

Desenvolvimento de uma célula robótica colaborativa para demonstração

Pedro António Moura Monforte

Dissertação de Mestrado

Orientador: Prof. Joaquim Gabriel Mendes

Coorientador na empresa Growskills: Rúben Almeida



Mestrado Integrado em Engenharia Mecânica

Setembro, 2021

Resumo

Para introduzir os futuros técnicos e engenheiros à robótica colaborativa, as células de demonstração têm sido amplamente utilizadas. Assim, o desenvolvimento das mesmas assume um contorno prático crucial para a formação nesta área. A relevância da robótica colaborativa prende-se com o facto de ser um ramo em expansão que favorece uma rápida e prática automatização a um custo reduzido, constituindo uma excelente tecnologia para pequenas empresas se lançarem num mercado cada vez mais competitivo. Além disso, as suas características únicas permitem que se distancie das complexas e dispendiosas soluções oferecidas pela robótica tradicional, inacessíveis a empresas em início de vida ou que pretendam uma solução mais compacta e flexível. Foi no sentido de desenvolver uma célula robótica colaborativa para demonstração que a presente tese surgiu.

Neste projeto, desenvolveu-se uma pequena estação de trabalho que realiza operações dos robôs colaborativos da Universal Robots incluindo-se um pequeno sistema pneumático e componentes eletrónicos. A célula consiste num ciclo no qual se incorporam operações de montagem e paletização além de se recorrer a procedimentos característicos dos robôs, como *Conveyor Tracking* e o controlo de força/binário. Utilizou-se um sistema de visão para a deteção de cor e posição de elementos cilíndricos tendo-se recorrido à câmara Intel RealSense D435i. A programação do sistema de visão foi efetuada em linguagem Python e a célula é remotamente controlada por uma aplicação desenvolvida em C#. O trabalho iniciou-se com o projeto da célula; terminada esta caracterização inicial, encomendou-se o material necessário e imprimiram-se algumas peças para permitir a montagem de todo o sistema projetado.

Seguidamente, programou-se o sistema de visão para identificar a posição, orientação e cor das peças. Por fim, desenvolveu-se a aplicação em C# e elaborou-se o código na consola de programação para o correto funcionamento da célula. Além do funcionamento regular, criou-se ainda um modo manual que permite a atuação de válvulas e motores bem como monitorização do estado de sensores introduzidos.

Development of a Demonstrative Collaborative Robotic Cell

Abstract

To introduce future technicians and engineers to collaborative robotics, demonstration cells have been widely used. Thus, their development assumes a crucial practical outline for training in this area. The relevance of collaborative robotics is linked to the fact that it is an expanding field that favours quick and practical automation at a reduced cost, constituting an excellent technology for small companies to launch themselves in an increasingly competitive market. In addition, its unique characteristics allow it to distance itself from the complex and costly solutions offered by traditional robotics, which are inaccessible to companies just starting out or those looking for a more compact and avant-garde solution. It was in order to develop a collaborative robotic cell for demonstration that this thesis emerged.

In this project, a small workstation was developed that performs operations of Universal Robots collaborative robots, including a small pneumatic system and electronic components. The cell consists of a cycle in which assembly and palletizing operations are incorporated, in addition to resorting to procedures characteristic of collaborative robots, such as Conveyor Tracking and force/torque control. A vision system was used to detect the color and position of cylindrical elements, using the Intel RealSense D435i camera. The programming of the vision system was carried out in Python language and the cell is remotely controlled by an application developed in C#. In the beginning, it was projected; Once this initial characterization was completed, all the material that allowed its conception was ordered, in addition to having designed the extraordinary by 3D printing. Then, the vision system was programmed to respond to the station's needs. Finally, the application was developed in C# and the code was prepared in the programming console for the correct functioning of the project. In addition to the regular functioning of the cell, there is also a manual command that allows the actuation of valves and motors as well as monitoring of the status of introduced sensors.

Agradecimentos

A realização desta última etapa não teria sido possível sem a colaboração, auxílio e empenho demonstrado por várias pessoas ao longo do meu percurso universitário e pessoal. Desta forma, encontram-se nesta secção os agradecimentos aos que mais influenciaram e contribuíram, direta ou indiretamente, para a conclusão da mesma.

Assim sendo, agradeço em primeiro lugar à minha família, sobretudo à minha irmã, mãe e pai, que me apoiaram incondicionalmente durante todo o meu percurso académico e pessoal e ainda mais nesta última fase.

À Growskills Robotics por me ter aceite nas instalações para a realização da dissertação em contexto empresarial. Ao Eng. Rúben Almeida por me ter acompanhado ao longo do processo com a maior dedicação e auxiliando-me, incansavelmente, com a realização da dissertação. Aos Eng. António Fernandes, Eng. Luís Rodrigues e Eng. Miguel Oliveira pelo excelente ambiente providenciado nas instalações – ambiente esse que facilitou a conclusão da dissertação.

Ao Prof. Joaquim Gabriel Mendes pelo acompanhamento, envolvimento, transmissão de conhecimentos, dedicação e empenho demonstrado ao longo deste projeto.

Índice

Resumo	ii
Abstract	iii
Agradecimentos.....	iv
Índice	v
Índice de Figuras	viii
1 Introdução	1
1.1 Enquadramento	1
1.2 Objetivo	1
1.3 Metodologia	2
1.4 Organização da dissertação.....	4
2 Estado da Arte.....	5
2.1 Comparação entre robôs colaborativos e tradicionais	5
2.2 Programação de robôs colaborativos	9
2.3 Programação de robôs tradicionais	12
2.4 Impressão 3D	17
2.5 Células Robóticas Colaborativas	17
2.5.1. MPS Robot Assembly	18
2.5.2. UBI Factory	18
3 Apresentação detalhada da célula robótica.....	19
3.1 Elementos produzidos por impressão 3D	23
3.1.1. Elementos que constituem a embalagem e auxiliam a montagem da mesma	24
3.1.1.1. Base cúbica	24
3.1.1.2. Cilindro com rasgos	25
3.1.1.3. Tampa.....	25
3.1.1.4. Plataforma de montagem	26
3.1.1.5. Caixa de paletização	26
3.1.1.6. Armazém de cilindros	27
3.1.2. Elementos que constituem o alimentador	27
3.1.2.1. Base.....	27
3.1.2.2. Corpo	28
3.1.2.3. Atuador	30
3.1.2.4. Canal.....	30
3.1.2.5. Armazém.....	31
3.1.2.6. Suporte principal	32
3.1.2.7. Suporte secundário.....	32
3.1.3. Elementos do <i>Conveyor</i>	33
3.1.3.1. Conectores.....	33
3.1.3.2. Placa de Suporte do Encoder.....	34
3.1.3.3. Placa de Suporte do Motor	34
3.1.3.4. Caixa de proteção.....	34
3.1.4. Elementos da Rampa	34
3.1.4.1. Nivelador de Rampa	34

3.1.4.2.	Conectores.....	34
3.1.4.3.	Plataforma de recolha.....	34
3.1.4.4.	Apoios da plataforma.....	35
3.1.5.	Outros	35
3.1.5.1.	Suporte de câmara de vídeo	35
3.1.5.2.	Apoio para cabo.....	35
3.1.5.3.	Suporte para cabo	35
3.1.5.4.	Batente.....	35
3.1.5.5.	Dedos.....	35
3.2	Elementos Mecânicos	36
3.2.1.	Conveyor.....	36
3.2.2.	Rampa	37
3.3	Elementos Elétricos e Eletropneumáticos.....	38
3.3.1.	Elementos Elétricos	38
3.3.1.1.	Motor DC.....	38
3.3.1.2.	Encoder.....	39
3.3.1.3.	Detetor fotoelétrico difuso.....	40
3.3.1.4.	Sensor de barreira	40
3.3.1.5.	Interruptor diferencial.....	41
3.3.1.6.	Disjuntor	41
3.3.1.7.	Controlador de Motor	41
3.3.1.8.	Fonte de alimentação	41
3.3.2.	Elementos Eletropneumáticos	41
3.3.2.1.	Cilindro de duplo efeito com amortecimento	41
4	Implementação do sistema de visão em Python	45
4.1	Calibração da câmara	45
4.1.1.	Captação de imagem.....	46
4.1.2.	Deteção de um contorno amarelo	47
4.1.3.	Deteção do centro do contorno e respetivas coordenadas	47
4.1.4.	Envio da informação	47
4.2	Seleção do Cilindro	48
4.2.1.	Determinação das posições de todos os cilindros	48
4.2.1.1.	Distância entre centros	48
4.2.1.2.	Ângulo formado pelo segmento de reta que une os centros e o eixo XX.....	49
4.2.1.3.	Ângulo que determina os limites do cilindro no espaço	50
4.2.2.	Determinação do cilindro a ser selecionado.....	53
5	Interface em C#.....	57
5.1	Componentes da Interface	57
5.1.1.	Botão Iniciar Programa	57
5.1.2.	Botão Parar Programa.....	57
5.1.3.	Botão Calibração	58
5.1.4.	Botão Iniciar Ciclo	58
5.1.5.	Botão Comando Manual	59
5.1.6.	Botão Sair da Simulação	59

5.1.7. Caixa de Seleção 1.....	59
5.1.8. Caixa de Seleção 2.....	59
5.1.9. Caixa de Seleção 3.....	59
5.1.10. Caixa de Seleção 4.....	60
5.1.11. Sinalizador	60
5.1.12. Tabela de registo de ciclos	60
5.1.13. Caixa de Texto 1 – Total ciclos.....	60
5.1.14. Caixa de Texto 2 – Total Paletizações	60
5.1.15. Caixa de Texto 3 – Total Transportes	60
5.1.16. Lista de anomalias	60
5.2 Componentes da interface de Comando Manual.....	61
5.2.1. Botão Iniciar Comando Manual	61
5.2.2. Botão Sair do Comando Manual.....	61
5.2.3. Quadros de seleção.....	61
5.2.4. Sinalizadores de sensores.....	62
5.3 Excertos de algoritmos	62
5.3.1. Comunicação via Dashboard.....	62
5.3.2. Comunicação via Socket	63
5.3.3. Algoritmo geral	63
6 Programação na consola	65
6.1 Comunicação via Socket.....	65
6.2 Conveyor Tracking	66
6.3 Controlo de força.....	68
6.4 Rotinas Paralelas	70
6.5 Paletização	71
6.6 Alguns movimentos gerais	71
6.6.1. Movimento de Juntas (Movimento J).....	71
6.6.2. Movimento Linear (Movimento L)	72
7 Conclusões.....	74
7.1 Complexidade e Inovação da célula	74
7.2 Eficácia do Sistema de Visão.....	74
7.3 Dificuldades Experienciadas	74
7.4 Trabalhos Futuros	75
Referências	76
ANEXO A: Vistas 2D das peças concebidas por impressão 3D.....	78
ANEXOS B: Esquema Elétrico em EPLAN	111

Índice de Figuras

Figura 2.1 - Linha de robôs tradicionais	5
Figura 2.2 - Operador em proximidade com um cobot	5
Figura 2.3 – TCP de um elemento terminal	9
Figura 2.4 - Referenciais de um robô.....	10
Figura 2.5 - Método dos quatro pontos	11
Figura 2.6 - Eixo Z do TCP desalinhado com eixo Z base	11
Figura 2.7 - Simulação em RoboDK com um robô da KUKA	14
Figura 2.8 - Linha de montagem simulada em V-REP.....	15
Figura 2.9 - Consola de um robô industrial	16
Figura 2.10 - Célula robótica colaborativa	18
Figura 3.1 - Fluxograma de um ciclo	20
Figura 3.2 - Colocação de uma base cúbica no transportador	21
Figura 3.3 - Execução do Conveyor Tracking	21
Figura 3.4 - Encaixe da base na plataforma	21
Figura 3.5 - Encaixe de uma tampa na base	22
Figura 3.6 - Encaixe de cilindro na base	22
Figura 3.7 - Utilização do sistema de visão para seleção do cilindro	22
Figura 3.8 – Colocação da embalagem no transportador final	23
Figura 3.9 - Fim de ciclo	23
Figura 3.10 - Base Cúbica.....	24
Figura 3.11 - Cilindro com rasgos	25
Figura 3.12 - Tampa Amarela	25
Figura 3.13 - Plataforma de montagem	26
Figura 3.14 - Caixa de paletização.....	26
Figura 3.15 - Armazém de cilindros	27
Figura 3.16 - Base do alimentador	28
Figura 3.17 - Corpo do alimentador	29
Figura 3.18 - Atuador linear.....	30
Figura 3.19 – Canal do alimentador	31
Figura 3.20 - Armazém de bases cúbicas.....	31
Figura 3.21 - Suporte principal	32
Figura 3.22 - Suporte secundário.....	32
Figura 3.23 - Alimentador	33
Figura 3.24 - Conveyor.....	37
Figura 3.25 - Rampa	38
Figura 3.26 - Curva característica do motor DC	39
Figura 3.27 - Sinais do encoder em quadratura.....	40
Figura 3.28 - Esquema Pneumático.....	42

Figura 3.29 - Modelo da célula robótica.....	42
Figura 3.30 - Secção elétrica da estação.....	43
Figura 3.31 - Conveyer inicial e alimentadores.....	43
Figura 3.32 - Rampas + conveyer final	44
Figura 4.1 – Algoritmo de calibração	46
Figura 4.2 - Cálculo da distância entre centros.....	48
Figura 4.3 - Cálculo do ângulo alfa	49
Figura 4.4 - Conceção do ângulo alfa	49
Figura 4.5 - Definição do alfa real	50
Figura 4.6 - Definição do ângulo beta	50
Figura 4.7 - Remoção da diagonal principal	51
Figura 4.8 - Criação de um vetor com normas.....	52
Figura 4.9 - Ordenação das matrizes.....	53
Figura 4.10 - Esquema de seleção de cilindros	54
Figura 4.11 - Ajuste da garra.....	55
Figura 4.12 - Captura de vídeo da câmara	56
Figura 5.1 - Interface em C#	57
Figura 5.2 - Interface do Comando Manual	61
Figura 5.3 - Excerto referente à comunicação via Dashboard.....	62
Figura 5.4 - Excerto referente à comunicação via Socket	63
Figura 5.5 - Excerto de Calibração.....	64
Figura 6.1 - Calibração	65
Figura 6.2 - Setup do Conveyer Tracking	67
Figura 6.3 - Conveyer Tracking.....	68
Figura 6.4 - Setup de força unidirecional	69
Figura 6.5 - Setup de força bidirecional	69
Figura 6.6 - Algoritmo de força.....	70
Figura 6.7 - Rotina Paralela de Força	70
Figura 6.8 - Paletização	71
Figura 6.9 - Setup de Movimento J	72
Figura 6.10 - Setup de Movimento L	73

1 Introdução

1.1 Enquadramento

A robótica colaborativa tem vindo a desempenhar um papel cada vez mais notório na indústria. As suas aplicações vão desde soluções pouco ergonómicas – monótonas e repetitivas – até soluções em ambientes nocivos e perigosos em que nenhum ser humano poderia trabalhar. Além disso, a robótica colaborativa não se traduz no desapareço pela mão-de-obra uma vez que o robô colaborativo, ao contrário do robô tradicional, opera com o trabalhador e não em vez dele. Atualmente, a robótica colaborativa encontra-se em crescimento exponencial sendo cada vez mais procurada pela sua eficiência e pela flexibilidade.

A necessidade de desenvolver uma célula robótica para demonstração advém precisamente deste aumento de procura de soluções colaborativas que justifica a elaboração de soluções de formação que preparem técnicos e engenheiros para uma situação que cada vez mais se torna realidade num futuro próximo: introdução da robotização em todas as áreas do mercado. A presente tese, realizada na Growskills, permitiu o desenvolvimento de uma estação robótica de formação de técnicos de escolas profissionais.

1.2 Objetivo

Como todo e qualquer objeto de formação, é necessário que a célula robótica seja capaz de fornecer ao estudante uma vasta gama de conhecimentos e de aptidões que o preparem não só para programar robôs colaborativos, mas também para compreender algumas das suas principais aplicações. Além do robô, pretendia-se também incluir um pequeno sistema pneumático e elétrico de modo a evidenciar-se como a robótica colaborativa consegue ser incorporada com áreas transversais ao mundo industrial. Assim, o objetivo principal do trabalho foi o seguinte: o desenvolvimento de uma célula robótica colaborativa para demonstração – com intuito de formação – na qual se realizassem operações comumente verificadas em ambientes industriais (montagens, paletizações, transporte de peças, etc). Além disso, procurou-se enfatizar a integração do robô colaborativo com outros sistemas (pneumática, visão) de modo a se realçar a sua versatilidade e campo de aplicação.

Para se cumprir com estes objetivos o trabalho abrangeu as seguintes etapas:

- Uma etapa de pesquisa para avaliar quais os tipos de movimentos e de operações que deveriam ser integrados na solução bem como é que a célula deveria ser organizada, isto é, desenvolver um projeto com cabeça, tronco e membros.

- Incluir um sistema de visão e um interface que permitisse demonstrar a comunicação entre o sistema robótico e o sistema informático.

A primeira etapa passou por analisar quais as operações que deveriam ser incluídas e a relevância das mesmas. Além disso, procurou-se desenvolver uma célula sequencial e que pudesse representar um ciclo de trabalho numa fábrica ou instalação empresarial.

A segunda etapa abrangeu a procura de uma solução que fizesse justiça à complexidade da célula e que contribuísse para introduzir o formando – a quem a célula se destina – à cooperação entre sistemas informáticos e sistemas industriais. Este vínculo entre os dois tipos de sistemas torna-se cada vez mais forte e a linha que os separa cada vez mais ténue pelo que é necessário realçar que qualquer solução robótica acabará por incorporar sistemas informáticos – indispensáveis à automatização.

1.3 Metodologia

Para se cumprir com os objetivos definidos acima, foi necessário construir uma estação de raiz pelo que se analisou células já existentes de forma a se perceber de que forma poderia a presente superar as restantes em termos de complexidade e inovação. Assim, optou-se por incluir as seguintes operações na célula robótica:

- *Conveyor Tracking*

Esta operação consiste, como o nome diz, no seguimento de um tapete transportador. Para que esta operação possa ser realizada, é necessário um encoder – quer absoluto quer incremental – cuja informação é processada por funções internas da Universal Robots, permitindo que o robô acompanhe o movimento de uma peça ao longo do tapete. Esta operação pode ser efetuada tanto em tapetes circulares como em tapetes lineares. No presente caso, optou-se por incluir uma operações deste tipo no fim do ciclo. Tratando-se de uma operação característica dos robôs colaborativos, classificou-se como absolutamente necessária e fundamental a sua implementação.

- Controlo de força/binário

No caso dos robôs da Universal Robots o controlo de força é possível através de um sensor de força. Este controlo de força permite que o robô se mova ou rode numa determinada direção ainda que haja oposição, isto é, permite que o robô se movimente contra uma carga resistiva até o sensor de força registar um determinado valor. Este controlo de força foi utilizado para se encaixar peças cilíndricas em bases cúbicas. Como ambos possuem rasgos – permitindo o encaixe somente de uma forma – o robô roda a peça cilíndrica em torno do seu eixo vertical até

que os rasgos do cilindro coincidam com os da base. Simultaneamente, o cilindro é empurrado contra a base de modo a permitir o encaixe. Quando este ocorrer – e o robô continuar a rodar – o valor de binário disparará uma vez que encontrará resistência ao movimento. Através da monitorização do valor da força – neste caso até de binário – consegue-se inferir quando é que o encaixe se realizou.

- Paletização

A operação de paletização é uma das mais comuns na indústria robótica pelo que se considerou ser fundamental a sua inserção. Apesar do UR5 possuir um modelo de paletização – sendo necessário somente definir a dimensão da peça a paletizar, a sua altura e o número de linhas e colunas da paletização, este método provou-se incapaz de resolver o problema dado que nem todas as embalagens tinham a mesma altura. Assim, utilizou-se o controlo de força já que este pode também ser utilizado neste tipo de situações. Desta forma, fica provada a flexibilidade do robô colaborativo nesta operação.

- Visão

O sistema de visão tinha como objetivo aumentar a complexidade da célula enquanto que simultaneamente se apresentava a incorporação de vários tipos de sistemas numa única célula compacta. Uma vez que o futuro industrial passa cada vez mais pela comunicação e incorporação de várias famílias de componentes, considerou-se vantajoso integrar um sistema de visão que permitisse a identificação de cor e posição de cilindros. Este algoritmo de visão foi desenvolvido em Python por duas razões: a primeira advém do facto de que o sistema de visão incorporado opera segundo essa linguagem; a segunda traduz-se na vastíssima gama de operações que podem ser realizadas através da programação em Python.

- Interface em C#

O objetivo da interface consistia em e enriquecer a célula uma vez que se pretendia, desta forma, que esta fosse controlada via remotamente por um computador no qual se inserisse a informação necessária para o correto funcionamento da mesma. Esta interface foi desenvolvida na linguagem mais apropriada - C# - e serviu o propósito de relacionar o sistema robótico ao sistema informático.

Além das operações que constituem a célula, projetaram-se os restantes elementos que serão abordados no capítulo 3.

1.4 Organização da dissertação

Termina-se este capítulo introdutório enunciando-se como é que a tese está organizada ao longo das restantes páginas.

Seguidamente a este capítulo, encontram-se os seguintes:

- O capítulo 2 – Estado da Arte – onde se comparam robôs colaborativos a tradicionais enfatizando-se as suas diferenças quer a nível de aplicação, flexibilidade, dificuldade de programação, etc. Além disso, apresenta-se os tipos de programação de robôs evidenciando-se em que casos é que se utiliza cada tipo de programação. Por fim, apresenta-se uma breve introdução acerca da impressão 3D.
- O capítulo 3 – Apresentação detalhada da célula robótica – no qual se demonstra todos os elementos incorporados na estação. Nestes elementos, destacam-se os que foram desenvolvidos por impressão 3D bem como os que foram encomendados (mecânicos, pneumáticos, etc) para que a célula funcionasse corretamente.
- O capítulo 4 – Implementação do sistema de visão em Python – é unicamente dedicado à análise do sistema de visão. Avalia o algoritmo por detrás da visão explicando-se o que se procurou fazer e como se implementou.
- O capítulo 5 – Interface em C# - apresenta a interface desenvolvida salientando-se que funções é que são executadas e como é que esta comunica com o robô.
- O capítulo 6 – Programação na consola – apresenta alguns excertos de código destacando-se os trechos relativos às principais operações realizadas. Além disso, procura-se introduzir o leitor a este tipo de programação.
- O capítulo 7 – Conclusões – avaliação do cumprimento dos requisitos e estabelecendo o que poderia ser implementado em trabalhos futuros.

2 Estado da Arte

Este capítulo é dedicado ao estudo teórico dos robôs colaborativos destacando-se o seu campo de aplicação e as características que os tornam colaborativos. Além disso, ao longo deste capítulo, encontrar-se-ão diferenças entre os dois tipos de robôs (tradicionais e colaborativos), bem como os métodos de programação utilizados em cada um. Tudo isto tem como efeito esclarecer o leitor face à divisória de robôs existentes no mercado bem como identificar as razões que determinam a crescente utilização de robótica colaborativa. Por fim, efetuar-se-à uma introdução à impressão 3D já que esta foi recorrente ao longo do projeto.

