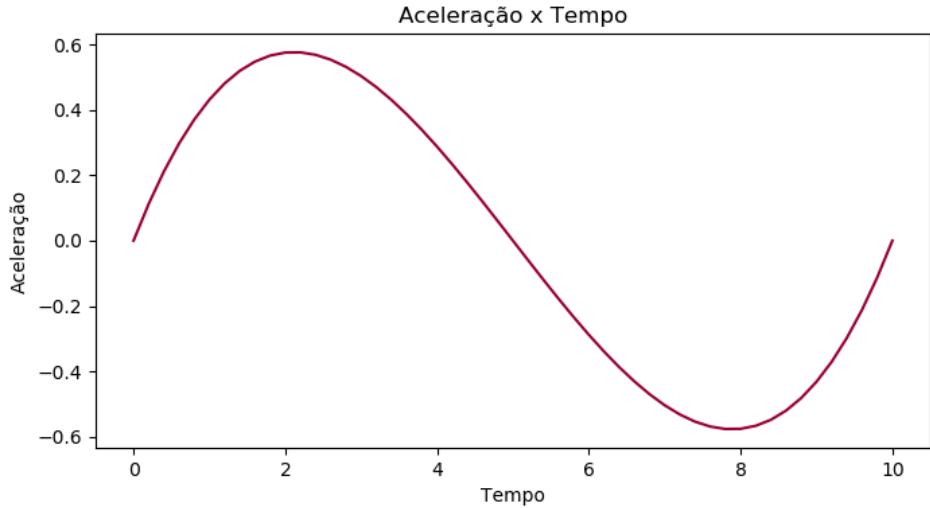


Figura 8 – Aceleração de junta em trajetória polinomial de quinto grau.



Uma trajetória polinomial de quinto grau pode ser descrita por n polinômios como o da Equação 6.

$$q(t) = a_5 t^5 + a_4 t^4 + a_3 t^3 + a_2 t^2 + a_1 t + a_0 \quad (6)$$

onde,

- n : número de juntas
- $q(t)$: posição da junta em um instante t
- a_i : coeficientes do polinômio, $i = 0, \dots, 5$

As expressões que descrevem a velocidade (Equação 7) e a aceleração (Equação 8) da junta ao longo do movimento podem ser obtidas através da derivação da Equação 6 em relação ao tempo (LAGES, 2005).

$$\dot{q}(t) = 5a_5 t^4 + 4a_4 t^3 + 3a_3 t^2 + 2a_2 t + a_1 \quad (7)$$

$$\ddot{q}(t) = 20a_5 t^3 + 12a_4 t^2 + 6a_3 t + 2a_2 \quad (8)$$

Para a geração da trajetória é necessário aplicar as condições iniciais e finais desejadas às expressões de posição, velocidade e aceleração (Equações 6, 7, 8, respectivamente).

Assim, é possível, ao resolver o sistema de seis equações (9 - 14), descobrir os coeficientes a_i do polinômio para que a junta se desloque como desejado (LAGES, 2005).

$$q(t_{in}) = a_5 t_{in}^5 + a_4 t_{in}^4 + a_3 t_{in}^3 + a_2 t_{in}^2 + a_1 t_{in} + a_0 \quad (9)$$

$$\dot{q}(t_{in}) = 5a_5 t_{in}^4 + 4a_4 t_{in}^3 + 3a_3 t_{in}^2 + 2a_2 t_{in} + a_1 \quad (10)$$

$$\ddot{q}(t_{in}) = 20a_5 t_{in}^3 + 12a_4 t_{in}^2 + 6a_3 t_{in} + 2a_2 \quad (11)$$

$$q(t_f) = a_5 t_f^5 + a_4 t_f^4 + a_3 t_f^3 + a_2 t_f^2 + a_1 t_f + a_0 \quad (12)$$

$$\dot{q}(t_f) = 5a_5 t_f^4 + 4a_4 t_f^3 + 3a_3 t_f^2 + 2a_2 t_f + a_1 \quad (13)$$

$$\ddot{q}(t_f) = 20a_5 t_f^3 + 12a_4 t_f^2 + 6a_3 t_f + 2a_2 \quad (14)$$

onde,

- t_{in} : instante inicial do movimento
- t_f : instante final do movimento

Uma das formas de resolver o sistema de equações acima consiste em representá-lo de forma matricial como mostra a Equação 15.

onde,

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} 1 & t_{in} & t_{in}^2 & t_{in}^3 & t_{in}^4 & t_{in}^5 \\ 0 & 1 & 2t_{in} & 3t_{in}^2 & 4t_{in}^3 & 5t_{in}^4 \\ 0 & 0 & 2 & 6t_{in} & 12t_{in}^2 & 20t_{in}^3 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix}^{-1} \times \begin{bmatrix} q(t_{in}) \\ \dot{q}(t_{in}) \\ \ddot{q}(t_{in}) \\ q(t_f) \\ \dot{q}(t_f) \\ \ddot{q}(t_f) \end{bmatrix} \quad (15)$$

Em muitas operações, como as de *pick-and-place*, pode-se considerar que o movimento começa no instante zero, ou seja, $t_{in} = 0$. Ainda, é possível definir que nesse momento

(t_{in}) , bem como no instante final do deslocamento (t_f), as velocidades e acelerações das juntas são nulas. Nesse caso, ao substituir os valores de t_{in} , $\dot{q}(t_{in})$, $\ddot{q}(t_{in})$, $\dot{q}(t_f)$ e $\ddot{q}(t_f)$ nas matrizes da Equação 15, tem-se a Equação 16.

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix}^{-1} \times \begin{bmatrix} q(t_{in}) \\ 0.0 \\ 0.0 \\ q(t_f) \\ 0.0 \\ 0.0 \end{bmatrix} \quad (16)$$

Assim, para obter os coeficientes a_i do polinômio de quinto grau, basta substituir os elementos t_f , $q(t_f)$ e $q(t_{in})$ e realizar a operação matricial indicada na Equação 16. Após a obtenção dos coeficientes a_i , as Equações 6, 7 e 8 podem ser utilizadas para determinar, respectivamente, a posição, a velocidade e a aceleração da junta em qualquer instante do movimento.

Na aplicação do projeto, para planejar as tarefas e executar as operações de cinemática a fim de alcançar um objeto em uma superfície, é necessário saber as coordenadas nas quais ele se encontra. Da mesma forma, para levá-lo até o lugar ao qual pertence, é necessário saber que tipo de objeto ele é. Para isso, as peças precisam ser devidamente detectadas, localizadas e classificadas, tarefas que podem ser desempenhadas através da implementação de algoritmos de inteligência artificial.

2.2 Inteligência Artificial

Não há um consenso na comunidade científica sobre uma definição precisa e única de inteligência artificial. No geral, é um campo de estudo que combina ciência da computação e conjuntos robustos de dados para resolução de problemas (IBM, 2020). Inteligência artificial é uma área muito ampla com as mais variadas aplicações. Ela compreende diversos ramos, como aprendizado de máquina, processamento de linguagem natural e visão computacional (WANG, 2017).

Visão computacional é uma área da inteligência artificial que tem como objetivo conferir às máquinas a habilidade de "enxergar", ou seja, processar conteúdos visuais e extrair informações relevantes desses (IBM, 2021). Outro ramo extremamente relevante dentro da inteligência artificial é o aprendizado de máquina. Esse campo compreende algoritmos capazes de adquirir conhecimento automaticamente, ou seja, sistemas com a habilidade de aprender (SAS, 2021a). Um algoritmo de aprendizado de máquina consiste em treinar um modelo, através de uma série de operações matemáticas, a fazer previsões ou classificações

baseadas em dados de entrada (IBM, 2020). O aprendizado pode se dar de várias maneiras, dentre elas, as mais comuns são o aprendizado supervisionado e o não-supervisionado (SAS, 2021a).

No aprendizado não-supervisionado, os algoritmos devem analisar um conjunto de exemplos cujas saídas não são definidas. Nesse método, os objetivos são explorar, encontrar padrões e definir a estrutura do conjunto de dados. Já no aprendizado supervisionado, os algoritmos são treinados usando exemplos devidamente rotulados, ou seja, que contêm dados de entrada e de saída já determinados (SAS, 2021a). Esses algoritmos, depois de treinados, são usados para prever saídas para entradas desconhecidas, segundo o aprendizado obtido dos exemplos. Portanto, os algoritmos de aprendizado supervisionado são empregados em problemas de regressão e classificação.

Métodos para problemas de regressão consistem em prever um valor numérico que se encontra em um espectro contínuo de valores, já os métodos para classificação devem prever uma classe dentre possibilidades limitadas (MONARD; BARANAUSKAS, 2003). Por exemplo, se é necessário, a partir de determinados parâmetros, prever o preço de produtos em reais, trata-se de um problema de regressão. Por outro lado, se o objetivo for usar os parâmetros fornecidos para determinar se um produto é caro ou barato, ou seja, para enquadrá-lo em uma classe, o problema deve ser abordado como uma classificação.

Vários algoritmos de aprendizado de máquina já foram idealizados e desenvolvidos para diversas aplicações. Para resolver problemas de classificação, ou seja, para aplicações em que se deseja usar dados de exemplo para treinar um modelo capaz de prever classes de dados futuros, pode-se citar algoritmos como o k-vizinhos mais próximos (KNN, do inglês *K-Nearest Neighbors*), a máquina de vetores de suporte (SVM, do inglês *Support Vector Machine*), a floresta aleatória (*random forest*) (SAS, 2021a), a rede neural convolucional (CNN, do inglês *Convolutional Neural Network*) e o perceptron multicamadas (MLP, do inglês *Multi-Layer Perceptron*) (DESAI; SHAH, 2020).

2.3 Aprendizado de máquina aplicado à visão computacional em sistemas *pick-and-place*

Operações de *pick-and-place*, em vários setores de indústria, precisam ser automatizadas. Na maioria desses sistemas, sensores indicam ao manipulador a localização da peça a ser alcançada. As coordenadas em que esses objetos devem ser reposicionados costumam ser predeterminadas. No entanto, esse tipo de método pode se tornar ineficiente em cenários que não sejam bem estruturados e restritos, que apresentem maior grau de versatilidade. É nesse tipo de cenário que o sensoriamento por câmeras combinado a algoritmos de aprendizado de máquina se torna atrativo (PICCININI et al., 2009).

O aprendizado de máquina aplicado à visão computacional habilita o computador a aprender a entender as imagens digitais processadas e, a partir do conhecimento adquirido, extrair informações relevantes delas (IBM, 2021). Esses sistemas podem ser utilizados para desempenhar tarefas como detecção e classificação de objetos para operações de *pick-and-place*.

A primeira etapa de um sistema de visão computacional é o pré-processamento das imagens digitais. Esse processo consiste em adequar a imagem às condições desejadas, como definição de tamanho e formato, por exemplo. Além disso, as imagens a serem tratadas podem ter ruídos provenientes da própria aquisição. Não necessariamente essas características indesejáveis são decorrentes de interferência no sinal da captura da imagem, os ruídos também podem ser consequência de condições do ambiente, como iluminação e posição, por exemplo. Assim, a etapa de pré-processamento de imagens inclui a aplicação de filtros capazes de conter a propagação desses ruídos para as próximas etapas do algoritmo de visão computacional. Depois de serem pré-processadas, as imagens podem ser utilizadas para diversas tarefas, desde segmentação e reconhecimento de padrões, até processos mais complexos como rastreamento de objetos (MARENGONI; STRINGHINI, 2009).

Para implementar as funcionalidades necessárias em um sistema de visão computacional, desde técnicas para pré-processamento de imagens até tarefas de alto nível associadas à visão humana, é possível utilizar a biblioteca OpenCv (*Open Source Computer Vision*) (MARENGONI; STRINGHINI, 2009). Ela é uma biblioteca *open source* para visão computacional e aprendizado de máquina que tem interfaces C++, Python, Java e MATLAB e suporta Windows, Linux, Android e Mac OS (OPENCV, c2021).

A detecção de objetos em uma imagem passa por um processo chamado segmentação, que consiste em particionar uma imagem em regiões. Essas podem ser estabelecidas de acordo com suas características, como por exemplo, semelhança de cor. É esse procedimento que permite detectar objetos em uma imagem. Um dos métodos utilizados para segmentação é a detecção de bordas, que podem ser vistas como variações, normalmente abruptas, no nível de intensidade dos píxeis em uma imagem. Ao encontrar esse tipo de comportamento, é possível distinguir contornos e, portanto, objetos (MARENGONI; STRINGHINI, 2009).

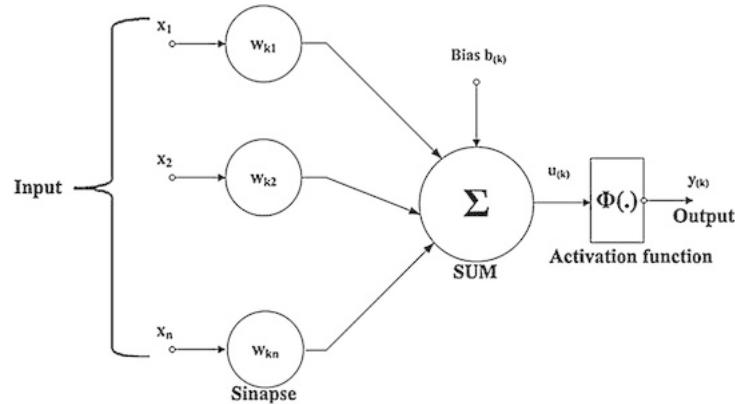
Além da segmentação, outro processo importante no contexto de visão computacional é o reconhecimento de padrões. Esse procedimento permite classificar os objetos detectados pela etapa de segmentação. Para isso, é necessário que o computador tenha um conhecimento prévio sobre os padrões que devem ser reconhecidos. Esse tipo de tarefa pode ser alcançada através do estabelecimento de um conjunto de regras explícitas ou através da utilização de técnicas de aprendizado de máquina. Nessa última abordagem, o conhecimento necessário para reconhecer os padrões desejados em uma imagem pode ser aprendido a partir

de um conjunto de amostras de treinamento ([MARENCONI; STRINGHINI, 2009](#)). Alguns algoritmos de aprendizado de máquina que podem ser usados para classificação de objetos são as florestas aleatórias ([AKAR; GÜNGÖR, 2012](#)) e as redes neurais artificiais como a rede neural convolucional e o perceptron multicamadas ([SILVA et al., 2020](#)).

Árvores de decisão são estruturas de tomada de decisão com nós que partem de uma raiz (o nó de origem) e se ramificam até chegarem às folhas (nós de resultado) de acordo com condições de decisão estabelecidas em cada um dos nós. As amostras de treinamento de uma árvore de decisão são categorizadas considerando determinados parâmetros e, assim, são criadas regras para cada um dos nós, tornando o modelo capaz de representar e reproduzir as características analisadas nas amostras ([LAURETTO, 2010](#)). Floresta aleatória (*Random forest*) é um classificador *ensemble*, ou seja, é um algoritmo que cria múltiplos classificadores que, posteriormente, processam individualmente os novos dados de entrada. Ainda, as *random forests* têm as características de um método de *bagging*, em que os votos de todos os classificadores influenciam na decisão final de forma igualitária, ou seja, a previsão será resultado da maioria simples dos votos. O que diferencia a *random forest* de algoritmos *ensemble* do tipo *bagging* padrão é que as características processadas por cada um dos classificadores de uma *random forest* são selecionadas de forma aleatória ([AKAR; GÜNGÖR, 2012](#)). Uma das maneiras de implementar uma *random forest* é através da Scikit-learn — biblioteca Python desenvolvida para aplicações de aprendizado de máquina. Essa ferramenta reúne funções para análise preditiva de dados e possibilita a interação com outras bibliotecas como Numpy, Scipy e Matplotlib ([SCIKIT-LEARN, c2020a](#)).