2.1 Comparação entre robôs colaborativos e tradicionais

Sendo o foco da dissertação o desenvolvimento de uma célula robótica utilizando um robô colaborativo, é necessário, primeiramente, saber distingui-lo do robô tradicional.

Uma forma simples de os distinguir é pela seguinte frase:

Um robô colaborativo é concebido para trabalhar com o operador enquanto que o robô tradicional é concebido para trabalhar em vez do operador.



Figura 2.1 - Linha de robôs tradicionais

VS



Figura 2.2 - Operador em proximidade com um cobot

Entre as funções desempenhadas pelos robôs – tanto colaborativos como tradicionais, encontram-se as tarefas pouco ergonómicas e até potencialmente perigosas. Apesar do robô colaborativo ser concebido para operar junto do ser humano, isto não significa que ele tenha de ser implementado, necessariamente, em soluções colaborativas. Este pode ser introduzido em aplicações tradicionais ainda que se verifique uma menor produtividade – reduzida em 2/3 – devido a menor velocidade de operação e menor capacidade de carga – ambas devido ao facto de ser colaborativo. Ou seja, um robô para ser colaborativo tem de, necessariamente, ser capaz

de trabalhar em conjunto com um ser humano. Contudo, isto não implica que ele não possa ser utilizado em soluções tradicionais. No caso de desempenharem a mesma função e serem do mesmo tamanho, o robô colaborativo mantém-se uma solução mais barata uma vez que os motores e mecanismos de transmissão de movimento implementados são menos exigentes.

Como os robôs colaborativos são desenhados para operarem junto do ser humano, não são isolados em jaulas, como é o caso dos robôs tradicionais. Apesar disso, isto não significa que sejam 100% seguros. No entanto, são portadores de equipamentos de segurança (fronteiras virtuais, sensores, etc) que detetam a presença de objetos no seu caminho de trabalho (outros robôs, operários, etc) facilitando os protocolos de segurança e diminuindo o risco de colisões e, consequentemente, lesões. Além disso, possuem sensores de força que forçam a paragem do robô caso este, não tendo sido capaz de detetar a presença de um objeto no seu espaço de trabalho, entre em contacto com o operador ou qualquer outro obstáculo. (Robotiq, “Collaborative Robots in Global Companies”).

Pode-se comparar os robôs tradicionais com os colaborativos, nas diversas vertentes e características, a partir da seguinte tabela:

Caraterística	Robô Tradicional	Robô Colaborativo
Segurança	Não são seguros; têm de estar cercados por jaulas a fim de proteger os operários e o ambiente empresarial.	São desenhados para serem seguros podendo trabalhar junto de outros robôs ou equipamentos.
Proximidade c/ operadores	Não é permitida.	Trabalham na maioria das vezes próximos de operadores e/ou em cooperação.
Dificuldade de programação	Elevada; apenas programadores experientes têm competência para os programar.	Muito reduzida; programadores inexperientes são capazes de o programar.
Capacidade de carga	Elevada (acima dos 20 kg).	Reduzida (abaixo dos 20 kg).
Alcance	Elevado (acima dos 1300 mm).	Máximo de 1300 mm.
Versatilidade	Reduzida.	Elevada.
Adaptabilidade/Flexibilidade	Reduzida.	Elevada.

Custo	Elevado; o retorno do investimento demora alguns anos.	Reduzido; o retorno do investimento demora, normalmente, um ano.
Espaço de trabalho	Elevado.	Limitado.
Espaço de alocação	Elevado.	Reduzido.
Volume de produção	Elevado	Limitado
Mobilidade	Reduzida; normalmente o robô não é movido após instalação.	Elevada; o cobot pode ser facilmente movido para outro local de trabalho.
Tempo de instalação	Elevado; (algumas semanas com operadores especializados).	Reduzido; poucas horas e não requer operadores especializados para tal tarefa.
Integração com outros robôs/máquinas	Fácil	Fácil
Velocidade de operação	Elevada	Limitada
Intervenção humana	Reduzida; apenas em caso de avaria	Elevada

(Robotiq, “Getting Started with Collaborative Robots”)

(Campbell, 2020)

(Cobot Trend Staff)

(Nota: as três referências foram utilizadas para a construção da tabela)

Observando a tabela, existem alguns aspectos fundamentais que destacam os robôs colaborativos dos tradicionais, nomeadamente:

- a programação;
- o tempo de instalação;
- adaptabilidade/flexibilidade.

Estas três características são, além da própria aparência do robô, as que melhor os distinguem.

Relativamente ao primeiro tópico - a dificuldade de programação - existe uma diferença abismal entre os dois robôs. No caso do robô tradicional é necessário, antes de testar em tempo real, de simular o comportamento do robô num *software* de simulação (existem alguns robôs que permitem guiamento direto, mas o *modus operandi* consiste em elaborar um programa offline tanto por questões de segurança como por questões práticas). Este processo de

programação offline é vagaroso e exige elevados conhecimentos e experiência de programação robótica bem como de linguagens de programação. Mesmo com a simulação em ambiente virtual é necessário cautela extraordinária já que existem fatores externos que influenciam o comportamento do robô e que, por razões naturais, são impossíveis de ser tidos em conta no ambiente virtual (Mccrea, 2018). No caso do robô colaborativo a situação é muito simplificada. A programação em si é, muitas vezes, efetuada em modo online por guiamento direto. Neste tipo de programação, o operador movimenta o robô para as posições desejadas, definindo um determinado trajeto que é memorizado pelo robô. Desta forma, torna-se trivial programar um cobot. Ademais, não é necessário um programador experiente. (Abreu, 2020)

Relativamente ao tempo de instalação, os robôs tradicionais apenas podem ser instalados por operadores muito qualificados e, mesmo assim, a instalação pode demorar várias semanas – em função do tamanho do robô. Este elevado tempo de instalação em conjunto com o tempo de programação torna difícil a utilização imediata do robô aquando do fornecimento. Os cobots têm um tempo de instalação muito reduzido. As pequenas dimensões e reduzido número de componentes – relativamente aos robôs tradicionais – permite que a sua instalação seja notavelmente mais eficiente.

Outro ponto chave é a flexibilidade/adaptabilidade. O robô tradicional é desenhado para trabalhar num determinado ambiente, sob condições pré-determinadas, isto é, o ambiente é que se adapta ao robô e não o contrário. Desta forma, a sua adaptabilidade é muito reduzida. Além disso, tende a efetuar sempre o mesmo conjunto de operações sobre o mesmo tipo de peças. O cobot é desenhado para operar em vários ambientes adaptando-se, dentro das suas capacidades, ao mesmo. Além disso, um único cobot é capaz de efetuar várias tarefas. Assim, a sua existência em si permite a sua adaptação a diversos ambientes e operações. No caso dos robôs tradicionais isto é raríssimo pelo facto de não serem movidos do local onde foram primeiramente instalados. (Cobot Trend Staff)

Além das referidas, existe uma razão para o mercado dos cobots estar a crescer. São uma etapa importante para pequenas e médias empresas seguirem o rumo da automatização. Ainda hoje, são maioritariamente utilizados em pequenas empresas que, dadas as exigências do mercado, procuram a automatização dos seus processos e, por outro lado, não possuem ainda o investimento para robôs tradicionais. Para estas, os robôs colaborativos constituem uma rampa de lançamento para a robotização. São um excelente método para um crescimento rápido de produção e competitividade de mercado. Num mercado cada vez mais competitivo, a robótica colaborativa proporciona a pequenas-médias empresas a oportunidade de lutar por um lugar (Robots, U. “Get Started with Cobots”).

2.2 Programação de robôs colaborativos

Mencionou-se anteriormente que os robôs colaborativos possuem um método muito simples de programação: programação online por guiamento direto. Este tipo de programação é tão simples que qualquer operador, sem qualquer experiência de programação, consegue elaborar um programa e testá-lo. O guiamento direto consiste em mover o robô para uma determinado ponto do espaço que é ensinado ao robô, isto é, a posição e orientação segundo os três eixos é guardada na memória. Existem alguns procedimentos que são necessários ter em conta quando se pretende simular um programa num robô colaborativo. São eles:

- Definição do TCP (Tool Center Point)
- Definição do Payload e centro de gravidade¹

O TCP é o ponto intermédio dos pontos de contacto do elemento terminal do robô. A figura 2.3 evidencia o TCP para um determinado elemento terminal.

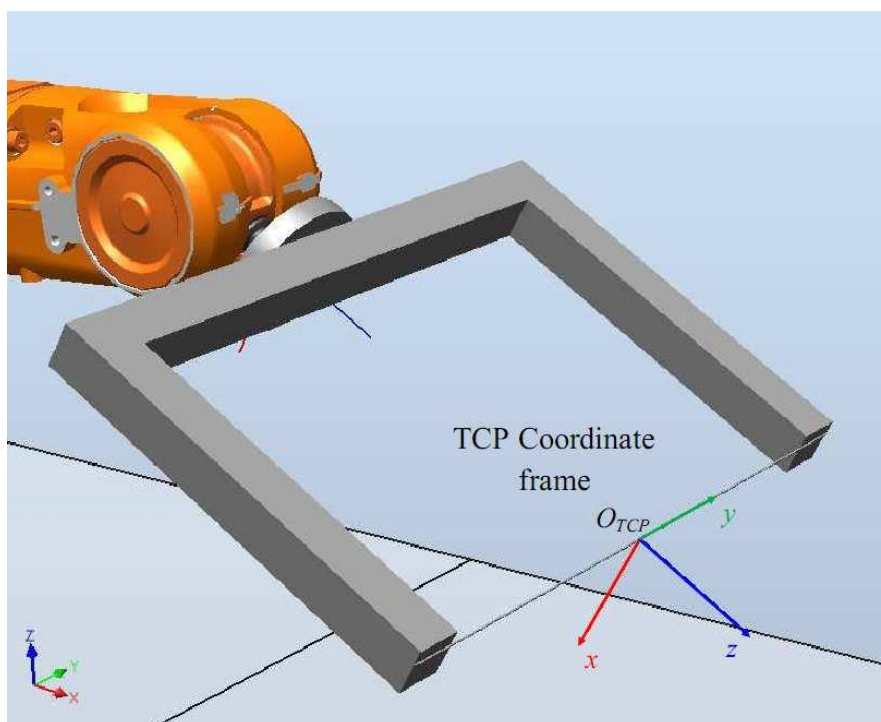


Figura 2.3 - TCP de um elemento terminal

O TCP é definido relativamente ao referencial ferramenta e nunca em relação ao referencial base. O referencial ferramenta possui a sua origem na flange do robô e relaciona-se com o referencial base mostrado na figura 2.4. (Abreu, 2020)

¹ Os robôs tradicionais não necessitam desta definição

Coordinate systems

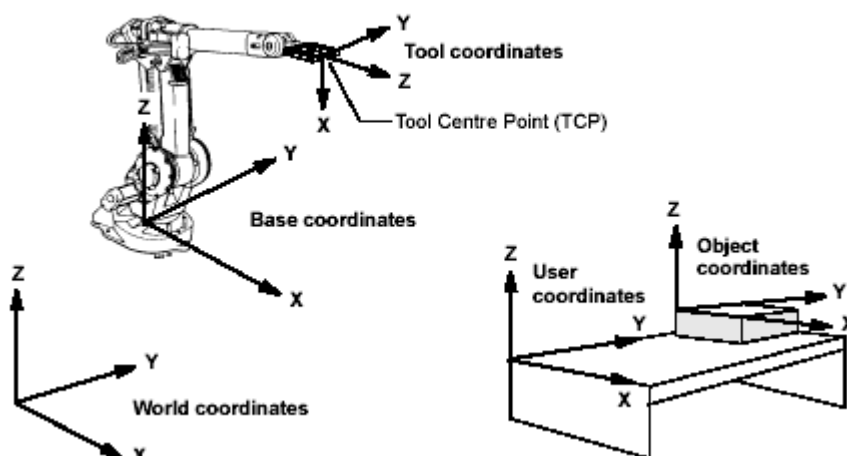


Figura 2.4 - Referenciais de um robô

Existem dois métodos para a definição do TCP. O primeiro método é o mais simples e consiste numa medição direta do TCP. Note-se que este método é empírico e pode, por vezes, conduzir a resultados incertos, sobretudo quando este possui uma componente em mais de um eixo.

O segundo método é o “método dos quatro pontos”. Este método consiste em posicionar o elemento terminal do robô quatro vezes num determinado ponto desde que a orientação das juntas tenha sido alterada. Após cada ponto ser ensinado, o sistema interno do robô automaticamente calcula o TCP. Para que o cálculo seja bem efetuado é necessário que haja uma variação significativa da orientação das juntas do robô. Caso tal não aconteça, o TCP não será calculado. No entanto, este procedimento não é infalível e, por isso, deve-se ser crítico face aos resultados fornecidos pelo robô. Caso se detete que o TCP está mal definido deve-se optar pela medição direta do mesmo ou a uma repetição do processo (Abreu, 2020). A figura 2.5 revela a configuração dos quatro pontos²:

² A ponta da ferramenta incide sempre no mesmo ponto apenas alterando-se a orientação das juntas.

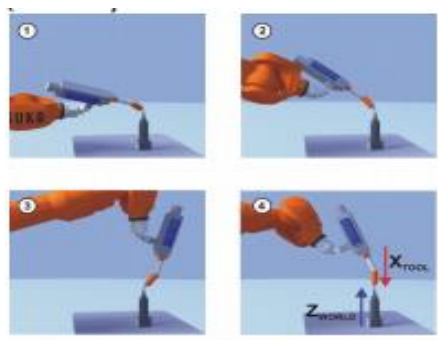


Figura 2.5 - Método dos quatro pontos

Para se definir o Payload e o centro de gravidade da ferramenta basta movimentar o robô para quatro pontos diferentes. Neste caso, quanto mais diversas forem as posições melhor calibrado o sistema ficará.

Além destes três parâmetros, pode ser necessário definir a orientação do TCP relativamente ao referencial base. Para tal, deve-se alinhar o eixo Z+ do TCP com o do referencial base. A figura 2.6 apresenta uma situação em que tal é necessário antes de começar a programação.

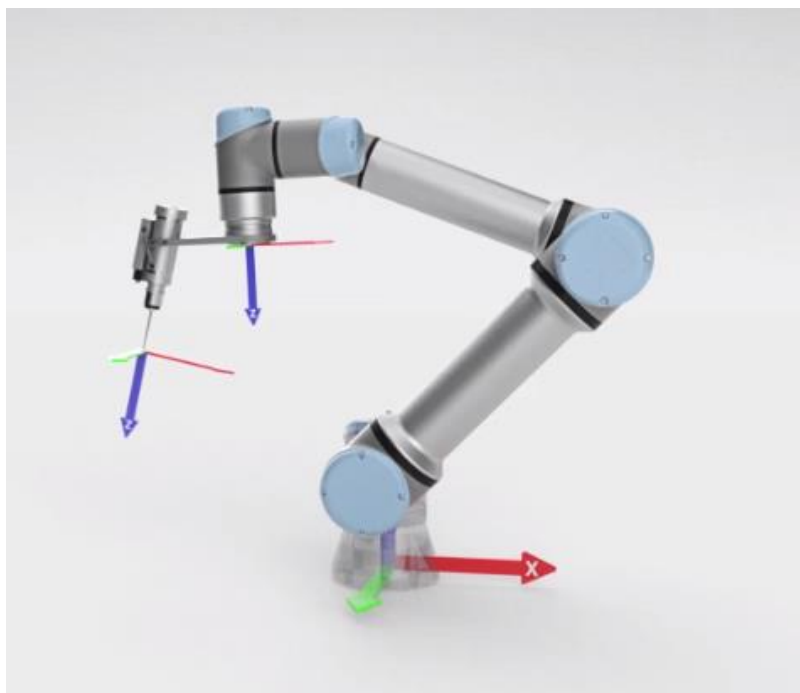


Figura 2.6 - Eixo Z do TCP desalinhado com eixo Z base

Por outras palavras, tal procedimento é necessário quando o eixo Z do TCP não é paralelo ao referencial Z da base.

Definidos estes parâmetros, pode-se iniciar a programação do robô. A grande vantagem do guiamento direto é a facilidade com que se movimenta o robô de uma posição para outra bem como a otimização de tempo, já que movimentar o robô unicamente pela consola é, em

geral, mais demorado. Poderá, contudo, ser necessário quando se pretende colocar o robô numa posição específica com muito rigor. Normalmente, o guiamento direto e por consola são conciliados de modo a diminuírem-se os tempos de programação. O operador coloca manualmente o robô numa posição pretendida e seguidamente, recorrendo à consola, ajusta a mesma aumentando o rigor e a precisão.

O guiamento direto oferece uma vantagem posicional mas não programacional, isto é, quer se programe utilizando a consola ou não, o algoritmo de programação é o mesmo.

2.3 Programação de robôs tradicionais

Relativamente aos robôs tradicionais, a tarefa de programação é mais fatigante e propensa a erros.

Existem dois métodos de programação:

- Programação offline;
- Programação por consola de programação.

A programação offline permite a programação do robô enquanto este está a trabalhar pelo que a perda de tempo útil de produção é muito reduzida.

A programação offline ocorre em simuladores robóticos e é comumente utilizada quando se programam robôs tradicionais embora também possa ser utilizada para robôs colaborativos. Este método de programação consiste em modelizar o robô, o ambiente de trabalho e as peças num ambiente virtual. Utiliza uma linguagem de programação própria do sistema ou, em alguns casos, idêntica à do robô. Para se programar em modo offline é necessário cumprir com várias etapas, nomeadamente as seguintes:

1. Criação de um modelo 3D da célula robotizada
 - a. Importação de modelos geométricos de um sistema CAD e/ou criação do ambiente por utilização das facilidades de CAD inerentes ao programa de simulação.
2. Calibração do modelo simulado
 - a. Calibração do TCP e de equipamentos auxiliares utilizados na célula (peças, gabaritos, sistemas de posicionamento, etc...).
3. Criação dos programas de operação do robô e equipamento periférico utilizando a linguagem nativa do programa de simulação
4. Verificação de colisões, deteção de potenciais colisões, limites de juntas e velocidades bem como análise dos tempos de ciclo

5. Envio dos programas criados para o controlador do robô e da célula (Abreu, 2020).

Em qualquer tipo de programação de um robô tradicional surge sempre a mesma adversidade: o ambiente em que este se encontra. É impossível saber, a 100%, como é o que ambiente se vai comportar em tempo real. Isto é, alterações de temperatura, humidade, etc são impossíveis de colmatar uma vez que são imprevisíveis. Esta é a principal desvantagem dos *softwares* de simulação 3D que apenas conseguem simular o ambiente natural (alterações de temperatura e humidade) com sinais elétricos. (Robotic, S. “Best Robotics Simulation Software”).

Apesar disso, a simulação robótica possui várias vantagens, bem como desvantagens. Destacam-se as seguintes:

Vantagens:

- Poupança de tempo de programação e de iterações;
- Teste de código em ambiente seguro;
- Facilidade de ajustes;
- Simulação de ambientes complexos sem custos acrescidos;
- Verificação da compatibilidade do sistema com uma certa linguagem de programação;
- Determinação da viabilidade de um sistema.

(Buskirk, “An Overview of Robotic Simulation Software”)

Desvantagens:

- Simulações 3D requerem um computador com elevada performance;
- Sensores e outros elementos elétricos podem responder de forma diferente no simulador;
- Simuladores apenas aproximam a complexidade de um ambiente em tempo real.

(Robotics, 2012)

(Robotic, S. “Best Robotics Simulation Software”)

Existem vários tipos de simuladores dos quais se podem destacar os seguintes:

- RoboDK
- CoppeliaSim
- RobotStudio
- Visual Components

- RoboLogix
- RoboGuide
- Workspace
- Etc...

A atenção debate-se sobre os dois primeiros já que são aqueles que foram mais explorados – ainda que levemente – durante a realização da tese.

O RoboDK é uma ferramenta de programação offline para robôs tradicionais que utiliza a linguagem Python. Todos os programas são convertidos para a linguagem específica do robô antes de serem enviados para o físico. (Robotics, S. “Best Robotics Simulation Software”)

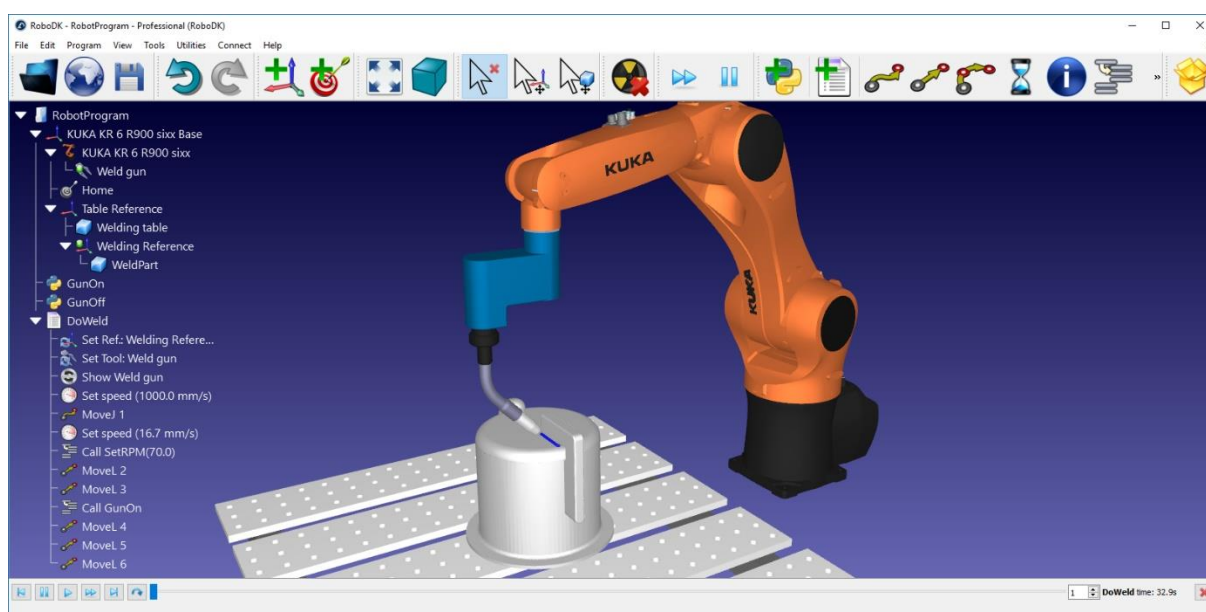


Figura 2.7 - Simulação em RoboDK com um robô da KUKA

A biblioteca do software oferece modelos 3D para cerca de 200 robôs tradicionais bem como ferramentas da ABB, KUKA e Yaskawa entre muitas outras. O RoboDK possui várias características capazes de gerar alertas perante certas condições ou detectar possíveis colisões.

Além disso, permite a realização de testes de precisão e repetibilidade que são de maior interesse para a empresa. É um *software* útil para demonstrações até mesmo em contextos educacionais. (Robotics, 2012)

O CoppeliaSim é um simulador robótico grátis para objetivos educacionais ou pago caso se pretenda utilizar a versão profissional para projetos comerciais. Pode-se “correr” o V-REP em Windows, Linux ou macOS e pode ser programado em seis linguagens diferentes de programação.

O CoppeliaSim suporta um sistema de simulação de ar e água para se poder apropriadamente modelar motores. Também permite simular uma larga gama de sensores, disponíveis na biblioteca, sendo que ainda se pode adicionar novos sensores via plug-in. (Buskirk, “An Overview of Robotic Simulation Software”)

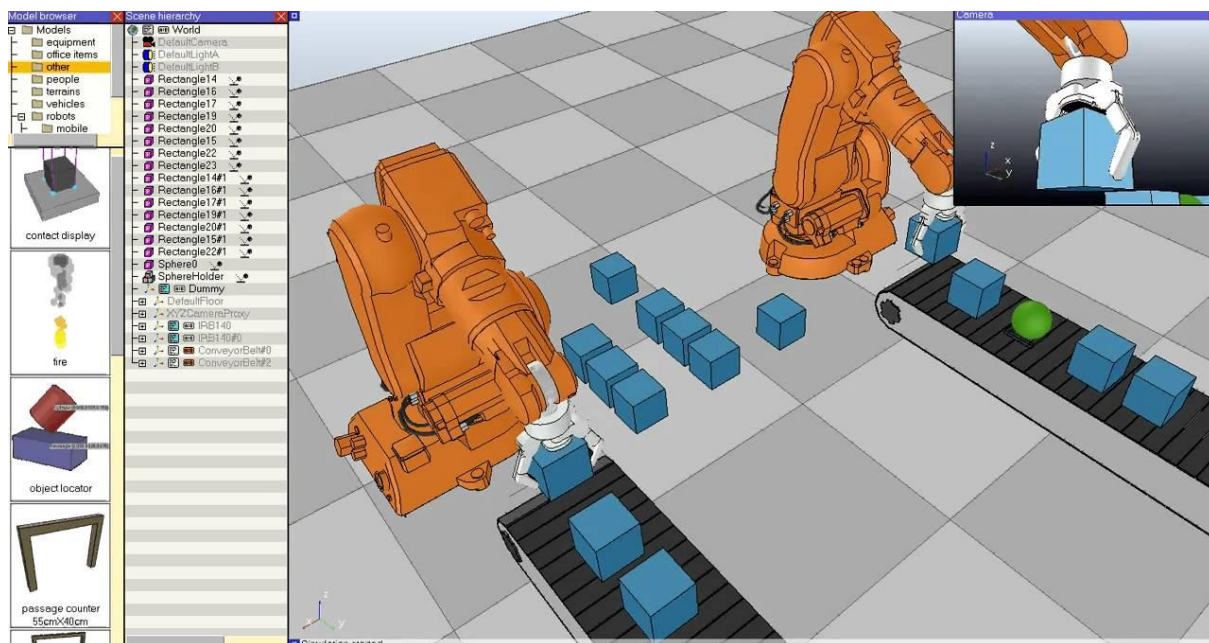


Figura 2.8 - Linha de montagem simulada em V-REP

De seguida, apresentam-se mais algumas funcionalidades disponíveis em *software* de simulação:

- Cálculo e análise de tempos de ciclo
- Calibração de robôs e células de trabalho
- Otimização da programação do robô e célula
- Interfaces diretas com sistemas de CAD
- Detecção automática de colisões
- Geração automática da cinemática de robôs
- Simulação da cinemática e dinâmica de robôs

A programação pela consola de programação é um tipo de programação online que requer a movimentação do robô de forma ativa. O robô é movimentado para os pontos de interesse, sendo então memorizados. O processo de programação é depois complementado por uma programação textual onde se define o tipo de movimento a utilizar, o tipo de interpolação entre os pontos ensinados, as velocidades pretendidas e toda a interação necessária para acomodar a realização das tarefas a programar. (Owen-Hill, 2012)



Figura 2.9 - Consola de um robô industrial

Possui as seguintes vantagens e limitações:

Vantagens:

- Permite um posicionamento preciso já que o robô pode ser programado utilizando coordenadas numéricas, quer seja no referencial mundo, robot ou noutro qualquer;
- A maioria dos robô tradicionais possui uma consola facilitando-se a tarefa de programação (os operadores estão habituados).

Limitações:

- É um processo mais complicado que a programação em consola dos robôs colaborativos;
- Requer mais tempo de programação;
- Enquanto o robô está a ser programado não pode estar a executar outras tarefas.

Embora os métodos principais de programação em robôs tradicionais sejam a simulação em *software* e a programação por consola, existem alguns que permitem o guiamento direto como opção de programação. Normalmente, esta alternativa de programação encontra-se disponível em robôs dedicados a pinturas, soldaduras, colagens entre outras operações da mesma família. Por exemplo, algumas marcas de robôs tradicionais já apresentam programação por guiamento direto:

- KUKA
- ABB
- Fanuc

2.4 Impressão 3D

A impressão 3D é um processo de fabrico a partir do qual se originam componentes através de sucessivas deposições de material. É, por isso, considerado um processo de fabrico aditivo. Aliando-se a existência de uma impressora 3D na Growskills com a necessidade de desenvolver várias peças que constituíssem uma célula operacional, optou-se por utilizar este processo de fabrico para realçar a importância do mesmo e flexibilidade bem como para facilitar a alteração das geometrias de componentes ao longo do projeto.

Os materiais utilizados para impressão são, principalmente, termoplásticos, nomeadamente PLA, PETG e ABS. Neste projeto, todos os componentes obtidos por impressão 3D foram fabricados em PLA. As temperaturas de impressão variam, salvo rara exceção, entre 180-220°C sendo normalmente efetuado uma “Torre de Temperaturas” a fim de se determinar qual a melhor temperatura de impressão. Frequentemente, peças mais complexas e com maiores exigências dimensionais requerem ser impressas numa base a uma temperatura mais elevada de modo a se impedir encurvamentos. Esta temperatura não é a de impressão – isto é – a do extrusor, mas sim a da plataforma onde a peça será impressa.

Os passos para obtenção de um componente por impressão 3D são os seguintes:

- Desenvolvimento de um modelo 3D do componente;
- Utilização de um *software* próprio (designado como *slicer* devido à desconstrução de uma peça em sucessivas camadas) de impressão 3D para geração do código G;
- Importação do código G para a impressora (via cartão de memória).

2.5 Células Robóticas Colaborativas

De entre as células disponíveis no mercado, encontram-se algumas características que foram inseridas na desenvolvida, nomeadamente operações de montagem, paletização, transporte, etc

A figura 2.10 apresenta uma célula robótica colaborativa com elementos pneumáticos e elétricos incorporados.

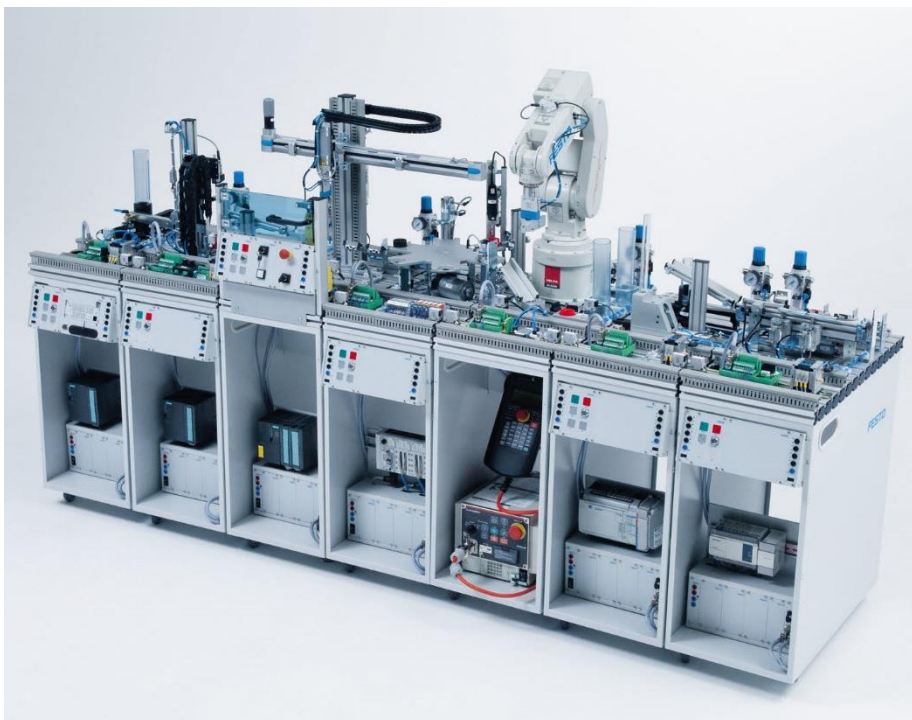


Figura 2.10 - Célula robótica colaborativa

Seguidamente, apresenta-se algumas células robóticas que foram analisadas para se poder desenvolver a presente.

2.5.1. MPS Robot Assembly

É uma célula robótica da Festo Didactic que permite a montagem de tampas em peças cilíndricas. Além disso, permite a interpretação da orientação da peça, permitindo a sua correta alocação para montagem. É uma célula auxiliada com um sistema pneumático e com sensorização.

2.5.2. UBI Factory

Trata-se de uma estação na qual se encaixam pequenos cilindros numa base cúbica. Seguidamente, coloca-se uma tampa sobre o conjunto que é selada via vácuo. Ademais, possui ainda um armazém automático para guardar as peças. A célula possui ainda um PLC Siemens S7/1200.

Como as células estão patenteadas, apenas se possui um pequeno vídeo de demonstração de cada uma – estando ambos na secção das Referências.

3 Apresentação detalhada da célula robótica

Neste capítulo, apresentar-se-à o desenvolvimento do projeto da célula robótica, desde os que foram concebidos via impressão 3D até aos que foram facultados por distribuidoras. Também neste capítulo se reforça a necessidade de estabelecimento de um problema inicial em torno do qual toda a projeção da estação girou. Este capítulo está então dividido em duas partes: a primeira consiste na formalização da estação robótica, isto é, na sequência de operações a realizar bem como nas várias alternativas possíveis de modo a torná-la flexível; a segunda parte constitui o grosso do capítulo e corresponde à apresentação de todos os elementos que fazem parte da célula.

Quanto à estação robótica, optou-se pela seguinte configuração:

- Dois alimentadores que colocam uma base cúbica a circula no *conveyor* inicial. Cada alimentador possui até quatro peças de uma determinada cor. Após esta base ser emitida para o primeiro tapete transportador, inicia-se a operação de *Conveyor Tracking*. Ao se atingir um sensor de fim-de-curso a base é transportada para uma pequena plataforma onde se irá montar o cilindro na mesma. Ainda antes desta primeira fase, recorre-se à calibração do sistema de visão. Seguidamente, o robô encaixará um cilindro de uma determinada cor na base. Segue-se a montagem de uma tampa constituindo, desta forma, a embalagem final. Por último, esta embalagem pode ser paletizada ou colocada noutro tapete transportador sobre o qual se processa outro *Conveyor Tracking*. O fluxograma da figura 3.1, esclarece alguma dúvida que possa existir face ao funcionamento do sistema.

A configuração representada no fluxograma é a de um ciclo normal. A embalagem não tem, necessariamente, de ser constituída pelos três elementos base + cilindro + tampa. Para que o sistema opere corretamente, tem de ser constituído forçosamente por uma base, vindo os restantes elementos por acréscimo. O mesmo acontece para o destino da embalagem, podendo ser paletizada, transportada ou nenhuma das anteriores. Optou-se por desenvolver a célula desta forma de modo a que apresentasse alguma flexibilidade deixando-se ao critério do utilizador decidir a composição de cada embalagem e os passos que cada ciclo efetuará.

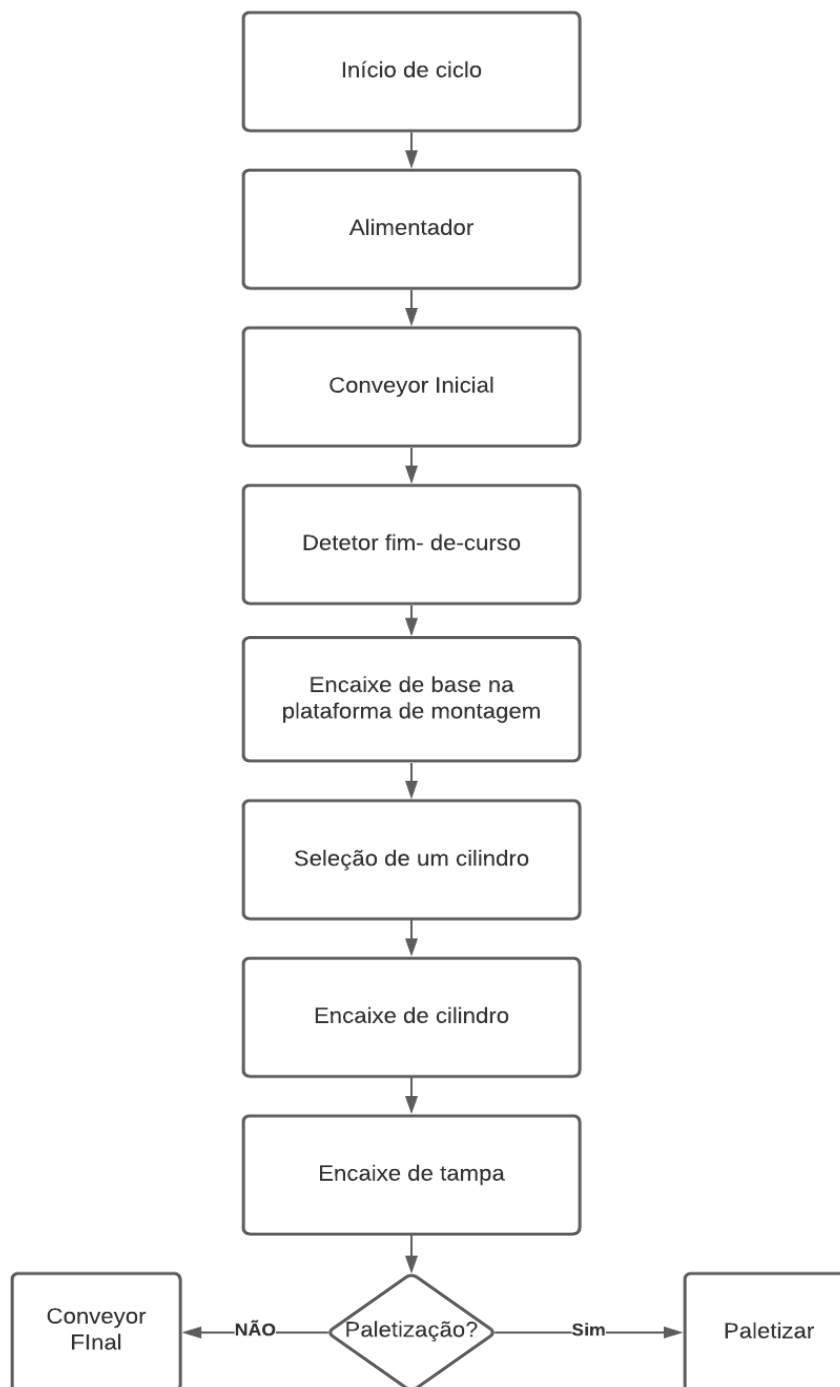


Figura 3.1 - Fluxograma de um ciclo

Seguidamente, serão apresentados todos os elementos que constituem a célula robótica, constituindo-se uma lista de material que foi incorporado. Como se tratam de vários constituintes, muitos deles semelhantes, e de modo a que o capítulo não se tornasse demasiado maçudo e de difícil leitura, dividiu-se o capítulo em três sub-capítulos: elementos produzidos por impressão 3D, elementos mecânicos e elementos elétricos e eletropneumáticos.

Ainda nas seguintes figuras se torna possível identificar um ciclo do robô:



Figura 3.2 - Colocação de uma base cúbica no transportador



Figura 3.3 - Execução do Conveyor Tracking

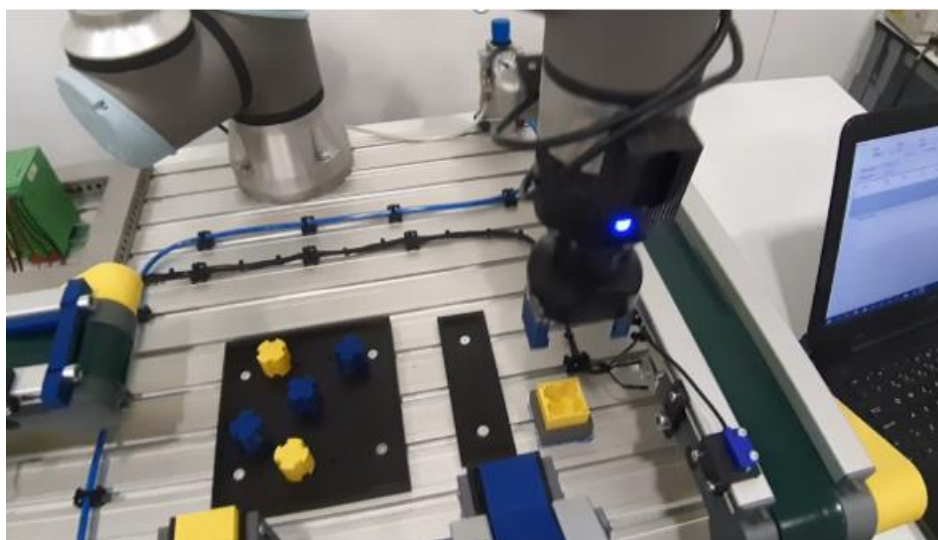


Figura 3.4 - Encaixe da base na plataforma



Figura 3.5 - Utilização do sistema de visão para seleção do cilindro

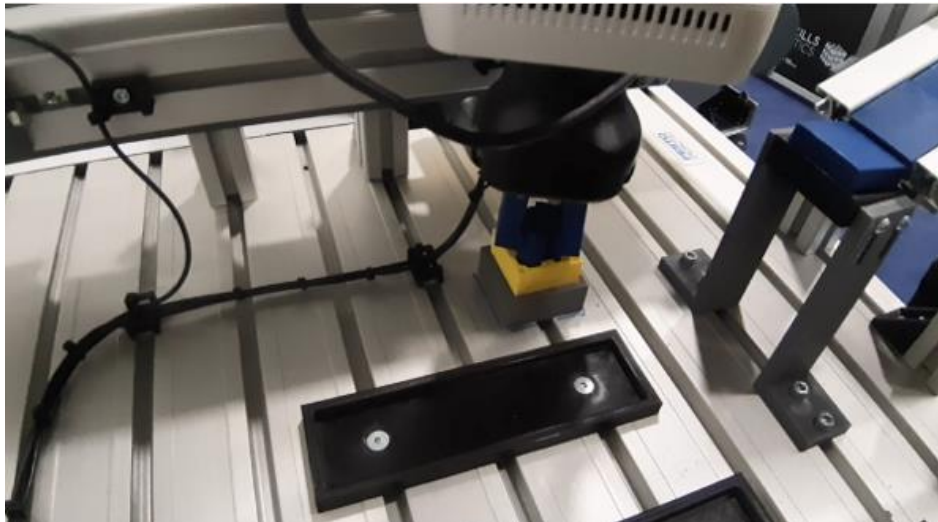


Figura 3.5 - Encaixe de cilindro na base



Figura 3.7 - Encaixe de uma tampa na base

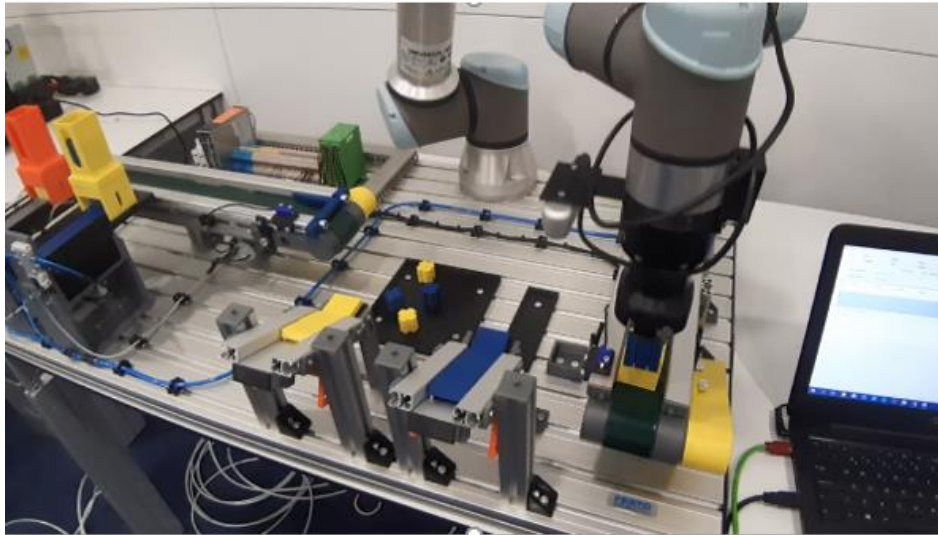


Figura 3.8 - Colocação da embalagem no transportador final

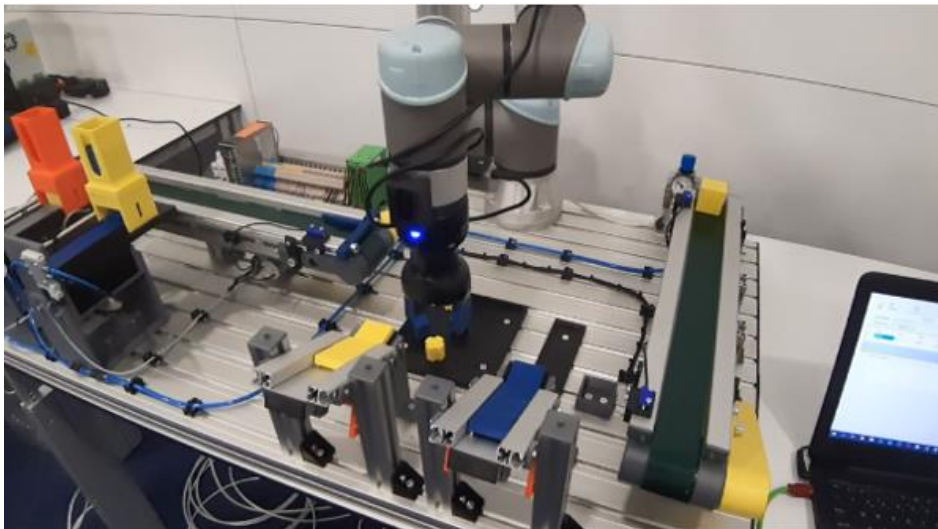


Figura 3.9 - Fim de ciclo

3.1 Elementos produzidos por impressão 3D

Nesta primeira secção, apresentam-se os elementos que foram desenvolvidos por impressão 3D. Qualquer uma destas peças poderia ter sido projetada noutro material qualquer que a sua funcionalidade manter-se-ia. Contudo, devido à facilidade de produção, reduzido custo e às sucessivas alterações tanto de dimensão como de geometria das peças, optou-se por este processo de fabrico.