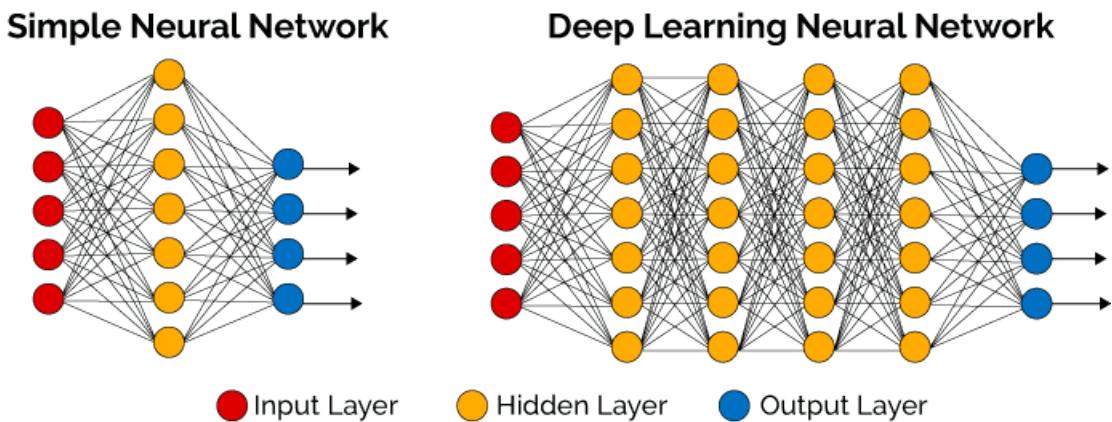
Redes neurais artificiais são algoritmos de aprendizado de máquina baseados na estrutura do cérebro humano. Elas são capazes não só de reconhecer padrões em dados brutos, agrupá-los e classificá-los, mas também, de aprender e se aperfeiçoar constantemente ao longo do processo ([SAS, 2021b](#)). Essas redes são compostas por nós, também chamados de neurônios, que são unidades computacionais que podem ser representadas matematicamente como mostra a Figura 9. O papel de um neurônio é processar os dados de entrada (*input*), aos quais são atribuídos pesos (w), e, de acordo com o resultado de uma função de ativação (*activation function*), propagar a informação processada para a próxima camada de neurônios.

Figura 9 – Modelo matemático de um neurônio (DATA SCIENCE ACADEMY, c2021).



Uma rede neural básica inclui uma camada de entrada (*input layer*), outra de saída (*output layer*) e, entre elas, uma camada oculta (*hidden layer*). Quando uma rede neural artificial é composta por múltiplas camadas ocultas, ela se torna mais profunda e poderosa, capaz de desempenhar tarefas complexas e, assim, passa a configurar uma arquitetura de aprendizado profundo (*deep learning*) (SAS, 2021b). A Figura 10 mostra modelos de redes neurais.

Figura 10 – Redes neurais simples e profunda (DATA SCIENCE ACADEMY, c2021).

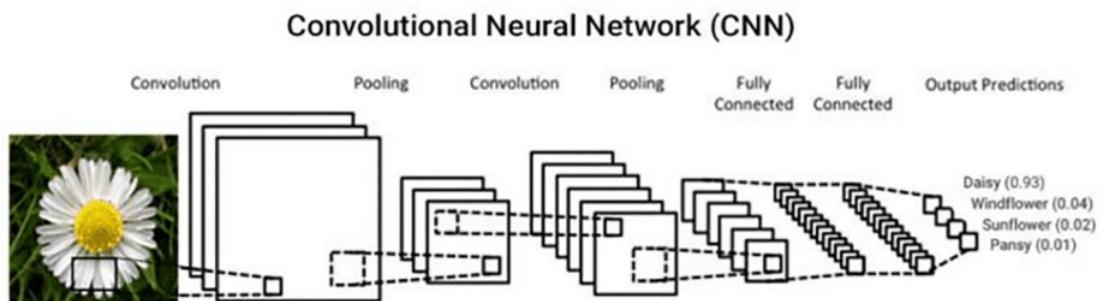


Um perceptron multicamadas, é um modelo de rede neural do tipo *feedforward* que usa a técnica de *backpropagation* para aprender (DESAI; SHAH, 2020). Em redes neurais *feedforward* cada neurônio de uma camada é conectado a todos os neurônios da camada seguinte (SAS, 2021b). A técnica de *backpropagation* consiste em, após realizar uma iteração de treinamento, percorrer a rede no sentido contrário, atualizando os pesos de modo a reduzir o erro (DESAI; SHAH, 2020). Esse algoritmo, assim como a *random forest* pode ser implementado com a biblioteca Scikit-learn na linguagem de programação Python (SCIKIT-LEARN, c2020a).

As CNNs (*Convolutional Neural Networks* ou Redes Neurais Convolucionais), são um tipo de rede neural artificial que também usam a técnica de *backpropagation* e que são

comumente utilizadas para examinar, identificar ou classificar imagens. Seu funcionamento consiste em desempenhar uma série de simplificações conforme a imagem passa pelas múltiplas camadas que compõem a rede, como mostra a Figura 11. Convoluçãoções medem o grau de sobreposição de duas funções. No caso dos filtros convolucionais usados nas CNNs, a convolução é feita entre os pesos do filtro e os píxeis da imagem (PODLOZHNYUK, 2007). Outra camada que faz parte da estrutura de uma rede neural convolucional é a de *pooling*. Essa camada é importante para garantir velocidade de processamento da rede, uma vez que a camada de *pooling* reduz o tamanho das amostras extraídas pela camada convolucional e, portanto, diminui a quantidade de parâmetros. Outra camada presente nas CNNs são as *fully-connected*, elas conectam todos os neurônios da camada anterior a todos os neurônios da camada seguinte (DESAI; SHAH, 2020). Uma CNN pode ser implementada através da API Keras na linguagem de programação Python. Essa é uma API de alto nível que opera sobre bibliotecas de baixo nível como Tensorflow. Ela conta com diversos modelos de camadas que podem ser montadas pelo usuário de modo a construir uma CNN personalizada (KERAS, c2021).

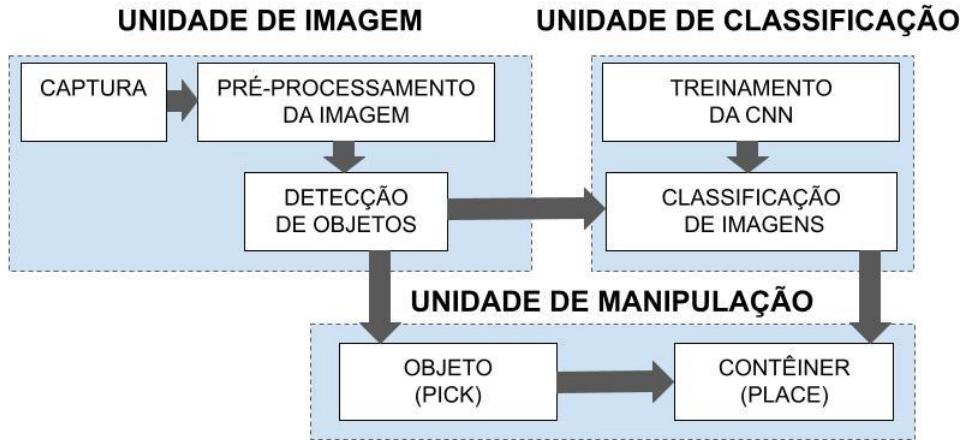
Figura 11 – Estrutura de uma CNN usada para classificar espécies de flores (DESAI; SHAH, 2020).



3 Projeto

O projeto de um Robô Inteligente Separador e Organizador consiste no desenvolvimento de um sistema robótico *pick-and-place* para detecção, classificação e organização de peças dispostas aleatoriamente em uma superfície delimitada. O espaço de trabalho onde os objetos serão detectados pode ser fixo, como uma mesa, ou pode ser uma área de uma esteira em uma linha de produção. Os objetos, detectados e classificados por um sistema de visão computacional, são apanhados e levados até o contêiner correspondente à sua classe. Como mostra a Figura 12, esse projeto é concebido em três módulos: a unidade de imagem, a unidade de classificação e a unidade de manipulação. O estudo de caso para validação do sistema consiste em detectar, classificar e organizar porcas e parafusos, ainda que o sistema possa ser facilmente generalizado para outros tipos de objetos.

Figura 12 – Diagrama das unidades do sistema



3.1 Unidades do projeto

O projeto, em sua concepção e em seu desenvolvimento, visa a produzir um sistema *open source* que possa contribuir para a comunidade científica da área de robótica. Dessa forma, a elaboração do projeto do sistema RISO considera que ele deve ser passível não só de ser reproduzido, mas também de ser expandido, tanto com otimizações quanto com agregação de novas funcionalidades. Além disso, o projeto deve ser versátil e, assim, permitir que seja adaptado para outros cenários e aplicações.

A unidade de manipulação do projeto consiste na modelagem, construção e controle do sistema robótico. Este deve ter dimensões e peso que permitam que ele seja facilmente montado e transportado, a fim de que possa ser usado em diferentes cenários. A automação desse robô é feita com o uso de um microcontrolador.

As tarefas a serem realizadas pelo manipulador são de *pick-and-place*. Assim, durante o deslocamento do robô pelo espaço de trabalho, ele deve ter uma elevação suficiente para não colidir com os objetos dispostos na mesa. No momento em que deve capturar um objeto ou deixá-lo dentro de um contêiner, o manipulador deve ser abaixado. Esse movimento configura um típico deslocamento de Schöenflies que pode ser realizado por um manipulador de quatro graus de liberdade ([SICILIANO et al., 2009](#)).

A unidade de imagem é parte do sistema de visão computacional. Ela engloba a etapa de pré-processamento das imagens e as técnicas para detecção e localização das peças dispostas na superfície de interesse. Essa unidade conta com uma câmera de resolução adequada além de um sistema de iluminação que permita uma boa aquisição de imagens. Essa câmera fica fixa em relação ao espaço de trabalho, acima do manipulador robótico. Além disso, é importante que a transferência de imagem da câmera para o processador do *software* de visão computacional seja rápida e que os elementos de *hardware* sejam compatíveis. Por

fim, as partes constituintes dessa unidade, assim como das demais, devem ser acessíveis, bem documentadas e versáteis.

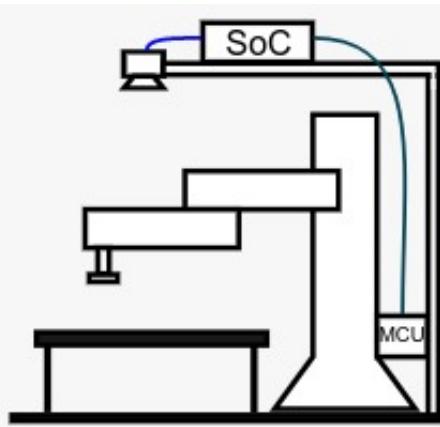
A unidade de classificação, quando agregada à unidade de imagem, completa o sistema de visão computacional. Ela diz respeito ao uso de um modelo de classificação, treinado através de um algoritmo de aprendizado de máquina, para inferir a classe dos objetos identificados. Para essa aplicação, que consiste na classificação de imagens, vários algoritmos de aprendizado de máquina podem ser utilizados. Para o projeto, buscou-se técnicas que pudessem ter uma implementação acessível, mas que, ao mesmo tempo, apresentassem resultados satisfatórios e flexibilidade para customização.

3.2 Constituição do sistema

Além do microcontrolador, utilizado para a automação do sistema robótico, um SoC (*System on a Chip*) é usado como plataforma para as operações de visão computacional. Ele é o responsável pelo acionamento da câmera, pelo processamento das imagens e pelas operações pertinentes à identificação, localização e classificação das peças.

Assim, foi idealizado um sistema que atenda às necessidades do projeto, já pontuadas anteriormente. Ele está ilustrado na Figura 13 e é composto por um manipulador de quatro graus de liberdade, câmera fixa com sistema de iluminação, um microcontrolador para controle do robô e um SoC como plataforma central. Com essa configuração, o manipulador fica no campo de visão da câmera. Por isso, antes da captura da imagem, o robô é movido para uma posição que não interfira na detecção dos objetos.

Figura 13 – Representação simplificada do sistema



Por simplicidade de projeto, fabricação e programação, foi escolhida uma cadeia cinemática do tipo PRRR, ou seja, um robô com uma junta prismática na base seguida por três juntas rotativas. Esse manipulador, esquematizado na Figura 4, é similar à configuração

clássica de um SCARA ([SICILIANO et al., 2009](#)). As juntas são controladas por motores de passo que são acionados através de *drivers*.

Para controlar o robô, foi escolhido um microcontrolador ESP32 32-bit dual core ([Espressif Systems, 2021](#)). Já como plataforma principal do projeto, ao avaliar a integração necessária entre *hardware* e *software* das unidades de imagem e classificação, optou-se por uma Raspberry Pi ([RPF, 2020](#)). Essa placa tem uma interface serial dedicada para módulos de câmera da *Raspberry Pi Foundation* ([RPF, 2021a](#)).

Para o desenvolvimento do *software* do projeto, optou-se por Python como linguagem de programação principal ([PSF, 2021a](#)). Python, além de ser uma linguagem suportada pelos modelos de Raspberry Pi ([RASPBERRY PI FOUNDATION, c2021](#)), tem uma grande variedade de APIs científicas e documentação ampla e acessível ([LINGE; LANGTANGEN, 2020](#)). Inclusive, Python tem suporte para a biblioteca OpenCV que, como já mencionado, conta com funcionalidades para desenvolvimento de *software* de visão computacional e, portanto, foi escolhida como ferramenta para o sistema do projeto. Para a plataforma ESP32, foi escolhida a linguagem Micropython a fim de padronizar o desenvolvimento de *software*, uma vez que a linguagem é uma adaptação de Python para microcontroladores ([NORRIS, 2016](#)).

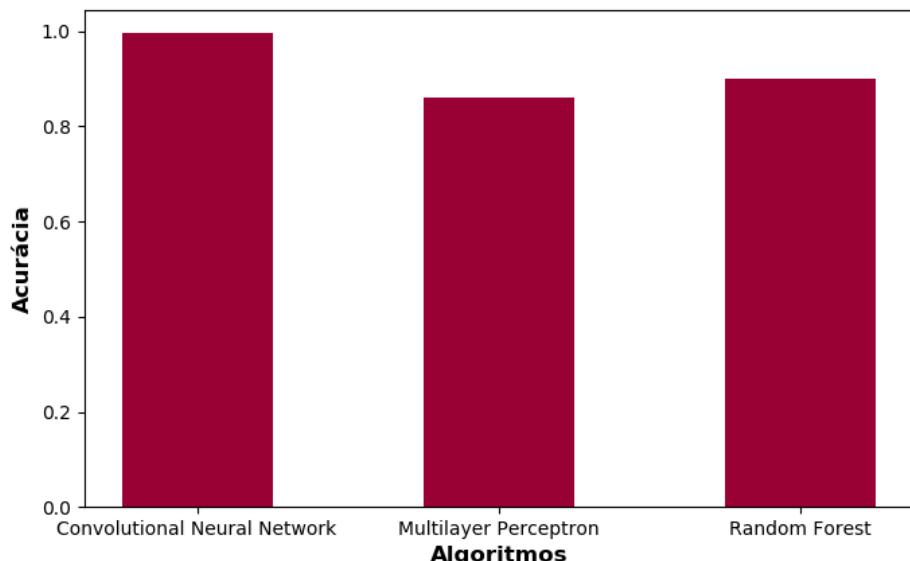
Para decidir a técnica de aprendizado de máquina mais adequada para os fins e necessidades do projeto, os algoritmos *random forest* ([AKAR; GÜNGÖR, 2012](#)), MLP e CNN ([DESAI; SHAH, 2020](#)) foram selecionados para estudo. Todos eles podem ser implementados em Python.