Ainda nestes elementos, pode-se proceder uma visão tripartida:

- Elementos que constituem a embalagem e que auxiliam a montagem da mesma

- Elementos que constituem o alimentador
- Elementos de ligação, auxiliares, etc

Inicia-se a análise dos primeiros:

3.1.1. Elementos que constituem a embalagem e auxiliam a montagem da mesma

3.1.1.1. Base cúbica

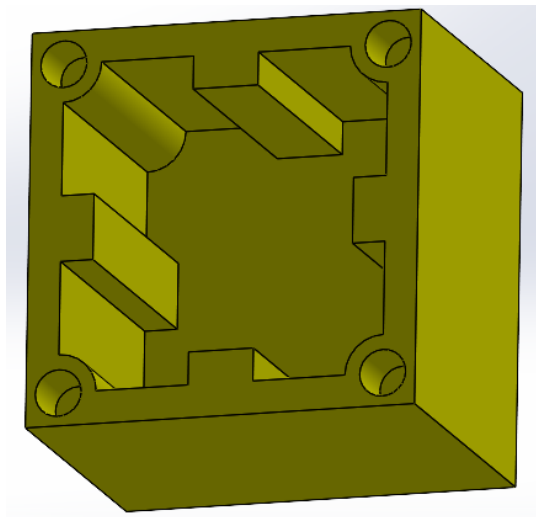


Figura 3.6 - Base Cúbica

A base cúbica é o primeiro constituinte de uma embalagem completa – cujos elementos incluem também o cilindro com rasgos e a tampa. Esta base é a peça que circulará no *conveyor*. A definição de uma base cúbica e não, por exemplo, paralelepípedica prendeu-se por motivos estéticos e funcionais uma vez que uma base cúbica é mais facilmente manipulada que uma base paralelepípedica.

Quanto às dimensões da mesma, teve de se garantir que não eram superiores ao alcance da garra Robotiq Hand-E (50 mm). Tratando-se de uma pequena peça produzida por impressão 3D, a sua massa (12 g) é desprezável.

Existem, ao todo, oito bases cúbicas: quatro azuis e quatro amarelas rigorosamente iguais entre si. Cada conjunto de quatro bases iguais é alocado no respetivo alimentador. A existência de alimentadores geometricamente iguais ainda que com cores diferentes, permite saber em qual é que se deve inserir cada base.

3.1.1.2. Cilindro com rasgos

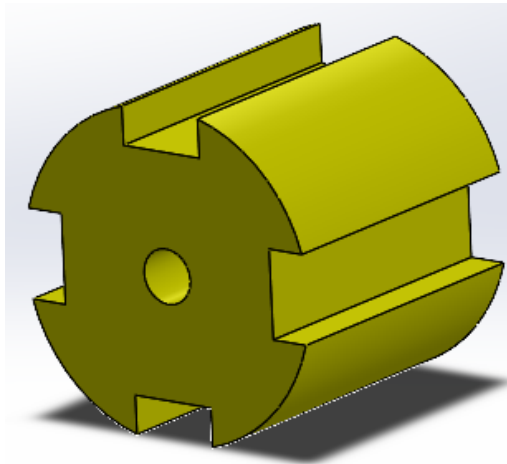


Figura 3.7 - Cilindro com rasgos

O cilindro possui rasgos que permitem o seu correto encaixe no elemento anterior. Devido ao atrito, estes rasgos são marginalmente maiores que os da base de modo a que se possa – sem problemas – encaixar e desencaixar as duas peças.

Como no caso anterior, existem também quatro cilindros amarelos e quatro azuis rigorosamente iguais entre si. Cada cilindro possui um furo que permite a sua alocação numa posição definida do reservatório. Por sua vez, podem ser dispostos ao longo do mesmo aleatoriamente.

3.1.1.3. Tampa

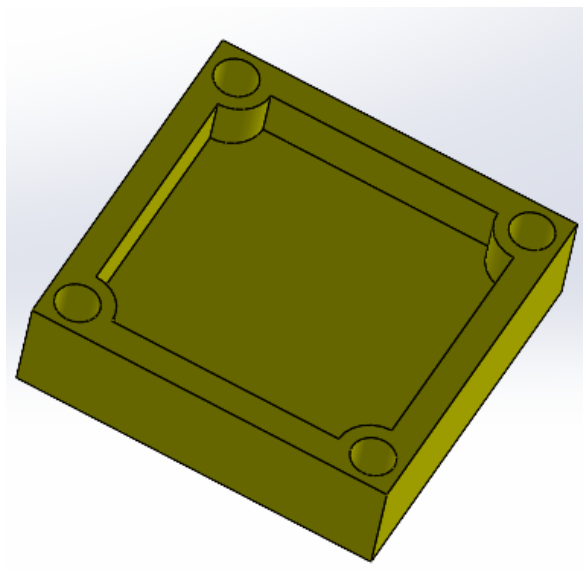


Figura 3.8 - Tampa Amarela

A tampa é dimensionalmente semelhante à base cúbica, sendo marginalmente mais baixa que a mesma. Tal como nos casos anteriores, existem também oito tampas – quatro de cada cor. As tampas estão depositadas em duas rampas – uma para cada cor. A montagem entre a tampa e a base é feita através de pinos de guiamento. Desta forma, não se trata de um encaixe propriamente dito, mas sim de um guiamento que impede que as peças se separem uma da outra.

3.1.1.4. Plataforma de montagem

A plataforma de montagem é, fundamentalmente, uma pequena caixa sem tampa que permite a inserção de uma base cúbica. A plataforma está fixada à placa perfilada e estabiliza o controlo de força. Existe somente uma plataforma já que o robô apenas desenvolve uma embalagem de cada vez. Esta peça é fundamental já que durante o controlo de força, se a base cúbica não estiver fixa, ou pelo menos, o menos móvel possível, incorre-se na possibilidade da montagem

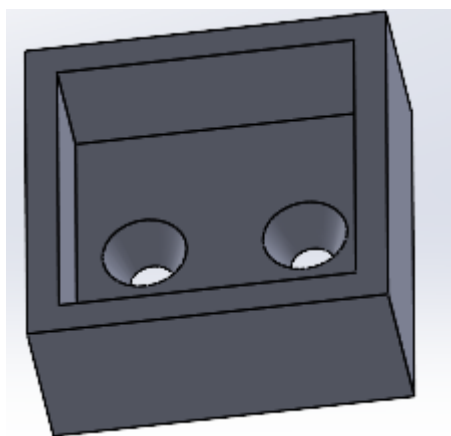


Figura 3.9 - Plataforma de montagem

entre as duas peças não se realizar. Esta impossibilidade traduz-se na oscilação da posição da base como consequência da força que lhe é aplicada.

3.1.1.5. Caixa de paletização

Após uma embalagem ter sido concluída, existem três caminhos possíveis: um segundo *conveyor*, paletização ou fim de ciclo. A paletização é feita num pequeno constituinte que



Figura 3.10 - Caixa de paletização

permite a montagem de até quatro peças. A altura desta caixa é reduzida para permitir ao robô inserir as peças com o maior rigor possível – a fim de se reduzir os desvios de posicionamento.

3.1.1.6. Armazém de cilindros

O armazém de cilindros é um pequeno armazém que ora permite o guiamento dos cilindros em posições definidas ora permite a sua colocação em posições aleatórias. As posições definidas são úteis para se definir a posição de calibração. Além disso, permite uma fácil e rápida testagem do sistema de força não sendo necessário desenvolver nenhum algoritmo para verificar se o encaixe entre as duas peças é possível.

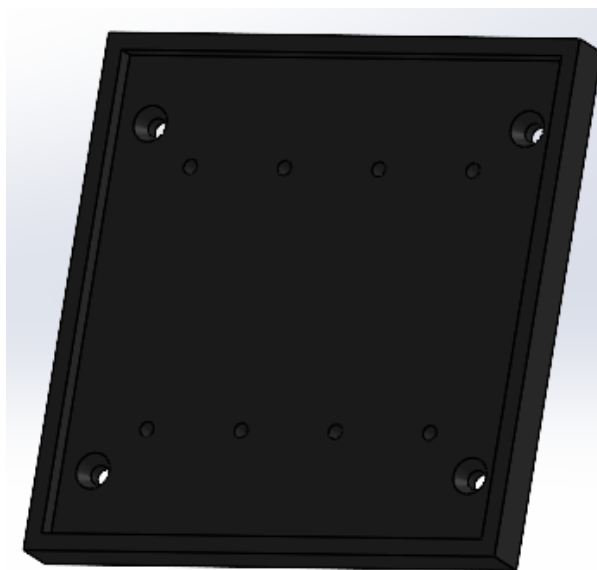


Figura 3.11 - Armazém de cilindros

Estas são as peças que, de uma forma ou de outra, estão envolvidas com o processo de montagem da embalagem, ora constituindo a mesma ora fornecendo suporte para os seus constituintes.

3.1.2. Elementos que constituem o alimentador

Deve-se realçar que, embora o alimentador tenha um papel fundamental no projeto, as suas especificidades exigem que este seja tratado em separado. Devido às limitações da impressora 3D, o alimentador teve de ser dividido em sete partes. Cinco constituem o alimentador em si enquanto que as outras duas correspondem a suportes – que permitem o ajuste da altura do alimentador. Deste modo, o alimentador é constituído pelos seguintes componentes:

3.1.2.1. Base

A base é um corpo paralelepipedico sobre o qual o corpo se fixa. Por sua vez, está fixa ao suporte secundário. As dimensões da base foram escolhidas de modo a que o alimentador ficasse o mais compacto possível bem como estável. Uma característica importante da base é a incorporação de ranhuras para alocar porcas. Uma vez que se pretende que o alimentador seja facilmente montado e desmontado, optou-se por incorporar porcas no mesmo a fim de diminuir o esforço envolvido no aperto e desaperto das mesmas. Por exemplo, caso estas não estivessem incorporadas, ter-se-ia de ser cauteloso aquando da desmontagem, de modo a garantir que as mesmas não se afastariam da sua posição.

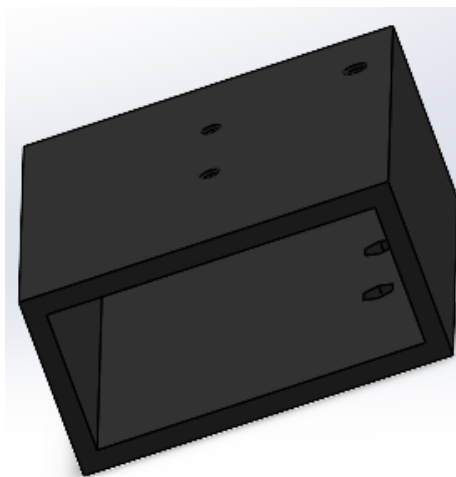


Figura 3.12 - Base do alimentador

3.1.2.2. Corpo

O corpo é a peça mais detalhada do alimentador. É uma peça totalmente preenchida (para ter a máxima rigidez possível) possuindo quatro particularidades:

- Ranhuras para encaixe do canal

A ligação entre o canal de passagem de bases cúbicas e o corpo é conseguido por atrito, isto é, o corpo possui ranhuras que encaixam justamente com as ranhuras do corpo. Essas ranhuras estão assinaladas com o número 1 na figura 3.9. Naturalmente, após algumas utilizações a

ligação tende a ser mais folgada. Apesar disso, e não sendo o propósito estar constantemente a retirar o canal, esta união provou ser eficiente.

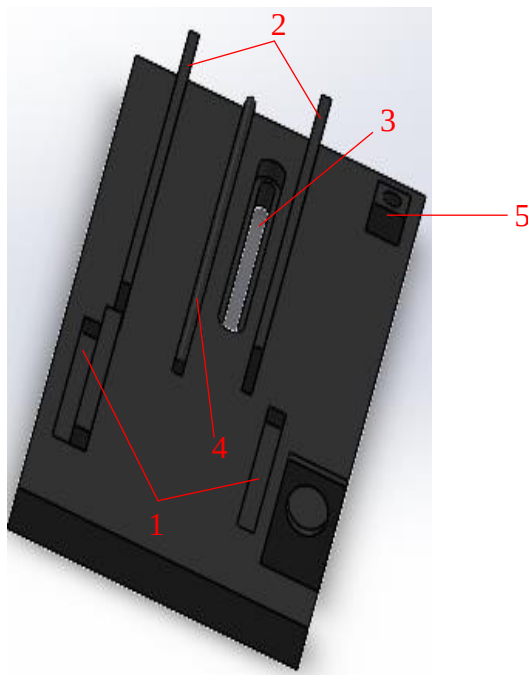


Figura 3.13 - Corpo do alimentador

- Limitadores do alimentador

No corpo, circula um atuador que será responsável pelo guiamento de bases cúbicas para o *conveyor*. Embora o movimento seja linear, optou-se por colocar dois limitadores – simbolizados pelo número 2 na figura 3.9, para impedir desvios e, simultaneamente, para ser mais facilmente identificável a posição do atuador no corpo.

- Rasgo para encaixe com base

O rasgo é uma pequena caixa que permite que o parafuso, a partir do qual a base e o corpo são unidos, não entre em contacto com o atuador. Este está marcado com o número 3 na figura 3.9.

- Ranhura para guiamento do atuador

Além dos limitadores, existe uma ranhura que guia o alimentador. Esta ranhura impede que, após várias utilizações, o alimentador rode aquando do seu movimento. Está assinalada com o número 4 na figura 3.9. Ainda que, mesmo que rodasse, a sua prestação não seria afetada, optou-se por colocar este suporte também por uma questão estética.

- Suporte do cilindro pneumático

Também no corpo, encontrar-se-à o cilindro pneumático responsável pelo movimento do atuador. De modo a que o cilindro não levite, existe uma saliência no corpo que permite a estabilidade do cilindro - número 5 na figura 3.9.

3.1.2.3. Atuador

O atuador foi desenhado após visualização de alguns exemplos tendo-se retirado inspiração de vários modelos. Além da ranhura – que emparelha com a do corpo – o atuador possui um batente permitindo que a base cúbica seja corretamente orientada e amparando a sua queda no *conveyor*. Este batente é fundamental porque, como se verá à frente, o *conveyor* possui duas guias perfiladas que funcionam como segurança – impedindo que a base cúbica caia. Devido ao desnível existente entre o alimentador e a tela, cada base cúbica tendia a rodar sobre si e a cair numa posição incorreta que comprometeria todo o sistema. Assim, este batente impede essa rotação e garante que a base é corretamente posicionada. O furo presente permite que o atuador

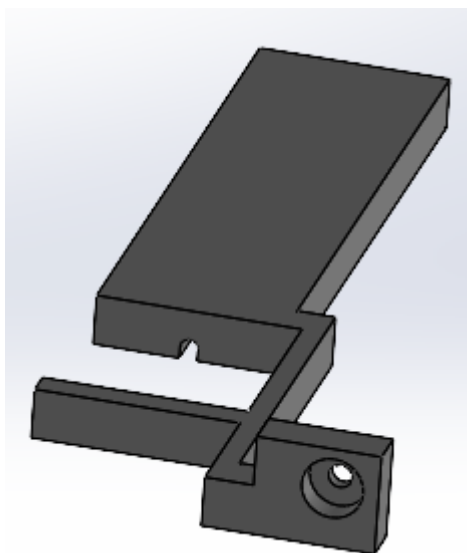


Figura 3.14 - Atuador linear

esteja solidário com o cilindro pneumático.

3.1.2.4. Canal

O canal é essencialmente um túnel por onde a base cúbica pronta a ser enviada para o *conveyor* circula. Acoplado ao mesmo está um armazém onde se encontram depositadas as restantes bases cúbicas. A ligação entre o armazém e o túnel não é de encaixe, estando o primeiro meramente apoiado no segundo. Desta forma, pode-se substituir, facilmente, as bases cúbicas sem ser necessário retirar o canal que, devido ao atrito, está firmemente fixado ao corpo. O canal possui dois rasgos que permitem a verificação, por parte de um sensor de barreira, da existência de

uma peça na fronteira do atuador, isto é, se existe pelo menos uma base cúbica no sistema. O alimentador de peças azuis possui um canal de cor amarela – de forma a distinguir os dois tipos de alimentador. O canal laranja identifica o alimentador de bases amarelas enquanto que o canal amarelo identifica o alimentador de bases azuis.

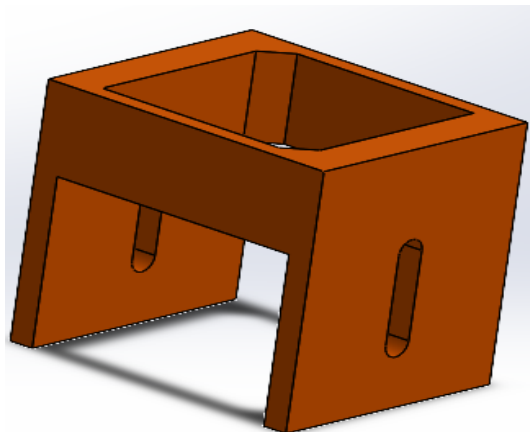


Figura 3.15 - Canal do alimentador

3.1.2.5. Armazém

O armazém é fundamentalmente uma coluna que assenta no canal e que permite alimentar bases cúbicas. Além disso, possui dois rasgos verticais que permitem ao utilizador conferir visualmente se existem bases cúbicas e quantas existem. Tanto o canal como o armazém foram produzidos com a técnica *poka-yoke*. Esta técnica japonesa, “à prova de erro”, apenas permite que as duas peças encaixem sob uma determinada orientação. Neste caso, o armazém apenas encaixa no canal caso os rasgos estejam alinhados com o eixo visual do utilizador. Analogamente ao canal, o alimentador de peças azuis possui um armazém em amarelo.

Como o alimentador operará a cerca de 145 mm da placa perfilada, torna-se necessário criar um suporte que o eleve. No entanto, para elevar o alimentador a essa altura o suporte seria uma peça muito densa e maciça pelo que, antes, preferiu-se construir dois suportes. O principal e o secundário.

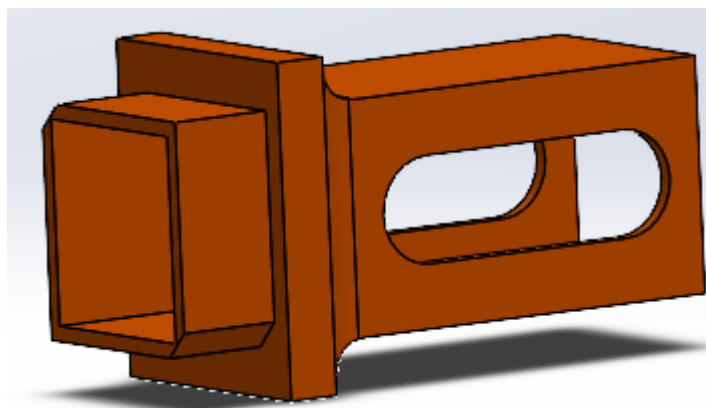


Figura 3.16 - Armazém de bases cúbicas

3.1.2.6. Suporte principal

O suporte principal está fixo à placa perfilada e possui uma estrutura em U. A ligação com o suporte 2 é conseguida através dois rasgos que possui de cada lado permitindo, deste modo, não só a fixação entre os mesmos, mas também a regulação da altura do alimentador.

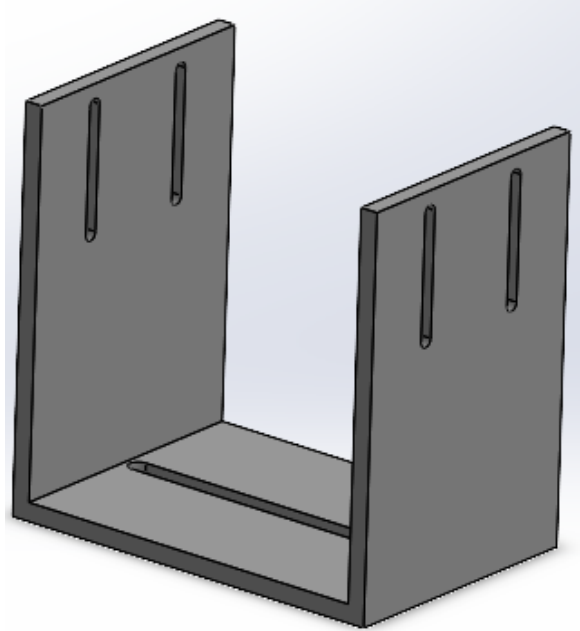


Figura 3.17 - Suporte principal

3.1.2.7. Suporte secundário

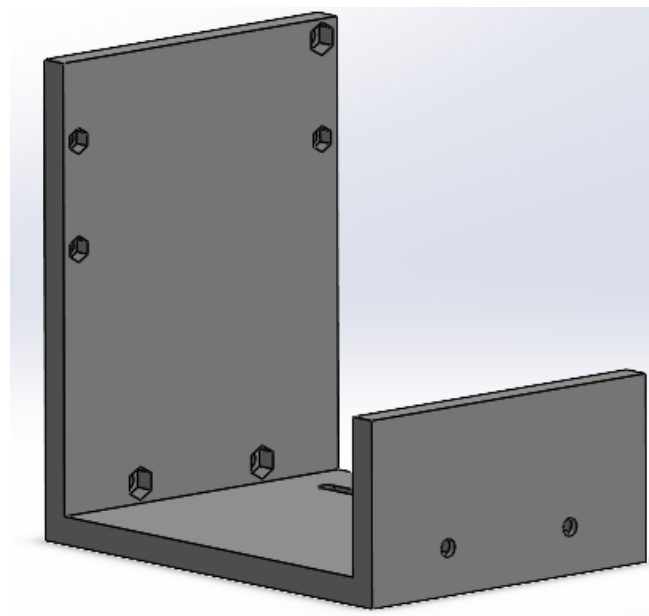


Figura 3.18 - Suporte secundário

O suporte secundário está conectado à base e ao suporte principal. A sua estrutura é também em U, com uma das extremidades significativamente inferior. Além disso, estão também fixos dois elementos externos: sensor e eletroválvula pneumática. Tal como na base, também o

suporte secundário possui ranhuras para incorporação de porcas. Uma das extremidades é inferior de modo a se impedir que o corpo chocasse com a parede do suporte.

Na figura 3.15, encontra-se a versão final do alimentador com todos os componentes ligados entre si:

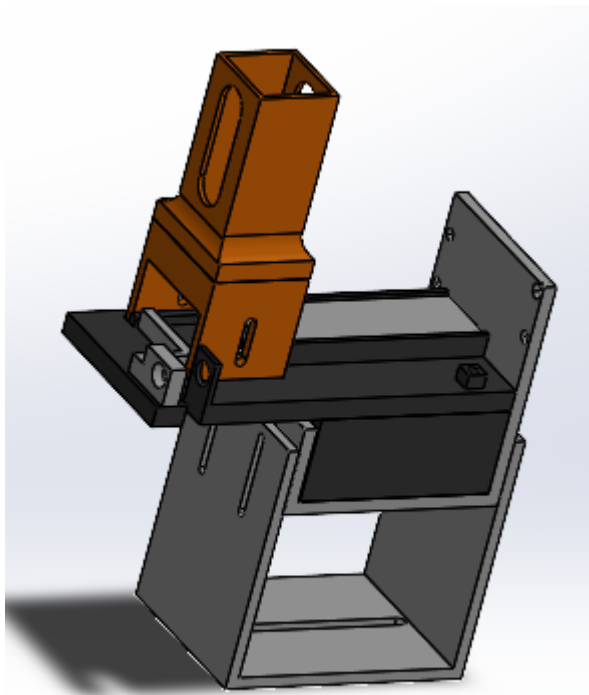


Figura 3.19 - Alimentador

Os elementos abordados anteriormente constituem os principais constituintes da célula ainda que existam outros componentes que favorecem ora o aspeto estético ora o aspeto funcional. Os restantes elementos serão abordados apenas via textual.