A biblioteca Scikit-learn possui funções que facilitam o tratamento dos dados de treinamento e, além disso, conta com vários modelos de aprendizado de máquina ([SCIKIT-LEARN, c2020a](#)), dentre eles, o *random forest* ([SCIKIT-LEARN, c2020b](#)) e o MLP ([SCIKIT-LEARN, c2020b](#)). Assim, foi a ferramenta usada para o estudo desses dois algoritmos. Para a implementação da CNN, utilizou-se a API Keras a partir da biblioteca Tensorflow ([KERAS, c2021](#)).

Os modelos escolhidos foram estudados e implementados a fim de fazer uma avaliação de seus desempenhos. Para isso, foram utilizadas 180 imagens RGB de 480x480 píxeis. Metade das amostras desse conjunto eram imagens de porcas e a outra metade, de parafusos. Para a captura dessas imagens de treinamento, os objetos foram posicionados de múltiplas formas, para que os modelos aprendessem a identificá-los de qualquer ângulo. Todos os algoritmos testados têm um fator de aleatoriedade, seja na construção das árvores de decisão, no caso das *random forests*, seja na definição inicial dos pesos, no caso das redes neurais (MLP e CNN). Além disso, o mesmo algoritmo pode ser implementado de diversas maneiras, visto que existem vários parâmetros que podem ser manipulados de acordo com a aplicação.

Portanto, os resultados obtidos, apresentados na Figura 14, podem variar entre diferentes execuções do treinamento.

Figura 14 – Comparaçāo de modelos para classificaçāo de porcas e parafusos

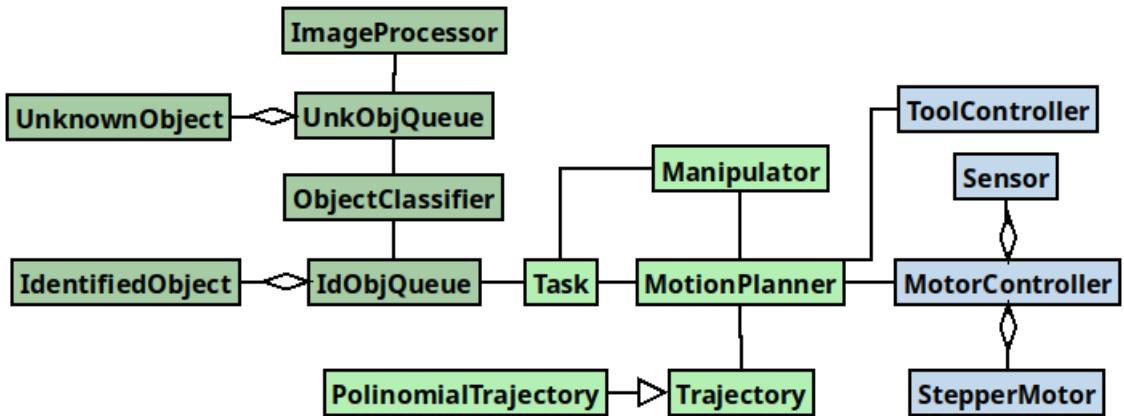


Assim, a implementação de uma CNN com a API Keras é uma alternativa que, além de possibilitar um desenvolvimento acessível e versátil, tem um bom desempenho em termos de assertividade nas predições. Portanto, é o algoritmo implementado para treinar o modelo de classificação de peças do projeto.

3.3 Funcionamento do sistema

O *software* do projeto é desenvolvido utilizando programação orientada a objetos de forma a modularizar a sua implementação. As classes, apesar de independentes, são integradas na rotina principal do sistema e trabalham em conjunto para alcançar os objetivos do projeto. O diagrama de classes da Figura 15 mostra os módulos idealizados e as interações entre eles. As classes representadas com fundo verde são processadas pela Raspberry Pi, já as com fundo azul, pelo ESP32.

Figura 15 – Diagrama de classes do sistema



A classe *ImageProcessor* é responsável pela captura de imagem e identificação das peças e sua localização. Assim, essa classe produz instâncias de *UnknownObject* correspondentes a cada peça detectada. Esses objetos são adicionados à fila *UnknownObjectQueue*, que é consumida pela classe *ObjectClassifier*. Cada instância consumida pelo *ObjectClassifier* passa pelo processo de classificação e assim, desde que tenha sido classificado com sucesso, uma instância de *IdentifiedObject* é produzida e adicionada à fila *IdObjQueue*.

Uma instância de *Task* é criada e sua função é consumir os elementos de *IdObjQueue*. Uma instância da classe *Manipulator*, que representa a cadeia cinemática do robô, é utilizada pela *Task* para resolver a cinemática inversa a partir das coordenadas operacionais obtidas pelo *ImageProcessor* e, assim, determinar as coordenadas correspondentes no espaço das juntas. Essas informações dos valores das juntas alimentam uma instância de *MotionPlanner* que planeja o movimento de cada junta desde sua posição corrente até a desejada.

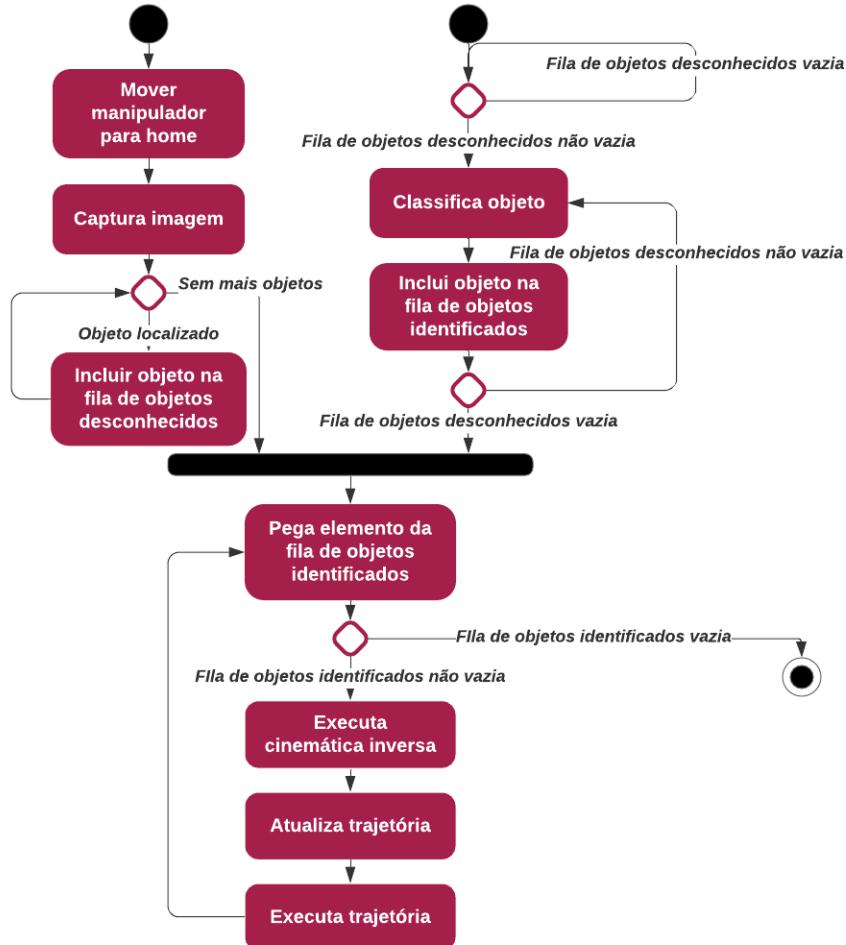
No planejamento feito pela instância de *MotionPlanner*, são geradas trajetórias para cada uma das juntas. Um objeto *MotorController* consome as trajetórias e controla o envio dos passos para os *drivers* a fim de rotacionar os motores de passo de forma sincronizada e com velocidades apropriadas. O projeto inicial do sistema é conceituado como uma malha aberta, uma vez que constitui uma implementação mais simples e menos custosa. Dessa forma, a instância de *MotorController* também é responsável por estimar, baseado na quantidade de pulsos emitida para cada *driver*, o deslocamento angular de cada junta. Essas estimativas são enviadas para o objeto *Manipulator* que, assim, pode atualizar, em *software*, a cadeia cinemática.

Ainda, por não haver um controle contínuo do posicionamento das juntas ao longo do tempo, sensores de fim de curso são posicionados e administrados pela instância de *MotorController*, não só a fim de fazer o posicionamento inicial do robô, mas também de garantir que os motores não excedam seus limites mecânicos. Da mesma forma que as trajetórias

geradas comandam a operação de *MotorController*, quando pertinente, elas também podem ser enviadas para uma instância de *ToolController* a fim de manipular o efetuador do robô.

O funcionamento do sistema é iniciado ao comando do usuário e opera segundo a rotina ilustrada na Figura 16.

Figura 16 – Diagrama de atividades



Primeiramente, o robô deve ser movido até sua configuração inicial, que é reconhecida por sensores de fim de curso estrategicamente posicionados, os quais são lidos pela instância de *MotorController*, processada pelo ESP32. Nessa etapa, o manipulador deve sair do campo de visão da câmera para que não interfira na detecção dos objetos. Feito isso, uma imagem é capturada e a classe *ImageProcessor* passa a detectar as peças existentes e adicioná-las a fila de *UnkObjQueue*. Enquanto isso, essa fila é consumida pelo *ObjectClassifier* que passa a rotular os objetos localizados e detectados no espaço de trabalho e adicioná-los a fila *IdObjQueue*.

Quando todos os elementos da *UnkObjQueue* já foram consumidos, começa a etapa da manipulação, que consiste em organizar os objetos em contêineres de acordo com sua

classe. Para cada peça em *IdObjQueue*, a cinemática inversa é calculada pela instância de *Manipulator* e, assim, de acordo com as coordenadas das juntas necessárias para alcançar a peça, são geradas as trajetórias. Assim, o robô se move até a peça, a apanha e, depois, se desloca até o contêiner correspondente à sua classe — cujas coordenadas são predefinidas — no qual o objeto é largado. Esse processo se repete até que todas as peças tenham sido organizadas, ou seja, até que a fila *IdObjQueue* esteja vazia. A execução dessa rotina se dá de forma cíclica e pode ser interrompida ao comando do usuário ou de acordo com um número máximo de varreduras pré-definido.

4 Desenvolvimento do Protótipo

O desenvolvimento do protótipo de um robô inteligente separador e organizador de peças se deu de forma modularizada, como proposto na fase de projeto. Os elementos de *software* foram implementados utilizando a ideia de programação orientada a objetos. Essa abordagem facilitou o desenvolvimento do projeto, já que as partes necessárias para o funcionamento do sistema podiam ser implementadas e de forma independente. O código principal do protótipo gerencia a utilização dos módulos bem como as interações entre eles.

4.1 *Software*

Os módulos implementados seguem a ideia proposta no diagrama de classes da Figura 15. A classe *MotorController* foi desenvolvida na linguagem de programação MicroPython, as demais, em Python.

Para a classe *ImageProcessor* foram utilizadas ferramentas da biblioteca OpenCV. As principais funções utilizadas são ilustradas na Figura 17.

Figura 17 – Processamento de imagem com OpenCV

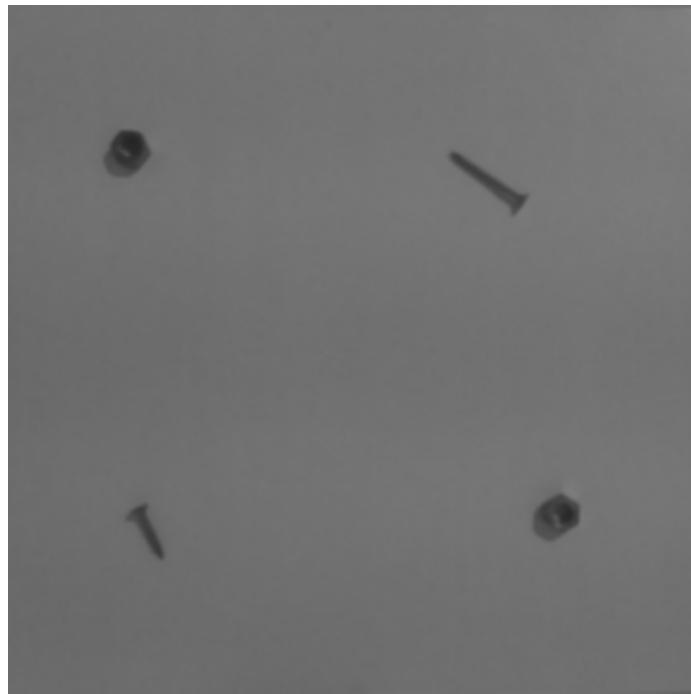
FILTRAÇÃO	Cor	cvtColor()
	Suavização	GaussianBlur()
	Binarização	threshold()
	Erosão	erode()
	Dilatação	dilate()
SEGMENTAÇÃO		findContours()
LOCALIZAÇÃO		boundingRect()

As imagens capturadas, como a da Figura 18, se encontram inicialmente, no espaço de cores RGB. Para prosseguir com o processamento necessário para detecção de objetos, é necessário converter a imagem para a escala de cinza. Essa conversão é feita pela função *cvtColor*. Após isso, a imagem é tratada com um filtro de suavização, *GaussianBlur*, o que resulta em uma imagem como a da Figura 19

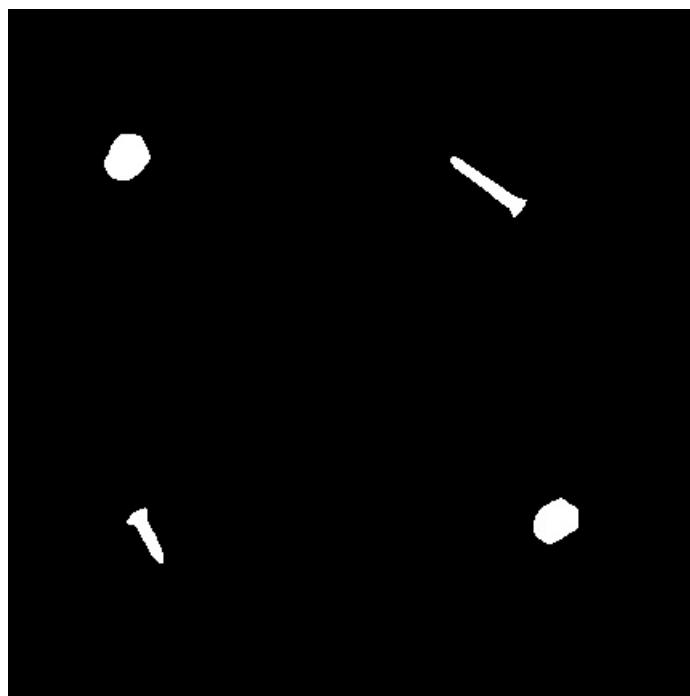
Figura 18 – Imagem capturada



Figura 19 – Imagem suavizada na escala de cinza

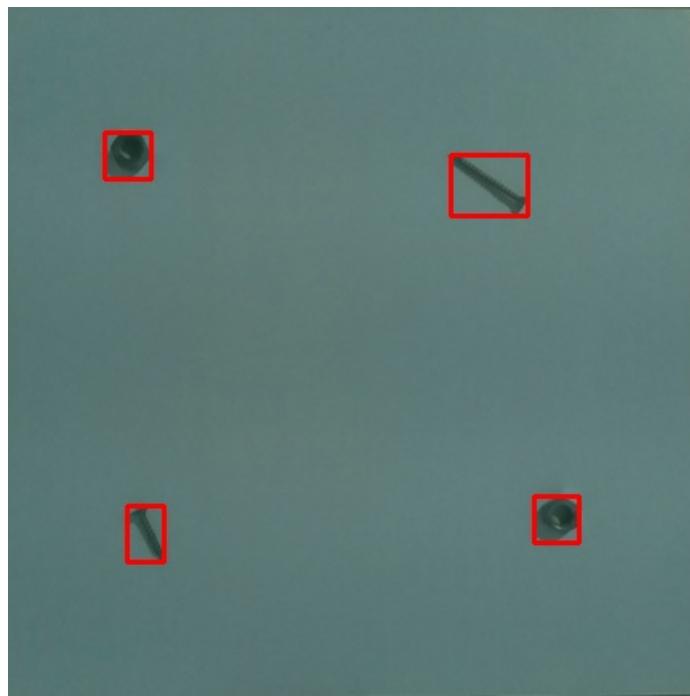


Posteriormente, a função *threshold* destaca os objetos na imagem, diferenciando-os do plano de fundo de forma binária, como mostra a Figura 20.

Figura 20 – Imagem depois da aplicação da função *threshold*

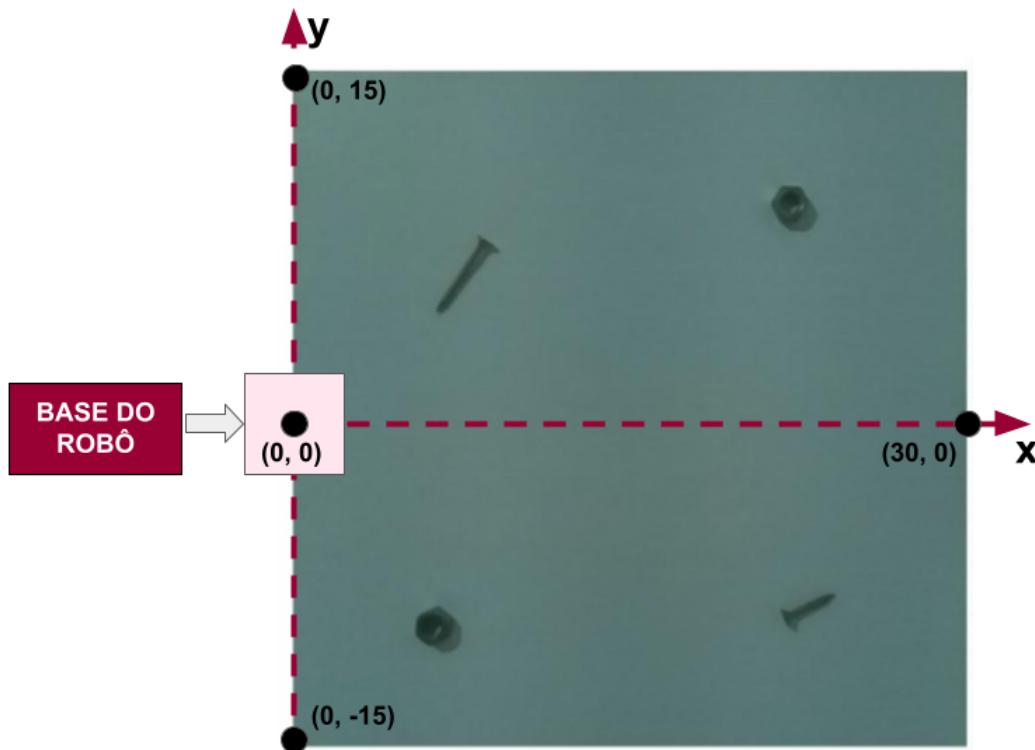
Depois, possíveis ruídos da imagem são removidos com a função *erode* e, após, os objetos são dilatados com a ferramenta *dilate*. Dessa forma, a imagem é adequada para a detecção de contornos. Esse procedimento de segmentação da imagem é feito pela função *findContours*. Ela utiliza o método de detecção de bordas para identificar os contornos dos objetos presentes na imagem binária. Através da função *boundingRect*, é traçado um retângulo em torno de cada objeto identificado, como mostra a Figura 21, e, além disso, são obtidas informações referentes à localização e à dimensão desse retângulo. É possível considerar que as coordenadas do objeto correspondem às do centro do retângulo e, assim, obtém-se a localização, em píxeis, do objeto na imagem.

Figura 21 – Identificação de objetos na imagem



Para converter os valores de x e y em píxeis para o sistema de coordenadas do robô, é necessário analisar o sistema de captura e encontrar a relação entre as unidades de medida para a realização dos cálculos. Para esse procedimento, foi definido um espaço de trabalho com medidas conhecidas (30cm x 30cm) e configurou-se a captura da imagem, bem como a posição da câmera, de modo que a imagem enquadrasse única e exclusivamente a área de interesse. Além disso, foi definida a posição do robô em relação ao espaço de trabalho, como mostra a Figura 22. Considera-se sua base a origem do sistema de coordenadas, a partir do qual é definido o espaço de trabalho.

Figura 22 – Sistema de coordenadas do espaço de trabalho



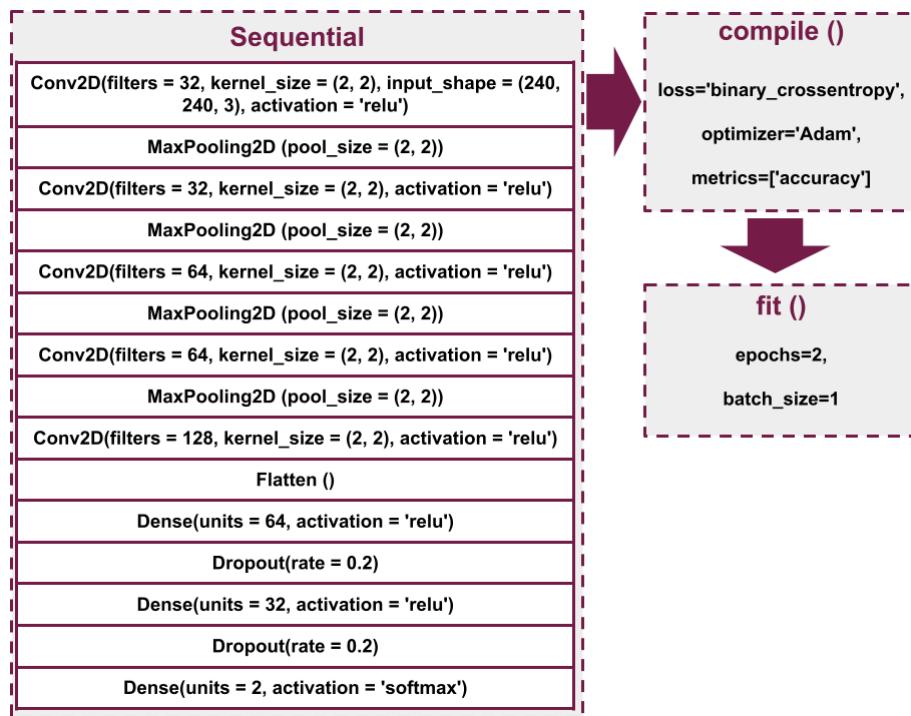
Assim, uma vez que as dimensões, em píxeis, da imagem são definidas e fixas e que sabe-se o tamanho no sistema métrico que corresponde a cada uma dessas dimensões, pode-se definir as coordenadas dos objetos no sistema de coordenadas real do robô. Depois de detectados e localizados, é possível retornar as imagens dos objetos, convertidas novamente para RGB. Para serem classificadas, no entanto, essas imagens precisam ser representadas de forma matricial. Para isso, é utilizada a função *asarray* da biblioteca *NumPy*, ([NUMPY, 2021](#)), que permite que, a partir de então, a imagem, possa ser manipulada e processada como uma matriz que, depois, é reorganizada em um vetor unidimensional passível de ser processado pela rede neural convolucional. Assim, os objetos passam a estar devidamente localizados e suas imagens prontas para serem classificadas pelo *ObjectClassifier*.

Na classe *ObjectClassifier*, foram desenvolvidas funções para carregamento de modelos e para a utilização destes na classificação de imagens. Para o treinamento da rede neural convolucional, foram utilizadas algumas ferramentas auxiliares de outras bibliotecas. Da *OpenCV*, foi utilizada a função *imread* a fim de carregar as imagens de treinamento, previamente capturadas e armazenadas, para serem utilizadas pela rede neural. Além disso, utilizou-se a função *LabelEncoder* da biblioteca *Scikit-learn* com o objetivo de automatizar a codificação e decodificação entre os rótulos em texto de cada imagem e seus correspondentes numéricos, uma vez que a rede neural só opera com números, não pode processar textos.

Os modelo de CNN foi desenvolvido através das camadas pré-implementadas dis-

ponibilizadas pela API Keras. Elas foram estruturadas sobre um modelo *Sequential*, classe dessa API responsável por montar e interligar camadas, definidas pelo usuário, de forma sequencial. A construção da rede neural convolucional consistiu, primeiramente, no estudo do funcionamento dos modelos de camadas oferecidos pela Keras bem como no entendimento das possíveis funções de ativação e dos parâmetros intrínsecos de cada camada. A partir de então, de acordo com os estudos realizados, a CNN foi desenvolvida através de um processo de ajuste de parâmetros e teste com diferentes arquiteturas de camadas e funções de ativação. Isso culminou em um modelo, ilustrado na Figura 23, adequado às necessidades do projeto.

Figura 23 – Implementação da rede neural convolucional (CNN)



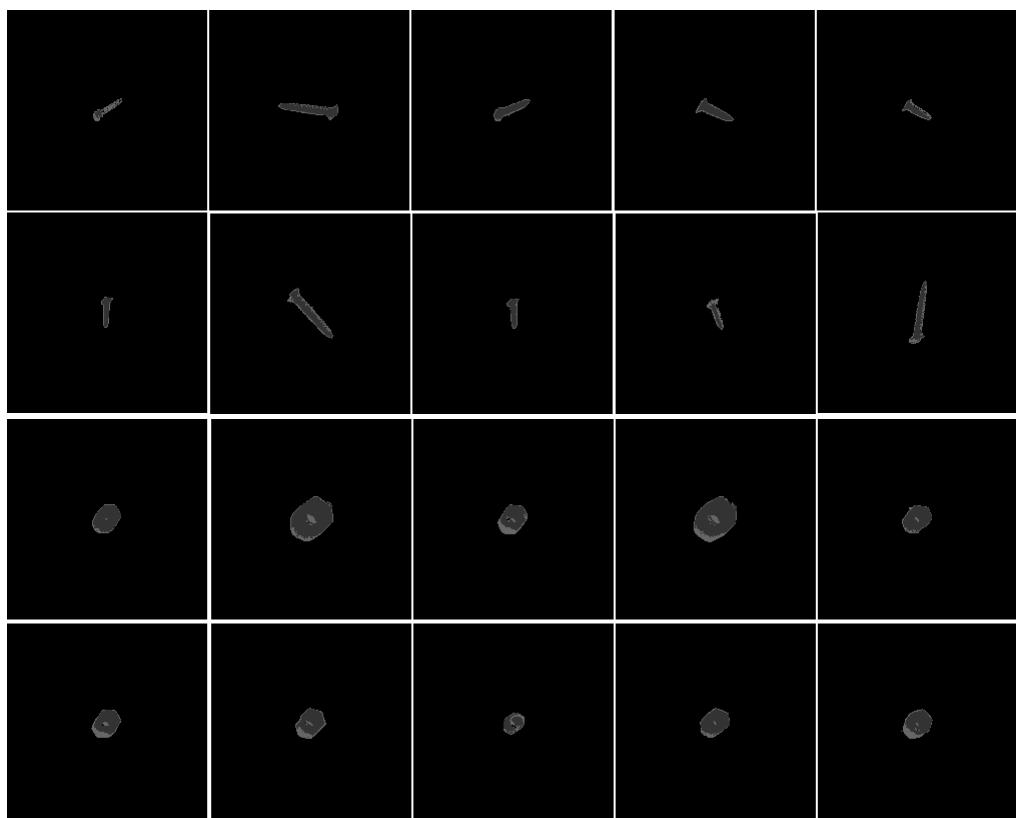
Como mostra a figura acima, a função de ativação da última camada – responsável por gerar a saída da rede neural, por indicar classificação da imagem – é a *softmax*. Essa função, retorna valores entre zero e um e, assim, seu resultado pode ser tratado como uma distribuição de probabilidades, o que possibilita inferir não só a classe da imagem (considerase a classe com a maior probabilidade), mas também as probabilidades associadas a cada uma das classes possíveis (SHARMA; SHARMA, 2017).

A rede neural apresentada na Figura 23, depois de devidamente compilada e treinada, é convertida e salva para o formato *tflite*. Esse tipo de arquivo faz parte de um conjunto de ferramentas do Tensorflow chamado Tensorflow Lite que visa à otimização da execução em dispositivos móveis como microcontroladores e *smartphones* de modelos de aprendizado de máquina criados com Tensorflow (TENSORFLOW, 2021). O modelo *tflite* é, posterior-

mente, carregado no *ObjectClassifier* e utilizado para realizar inferências sobre as imagens e, assim, classificar os objetos.

Para o estudo de caso proposto, foram utilizadas 450 imagens, 225 de porcas e 225 de parafusos, para o treinamento da rede neural convolucional desenvolvida. Originalmente, o conjunto de amostras totalizava 500 imagens. Porém, 50 delas foram separadas aleatoriamente para utilização na validação da rede. Isso foi feito através da função *train_test_split* da biblioteca *Scikit-learn*. As imagens de treinamento, como as da Figura 24, foram capturadas e processadas de modo a ficarem com fundo preto e tamanho igual a 200 x 200 x 3.

Figura 24 – Amostra das imagens utilizadas para treinamento da CNN



A classe *Manipulator* consiste na representação da cadeia cinemática do robô. Para isso, foram criadas uma classe para a representação de juntas e outra para elos. Ao elo é atribuído um tamanho e à junta está associado um atributo de tipo, que define se a junta é prismática ou rotativa. Ainda, é atribuído um valor para objetos de junta que corresponde à sua posição.

Na classe *Manipulator* a cinemática direta foi desenvolvida a partir do método de Denavit-Hartenberg. Para tal, foi criada uma classe *DHMatrix* que representa uma matriz de Denavit-Hartenberg, como a da Equação 1. Para cada junta adicionada à cadeia cinemática, é criado um objeto *DHMatrix* associado a ela. Assim, a alteração do valor das juntas

modifica a matriz de Denavit-Hartenberg correspondente e, a partir disso, a cadeia cinemática é atualizada. O tipo da junta, prismática ou rotativa, determina qual o parâmetro da *DHMatrix* que deve ser afetado pelo seu movimento: o d ou o θ .

As operações de cinemática direta com o método de Denavit-Hartenberg exigem a manipulação de matrizes. Assim, desde a criação dos objetos *DHMatrix* até a multiplicação matricial e alteração dos elementos foi utilizada a biblioteca *numpy*.

Para a cinemática inversa utilizou-se o método geométrico descrito pelas Equações 3, 4 e 5. A função de *Manipulator* que executa a cinemática inversa do robô recebe as coordenadas desejadas e, a partir das informações das juntas e elos inseridos na cadeia cinemática, calcula e retorna os valores necessários para a junta prismática da base e para as duas juntas rotativas que a seguem. Vale ressaltar que os cálculos foram implementados para atender, a princípio, exclusivamente a cadeia cinemática proposta pelo projeto (PRRR) e consideraram que a última junta rotativa, responsável pela orientação do efetuador, permanecerá fixa. Os cálculos de trigonometria são realizados através de funções da biblioteca *math* (PSF, 2021b).

A classe *MotionPlanner* define um caminho a partir de instantes e posições das juntas iniciais e finais. A implementação dessa classe considera que entre a partida e o destino não existem obstáculos. É a partir de um objeto de *MotionPlanner* que a classe *Trajectory* gera a trajetória do robô.

Trajectory é uma classe genérica de um gerador de trajetória. Nessa classe são definidos dois métodos especiais da linguagem Python: *iter* e *next*. Dessa forma, ela passa a funcionar como um iterador, ou seja, passa a ser possível iterar pelos seus elementos que, no caso, são posições de juntas ao longo da trajetória. Essa classe tem um parâmetro *step*, intervalo de tempo entre um ponto e outro da trajetória, que influencia na suavidade do movimento.