3.1.3. Elementos do *Conveyor*

Além dos elementos que constituem um bloco por si só – embalagem e alimentador – existem ainda componentes que exercem a função auxiliar de ligação e de conexão entre várias peças. Tão úteis como as anteriores, serão estes elementos que serão abordados.

3.1.3.1. Conectores

Sendo o *conveyor* constituído essencialmente por um perfil principal e por duas guias perfiladas que definem a trajetória retilínea da peça, é necessário que estes estejam conectados de algum modo. Assim, existem três tipos de conectores que, embora difiram mediante a sua aplicação, possuem fundamentalmente a mesma geometria. Ao todo, empregam-se quatro conectores: dois de cada lado do *conveyor*. Desses dois, cada um encontra-se numa extremidade do mesmo. A

necessidade dos três tipos de conectores prende-se com o facto das chumaceiras não serem todas iguais e também pela incorporação de uma placa de suporte do encoder. Estes dois fatores provocam a alteração dimensional e geométrica dos conectores.

3.1.3.2. Placa de Suporte do Encoder

Para alocar o encoder a um dos veios do *conveyor*, foi necessário desenvolver uma placa que assegure a fixação do primeiro. É uma peça semelhante a uma das chumaceiras, possuindo também porcas incorporadas que fixam o encoder. Esta placa apenas é necessária uma vez que, para se realizar o *Conveyor Tracking* ser necessário acoplar um encoder a um dos veios do *conveyor*. Desta forma, esta placa – que está fixada ao *conveyor* – permite a alocação do encoder impedindo que este se desloque permitindo a correta realização desta operação.

3.1.3.3. Placa de Suporte do Motor

É uma peça em L e está fixa a um dos apoios do tapete transportador e permite a fixação do motor, transmitindo-se, assim, mediante um sistema de polias, o movimento do mesmo ao tapete transportador.

3.1.3.4. Caixa de proteção

Esta caixa de proteção impede que o sistema polia + correia fique à vista. Deste modo, tenta-se proteger esta parte do sistema de poeiras, choques, etc. Além de contribuir para o aspeto estético, a caixa de proteção protege o sistema caso a correia se parta ou alguma polia se desparafuse do veio. Ainda além desta proteção, esta caixa impede que um operador acidentalmente toque com o braço ou com a mão no sistema das polias, impedindo eventuais lesões.

3.1.4. Elementos da Rampa

3.1.4.1. Nivelador de Rampa

O nivelador da rampa permite ajustar tanto a altura como a inclinação da mesma. É uma peça em U que se fixa entre os dois apoios da mesma. Os dois furos no centro permitem a fixação do corpo da rampa – um perfil 19x45.

3.1.4.2. Conectores

Existe somente um tipo de conectores na rampa que permite a ligação de duas guias perfiladas a um outro perfil principal. São colocados dois, um em cada extremidade. A geometria destes apoios foi concebida de modo a que a rampa se tornasse o mais compacta possível.

3.1.4.3. Plataforma de recolha

A plataforma de recolha está situada à saída da rampa, permitindo ao robô recolher a tampa. Esta plataforma é ligeiramente mais larga que uma tampa uma vez que a trajetória da mesma nunca é 100% linear e tem de se precaver qualquer desvio aquando da descida.

3.1.4.4. Apoios da plataforma

A plataforma de recolha é apoiada por dois elementos em forma de L, que asseguram o seu correto posicionamento bem como estabilidade. Como neste caso, nenhuma operação será realizada sobre a plataforma – somente a recolha de uma tampa – não é necessário que esta esteja fixa sobre a placa perfilada. Estas pernas permitem, em concílio com a elevação, ajustar a altura da rampa e da plataforma da mesma.

3.1.5. Outros

3.1.5.1. Suporte de câmara de vídeo

O suporte de câmara de vídeo permite fixar a câmara ao robô. Como consequência desta fixação, o TCP da garra altera-se de acordo com a espessura do mesmo. A fixação da câmara ao suporte é conseguida mediante dois furos horizontais, em vez de rasgos – garantindo que a sua posição é sempre a mesma.

3.1.5.2. Apoio para cabo

Existe uma outra peça que permite que o cabo da garra Robotiq Hand-E esteja protegido do exterior. Apesar de já existir uma peça da Robotiq, foi necessário criar uma nova devido à introdução do suporte de visão. A sua geometria é análoga à já existente, tendo-se tido o cuidado de aumentar a sua dimensão como consequência do suporte de visão.

3.1.5.3. Suporte para cabo

Este suporte para cabos impede que os mesmos circulem livres no meio da estação.

3.1.5.4. Batente

Como a base que circula no primeiro *conveyor* raramente atinge a mesma posição ao longo dos diversos ciclos, torna-se necessário criar um batente que assegure que esta alcança sempre a mesma posição final. Por outras palavras, este batente foi criado para melhorar a repetibilidade do sistema.

3.1.5.5. Dedos

Desenvolveram-se dedos com a geometria angular devido ao facto de ser necessário centrar as peças cilíndricas. O sistema de visão não é “perfeito” e possui várias imprecisões na deteção de peças pelo que é necessário garantir que mesmo que não esteja centrado, ao se fechar, se garanta uma correta centragem da peça cilíndrica.

Terminada a análise dos elementos produzidos por impressão 3D, analisa-se agora os elementos mecânicos que foram incorporados nesta solução.

3.2 Elementos Mecânicos

3.2.1. Conveyor

O *conveyor* é, fundamentalmente, constituído por um perfil em alumínio 45x60 (perfil principal) e por dois perfis 23x16 que servem de guias para a peça que circula no mesmo. O perfil principal está apoiado por dois perfis 30x30 que estão fixos à plataforma perfilada. A escolha dos perfis prendeu-se o seguinte:

- Dimensão da peça a circular e guias
Tendo a base cúbica uma dimensão ligeiramente inferior a 40 mm (~36 mm), optou-se por escolher um perfil base que tivesse uma dimensão de pelo menos 40 mm. No entanto, isto por si só não é suficiente já que, nesse perfil principal, teriam de assentar dois perfis 23x16. Daí que, e recorrendo-se ao *software* Cad Assembler – fornecido pela MiniTec – se tenha escolhido um perfil principal com 60 mm de base.

Os comprimentos dos perfis foram escolhidos de modo a suportar um tapete com dimensões adequadas. Como a placa perfilada tem dimensões 700x1100 mm, optou-se por colocar um tapete com comprimento de 500 mm. As guias perfiladas são marginalmente menores, apresentando dimensões de 480 mm, cada.

Além dos perfis, o *conveyor* é constituído por quatro chumaceiras, dois veios e um sistema de polias, além da tela. O modelo do *conveyor* foi elaborado no *software* mencionado acima pelo que se destaca ainda o seguinte:

- Chumaceiras

Existem três tipos de chumaceiras no *conveyor*. Uma chumaceira normal sem furo; outra igual à anterior mas com um furo – onde se encontra o veio que estará conectado ao encoder. A outra chumaceira estará ligada ao veio do motor e é diferente das demais.

- Veios

Os veios foram maquinados de modo a permitirem um correto funcionamento do motor, isto é, foram adaptados à dimensão das chumaceiras bem como das polias, encoder e motor DC.

- Sistema de polias

Apenas se pode transmitir o movimento do motor DC ao veio mandante por duas formas:

- Utilização de um motor de veio oco
- Utilização de um sistema de polias

Neste caso, optou-se por utilizar um sistema de polias uma vez que se escolheu um motor sem veio oco e que se adaptava aos requisitos do problema. O motor DC utilizado – que será abordado mais à frente – tem uma velocidade de rotação nominal de 58 rpm. Tendo o veio mandante um diâmetro de 25 mm, a velocidade linear do *conveyor* seria de, aproximadamente, 76 mm/s. Considerou-se que este valor era demasiado alto para se poder efetuar o *conveyor tracking* pelo que se optou por reduzir a velocidade. Já que seria necessário utilizar um sistema de polias para transmitir o movimento, optou-se, também, por efetuar uma redução de 2:1 (ou o mais próximo possível). Após uma pesquisa intensiva, optou-se por utilizar duas polias do tipo GT2, de 20 e 36 dentes – tendo-se assim uma redução de 1.8:1, sendo, por isso, a velocidade linear do tapete cerca de 42 mm/s – um valor mais aceitável. Note-se que a escolha da velocidade não se prendeu com a capacidade do robô executar o *conveyor tracking*, mas tratou-se de uma escolha do mestrando de modo a que este processo não fosse demasiado curto. A correia escolhida foi também uma GT2 com um comprimento de 400 mm.

A ligação entre os apoios e a placa perfilada bem como com o perfil principal é conseguido através de conectores próprios que a FluidoTronica enviou juntamente com os perfis.

Na figura 3.16, encontra-se uma representação tridimensional do mesmo.

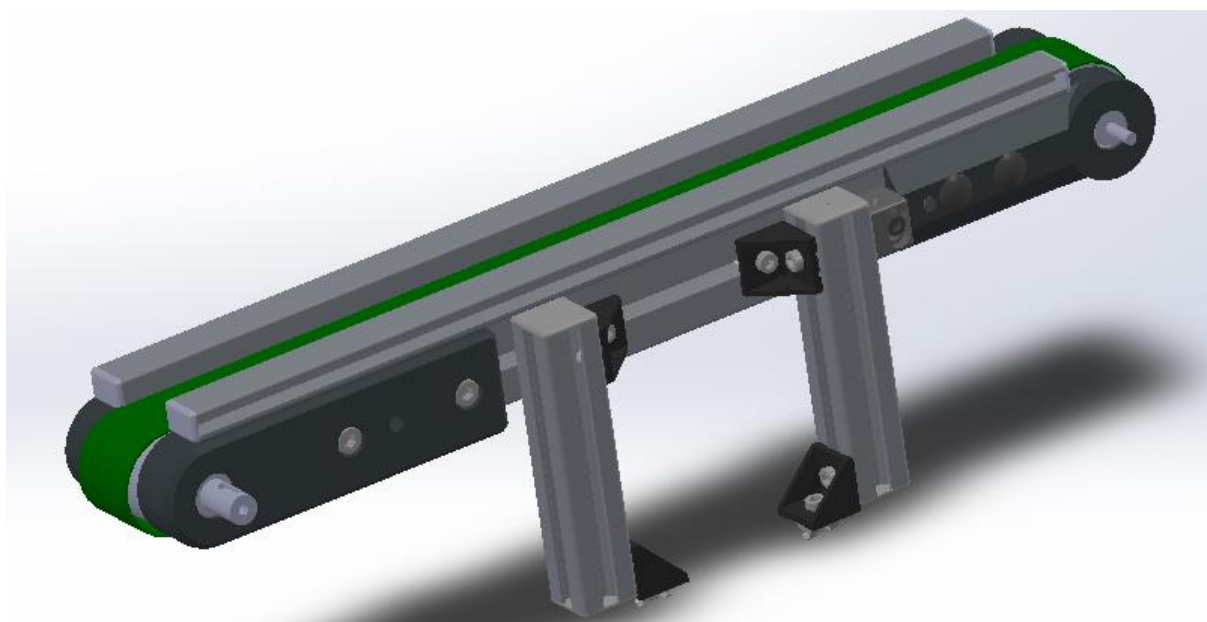


Figura 3.20 - Conveyor

3.2.2. Rampa

A rampa é constituída por um perfil principal 45x19 e por duas guias 23x16 – cuja função é análoga à das do *conveyor*. Tal como no caso anterior, o perfil principal está também apoiado em dois perfis 30x30. O perfil principal e as guias têm um comprimento de 140 mm, enquanto que os apoios têm um comprimento de 180 mm. Contrariamente ao *conveyor*, em que as guias

assentam no mesmo, na rampa as guias têm apenas um pequena área de contacto devido ao tipo de conectores utilizados. Desta forma, os 45 mm úteis são transformados em cerca de 40 mm. Como as tampas que circulam na rampa têm um dimensão de ~36 mm, os 40 mm úteis permitem um movimento fluido ainda que fundamentalmente linear. A escolha de um rampa prendeu-se com a facilidade que esta apresenta na recolha de elementos além de ocupar pouco espaço físico na célula total enquanto que representa uma etapa fundamental da mesma.

Também na seguinte tabela se encontram todos os elementos que constituem a rampa:

Também se encontra na figura 3.17 uma representação tridimensional da mesma.

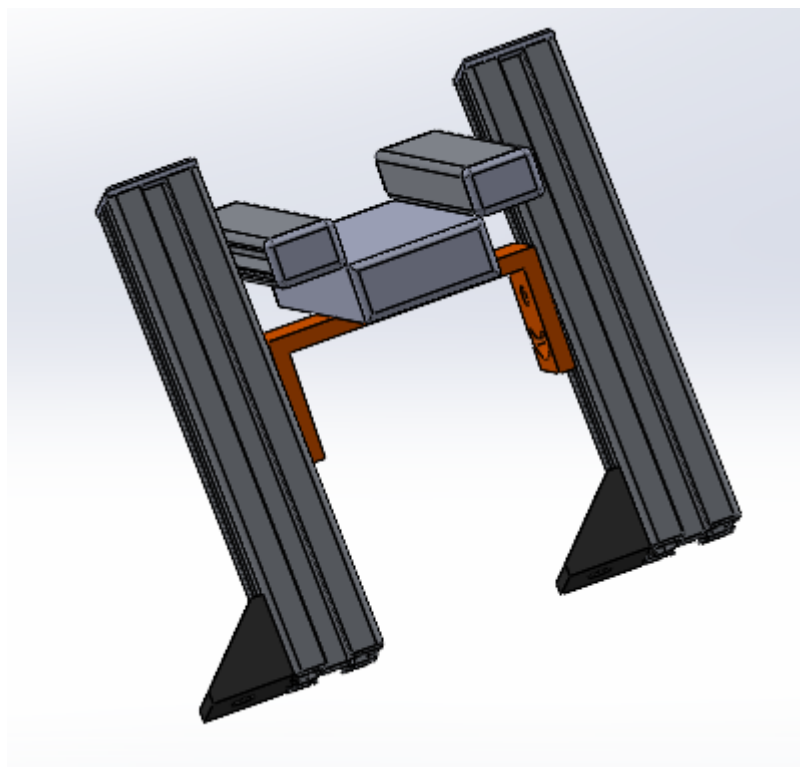


Figura 3.21 - Rampa

3.3 Elementos Elétricos e Eletropneumáticos

Estes componentes constituem uma outra parte de controlo da estação nos quais se inserem os elementos que permitem o acionamento de cilindros e de *conveyors*. Sem estes elementos de controlo seria impossível que a célula funcionasse.

3.3.1. Elementos Elétricos

Houveram vários elementos elétricos que foram incorporados a fim da célula ser robusta e eficiente. São eles:

3.3.1.1. Motor DC

A escolha do motor elétrico teve por base três fatores: velocidade, binário e atravancamento. Relativamente ao último, dado que a aplicação é de diminuta exigência, optou-se por escolher um motor DC com escovas. Além de de pequena dimensão, é de reduzido custo e elevada eficiência. Para se calcular o binário necessário para o *conveyor*, recorreu-se a um pequeno simulador da Oriental Motors (Oriental Motor USA, 2020), tendo-se chegado a um valor de 0.7 Nm.

O simulador encontra-se nas referências (Oriental Motor USA, 2020).

Desta forma, foi necessário encontrar um motor com um binário nominal de pelo menos 1 Nm e com uma velocidade nominal na ordem de 60-70 rpm. Esta velocidade foi calculada tendo em conta que, após a redução 1.85:1, a velocidade linear encontrar-se-ia na ordem dos 30-50 mm/s. Logo, escolheu-se o motor 150:1 Metal Gearmotor 37D 24V da Polulu. A sua curva encontra-se na figura 3.18:

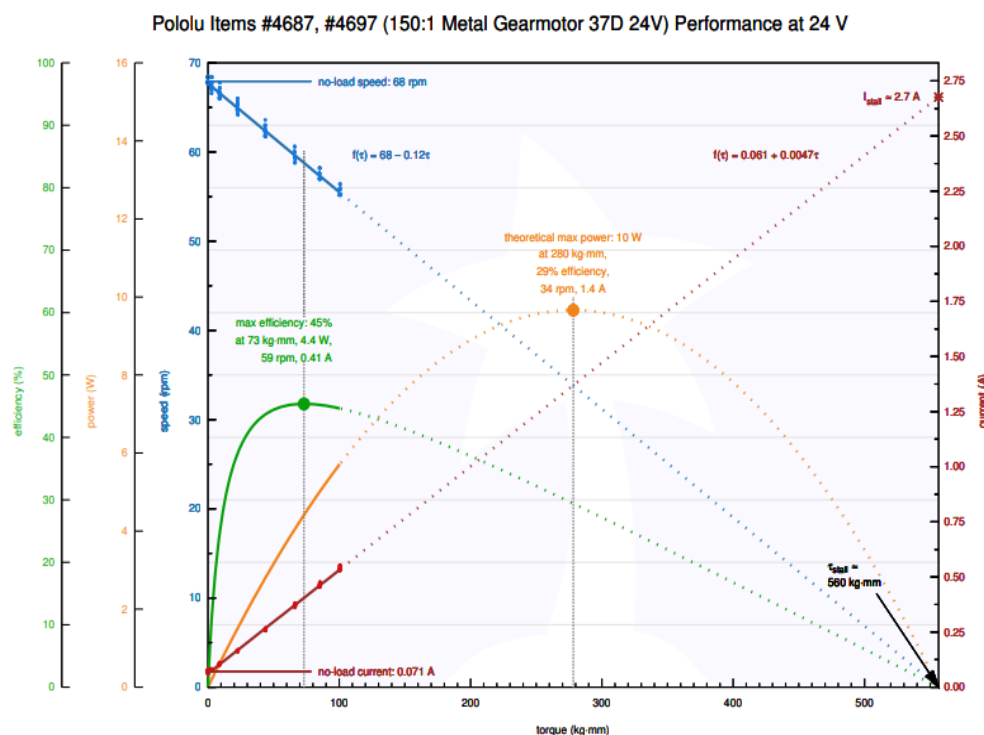


Figura 3.22 - Curva caraterística do motor DC

A escolha do motor prendeu-se, além dos requisitos funcionais, com o preço e prazo de entrega. Este motor era o que apresentava melhor relação custo-prazo de entrega.

3.3.1.2. Encoder

Como já anteriormente se mencionou, o encoder é fundamental para a realização do *Conveyor Tracking*. Além de partilhar a alimentação do conveyor, o encoder tem de ser de quadratura. Um encoder de quadratura é um encoder incremental com dois canais de saída desfasados 90 graus. Em função do sinal que é lido primeiramente, o encoder consegue identificar qual o

sentido de rotação. Os encoders de quadratura fornecem uma maior precisão tendo-se, neste projeto, escolhido um encoder com uma resolução de 1 grau por rotação. O encoder escolhido foi o Enc 51 da DataLogic. A figura 3.19 apresenta os sinais em quadratura:

Escolheu-se este encoder uma vez que a DataLogic possui uma parceria com a Growskills pelo que o custo do sensor é reduzido em 10%.

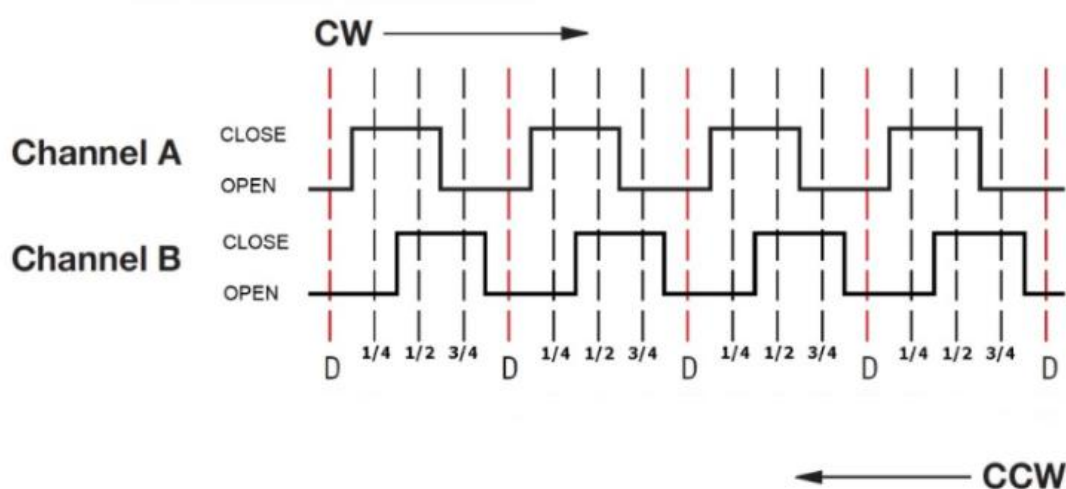


Figura 3.23 - Sinais do encoder em quadratura

3.3.1.3. Detetor fotoelétrico difuso

Um detetor fotoelétrico difuso tem o seguinte *modus operandi*: um feixe atinge um obstáculo e é refletido. Nesta célula, verificou-se que os cilindros azuis não são tão bem detetados como os amarelos uma vez que a superfície azul é menos refletora que a amarela. Contudo, ambas são detetadas por estes sensores.

Em cada *conveyor*, existe um detetor de fim-de-curso que assinala o fim do *Conveyor Tracking*. Para a escolha deste sensor, procurou-se uma alternativa simples e de pequena dimensão. Após uma pesquisa de várias fornecedores, optou-se por escolher o sensor SM-ALL da DataLogic. O único cuidado a ter na escolha do sensor foi o seguinte: o robô só interpreta sinais PNP pelo que foi este o formato escolhido.

3.3.1.4. Sensor de barreira

O sensor de barreira emite um feixe luminoso que se encontrar um obstáculo permutará o seu contacto. Este sensor não traduz o seu comportamento numa reflexão do feixe mas sim na interrupção do mesmo.

Além do sensor de fim-de-curso, implementaram-se também sensores de barreira que permitem a deteção de uma base cúbica em cada alimentador. O sensor utilizado foi o S7, que permite operar quer como barreira quer como difuso em função das fibras óticas selecionadas.

3.3.1.5. Interruptor diferencial

Selecionou-se um disjuntor diferencial para se desligar o circuito sempre que se deteta uma corrente de fuga superior à nominal.

3.3.1.6. Disjuntor

A função do interruptor diferencial é análoga à da do disjuntor pelo que oferece somente proteção extraordinária.

3.3.1.7. Controlador de Motor

Um controlador de motor é um equipamento elétrico que comanda a performance de um motor elétrico. O controlador pode incluir rampas de aceleração, controlo de velocidade, controlo de sentido de rotação, controlo de binário bem como proteção contra correntes superiores à nominal. No presente caso, o controlo do sentido de rotação, controlo de velocidade e de binário foram as principais características que determinaram a escolha do controlador de motor. O controlador de motor utilizado foi o Kaleja 06.04.025.

3.3.1.8. Fonte de alimentação

Naturalmente, sendo todos os elementos alimentados a 24VDC, é necessário uma fonte de alimentação que forneça a respetiva tensão. Como estes motores operam somente a uma corrente de 1.5-2 A, optou-se por escolher uma fonte de alimentação com uma corrente nominal de 5A – suficiente para suportar todo o equipamento.

3.3.2. Elementos Eletropneumáticos

No domínio pneumático, destacam-se os seguintes elementos:

3.3.2.1. Cilindro de duplo efeito com amortecimento

Para se controlar eficazmente o movimento do atuador do alimentador, optou-se por utilizar um cilindro de duplo efeito com amortecimento. Em função do desenho do alimentador, considerou-se que um cilindro com 100 mm de curso era suficiente para corresponder com as exigências do projeto. Além do cilindro, existem outros elementos que foram incorporados no mesmo (por exemplo, válvulas unidirecionais de caudal, etc...)

A figura 3.20 apresenta o diagrama pneumático da célula robótica.

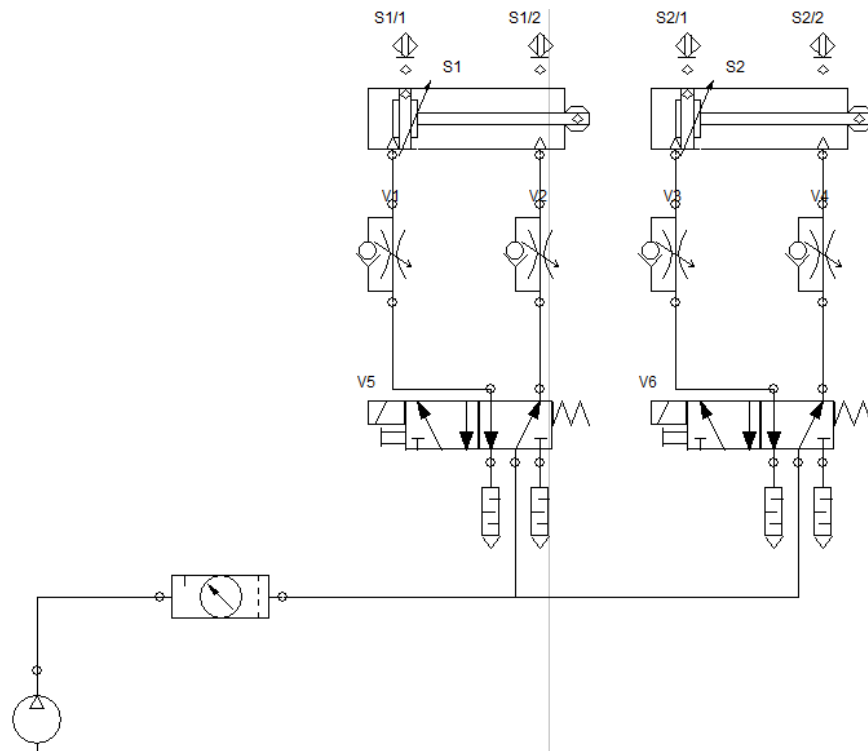


Figura 3.28 - Esquema Pneumático

Terminada a análise de todos os constituintes da célula robótica, a figura 3.21 representa um

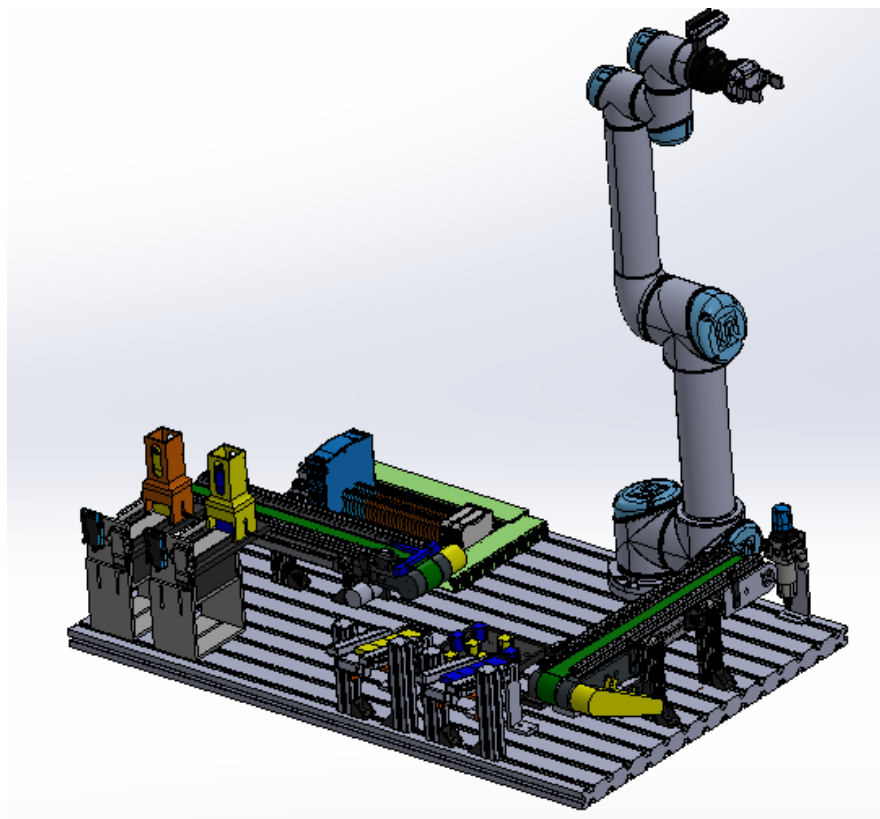


Figura 3.29 - Modelo da célula robótica

modelo, ainda que muito simplificado, da célula robótica:

Por fim, apresentam-se algumas fotografias tiradas à estação de modo a evidenciar-se a realização do projeto. De forma a não se colocar demasiadas fotografias, optou-se por colocar somente três que conferem um panorama de cada parte da célula.



Figura 3.30 - Secção elétrica da estação



Figura 3.31 - Conveyer inicial e alimentadores

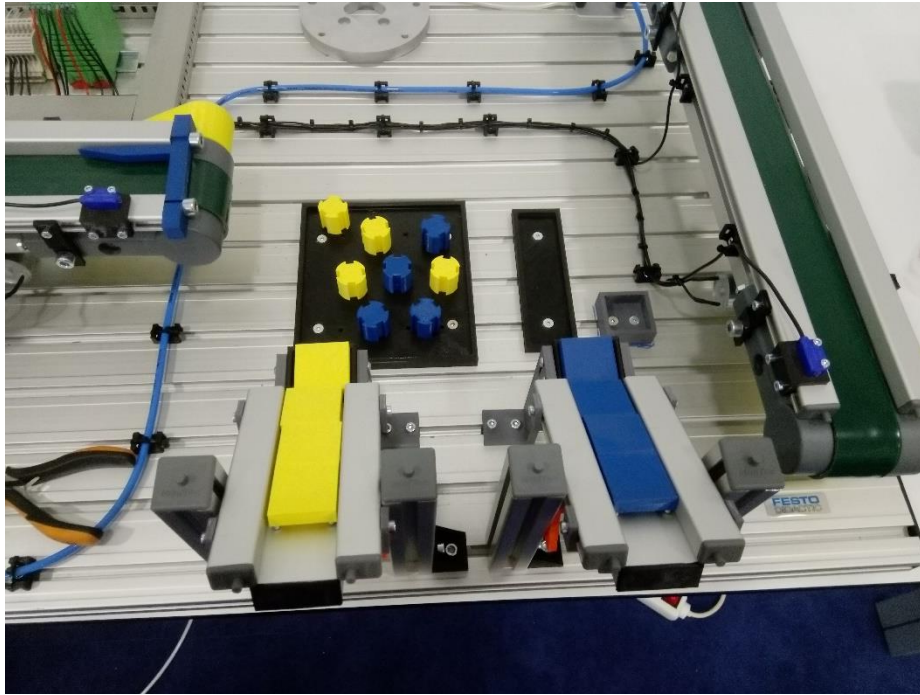


Figura 3.32 - Rampas + conveyor final

4 Implementação do sistema de visão em Python

A parte mais desafiante do projeto foi a concepção de um sistema de visão que se adequasse às exigências da célula robótica. Após alguma pesquisa, e devido a se utilizar a câmara Intel RealSense D435i, optou-se pela linguagem Python já que é esta a mais adequada para operar com este sistema de visão. Antes de se proceder à análise do código e, de modo a se simplificar a análise do mesmo, dividiu-se o mesmo em:

- Calibração da câmara
- Seleção de um cilindro

O algoritmo relativo à seleção de um cilindro prende-se com o facto de estes poderem assumir posições variáveis e, por isso, ser necessário ao robô identificar qual a que impede qualquer colisão. Apesar de existirem posições definidas – marcadas pelos pinos de guiamento – caso os cilindros se restringissem às mesmas, não seria necessário recorrer a nenhum sistema de visão. Resumidamente, o sistema de visão identificará – caso possível – qual o cilindro que pode ser recolhido pelo *gripper*.

4.1 Calibração da câmara

Durante a calibração, o algoritmo em Python deteta a posição de um cilindro – amarelo ou azul – e, em função da mesma, obtém a origem do referencial a que essa posição se refere. Antes de se fundamentar este procedimento é necessário perceber que a posição de um determinado ponto é dado em pixels. O primeiro passo consiste em converter estas coordenadas em mm. Para tal, utilizou-se uma folha axadrezada – com quadrados de 5 mm de lado. Colocou-se o cilindro em várias posições dessa folha convertendo-se as coordenadas de pixels em milímetros. Este processo é lento tendo sido necessário utilizar cerca de 40 posições. Após estes ensaios, tomou-se o valor médio como razão entre pixel e coordenada em milímetro.

A calibração consiste em criar um referencial cartesiano com origem na origem do referencial câmara. Seguidamente, a informação que é enviada ao robô – a coordenada do centro do cilindro em mm, em relação ao referencial câmara – permite criar um referencial com a mesma origem que o da câmara. Assim, todas as demais coordenadas têm por base uma mesma origem, não sendo necessário realizar mais nenhuma conversão. O referencial é criado através da função *pose_trans()* a qual torna possível uma translação da posição original para a origem – estabelecendo-se esta última. Apresenta-se um excerto do algoritmo de programação, de modo a facilitar a compreensão desta etapa.

No excerto apresentado, destacam-se as seguintes etapas:

- Captação de imagem;
- Detecção de um contorno amarelo;
- Detecção do centro desse contorno e respectivas coordenadas;
- Envio da informação.

```

1 import cv2
2 from realsense_depth import *
3 def div_int(x, y):
4     if y == 0:
5         return 0
6     return int(x / y)
7
8 dc = DepthCamera()
9 dados_calibracao = []
10 calibracao = False
11 while calibracao != True:
12     ret, dep_frame, color_frame = dc.get_frame()
13     cap = color_frame
14     grey_cap = cv2.cvtColor(cap, cv2.COLOR_BGR2GRAY)
15     hsv = cv2.cvtColor(cap, cv2.COLOR_BGR2HSV)
16     lower_yellow = np.array([15, 70, 10])
17     upper_yellow = np.array([35, 255, 255])
18     mask = cv2.inRange(hsv, lower_yellow, upper_yellow)
19     cv2.imshow("Original", cap)
20     contours, _ = cv2.findContours(mask, cv2.RETR_TREE,
21 cv2.CHAIN_APPROX_SIMPLE)
22     for cnt_amarelo in contours:
23         approx_amarelo = cv2.approxPolyDP(cnt_amarelo, 0.01 *
24 cv2.arcLength(cnt_amarelo, True), True)
25         area_amarelo = cv2.contourArea(approx_amarelo)
26         if area_amarelo > 3000:
27             cv2.drawContours(cap, [approx_amarelo], 0, (0, 0, 0), 2)
28             M_amarelo = cv2.moments(approx_amarelo)
29             cx_amarelo = div_int(M_amarelo["m10"], M_amarelo["m00"])
30             cy_amarelo = div_int(M_amarelo["m01"], M_amarelo["m00"])
31             cv2.circle(cap, (cx_amarelo, cy_amarelo), 3, (0, 0, 0), -1)
32             cv2.putText(cap, "Centro", (cx_amarelo - 20, cy_amarelo - 20),
33 cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
34             distance_amarelo = dep_frame[cy_amarelo, cx_amarelo]
35             mmx_amarelo = cx_amarelo / 2.276
36             mmy_amarelo = cy_amarelo / 2.202
37             mmx_amarelo = round(mmx_amarelo / 1000, 3)
38             mmy_amarelo = round(mmy_amarelo / 1000, 3)
39             if len(dados_calibracao) == 0:
40                 dados_calibracao = [mmx_amarelo, mmy_amarelo, 0, 0, 0, 0]
41                 dados_calibracao = str(dados_calibracao)
42                 calibracao = True
43                 print(dados_calibracao)
44             cv2.imshow("Original", cap)
45             key = cv2.waitKey(1)
46             if key == 27:
47                 if calibracao == True:
48                     break
49             break

```

Figura 4.1 - Algoritmo de calibração

4.1.1. Captação de imagem

A captação de imagem é conseguida através de uma função da Intel RealSense .Apenas é necessário chamar essa função neste algoritmo – precisamente o que acontece no início do programa na linha 2.

4.1.2. Detecção de um contorno amarelo

De seguida é necessário desenhar um contorno sobre tudo que seja da cor amarela. A forma mais fácil de o fazer é convertendo a imagem para HSV e depois filtrar esta última em função dos valores de H – Hue (tonalidade), S – Saturation (saturação) e V – Value (valor/brilho). Isto ocorre entre as linhas 15-19 – nesta última encontra-se os contornos correspondentes a estes valores de HSV – estes valores foram determinados experimentalmente. Contudo, por mais precisos que estes valores sejam, encontram-se sempre “resíduos”, isto é, pequenos contornos que não correspondem à cor amarela e que induzem o utilizador em erro. Por isso, apenas se tem em consideração os contornos com uma determinada área – em pixel. Esta área foi também determinada experimentalmente tendo-se chegado a um valor de 3000.

4.1.3. Detecção do centro do contorno e respetivas coordenadas

Após se desenhar os contornos com uma área suficientemente grande, determina-se os centros dos mesmos. Esta determinação é conseguida nas linhas 28-33. A função *cv2.moments* determina os momentos do contorno; as coordenadas do centróide são dadas pelas seguintes equações:

$$C_x = \frac{M_{10}}{M_{00}}; C_y = \frac{M_{01}}{M_{00}}$$

Estas operações têm lugar nas linhas 29 e 30, respetivamente. As restantes duas linhas limitam-se a desenhar um pequeno círculo no centro e em colocar uma linha de texto por cima do mesmo – tudo isto para ser mais fácil para o utilizador.

Segue-se a conversão das coordenadas em mm. Os valores foram determinados experimentalmente. Como o robô interpreta as coordenadas em metros, é necessário dividir por 1000.

4.1.4. Envio da informação

Procede-se, finalmente, à criação de um vetor com as coordenadas do cilindro que é posteriormente enviado para o robô. Apenas x e y é que têm relevância já que não se pretende nem rotação segundo qualquer dos eixos nem translação segundo o eixo zz.

Nota: Como curiosidade e complemento ainda se determina a distância do cilindro à câmara, isto é, a profundidade ou dimensão zz. Esta não tem qualquer uso prático na dissertação. O algoritmo de calibração utilizando um cilindro amarelo é rigorosamente igual, alterando-se unicamente os intervalos de HSV.

4.2 Seleção do Cilindro

A seleção do cilindro é o processo mais exaustivo do sistema de visão já que, caso se pretenda que o robô recolha um cilindro de uma determinada cor, tem de se ter em conta não só os cilindros dessa cor selecionada, mas também os outros já que a sua posição influencia a orientação com que a garra o pode abordar. A seleção do cilindro está dividida em três partes:

- Determinação das posições de todos os cilindros presentes
- Cálculo de parâmetros
- Análise de parâmetros

4.2.1. Determinação das posições de todos os cilindros

Na situação mais abrangente possível, existem oito cilindros: quatro de cada cor. O primeiro passo para se determinar os parâmetros consiste em recolher as posições destes cilindros. Após a calibração ter sido terminada, reúnem-se as condições para se determinar as posições de todos os cilindros em relação à origem do referencial câmara. Este procedimento é análogo ao utilizado na calibração. Seguidamente, colocam-se estas coordenadas em duas matrizes: uma para cada cor. Devido a limitações de algoritmo, mesmo que exista, por exemplo, somente um cilindro amarelo, a matriz terá na mesma quatro linhas – só que serão todas iguais. Após se tentar impedir que isto acontecesse – através da contagem de contornos ou métodos semelhantes – não se conseguiu solucionar este problema. Na prática, não advém qualquer perturbação deste tratamento de informação. Verificar-se-à, em secções posteriores, que o algoritmo e a sua funcionalidade não é afetada por esta redundância de informação. Antes de se proceder à avaliação desta etapa, sempre os eixos xx e yy forem mencionados, estes assumem a sua posição cartesiana típica alterando-se somente o sentido do eixo yy.

Para, eficientemente, se determinar qual o cilindro que deve ser recolhido primeiramente foi necessário calcular vários parâmetros que influenciam e determinam esta escolha. Os parâmetros são lineares e angulares:

4.2.1.1. Distância entre centros

A distância entre centros é facilmente calculada pela norma do vetor que une dois centros. A distância entre centros é calculada como se observa na figura 4.2:

```
dist_centros[i][j] = dist_x[i][j] ** 2 + dist_y[i][j] ** 2
dist_centros[i][j] = math.sqrt(dist_centros[i][j])
```

Figura 4.2 - Cálculo da distância entre centros

A variável *dist_centros* é uma matriz que contém em cada linha a distância do cilindro *i* ao cilindro *j* (sendo *i* o cilindro da linha e o *j* os restantes cilindros). As variáveis *dist_x* e *dist_y* representam a diferença de coordenadas *x* e *y* entre cilindros, respetivamente. É fundamental calcular a distância entre centros de cilindros já que, para o robô poder agarrar o cilindro, é necessário que este consiga inserir a garra num espaço que é ligeiramente superior ao ocupado pelo cilindro.

4.2.1.2. Ângulo formado pelo segmento de reta que une os centros e o eixo XX

O ângulo formado entre o segmento de reta que une os centros e o eixo XX é necessário já que servirá para determinar a orientação da garra. Este ângulo é calculado como o excerto da figura 4.3 apresenta:

```
tangente_alfa[i][j] = div_real(abs(dist_y[i][j]), abs(dist_x[i][j]))
alfa[i][j] = math.atan(tangente_alfa[i][j])
```

Figura 4.3 - Cálculo do ângulo alfa

A tangente do ângulo, *alfa*, corresponde à razão entre a diferença em *y* e a diferença em *x*. De modo a se obter somente ângulos positivos, recorreu-se a uma divisão em módulo. A figura 4.4 evidencia o que se considera como *alfa*:

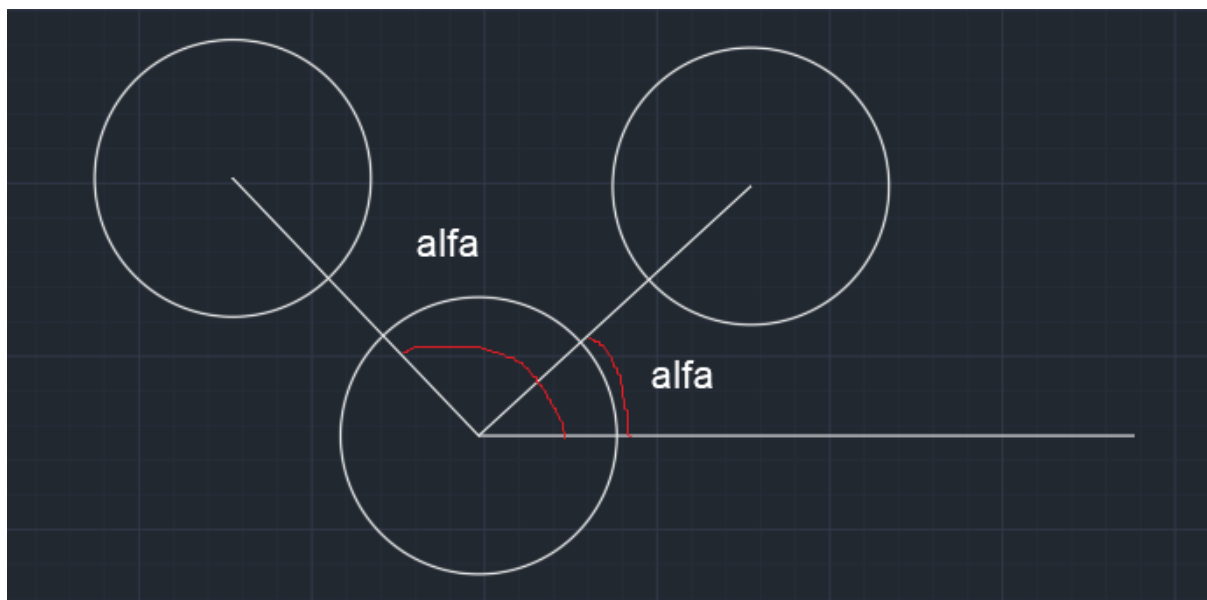


Figura 4.4 - Conceção do ângulo alfa

A partir deste desenho, entende-se que o *alfa* calculado não apenas corresponde ao real caso a diferença segundo *x* seja menor que 0 e a diferença segundo *y* seja maior que 0. Assim, este *alfa* tem de ser ajustado. A variável *alfalinha* corresponde ao *alfa* real e não ao calculado. Na figura 4.5, encontra-se a definição do *alfalinha*.

(Nota: A direção do eixo x e y correspondem às suas típicas direções cartesianas. O sentido de y está alterado uma vez que é este o eixo cartesiano da câmara)

```

if dist_x[q][w] > 0 and dist_y[q][w] > 0:
    alfalinha[q][w] = math.pi - alfa[q][w]
    limite_up[q][w] = alfalinha[q][w] + beta[q][w]
    limite_down[q][w] = alfalinha[q][w] - beta[q][w]
elif dist_x[q][w] > 0 and dist_y[q][w] < 0:
    alfalinha[q][w] = math.pi + alfa[q][w]
    limite_up[q][w] = alfalinha[q][w] + beta[q][w]
    limite_down[q][w] = alfalinha[q][w] - beta[q][w]
elif dist_x[q][w] < 0 and dist_y[q][w] > 0:
    alfalinha[q][w] = alfa[q][w]
    limite_up[q][w] = alfalinha[q][w] + beta[q][w]
    limite_down[q][w] = alfalinha[q][w] - beta[q][w]
elif dist_x[q][w] < 0 and dist_y[q][w] < 0:
    alfalinha[q][w] = 2 * math.pi - alfa[q][w]
    limite_up[q][w] = alfalinha[q][w] + beta[q][w]
    limite_down[q][w] = alfalinha[q][w] - beta[q][w]
elif dist_x[q][w] > 0 and dist_y[q][w] == 0:
    alfalinha[q][w] = alfa[q][w] + math.pi
    limite_up[q][w] = alfalinha[q][w] + beta[q][w]
    limite_down[q][w] = alfalinha[q][w] - beta[q][w]
elif dist_x[q][w] < 0 and dist_y[q][w] == 0:
    alfalinha[q][w] = alfa[q][w]
    limite_up[q][w] = alfalinha[q][w] + beta[q][w]
    limite_down[q][w] = alfalinha[q][w] - beta[q][w]
elif dist_x[q][w] == 0 and dist_y[q][w] > 0:
    alfalinha[q][w] = alfa[q][w]
    limite_up[q][w] = alfalinha[q][w] + beta[q][w]
    limite_down[q][w] = alfalinha[q][w] - beta[q][w]
elif dist_x[q][w] == 0 and dist_y[q][w] < 0:
    alfalinha[q][w] = alfa[q][w] + math.pi
    limite_up[q][w] = alfalinha[q][w] + beta[q][w]
    limite_down[q][w] = alfalinha[q][w] - beta[q][w]

```

Figura 4.5 - Definição do alfa real

Os limites do ciclo, $len(lista_completa) - 1$, correspondem ao número de linhas e colunas da matriz que possui toda a informação dos cilindros azuis e amarelos.