Ainda, foi implementada a classe *PolynomialTrajectory*. Ela é uma especialização de *Trajectory* que gera trajetórias de acordo com polinômios de quinto grau. Esse método foi implementado conforme mostra a Equação 16. Assim como para as operações com as matrizes de Denavit-Hartenberg, os cálculos matriciais necessários para a determinação dos polinômios que descrevem posição, velocidade e aceleração das juntas, são realizados com a biblioteca *NumPy*.

RISO é um sistema robótico com visão computacional para classificação de objetos e, portanto, possui as funcionalidades tanto de *ImageProcessor* quanto de *ObjectClassifier* quanto de *Manipulator*. Essas classes foram desenvolvidas de forma genérica, sem se restringirem aos objetivos específicos da aplicação desse projeto. No entanto, no caso do RISO,

essas classes devem trabalhar de forma interligada, formando um único sistema.

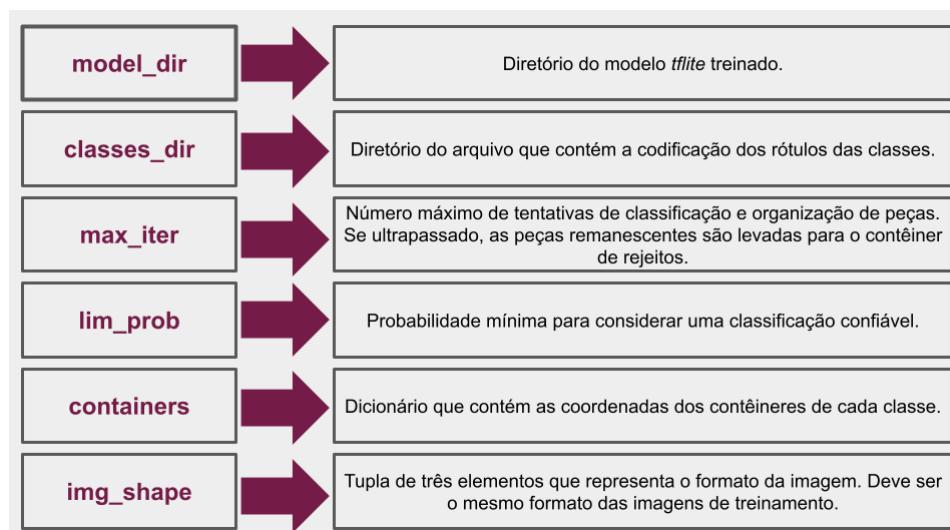
Portanto, criou-se uma classe RISO que herda as três classes citadas e as utiliza de acordo com as necessidades específicas do projeto. Isso inclui a configuração da cadeia cinemática de *Manipulator* de acordo com os elementos propostos para o robô real. Além disso, inclui a funcionalidade de movimentação que gerencia as etapas, funções e ações necessárias para movimentar o robô até coordenadas desejadas.

A função da classe RISO responsável por mover o robô cria uma instância de *MotionPlanner* e de um gerador de trajetória polinomial, *PolynomialTrajectory*. Essa função, recebe as coordenadas cartesianas desejadas como destino e, através de *MotionPlanner*, *PolynomialTrajectory* e das funções de cinemática direta e inversa herdadas de *Manipulator*, converte as coordenadas para o espaço das juntas, planeja o movimento necessário de cada junta, gera a trajetória e, ainda, atualiza os valores das juntas do seu modelo cinemático.

Além disso, a biblioteca PySerial ([LIECHTI, c2017](#)) é utilizada para promover uma interface serial pela qual os pontos gerados pelo gerador de trajetória possam ser enviados para o microcontrolador que comanda os motores. Essa mesma interface é utilizada para receber mensagens que informam a estimativa do ângulo real de cada junta ao fim de cada movimento. São essas estimativas que são utilizadas para a atualização dos valores das juntas do modelo cinemático do objeto RISO.

Apesar de várias características do sistema já terem sido pré-definidas na classe RISO, como a estrutura da cadeia cinemática do robô, alguns parâmetros podem variar a cada operação e, por isso, devem ser definidos pelo usuário na execução do código principal. Essas variáveis são listadas na Figura 25.

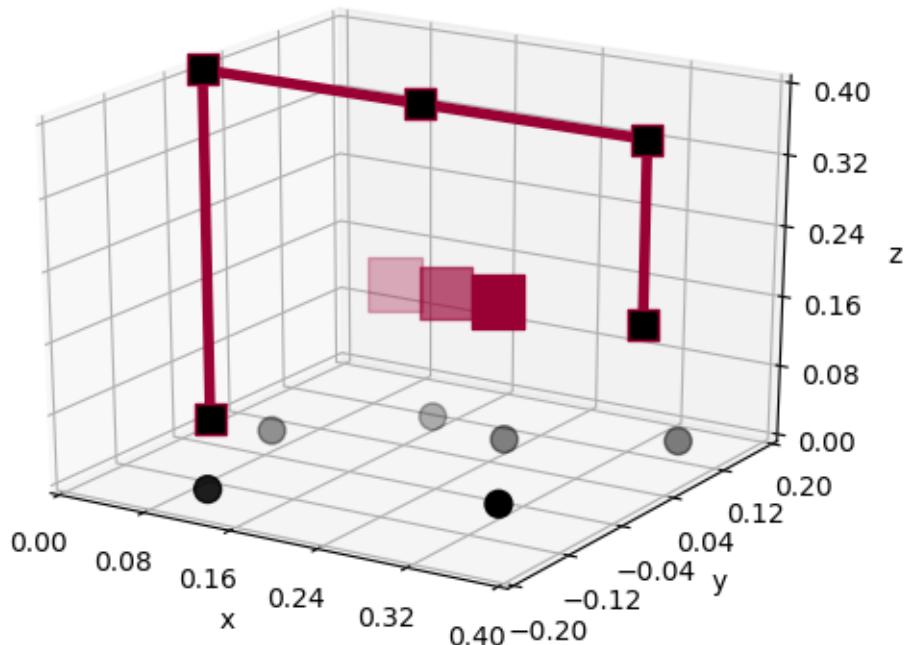
Figura 25 – Variáveis a serem definidas pelo usuário



Após definidas, um objeto RISO é criado e inicializado para que a rotina de identificação, classificação e organização de peças possa começar. Essa rotina é descrita pela Figura 16. As filas de objetos desconhecidos (*UnknownObjectQueue*) e de objetos identificados (*IdObjQueue*) são implementadas como objetos *Queue* disponibilizados pela biblioteca Python *queue* (PSF, 2021c). Assim, o código principal atua, basicamente, como um gerenciador das tarefas implementadas na classe RISO.

Para a validação do *software* do projeto, foi implementada uma classe de simulação do robô, que permite avaliar o funcionamento do sistema mesmo sem o robô montado. Ela funciona a partir de um objeto da classe RISO e, portanto, tem acesso às informações sobre a cadeia cinemática do robô. Essa funcionalidade permite mostrar o manipulador e seu espaço de trabalho. Além disso, é possível mostrar a representação dos contêineres definidos pelo usuário bem como das peças identificadas pelo sistema de visão computacional. Assim, é possível avaliar a interação do robô com os demais elementos do sistema (contêineres e objetos) e, ainda, validar o funcionamento das unidades de *software* do projeto em conjunto.

Figura 26 – Simulação do sistema robótico



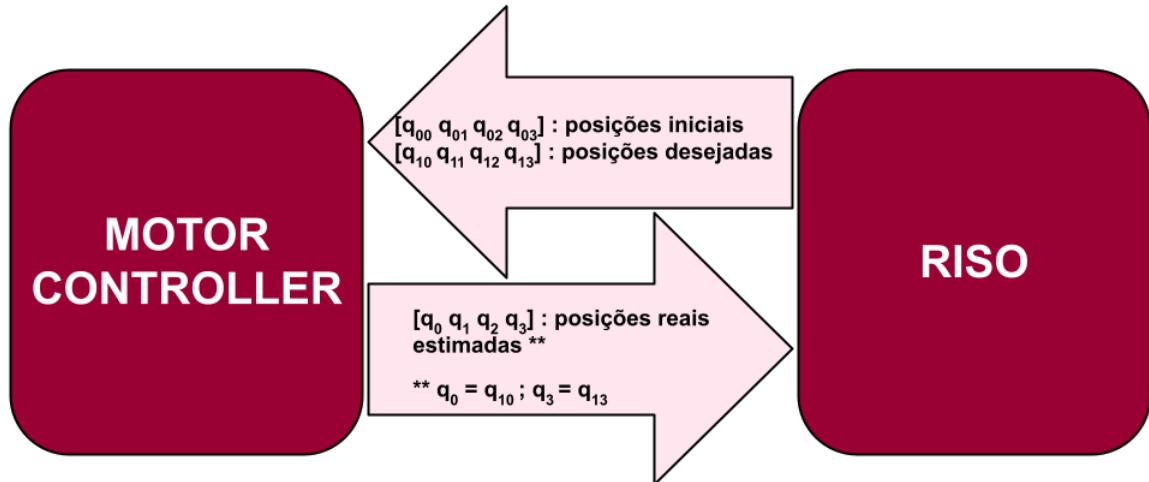
A simulação foi feita através de plotagens implementadas com a biblioteca *Matplotlib* (HUNTER, 2007). As coordenadas das juntas são utilizadas para traçar um gráfico de linhas. Assim, cada junta corresponde a um ponto marcado no gráfico e as linhas que as conectam são os elos do manipulador. Dessa forma, conforme são alterados os valores das juntas da cadeia cinemática, é possível atualizar o gráfico para as novas coordenadas e, assim, a simulação acompanha o movimento do robô. Os contêineres e os objetos, por sua vez, são representados por pontos em um gráfico de dispersão. As coordenadas dos pontos que

representam os contêineres são fixas, definidas pelo usuário, como mostra a Figura 25. Já as coordenadas dos pontos que simbolizam as peças são obtidas do processamento das imagens reais do espaço de trabalho, no qual os objetos estão dispostos.

A classe de *MotorController* foi implementada na linguagem MicroPython. Essa unidade recebe comandos via serial através do objeto UART ([GEORGE; SOKOLOVSKY, 2021a](#)) do módulo *machine*, que contém funções relacionadas diretamente ao *hardware* da placa ([GEORGE; SOKOLOVSKY, 2021b](#)). Os dados recebidos são processados e, assim, tem-se a quantidade de pulsos necessária em cada motor, bem como a direção do movimento. Apesar de os dados de entrada conterem comandos para todos os motores, estão sendo acionados, até então, apenas os motores responsáveis pelo posicionamento do robô nos eixos *x* e *y*. O motor que seria responsável pelo movimento vertical do robô bem como o motor que ditaria a orientação do efetuador, não estão sendo acionados.

Após o movimentos dos motores, a classe *MotorController* retorna uma estimativa da posição real de cada junta, dada a quantidade de passos efetuados e a resolução do motor. No caso dos dois motores que não estão sendo acionados, considera-se que o movimento foi completamente preciso, ou seja, a posição retornada é igual à posição final que havia sido requisitada. Essa troca de informações entre a classe *MotorController* e a RISO, que é executada no código principal do sistema, é ilustrada na Figura 27.

Figura 27 – Comunicação entre as classes RISO e *MotorController*



4.2 Hardware

O *software* principal do projeto foi implementado em uma RaspberryPi 3 Model B+ ([RPF, 2021b](#)) que, com seu SoC quad-core 1.4GHz, se mostrou adequada para processar

o *software* necessário no projeto. Conectado a essa placa, através da conexão CSI, fica o módulo de câmera Rev 1.3 de 5Mp – modelo oficial da Raspberry Pi ([RPF, 2021a](#)). O microcontrolador utilizado para controle do manipulador foi é um ESP32 32-bit dual core ([Espressif Systems, 2021](#)). Essas duas placas foram conectadas de modo a possibilitar a comunicação serial entre elas.

Para validação inicial do sistema, sem o robô real, apenas com a simulação, foi montada uma estrutura caseira para suporte da *Raspberry Pi* e da câmera, como mostra a Figura 28. Além disso, uma superfície branca foi utilizada como tabuleiro onde as peças são dispostas, como mostra a Figura 29.

Figura 28 – *Raspberry Pi 3 Model B+* e módulo de câmera Rev 1.3

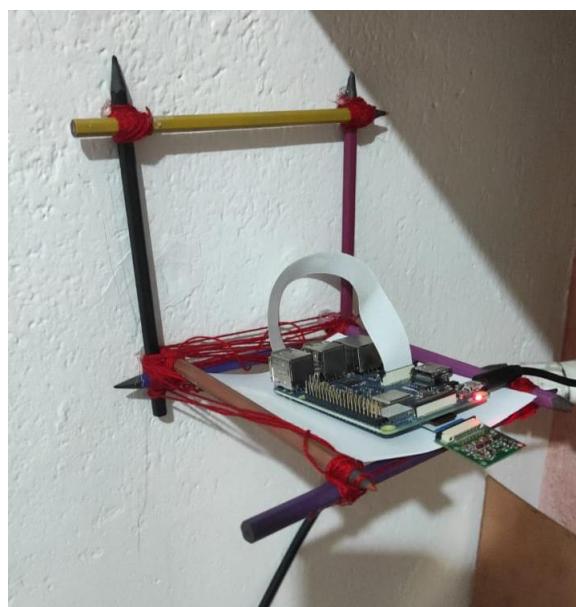
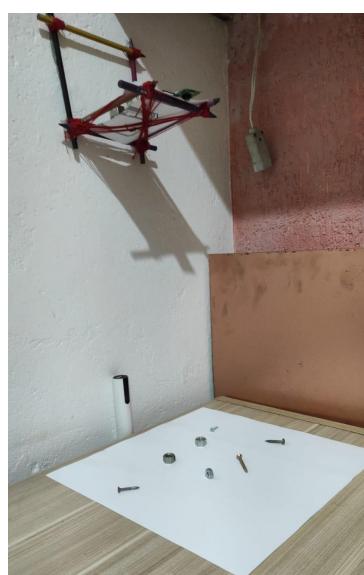


Figura 29 – Protótipo do sistema RISO



Apesar de a estrutura mecânica do robô não ter sido montada, foi desenvolvida uma placa para o acionamento dos motores responsáveis por movimentar as juntas do manipulador que o deslocam nos eixos x e y . Foram utilizados dois motores de passo Nema 23 - 7 kgf.cm - 1.4A AK23 / 7.0F8FN1.8. Para o acionamento desses, usou-se Driver para Motor de Passo DRV8825. Com esses componentes, foi possível obter uma resolução de 1.8°.

O circuito foi implementado em uma placa, como mostram as Figuras 30 e 31, em que o ESP32 está conectado à Raspberry Pi e aos *drivers* que controlam os motores de passo.