4.2.1.3. Ângulo que determina os limites do cilindro no espaço

Definiu-se como beta o ângulo formado entre o segmento de reta que une os centros de dois pontos e o ponto extremo do mesmo, isto é, este ângulo define o intervalo de ângulos a que pertence um determinado cilindro no espaço. Tendo todos os cilindros diâmetros de 27 mm, o beta é calculado desta forma:

```

tangente_beta[i][j] = div_real(0.0135, dist_centros[i][j])
beta[i][j] = math.atan(tangente_beta[i][j])

```

Figura 4.6 - Definição do ângulo beta

Contudo, para se obter o intervalo de ângulos a que pertence o cilindro, tem de se recorrer às variáveis *limite_up* e *limite_down*, expressas na figura 4.5. A partir destas variáveis é que se consegue definir se a garra do robô se encontra em risco de choque.

Ainda antes de se proceder ao último trecho do algoritmo, deve-se salientar dois pontos:

- A esta altura do algoritmo, as diversas matrizes que possuem informações relativas a distâncias, ângulos, limites, etc possuem todas – na diagonal principal – o valor zero ou $\pi/2$ caso se trata de uma matriz angular. Isto acontece uma vez que os ciclos não excluem o cálculo dos diversos parâmetros entre o mesmo cilindro. Daí que, se utilize o trecho da figura 4.7 para se retirar a diagonal principal:

```
for idx, ele in enumerate(dist_x):
    dist_x_final.append([el for idxx, el in enumerate(ele) if idxx != idx])
for idx, ele in enumerate(dist_y):
    dist_y_final.append([el for idxx, el in enumerate(ele) if idxx != idx])
for idx, ele in enumerate(dist_centros):
    dist_centros_final.append([el for idxx, el in enumerate(ele) if idxx !=
idx])
for idx, ele in enumerate(alfa):
    alfa_final.append([el for idxx, el in enumerate(ele) if idxx != idx])
for idx, ele in enumerate(beta):
    beta_final.append([el for idxx, el in enumerate(ele) if idxx != idx])
for idx, ele in enumerate(limite_up):
    limite_up_final.append([el for idxx, el in enumerate(ele) if idxx !=
idx])
for idx, ele in enumerate(limite_down):
    limite_down_final.append([el for idxx, el in enumerate(ele) if idxx !=
idx])
for idx, ele in enumerate(tangente_alfa):
    tangente_alfa_final.append([el for idxx, el in enumerate(ele) if idxx !=
idx])
```

Figura 4.7 - Remoção da diagonal principal

Com este trecho a matriz *dist_x*,

$$\begin{array}{ccccccc} 0 & i & p & w & d' & k' & r' & y' \\ b & 0 & q & x & e' & l' & s' & z' \\ c & j & 0 & y & f' & m' & t' & a'' \\ d & k & r & 0 & g' & n' & u' & b'' \\ e & l & s & z & 0 & o' & v' & c'' \\ f & m & t & a' & h' & 0 & w' & d'' \\ g & n & u & b' & i' & p' & 0 & e'' \\ h & o & v & c' & j' & q' & x' & 0 \end{array}$$

,

transforma-se na matriz *dist_x_final*,

i p w d' k' r' y'
b q x e' l' s' z'
c j y f' m' t' a''
d k r g' n' u' b''
e l s z o' v' c''
f m t a' h' w' d''
g n u b' i' p' e''
h o v c' j' q' x'

- De modo a melhorar e facilitar o processo de seleção de um cilindro, optou-se pelo seguinte critério: para qualquer cor, o algoritmo começará por avaliar as condições espaciais do cilindro que estiver mais distante dos demais. Para se conseguir isto, optou-se por ordenar as diversas matrizes por ordem decrescente (tem de se ordenar todas de modo a que fiquem coerentes, caso contrário induzir-se-ia em erro). No entanto, é necessário ter em conta que a ordenação tem de ser feita em duas partes: a parte correspondente aos cilindros amarelos e a parte correspondente aos cilindros azuis. Se assim não fosse feito, correr-se-ia o risco de, em vez de se ter as linhas amarelas separadas das azuis, ter-se uma mistura de linhas. A ordenação é feita, obviamente, tendo em conta as distâncias entre os centros. O cilindro que estiver mais longe dos outros será a primeira linha, e por aí adiante. Para isso recorre-se a um vetor auxiliar que possui a norma das distâncias de cada cilindro aos demais. A norma é uma forma prática de avaliar a proximidade já que o que possuir a maior norma é o que se encontra mais distante (ainda que possa estar próximo de um ou outro, estará mais longe dos demais). A figura 4.8 revela a criação desse vetor auxiliar:

```

m = 0
norma_dist = [0 for i in range(dimensao)]
while m <= (dimensao - 1):
    n = 0
    while n <= (dimensao - 2):
        norma_dist[m] += dist_centros_final[m][n] ** 2
        if n == (dimensao - 2):
            break
        n += 1
    if m == (dimensao - 1):
        break
    m += 1
k = 0
while k <= (dimensao - 1):
    norma_dist[k] = math.sqrt(norma_dist[k])
    if k == (dimensao - 1):
        break
    k += 1

```

Figura 4.8 - Criação de um vetor com normas

Neste vetor, o primeiro valor corresponde à norma da distância entre o cilindro 1 e os demais. O segundo valor corresponde à norma da distância entre o cilindro 2 e os demais e por aí fora.

De seguida, ordena-se este valor. Ao se trocar as posições troca-se, simultaneamente, nas matrizes as linhas. Desta forma, assegura-se essa ordenação das matrizes em função da distância entre centros. Este procedimento está dividido em duas partes (uma para cada cor) pelo que se apresentará somente o trecho de um já que são análogos, alterando-se apenas as linhas avaliadas.

```
while p < (dimensao_amarelo - 1):
    o = p + 1
    while o <= (dimensao_amarelo - 1):
        if norma_dist[p] < norma_dist[o]:
            aux = norma_dist[o]
            norma_dist[o] = norma_dist[p]
            norma_dist[p] = aux
            aux1 = dist_centros_final[o][:]
            dist_centros_final[o] = dist_centros_final[p]
            dist_centros_final[p] = aux1
            aux2 = alfa_final[o]
            alfa_final[o] = alfa_final[p]
            alfa_final[p] = aux2
            aux3 = beta_final[o]
            beta_final[o] = beta_final[p]
            beta_final[p] = aux3
            aux4 = limite_up_final[o]
            limite_up_final[o] = limite_up_final[p]
            limite_up_final[p] = aux4
            aux6 = limite_down_final[o]
            limite_down_final[o] = limite_down_final[p]
            limite_down_final[p] = aux6
            lista_completa[[o, p], :] = lista_completa[[p, o], :]
        if o == (dimensao_amarelo - 1):
            break
        o += 1
    if p == (dimensao_amarelo - 2):
        break
    p += 1
```

Figura 4.9 - Ordenação das matrizes

Desta forma, termina o processo de análise e avaliação de parâmetros necessários para a seleção do cilindro, estando-se em condições de entrar na seleção propriamente dita do cilindro.

4.2.2. Determinação do cilindro a ser selecionado

Esta etapa é a mais importante do sistema de visão uma vez que é a responsável pela seleção de um cilindro a ser encaixado numa base cúbica. O algoritmo por detrás desta etapa não é simples nem fácil de ser perceptível pelo que se recorrerá a um desenho e texto para se explicar o processo adjacente. Tome-se o desenho da figura 4.10,

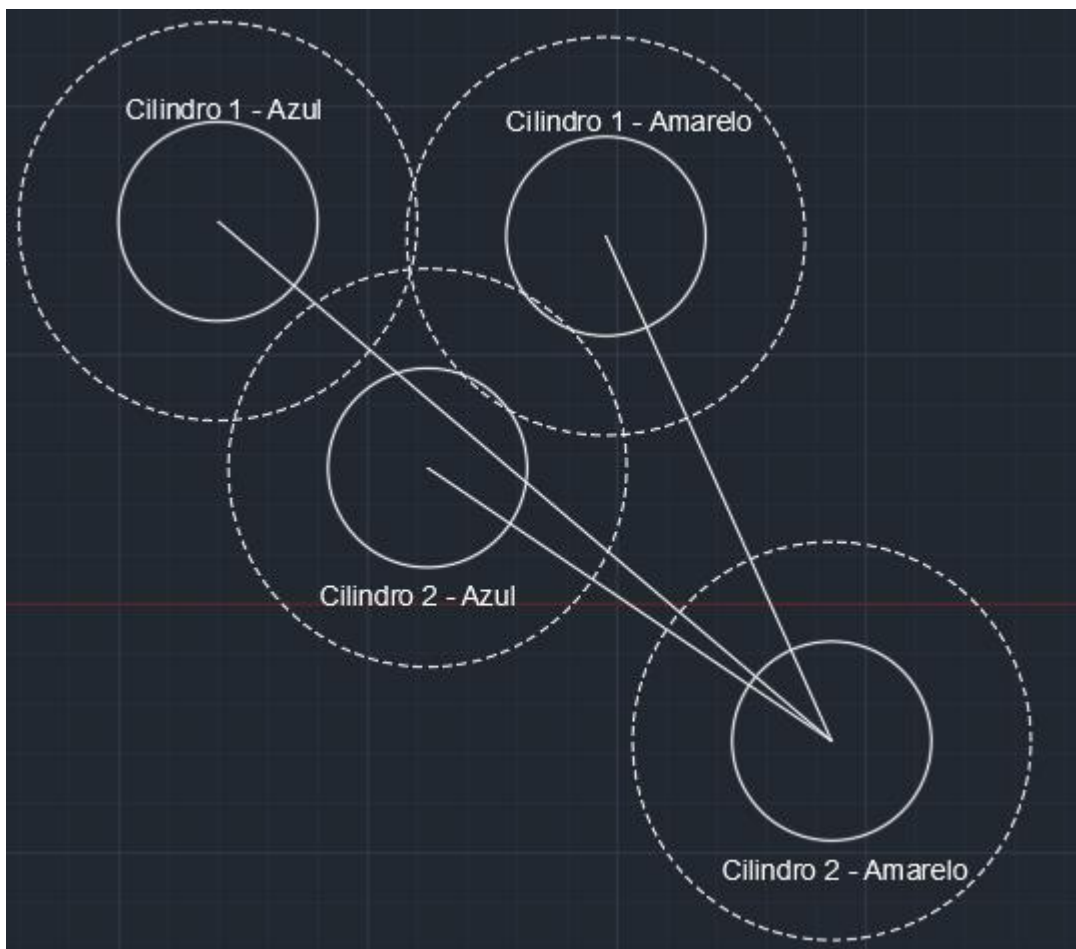


Figura 4.10 - Esquema de seleção de cilindros

As circunferências preenchidas representam o espaço ocupado pelo cilindro no reservatório enquanto que as circunferências a tracejado representam os limites de intervenção da garra do robô, isto é, corresponde ao espaço necessário em redor do cilindro de modo a que se impeça que haja choque entre a garra e um outro cilindro. A forma mais fácil de explicar o processo de de seleção é da seguinte forma:

- O primeiro cilindro a ser avaliado é, como foi dito anteriormente, aquele que se encontra mais distante dos demais. Neste caso é o Cilindro 2 – Amarelo. O algoritmo depois avaliará se na coroa circular existe algum cilindro. Neste caso, não existe e por isso o cilindro 2 é uma solução viável. O algoritmo terminaria neste momento. Contudo, consideremos a avaliação do Cilindro 2 – Azul. Existe uma pequena área de contacto entre a sua coroa circular e o Cilindro 1 Amarelo. Esta área de contacto exige uma rotação do gripper ao longo do seu eixo, como a figura 4.11 demonstra:

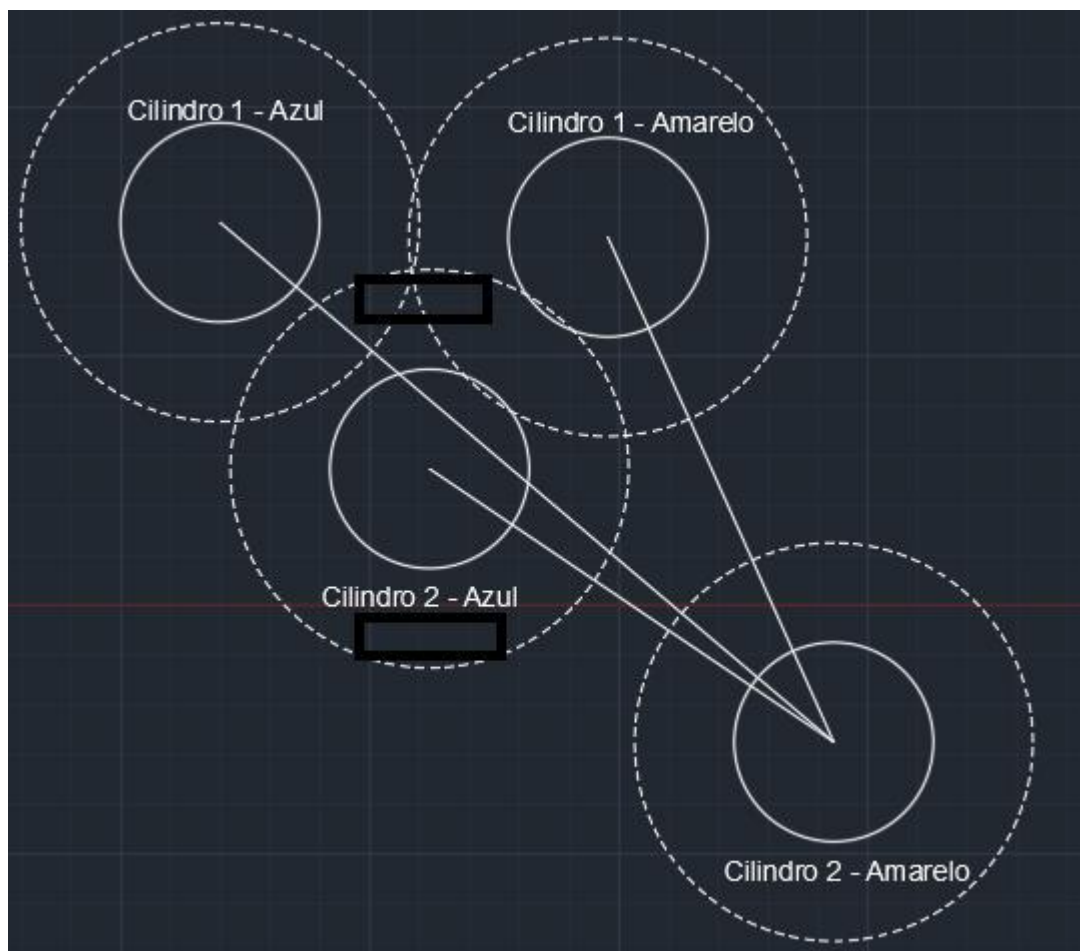


Figura 4.11 - Ajuste da garra

Na figura 4.11, os retângulos pretos representam – de forma muito simples – os dedos da garra. Estes depois podem-se posicionar ao longo da circunferência tracejada. Desta forma, o que o sistema realmente avalia é se existe, para uma dada configuração dos dedos, algum entrave que impeça a recolha do cilindro. Este processo iterativo é realizado para todos os cilindros até que se tenha encontrado uma solução ou caso seja impossível – o que se traduzirá num rearranjo da posição dos cilindros. A posição do cilindro será depois enviada para a aplicação C# que, por sua vez, a envia para o robô.

Finalmente, de modo a se verificar como é que os contornos são desenhados a figura 4.12 apresenta uma captura da câmara onde se pode comprovar que o desenho dos contornos é preciso ainda que o centro dos mesmos se tenda a afastar do real quanto mais próximo o cilindro se encontra dos limites da captura:

Através da figura 4.12, é fácil verificar que a detecção dos centros dos cilindros não é idílica apresentando um desvio tanto maior quanto mais próximo dos limites da captura o cilindro se encontra.

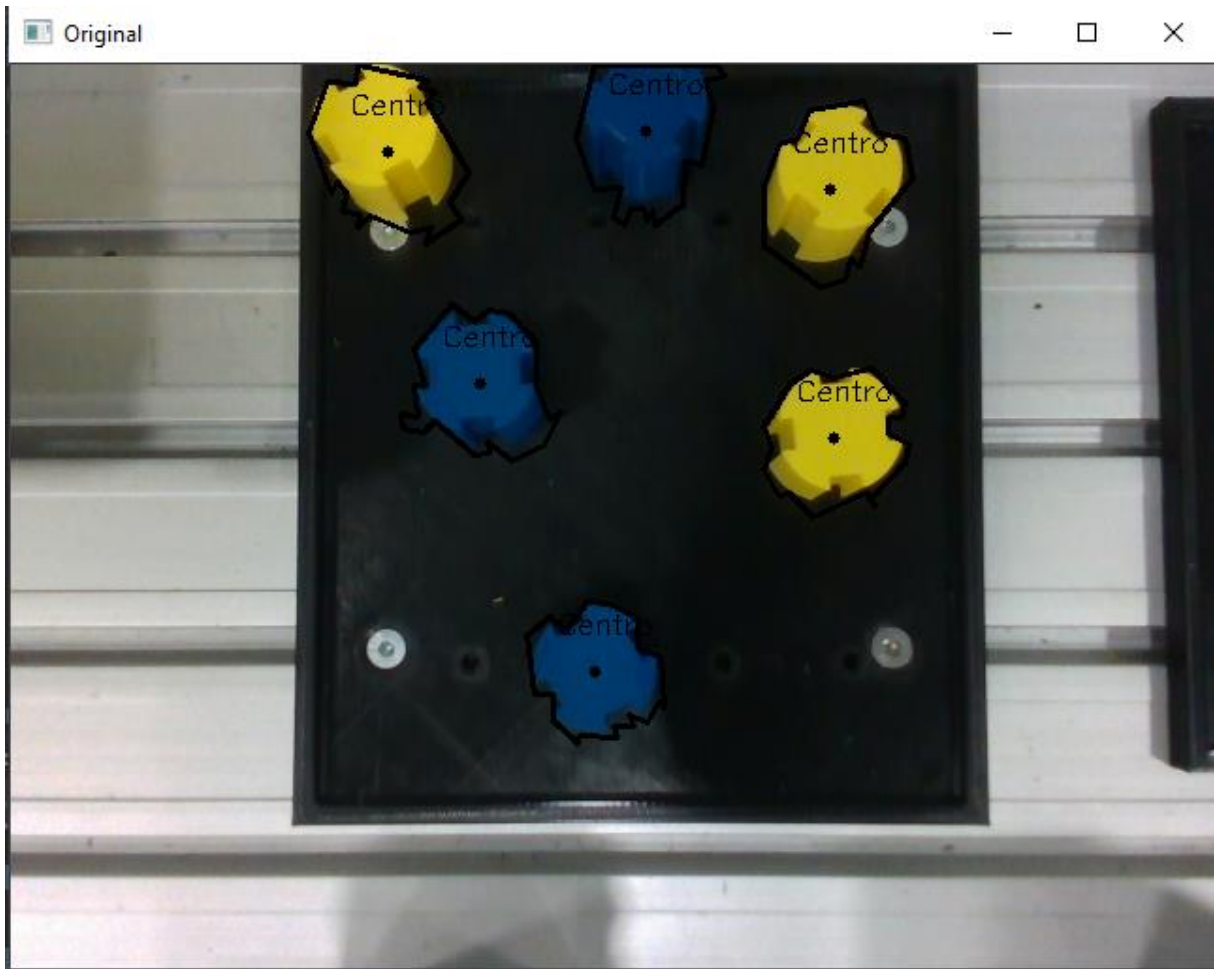


Figura 4.12 - Captura de vídeo da câmara

5 Interface em C#

O interface em C# surgiu da necessidade de se pretender controlar a célula robótica remotamente, isto é, a partir de um computador. O algoritmo por detrás da interface será enunciado ao longo do capítulo enfatizando-se a comunicação entre a aplicação C# e o robô, bem como a recolha de informação do ficheiro Python.

Para já, procede-se à apresentação da interface analisando-se os seus componentes e a função de cada um.

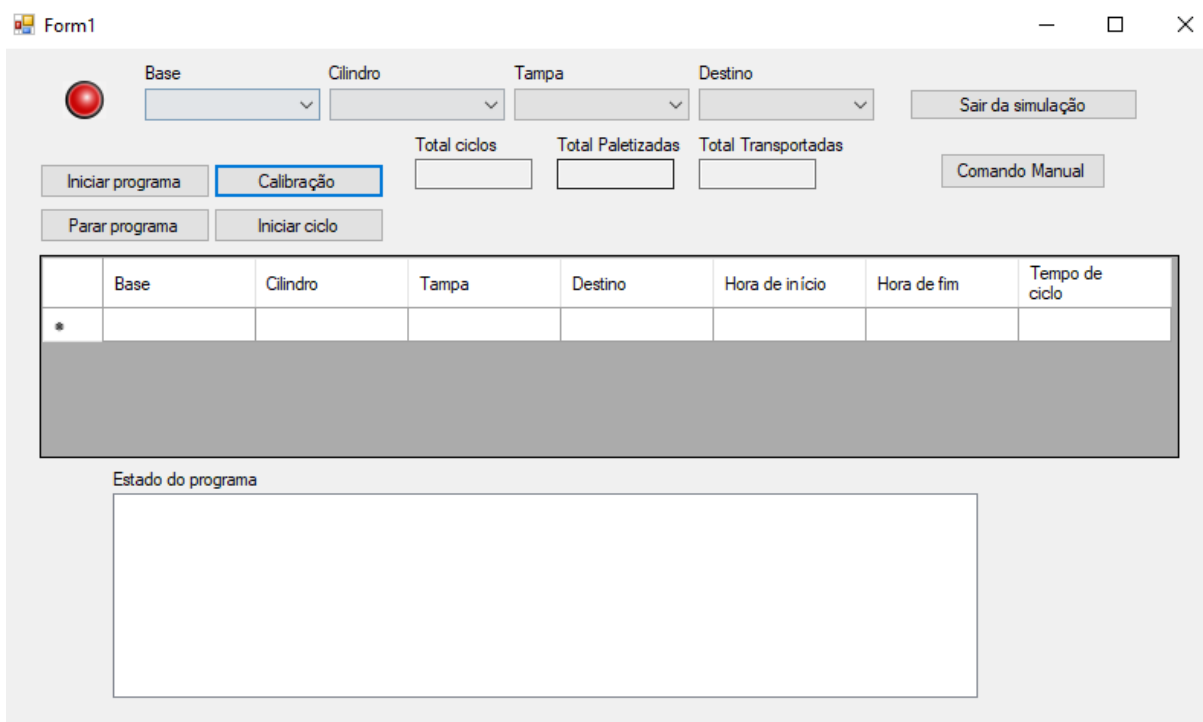


Figura 5.1 - Interface em C#

5.1 Componentes da Interface

5.1.1. Botão Iniciar Programa

Este botão envia a mensagem *play* pela Dashboard do UR5 que, encontrando-se em controlo remoto, inicia a execução do programa. Este botão deve ser premido enquanto o sinalizador possuir a cor vermelha. Caso seja premido noutra circunstância, isto é, caso o sinalizador não exiba a cor vermelha, então o programa já está em execução pelo que o utilizador recebe uma notificação a avisá-lo disso mesmo. Ou seja, este botão apenas exerce algum efeito quando o programa está em modo Parado.