Figura 30 – Esquemático da placa

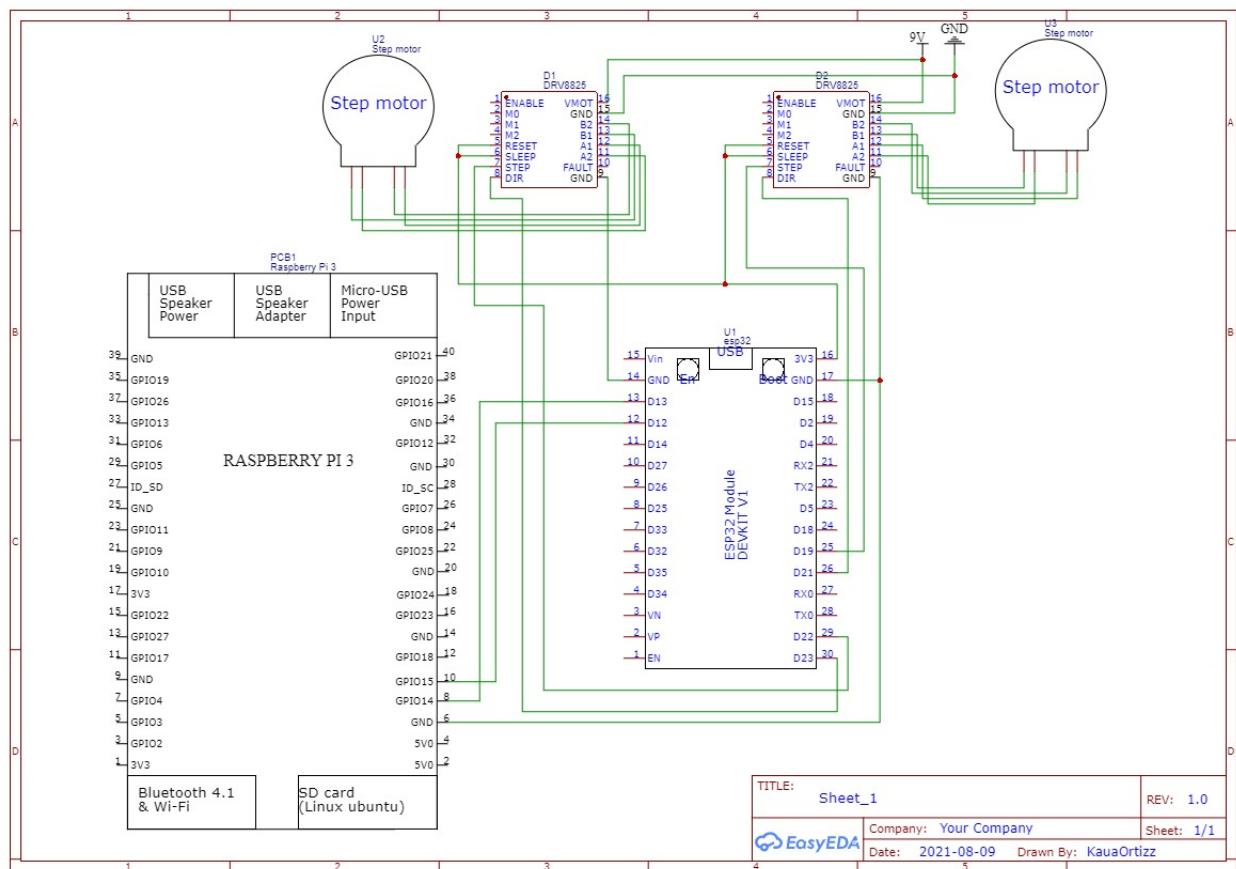
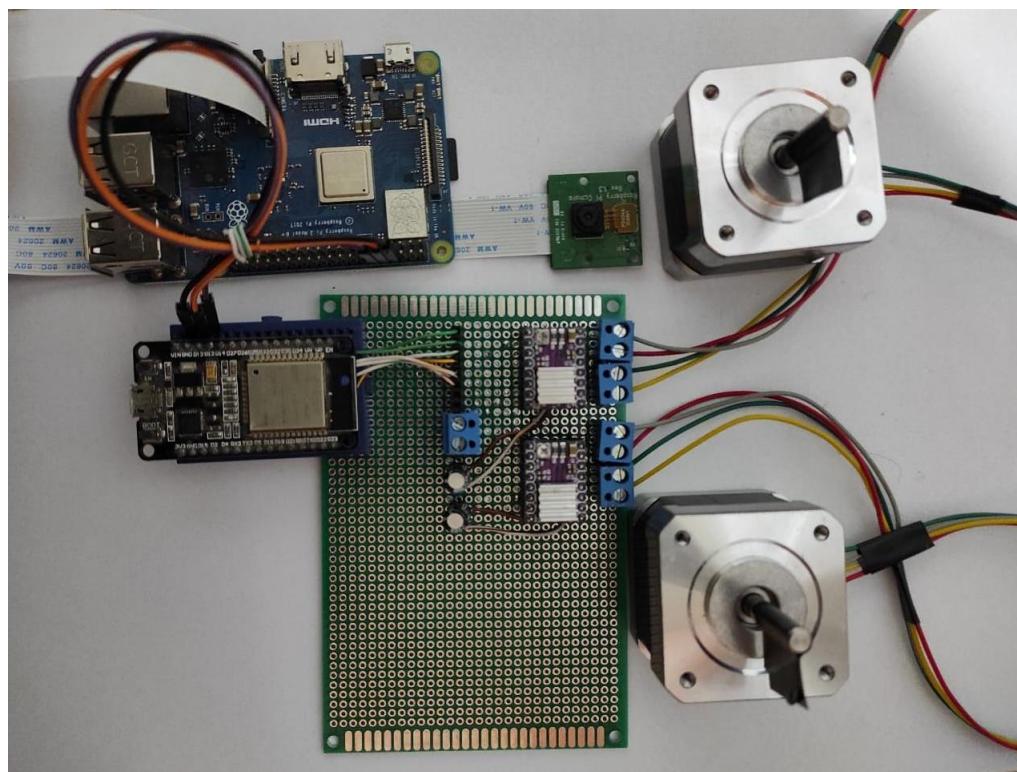


Figura 31 – Placa montada



5 Experimentação e Resultados

A experimentação do protótipo desenvolvido foi feita em um estudo de caso de organização de porcas e parafusos. Primeiramente, cada classe criada foi testada de forma individual, antes de ser integrada ao sistema.

O teste do funcionamento da rede neural convolucional desenvolvida se deu através da análise da acurácia fornecida pela própria API Keras e também pelo teste em um conjunto de 50 imagens separadas para essa finalidade. Depois de duas iterações de treinamento obteve-se 99,56% de acurácia, como mostra a Figura 32, ou seja, 99,56% das predições resultaram na classe correta. Para as 50 imagens, que não fizeram parte do processo de treinamento, o modelo foi capaz de acertar todas as classificações.

Figura 32 – Resultado do treinamento da CNN

```

Console IPython
Console 11/A
Python 3.6.9 (default, Jan 26 2021, 15:33:00)
Type "copyright", "credits" or "license" for more information.

IPython 5.5.0 -- An enhanced Interactive Python.
?           -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help        -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.

In [1]: runfile('/home/amanda/Área de Trabalho/RISO/CódigosImportantes/Treinamento/treinamento.py', wdir='/home/amanda/Área de Trabalho/RISO/CódigosImportantes/Treinamento')
Epoch 1/2
450/450 [=====] - 12s 27ms/step - loss: 0.2488 - accuracy: 0.8956 -
val_loss: 0.0036 - val_accuracy: 1.0000
Epoch 2/2
450/450 [=====] - 12s 26ms/step - loss: 0.0161 - accuracy: 0.9956 -
val_loss: 8.9045e-05 - val_accuracy: 1.0000

In [2]:

```

Os testes do algoritmo responsável por detectar os objetos na imagem também tiveram sucesso. Apesar de a iluminação influenciar no seu funcionamento, após o ajuste da sensibilidade da detecção e das condições do espaço de trabalho, ele também funcionou sem nenhuma falha, detectando todos e apenas os objetos desejados. O ajuste da sensibilidade é feito através da alteração do limiar da função *threshold*.

A localização das peças no espaço de trabalho também funcionou. Foram feitas medições com régua e comparadas aos resultados indicados pelo *software*. Os dois métodos de medição das coordenadas apresentaram uma diferença de cerca de $\pm 1,025$ cm. No entanto, o fato de o sistema não ter sido calibrado da maneira mais adequada e pela falta de precisão do método e do instrumento de medição, não foi possível calcular qual o erro exato do *software* de localização das peças. Ainda assim, por não ser tratar de um sistema robótico de precisão, os resultados foram considerados adequados para os objetivos inciais do projeto.

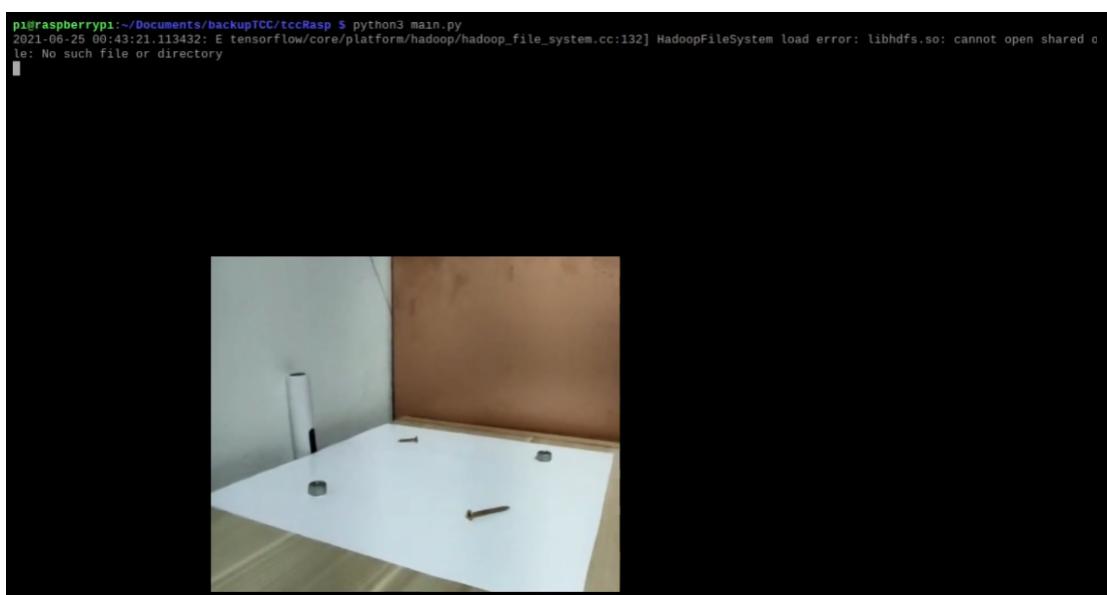
Após a validação de cada módulo de *software* do projeto, foi conduzida uma bateria de testes do sistema completo, operando em um cenário real de organização de porcas e parafusos. Nessa etapa, ainda sem elementos mecânicos, utilizou-se a simulação do robô para validar o funcionamento dos módulos relacionados ao movimento do manipulador e, assim, analisar se ele chegaria nas coordenadas desejadas.

Após o ajuste da iluminação e do limiar da função *threshold* para detecção das peças,

foram feitos testes com porcas e parafusos em diferentes quantidades e distribuições. Em todos esses experimentos, obteve-se 100% de êxito: não só todas as peças foram corretamente detectadas, classificadas e localizadas, mas também o robô simulado movimentou-se como desejado, alcançando cada objeto com a precisão esperada. Uma vez que o manipulador real não estava sendo utilizado, os objetos foram sendo retirados manualmente do espaço de trabalho.

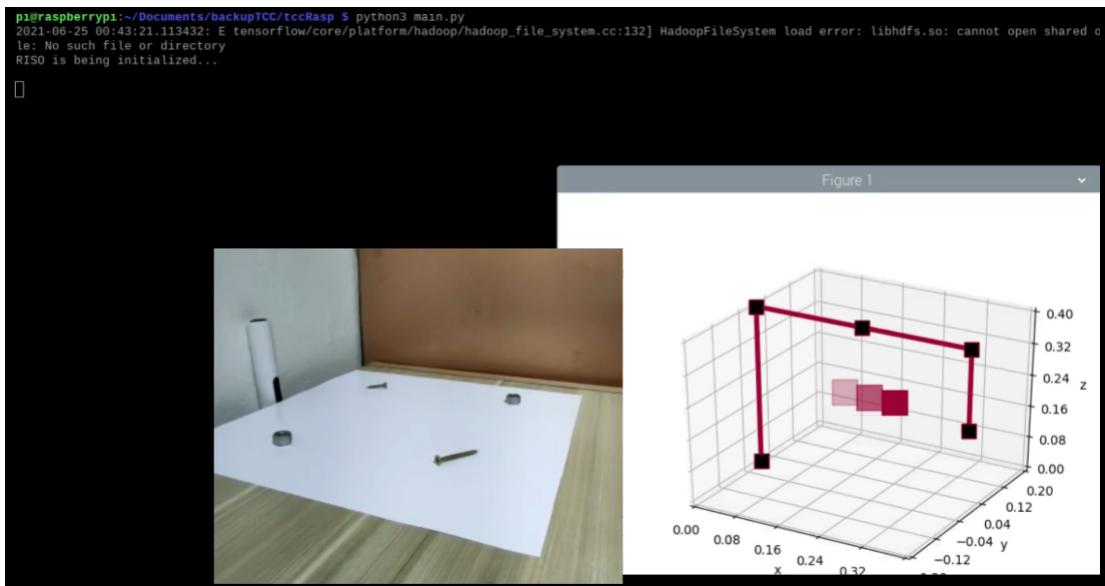
As Figuras 33, 34, 35, 36, 37, 38, 39, 40 e 41 mostram parte do processo ocorrido em um dos testes. A Figura 33 mostra o espaço de trabalho no momento em que o código principal começa a ser executado.

Figura 33 – Início do teste



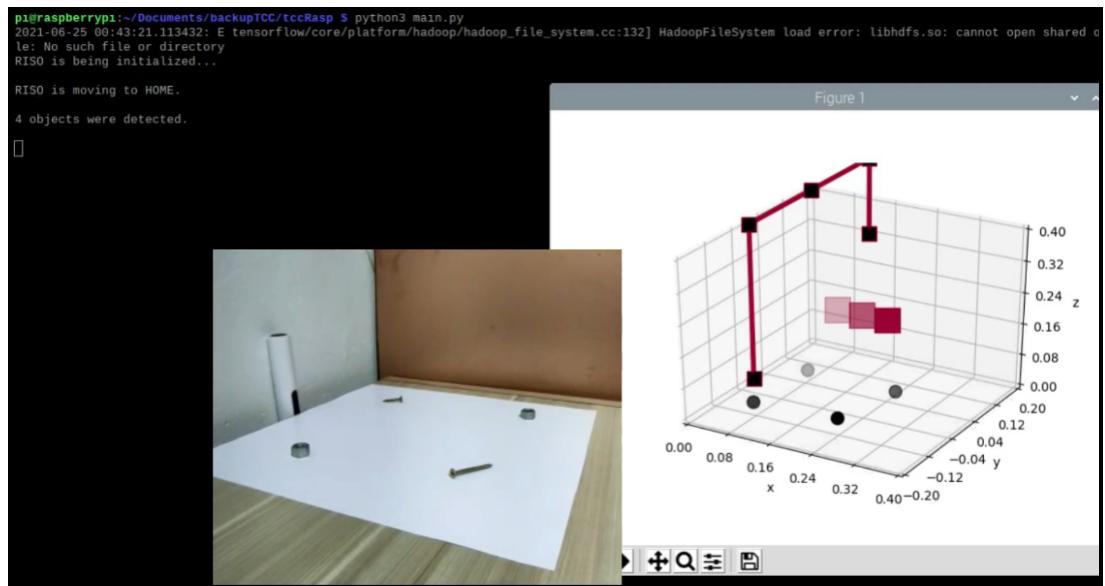
A Figura 34 mostra o início da simulação durante o processo de inicialização do sistema.

Figura 34 – Inicialização do sistema e da simulação



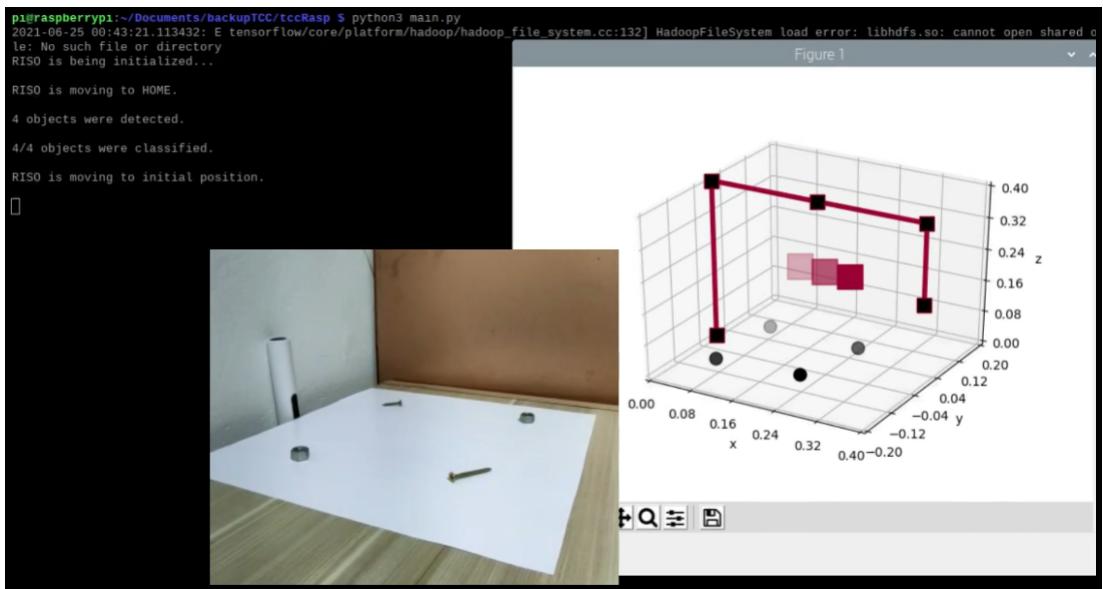
Na Figura 35 é possível ver que o sistema identificou os quatro objetos que estavam distribuídos na mesa. Além disso, nesse instante, o robô se encontra em sua posição de *home*, na qual é possível capturar a imagem do espaço de trabalho sem a interferência do manipulador.

Figura 35 – Detecção de objetos



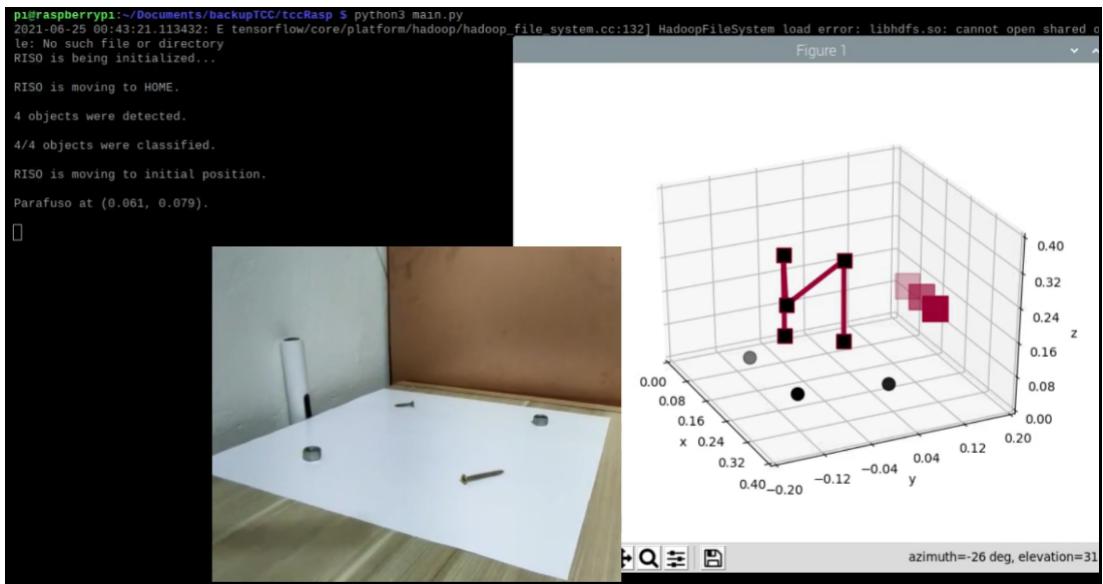
A Figura 36 mostra que dos quatro objetos detectados, todos conseguiram ser classificados com sucesso, ou seja, a probabilidade de a classe ser a certa ficou acima do limiar definido que, nesse caso, era de 90%. Além disso, nessa figura pode-se notar que o robô voltou ao centro do tabuleiro, que é sua posição inicial, ou seja, ele já está preparado para começar a organização das peças.

Figura 36 – Preparação do manipulador após a classificação



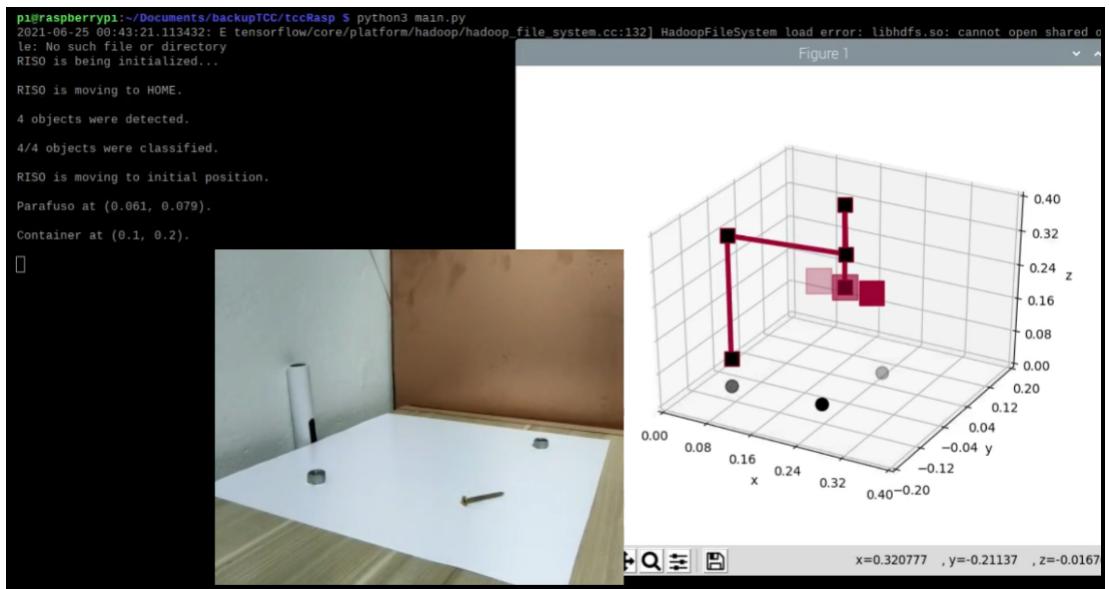
Na Figura 37 constam as coordenadas da primeira peça a ser organizada: um parafuso em $(0,061, 0,079)$. As coordenadas estão em metros. Dessa forma, o robô é deslocado até essas coordenadas e recolhe o objeto.

Figura 37 – Deslocamento do robô até o parafuso em $(0,061, 0,079)$



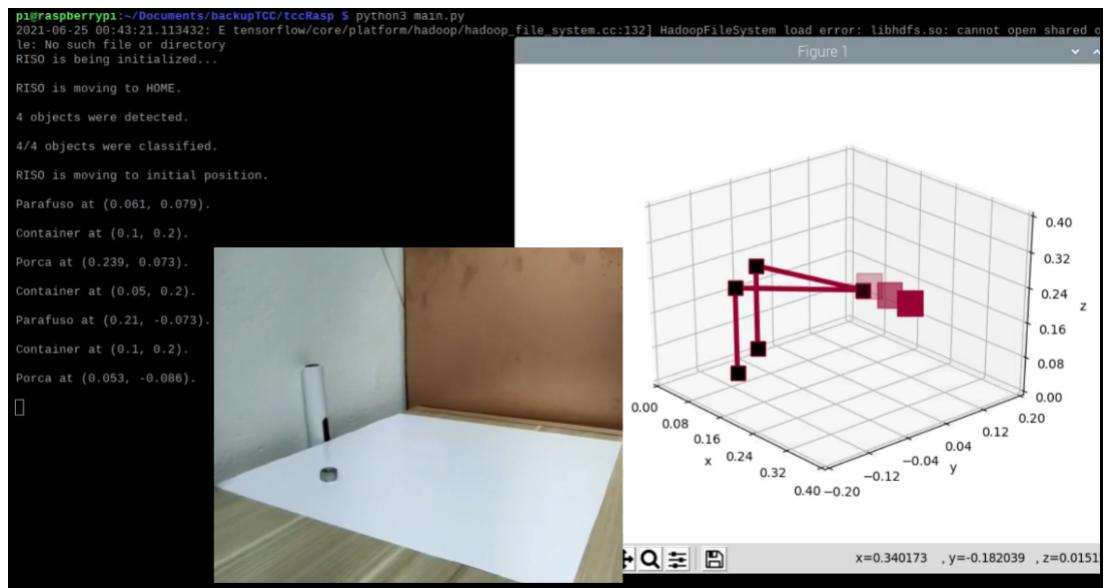
A Figura 38 mostra o parafuso, já retirado do espaço de trabalho, sendo levado até seu contêiner correspondente. Após isso, o mesmo processo de deslocamento até as coordenadas do objeto seguido pelo carregamento deste até o contêiner de sua classe se repete para as demais peças.

Figura 38 – Organização do parafuso em seu contêiner



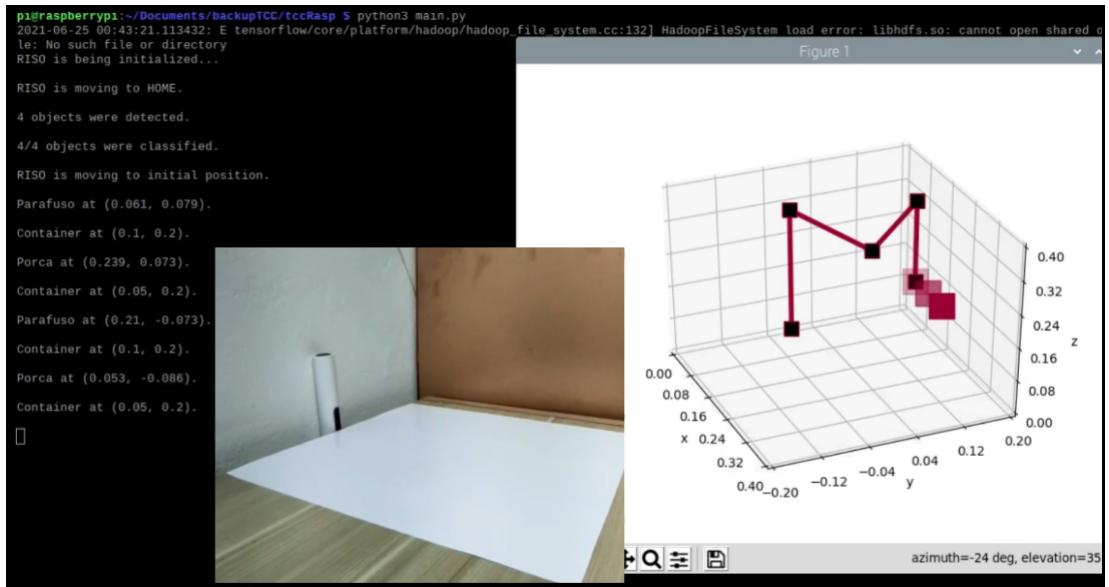
Na Figura 39, o último objeto está sendo retirado da mesa. Ele é classificado como uma porca e localizado nas coordenadas (0,053, -0,086).

Figura 39 – Deslocamento do robô até a porca em (0,053, -0,086)



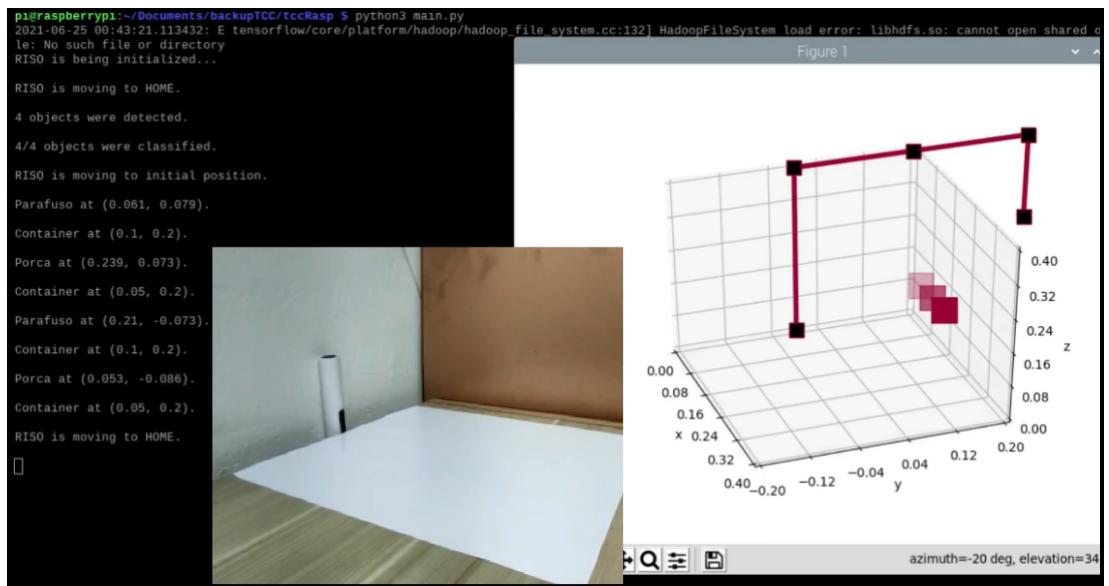
A Figura 40 mostra a porca sendo levada até o seu devido contêiner.

Figura 40 – Organização da porca em seu contêiner



Por fim, a Figura 41 mostra que não existem mais objetos a serem organizados e, portanto, o manipulador volta para sua posição de *home*.

Figura 41 – Fim da rotina do robô



Após o teste documentado acima, foram conduzidas experimentações do sistema em conjunto com o acionamento dos motores. O funcionamento pôde ser constatado através da comparação entre a posição dos motores e simulação do manipulador. Essa comparação pode ser observada nas Figuras 42, 43 e 44 em que o motor mais à direita da imagem corresponde ao movimento do elo mais próximo da base do robô e o outro, ao movimento do segundo elo.

Figura 42 – Posicionamento das juntas na inicialização do robô

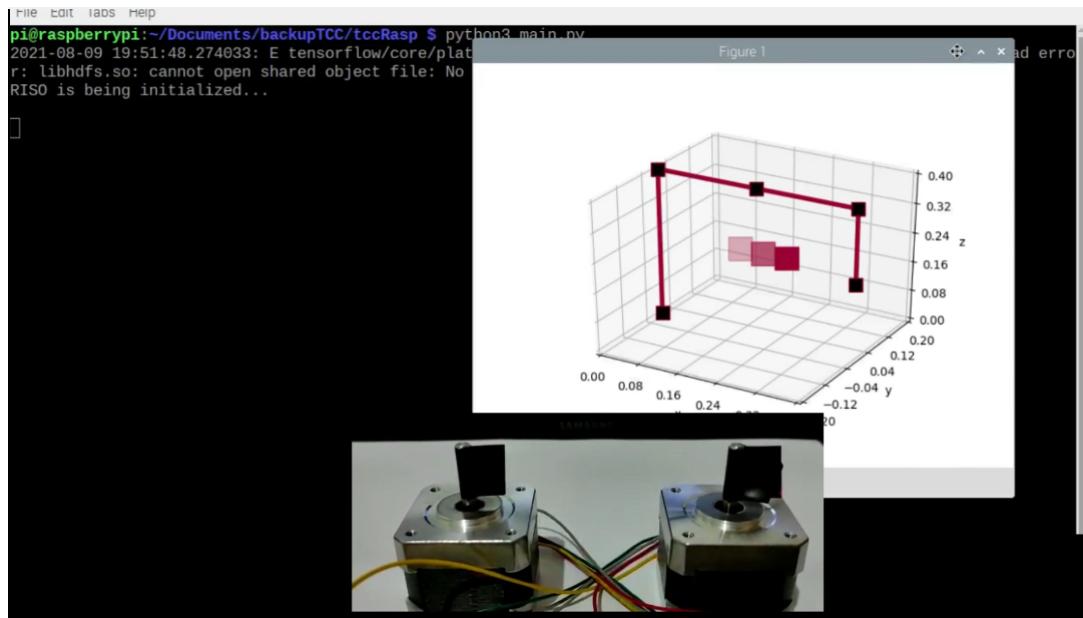


Figura 43 – Posicionamento das juntas no momento de captura da imagem

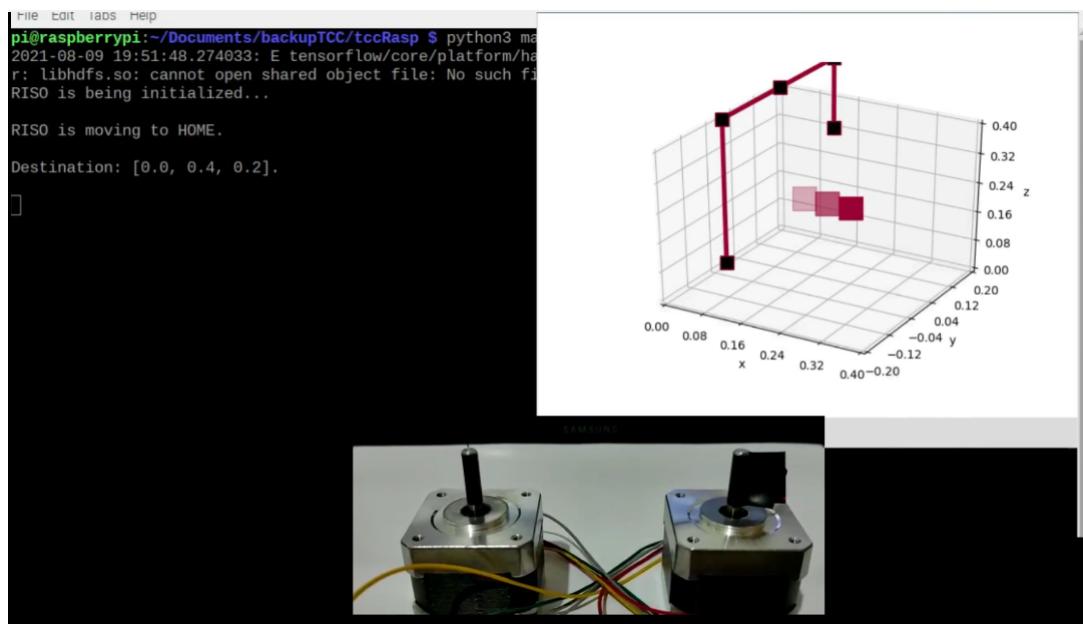
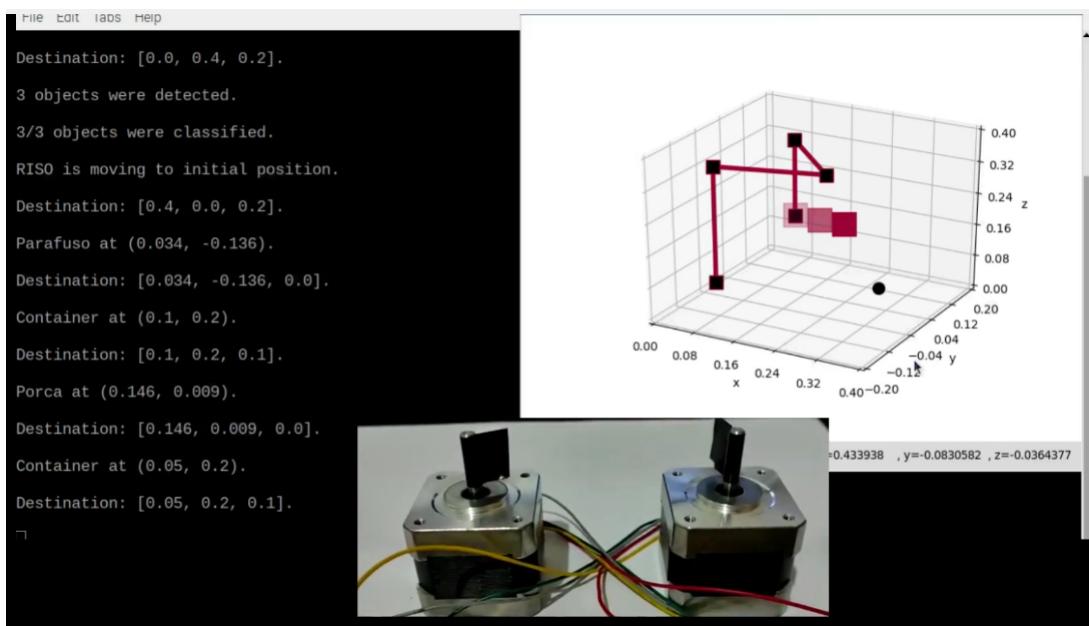


Figura 44 – Posicionamento das juntas para alcançar o contêiner das porcas



6 Conclusão

O desenvolvimento do projeto permitiu o estudo e o entendimento de aspectos dos ramos de visão computacional e aprendizado de máquina. Além do aspecto de pesquisa, foi possível cumprir o objetivo geral do projeto de desenvolver um sistema inteligente para separação e organização de peças.

Devido às restrições enfrentadas no decorrer do projeto, não foi possível montar a parte mecânica do protótipo. No entanto, os aspectos básicos de um sistema automatizado foram contemplados, uma vez que as etapas de sensoriamento, controle e atuação foram todas cumpridas. Assim, apesar de não ter atingido exatamente o que foi idealizado, todo o *software* foi devidamente desenvolvido, desde a parte de visão computacional até a parte da robótica. Além disso, foi possível comprovar o funcionamento do sistema através do sucesso no controle dos motores que movimentariam o robô.

A aplicação de visão computacional pôde ser validada nos testes conduzidos. Apesar da ausência de erros nessa unidade, entende-se que a falta de dados e testes mais abrangentes possa ter contribuído para essa taxa de 100% de acertos do sistema. Em uma continuação do projeto seria possível estender esses testes e coletar mais dados para treinamento da rede, conferindo assim maior variabilidade às amostras e, portanto, uma medição mais precisa do resultado para uma validação mais ampla.

Além disso, a localização dos objetos também é passível de evolução. Com uma estrutura adequada, é possível realizar uma calibração precisa do sistema e, assim, possibilitar que a determinação das coordenadas das peças seja mais certeira.

A operação do sistema também pode ser melhorada. A rotina do robô e a modularização do projeto foi desenvolvida justamente de modo a proporcionar uma expansão de funcionalidades. A escolha de partitionar a operação por etapas e não por objeto visou à aplicação de uma possível otimização do movimento. Assim, da forma que foi implementado, é possível analisar a fila de objetos a serem organizados e ordená-los de forma a otimizar o processo de movimentação do robô.

Quanto ao acionamento dos motores, sua posição foi analisada de forma visual, sem o uso de sensores ou medições precisas. Ademais, os motores utilizados se movem 1.8° a cada passo o que, em alguns casos, pode resultar em um erro de posicionamento. No entanto, como não se trata de um sistema de precisão, considerou-se que os resultados aproximados obtidos atendem ao propósito do projeto.

Além disso, outro avanço do projeto consiste em produzir as peças do robô e montá-lo. Assim, seria possível observar o sistema completo em funcionamento em operações reais em que os objetos pudessem, de fato, ser organizados em contêineres.

Por fim, o sistema, apesar de ter sido validado apenas para um estudo de caso de organização de porcas e parafusos, pode ser expandido para outros cenários, inclusive em uma operação dinâmica em que os objetos possam estar em uma esteira, por exemplo. Assim, outras aplicações similares podem ser exploradas a partir do que foi desenvolvido.

Referências

- AKAR, Özlem; GÜNGÖR, Oguz. Classification of multispectral images using random forest algorithm. *Journal of Geodesy and Geoinformation*, v. 1, n. 2, p. 105–112, 2012.
- ANGELES, Jorge. *Fundamentals of Robotic Mechanical Systems: Theory, Methods, and Algorithms*. Boston, MA: Springer US, 2007. 233–256 p. ISBN 978-0-387-34580-2. Disponível em: <https://doi.org/10.1007/978-0-387-34580-2_6>.
- DATA SCIENCE ACADEMY. *Deep Learning Book*. c2021. Acesso em: 13/05/2021.
- DESAI, Meha; SHAH, Manan. An anatomization on breast cancer detection and diagnosis employing multi-layer perceptron neural network (mlp) and convolutional neural network (cnn). *Clinical eHealth*, Elsevier, 2020.
- Espressif Systems. *ESP32 Datasheet*. Shangai, 2021.
- GEORGE, Damien; SOKOLOVSKY, Paul. *class UART – duplex serial communication bus*. 2021. Disponível em: <<https://docs.micropython.org/en/latest/library/machine.UART.html>>. Acesso em: 09/08/2021.
- GEORGE, Damien; SOKOLOVSKY, Paul. *machine — functions related to the hardware*. 2021. Disponível em: <<https://docs.micropython.org/en/latest/library/machine.html>>. Acesso em: 09/08/2021.

- HUNTER, J. D. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007.
- IBM. *Artificial Intelligence (AI)*. 2020. Disponível em: <<https://www.ibm.com/cloud/learn/what-is-artificial-intelligence/>>. Acesso em: 06/05/2021.
- IBM. *Computer vision*. 2021. Disponível em: <<https://www.ibm.com/topics/computer-vision/>>. Acesso em: 06/05/2021.
- KAVRAKI, Lydia E; LAVALLE, Steven M. *Motion Planning*. 2008.
- KERAS. *Keras*. c2021. Disponível em: <<https://keras.io/>>. Acesso em: 07/05/2021.
- KUCUK, Serdar; BINGUL, Zafer. *Robot kinematics: Forward and inverse kinematics*. [S.l.]: INTECH Open Access Publisher, 2006.
- LAGES, Walter Fetter. Geração de trajetórias. *Brasil: Universidade Federal do Rio Grande do Sul*, 2005.
- LAURETTO, Marcelo S. Árvores de decisão. 2010.
- LIECHTI, Chris. *pySerial — pySerial 3.4 documentation*. c2017. Disponível em: <<https://pyserial.readthedocs.io/en/latest/>>. Acesso em: 12/08/2021.
- LINGE, Svein; LANGTANGEN, Hans Petter. *Programming for Computations - Python*. Cham: Springer International Publishing, 2020. ISBN 978-3-030-16877-3.
- MARENCONI, Maurício; STRINGHINI, Stringhini. Tutorial: Introdução à visão computacional usando opencv. *Revista de Informática Teórica e Aplicada*, v. 16, n. 1, p. 125–160, 2009.
- MONARD, Maria Carolina; BARANAUSKAS, José Augusto. Conceitos sobre aprendizado de máquina. *Sistemas inteligentes-Fundamentos e aplicações*, Manole, v. 1, n. 1, p. 32, 2003.
- NEDELKOVSKI, Dejan. *SCARA ROBOT / HOW TO BUILD YOUR OWN ARDUINO BASED ROBOT*. c2019. Disponível em: <<https://howtomechatronics.com/projects/scara-robot-how-to-build-your-own-arduino-based-robot/>>. Acesso em: 03/09/2021.
- NORRIS, Donald. *Python for Microcontrollers: Getting Started with MicroPython*. New York: McGraw-Hill Education, 2016.
- NUMPY. *NumPy*. 2021. NumPy. Disponível em: <<https://numpy.org/>>. Acesso em: 12/07/2021.
- OPENCV. *About - OpenCV*. c2021. Disponível em: <<https://opencv.org/about/>>. Acesso em: 04/06/2021.
- PICCININI, Paolo et al. Multiple object detection for pick-and-place applications. In: *MVA*. [S.l.: s.n.], 2009. p. 362–365.
- PODLOZHNYUK, Victor. Image convolution with cuda. *NVIDIA Corporation white paper, June*, v. 2097, n. 3, 2007.
- PSF. *About Python*. 2021. Python Software Foundation. Disponível em: <<https://python.org/about/>>. Acesso em: 23/06/2021.

PSF. *math — Mathematical functions*. 2021. Python Software Foundation. Disponível em: <<https://docs.python.org/3/library/math.html>>. Acesso em: 12/07/2021.

PSF. *queue — A synchronized queue class*. 2021. Python Software Foundation. Disponível em: <<https://docs.python.org/3/library/queue.html>>. Acesso em: 19/07/2021.

RASPBERRY PI FOUNDATION. *Python - Raspberry Pi Documentation*. c2021. Disponível em: <<https://www.raspberrypi.org/documentation/usage/python/>>. Acesso em: 20/05/2021.

ROCHA, CR; TONETTO, CP; DIAS, Altamir. A comparison between the denavit-hartenberg and the screw-based methods used in kinematic modeling of robot manipulators. *Robotics and Computer-Integrated Manufacturing*, Elsevier, v. 27, n. 4, p. 723–728, 2011.

ROMANO, V; DUTRA, M. Introdução à robótica industrial. *Robótica Industrial: Aplicação na Indústria de Manufatura e de Processo*, São Paulo: Edgard Blücher, p. 1–19, 2002.

RPF. *Raspberry Pi*. 2020. Raspberry Pi Foundation. Disponível em: <<https://www.raspberrypi.org/products/>>. Acesso em: 14/05/2021.

RPF. *Camera Module - Raspberry Pi Documentation*. 2021. Raspberry Pi Foundation. Disponível em: <<https://www.raspberrypi.org/documentation/hardware/camera>>. Acesso em: 23/06/2021.

RPF. *Raspberry Pi 3 Model B+*. 2021. Raspberry Pi Foundation. Disponível em: <<https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/>>. Acesso em: 14/05/2021.

SANTOS, Flávia Moreira dos; WATANABE, Émerson Teruhiko; CARRARA, Valdemir. Estudo da cinemática inversa aplicada num braço robótico. *Anais do 12º Encontro de Iniciação Científica e Pós-Graduação do ITA-XII ENCITA*, 2006.

SAS. *Machine Learning: What it is and why it matters*. 2021. Disponível em: <https://www.sas.com/en_us/insights/analytics/machine-learning.html>. Acesso em: 06/05/2021.

SAS. *Redes Neurais: O que são e qual sua importância?* 2021. Disponível em: <https://www.sas.com/pt_br/insights/analytics/neural-networks.html>. Acesso em: 07/05/2021.

SCIKIT-LEARN. *Scikit-Learn*. c2020. Disponível em: <<https://scikit-learn.org/stable/>>. Acesso em: 07/05/2021.

SCIKIT-LEARN. *sklearn.ensemble.RandomForestClassifier*. c2020. Disponível em: <<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>>. Acesso em: 14/05/2021.

SHARMA, Sagar; SHARMA, Simone. Activation functions in neural networks. *Towards Data Science*, v. 6, n. 12, p. 310–316, 2017.

SHERBINY, Ahmed el; EL-HOSSEINI, Mostafa; HAIKAL, Amira. A comparative study of soft computing methods to solve inverse kinematics problem. *Ain Shams Engineering Journal*, v. 9, 2017.

SICILIANO, B. et al. *Robotics: Modelling, Planning and Control*. London: Springer, 2009. 632 p. (Advanced Textbooks in Control and Signal Processing).

SILVA, Samuel et al. Uso de redes neurais convolucionais para identificar displasia cortical focal em pacientes com epilepsia refratária. In: SBC. *Anais do XVII Encontro Nacional de Inteligência Artificial e Computacional*. [S.l.], 2020. p. 211–221.

TENSORFLOW. *Guia do TensorFlow Lite*. 2021. Disponível em: <<https://www.tensorflow.org/lite/guide>>. Acesso em: 27/05/2021.

WANG, Ming-Hwa. Artificial intelligence and subfields. *Santa Clara University, Department of Computer Engineering*, 2017.