5.1.2. Botão Parar Programa

A função deste botão é muito semelhante ao do seu antagónico. Ao se premir o mesmo, envia-se a mensagem *stop* pela Dashboard do robô parando-se a execução do programa. Este botão

deve ser premido caso se queira interromper o programa e, como tal, apenas surte efeito caso o sinalizador não exiba a cor vermelha. Noutra situação, o utilizador será notificado que o programa já se encontra parado.

5.1.3. Botão Calibração

Este botão permite calibrar o sistema de visão. Como se verificou no capítulo anterior, a calibração é necessária para criar um sistema de coordenadas em relação ao qual o robô recebe as coordenadas dos cilindros e em relação ao qual se movimenta. Este botão apenas pode ser premido uma vez após o programa entrar em execução. Após se premir este botão, aparecerá uma caixa de entrada a pedir para o utilizador escolher a cor do cilindro utilizado para calibração. O utilizador deve depois digitar “Amarelo” ou “Azul” de modo a que calibração seja executada. Seguidamente, a aplicação C# inicia um ficheiro Python que procederá à calibração propriamente dita, enviando as coordenadas do cilindro que serão, por sua vez, enviadas para o robô. Após esta se terminar, aparece uma mensagem para informar o utilizador.

5.1.4. Botão Iniciar Ciclo

Este botão permite que um ciclo se inicie desde que as quatro caixas de seleção tenham uma opção válida. Caso contrário, o utilizador receberá uma mensagem para preencher as mesmas. Além disso, o ciclo só se inicia caso se tenha realizado previamente a calibração do sistema de visão. Ademais, aparecerão caixas de entrada ao utilizador caso se detete uma anomalia – registada na lista. Estas anomalias podem ser de quatro tipos:

- Falta de bases

Caso o utilizador pretenda uma base amarela mas tal não exista no sistema, ele será avisado e poderá escolher entre colocar uma base amarela no respetivo alimentador ou utilizar uma base de outra cor (contanto que esta se encontra disponível). O ciclo não arranca até o utilizador ter escrito “Retomar Ciclo” – depois de colocar uma base amarela no sistema – ou “Utilizar uma base de outra cor” – tendo verificado que tal existe no sistema. O mesmo é análogo caso seja uma base azul.

- Falta de cilindros/Impossibilidade de seleção

Relativamente aos cilindros podem surgir duas situações: a não-existência de cilindros de uma determinada cor ou a não-acessibilidade a um cilindro da cor pretendida. Tanto num caso como noutro, o utilizador tem sempre a possibilidade de rearranjar ou introduzir cilindros no reservatório dos mesmos para depois prosseguir com o ciclo. Além destas duas alternativas, pode ainda escolher avançar sem a seleção de um cilindro. As alternativas são: “Alterar a posição dos cilindros”, “Utilizar um cilindro de outra cor” e “Prosseguir sem o cilindro”.

- Falta de tampas

A situação das tampas é análoga à das bases possuindo-se somente uma terceira opção que consiste em prosseguir sem a tampa. As opções de escolha são “Retomar ciclo”, “Utilizar uma tampa de outra cor” e “Prosseguir sem a tampa”.

- Limite de paletização atingido

Caso o limite de paletização tenha sido atingido, o utilizador poderá escolher entre colocar a embalagem no *conveyor* ou terminar o ciclo. As opções são “Transportar para o Conveyor” e “Terminar o ciclo”.

Terminando-se o ciclo, o utilizador recebe uma mensagem a notificá-lo além de ser introduzida uma nova linha na tabela de registos. Qualquer alteração da embalagem ou do destino da mesma – devido a anomalias – é tida em conta de modo a que não se registre as condições iniciais, mas sim as que se concretizaram.

5.1.5. Botão Comando Manual

O botão Comando Manual abre uma nova interface que permite o acionamento manual das eletro-válvulas e dos *conveyors*. O Comando Manual só se pode iniciar caso já se tenha realizado a calibração e caso nenhum ciclo esteja a ocorrer. Por questões óbvias, o controlo manual só deve ser possível se o sistema não estiver em funcionamento. Caso contrário, poder-se-ia impedir que um ciclo terminasse ou iniciasse sequer (por exemplo, caso se impedisse o acionamento da válvula ou do *conveyor*).

5.1.6. Botão Sair da Simulação

O botão Sair da Simulação fecha a aplicação C#. Devido a limitações da aplicação, este botão apenas surte efeito se nenhuma ação estiver a ocorrer (ciclo ou calibração).

5.1.7. Caixa de Seleção 1

A caixa de seleção 1 possui duas opções: Amarela e Azul. É através desta caixa que o utilizador escolhe a cor da base. Como se mencionou anteriormente, o ciclo não arranca até as quatro caixas terem uma opção. Por motivos logísticos, é impossível um ciclo se processar caso não haja uma base, daí que a opção Nenhuma não esteja presente como está nas outras.

5.1.8. Caixa de Seleção 2

A caixa de seleção 2 possui três opções: Amarelo, Azul e Nenhum. É através desta caixa que o utilizador decide se quer um cilindro e, caso queira, qual a cor do mesmo.

5.1.9. Caixa de Seleção 3

Análogo à caixa de seleção 2 só que relativa à tampa.

5.1.10. Caixa de Seleção 4

A caixa de seleção 4 possui três opções: *Conveyor*, *Paletizar* e *Nenhum*. É através desta que o utilizador escolhe o destino final da embalagem.

5.1.11. Sinalizador

O sinalizador indica o ponto de situação do programa. Caso seja vermelho, tal significa que o programa está parado; se for verde, um ciclo está a decorrer. A cor amarelo reflete as restantes situações.

5.1.12. Tabela de registo de ciclos

Ao fim de cada ciclo, regista-se a constituição da embalagem, o destino da mesma bem como o marco temporal de início e fim de ciclo, além da duração do mesmo. Esta tabela constitui um histórico de ciclos realizados.

5.1.13. Caixa de Texto 1 - Total ciclos

Esta caixa de texto regista o número total de ciclos que foram realizados desde que o programa está em execução. Este valor corresponde à soma de ciclos que terminaram em paletizações, transportes e os que não possuíram nenhum destino final.

5.1.14. Caixa de Texto 2 - Total Paletizações

Esta caixa de texto regista o número total de embalagens paletizadas. Como se mencionou no Capítulo 3, apenas se pode paletizar no máximo quatro peças. Assim, este é o valor limite que poderá aparecer nesta caixa. Caso nenhuma se realize, tomará o valor 0.

5.1.15. Caixa de Texto 3 - Total Transportes

Esta última caixa de texto indica o número total de embalagens cujo destino final foi circular no *conveyor* final. Caso se pretendesse que uma embalagem fosse paletizada, mas tal seja impossível – devido a se ter atingido o limite máximo da palete, e como alternativa se optasse por a transportar para o *conveyor*, essa embalagem será contabilizada.

5.1.16. Lista de anomalias

A lista presente na interface regista todos os erros que surjam num ciclo. A melhor forma de enunciar o que um erro representa é por via de exemplo. Suponhamos um ciclo em que se pretendesse uma embalagem constituída por uma base amarela e cilindro azul. O destino final pode ser paletização. Caso não existissem bases amarelas no sistema ou cilindros azuis, ou caso se tivesse atingido um limite de paletização, o utilizador seria notificado com uma mensagem que apareceria nesta lista.

A interface referente ao Comando Manual encontra-se na figura 5.2,



Figura 5.2 - Interface do Comando Manual

5.2 Componentes da interface de Comando Manual

5.2.1. Botão Iniciar Comando Manual

Este botão inicia um temporizador que enviará a cada 50 milissegundos a informação relativa ao estado dos quadros de seleção e, simultaneamente, receberá *feedback* do estado dos sensores. O temporizador envia um vetor com 7 componentes (6 relativos ao estado das saídas, o último igual a 1 para sinalizar que o comando manual está ativo).

5.2.2. Botão Sair do Comando Manual

Este botão pára o temporizador e envia uma última o vetor de 7 componentes só que com todas igual a 0 – simbolizando o fim do comando manual. Além disso, ao se sair deste modo, a janela da interface é fechada, retornando-se à principal.

5.2.3. Quadros de seleção

Os quadros de seleção exercem todos uma função semelhante. Cada um deles corresponde a um elemento de um vetor que é enviado para o robô. Caso o quadro esteja selecionado, o elemento possui o valor 1; caso contrário, terá o valor 0. Em função destes valores, o robô colocará as suas saídas ora a 1 ora a 0, respetivamente. Os quadros de seleção dos *Conveyors* não podem ser selecionados em simultâneo, isto é, não se pode selecionar Conveyor 1 para a

direita e Conveyor 1 para a esquerda. Caso um esteja selecionado e se selecione o outro, o primeiro terá a sua seleção anulada e vice-versa.

5.2.4. Sinalizadores de sensores

Os sinalizadores de sensores são atualizados conforme a atuação dos mesmos. O robô envia informação para a aplicação C# com um vetor que traduz o estado de cada sensor – 0 ou 1. Caso seja 0, o sensor terá a cor vermelha; caso contrário, terá a cor verde.

Terminada a análise das interfaces, apresentam-se alguns excertos de programação para salientar a comunicação existente entre a aplicação C# e o robô. Nesta última parte deste capítulo encontrar-se-ão pequenos trechos de algoritmo utilizados quer para iniciar ou parar o programa do robô, quer para enviar ou receber informação do mesmo. De modo a que este capítulo não se torne demasiado extenso e maçudo, apresentar-se-à somente um trecho de cada tipo, isto é, um trecho relativo à comunicação via Dashboard, outro trecho relativo à comunicação via Socket, e um trecho com algoritmo geral. Antes destes exemplos serem apresentados, deve-se salientar a diferença entre a comunicação Dashboard e Socket.

Na comunicação Dashboard, o robô é o servidor enquanto que a aplicação C# é o cliente. A porta da Dashboard é, impreterivelmente, a 29999 e através desta consegue-se ordenar o início e a paragem de um programa; retirar o estado (a correr, parado, etc); carregar uma instalação ou um programa, etc... Como se mencionou anteriormente, esta comunicação foi utilizada somente para ordenar o início e fim da paragem do programa.

Na comunicação Socket, o robô é o cliente enquanto que a aplicação C# é o servidor. Pode-se utilizar qualquer porta neste tipo de comunicação, desde que não seja uma à qual já esteja atribuída alguma função interna - como por exemplo a 29999. Através desta comunicação pode-se enviar strings, bytes, inteiros, etc...Ao longo do projeto, esta comunicação foi utilizada para enviar strings e listas (vetores).

De seguida, referencia-se alguns desses trechos que revelam os três tipos de algoritmos:

5.3 Excertos de algoritmos

5.3.1. Comunicação via Dashboard

```
IPEndPoint ip = new IPEndPoint(IPAddress.Parse("192.168.0.150"), 29999);
Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
server.Connect(ip);
string input = "play";
server.Send(Encoding.ASCII.GetBytes(input));
server.Shutdown(SocketShutdown.Both);
server.Close();
```

Figura 5.3 - Excerto referente à comunicação via Dashboard

Neste excerto, o servidor é definido na primeira linha através do IP e referencia-se a porta que será utilizada para a comunicação. O IP “192.168.0.150” corresponde ao UR5. A *string* “play” será interpretada pelo robô como uma ordem para iniciar o programa. Esta ordem é enviada para o robô, terminando-se assim esta breve comunicação. Este excerto está incluído no botão Iniciar Programa da interface principal.

5.3.2. Comunicação via Socket

```
const int PORT_NO = 10;
const string SERVER_IP = "192.168.0.100";
IPAddress localAdd = IPAddress.Parse(SERVER_IP);
TcpListener listener = new TcpListener(localAdd, PORT_NO);
listener.Start();
TcpClient client = listener.AcceptTcpClient();
NetworkStream nwStream = client.GetStream();
byte[] coordenada = ASCIIEncoding.ASCII.GetBytes(calibracao);
nwStream.Write(coordenada, 0, coordenada.Length);
listener.Stop();
client.Close();
```

Figura 5.4 - Excerto referente à comunicação via Socket

No trecho da figura 5.4, encontra-se nas duas primeiras linhas a definição de uma porta e do IP do servidor. O IP “192.168.0.100” corresponde ao IP do computador e a porta 10 é simplesmente uma porta que se encontra disponível. Para se enviar a informação por este método é preciso primeiro converter a mensagem em bytes que são depois enviados e convertidos na mensagem original. Este processo é sempre realizado quer se trate de uma *string* ou de outra variável qualquer. Este pequeno excerto corresponde a parte da calibração, que será aprofundada no ponto 5.3.3, como exemplificação de algoritmo geral.

5.3.3. Algoritmo geral

Existem várias linhas de código presentes ao longo da interface e todas elas merecem a sua atenção já que são igualmente relevantes para o correto funcionamento da mesma. Contudo, sendo impossível analisar linha a linha, optou-se por colocar o pequeno excerto relativo à calibração do sistema de visão.

```

const int PORT_NO = 10;#1
const string SERVER_IP = "192.168.0.100";#2
IPAddress localAdd = IPAddress.Parse(SERVER_IP);#3
TcpListener listener = new TcpListener(localAdd, PORT_NO);#4
listener.Start();#5
TcpClient client = listener.AcceptTcpClient();#6
string cor_cilindro = "";#7
while (cor_cilindro != "Amarelo" && cor_cilindro != "Azul")#8
{#9
    cor_cilindro = Interaction.InputBox("Insira a cor do cilindro", "Calibração", "", -1, -
1);#10
    if (cor_cilindro != "Amarelo" && cor_cilindro != "Azul")#11
    {#12
        MessageBox.Show("Digite Amarelo ou Azul");#13
    }#14
}#15
if (cor_cilindro == "Amarelo")#16
{#17
    NetworkStream nwStream = client.GetStream();#18
    var psi = new ProcessStartInfo();#19
    psi.FileName = @"C:\Users\Pedro Monforte\output\Calibracao_Amarelo.exe";#20
    var script = @"D:\Projetos_python\pythonProject\Calibracao_Amarelo.py";#21
    var start = "2019-1-1";#22
    var end = "2019-1-22";#23
    psi.Arguments = $"{script}\" \"{start}\" \"{end}\"";#24
    psi.UseShellExecute = false;#25
    psi.CreateNoWindow = true;#26
    psi.RedirectStandardOutput = true;#27
    psi.RedirectStandardError = true;#28
    var error = "";#29
    var calibracao = "";#30
    using (var process = Process.Start(psi))#31
    {#32
        error = process.StandardError.ReadToEnd();#33
        calibracao = process.StandardOutput.ReadToEnd();#34
    }#35
    byte[] coordenada = ASCIIEncoding.ASCII.GetBytes(calibracao);#36
    nwStream.Write(coordenada, 0, coordenada.Length);#37
    listener.Stop();#38
    client.Close();#39
}

```

Figura 5.5 - Excerto de Calibração

Na figura 5.5 encontra-se metade do algoritmo do botão Calibração. Não se introduziu a totalidade do código uma vez que seria redundante. O algoritmo em si é igual caso o cilindro seja azul, alterando-se somente o ficheiro Python a ser corrido. A única parte que se acrescentaria seria uma condição semelhante mas caso o cilindro selecionado fosse amarelo. A parte inicial corresponde à comunicação Socket já analisada. Seguidamente, as linhas 8-15 representam a caixa de entrada na qual o utilizador digita se pretende utilizar um cilindro amarelo ou azul para a calibração. Esta caixa não desaparece até a resposta ser válida. As restantes linhas evidenciam a aplicação C# a correr um ficheiro Python que permite a realização da calibração. O resultado dessa calibração – guardado na variável “calibracao” – é depois enviado para o robô.

6 Programação na consola

Neste último capítulo, apresentam-se alguns excertos da consola de programação de modo a se introduzir esta linguagem de programação que é própria de robôs. Focou-se nas principais operações que o robô efetua de modo a se salientar as suas especificidades. Além destas, introduziram-se segmentos de movimentos de modo a que se possa classificar e distinguir os dois tipos de movimentos utilizados: movimento de juntas e movimento linear. Estes excertos não cobrem, naturalmente, todo o processo de programação tendo-se tido o cuidado de colocar os mais importantes e/ou relevantes.

6.1 Comunicação via Socket

O excerto da figura 6.1 apresenta a primeira fase do programa – correspondente à calibração. Neste excerto estabelece-se a comunicação via Socket e recebe-se a coordenada do cilindro utilizado pelo sistema de calibração. Optou-se colocar excerto para mostrar como é que o processamento dessa informação é realizado.

```

7      open_calibracao:=socket_open("192.168.0.100",10)
8      Loop open_calibracao≠ False
9          Aguardar: 0.3
10         open_calibracao:=socket_open("192.168.0.100",10)
11      Loop passo_inicial[0]≠6
12          passo_inicial:=socket_read_ascii_float(6)
13          Aguardar: 0.1
14          x:=passo_inicial[1]
15          y:=passo_inicial[2]
16          z:=passo_inicial[3]
17          rx:=passo_inicial[4]
18          ry:=passo_inicial[5]
19          rz:=passo_inicial[6]
20          ponto_calib:=p[x,y,z,rx,ry,rz]
21          origem:=pose_trans(Ponto_Ref,ponto_calib)
22      socket_close()

```

Figura 6.1 - Calibração

A figura 6.1 comprova que o programa espera que a comunicação Socket seja estabelecida – linhas 7 - 10. Após a comunicação ter sido estabelecida, o robô aguarda a receção das coordenadas do cilindro calibrado. O vetor `passo_inicial` é um vetor de dimensão sete, em que a primeira coordenada corresponde ao número 6. Esta forma de processamento de vetores é tipicamente característica desta linguagem de programação. Sempre que um vetor é enviado, por exemplo, o vetor `[0, 0, 0]`, o robô interpreta-o da forma `[3, 0, 0, 0]`, sendo que a primeira

coordenada corresponde à dimensão do vetor. Ou seja, para o robô ler um vetor com seis coordenadas, precisa de ter uma dimensão de, pelo menos, sete. O segundo *Loop* garante que o programa só corre caso o vetor seja não nulo – daí se aguardar que a primeira coordenada seja igual a 6. Caso contrário, seria 0 – como é inicialmente. Estabelecido esse vetor, retira-se a informação útil do mesmo, isto é, as coordenadas propriamente ditas, criando-se um ponto com as mesmas – linhas 14 – 20. Para se definir a origem do referencial, basta realizar uma operação de *pose_trans* entre o ponto de referência e o ponto de calibração. O ponto de referência é um ponto guardado na memória do robô que corresponde à posição de calibração, isto é, a calibração – para ser realizada corretamente – deve ocorrer sempre para a mesma posição. Esta posição – onde o cilindro é colocado – é igual (em coordenadas x e y) à do ponto de referência. A função *pose_trans* permite que o robô se movimente em relação a um determinado ponto ou referencial. Ao se mover o robô nas coordenadas recebidas em relação ao ponto de referência, estar-se-ia a mover o robô para a origem do referencial. O ponto origem é gravado com essa posição que corresponde à origem do referencial. O seguinte exemplo permite perceber o que acontece com a calibração. Suponhamos que temos o ponto de referência com coordenadas [0.2, 0.1, 0.2, 0, 0, 0]. Estas coordenadas estão definidas não em relação à origem do referencial da câmara, mas sim em relação ao referencial da base. Se as coordenadas do ponto de calibração forem [0.1, 0.3, 0, 0, 0, 0], então isto significa que a origem tem as seguintes coordenadas [0.1, -0.2, 0.2, 0, 0, 0] – uma vez que o ponto de referência e o ponto de calibração são iguais, apenas a componente em z foi alterada por motivos de segurança. Contudo, deve-se mencionar que caso o referencial da câmara fosse paralelo ao referencial base, a coordenada da origem seria [0.3, 0.4, 0.2, 0, 0, 0] – uma vez que a função *pose_trans* trata as coordenadas como tendo o sentido dos eixos do referencial base. Devido a esta característica, quando as coordenadas do cilindro a ser encaixado forem recebidas, terão de ser multiplicadas por -1 e interpretadas como uma translação relativa à origem e não em relação ao ponto de referência.

6.2 Conveyor Tracking

O *Conveyor Tracking* foi realizado no tapete transportador final, optando-se para que este se movesse na vertical enquanto acompanhava o movimento do *conveyor*. É necessário realçar que esta é a única parte de algoritmo que nunca foi testada uma vez que os encoders nunca foram recebidos. A situação pandémica interferiu com os prazos de entrega e, consequentemente, os encoders – encomendados em abril – nunca chegaram a ser entregues. Apesar disso, elaborou-se o algoritmo que afeta o mesmo embora, na simulação propriamente dita, este nunca seja efetuado.

Para se realizar esta operação é preciso primeiro defini-la. Este setup é realizado na figura 6.2:

Configuração da correia transportadora

Conveyor 1

☒ Ativar acompanhamento da correia transportadora

Parâmetros da Correia Transportadora

Tipo de Codificador

Incremental

Modo

Quadratura

A Entrada

digital_in[8]

B Entrada

digital_in[9]

Acompanhamento de parâmetros

Selecione o tipo de correia transportadora

Linear

Ocorrências por metro

2547

Direção

✎

Linha_1

⇌ Inverter sentido

Figura 6.2 - Setup do Conveyor Tracking

O encoder é do tipo incremental – como já foi mencionado – e em quadratura. As entradas 8 e 9 correspondem a entradas específicas para os sinais A e B do encoder. O movimento é linear e as ocorrências por metro podem ser calculadas da seguinte forma:

O encoder possui uma resolução de 1 impulso por grau. O diâmetro do tambor é de 45 mm, o que resulta num perímetro de, aproximadamente, 141.4 mm. Logo, cada impulso cobre cerca de 0.393 mm. Logo, existem 2.547 impulsos por mm, isto é, 2547 por metro. A direção corresponde à direção do tapete transportador, sendo definida por uma linha.

O algoritmo de programação encontra-se na figura 6.3,



Figura 6.3 - Conveyor Tracking

Neste algoritmo, o robô vai-se movimentar na vertical – segundo um movimento linear – ao longo do tapete transportador. Mais uma vez salienta-se que foi impossível testar este código uma vez que os encoders nunca foram entregues. Caso alguma alteração fosse necessária, apenas poderia surgir depois de se testar o programa.

6.3 Controlo de força

O controlo de força pode ser aplicado tanto em robôs colaborativos como tradicionais, embora a solução em robôs colaborativos seja mais eficiente e prática uma vez que já está incorporada. Tal como na etapa anterior, existe um setup de controlo de força onde se escolhe sobre que referencial se pretende exercer a mesma; o seu módulo e direção.

Serão apresentadas duas formas de controlo de força, já que são utilizados dois tipos do mesmo durante o encaixe do cilindro. Na primeira fase, é aplicada força somente na direção vertical até o cilindro ter atingido a base cúbica; atingindo-se esta fase, além de se aplicar força na vertical, aplica-se um binário para a peça rodar em torno do seu eixo de modo a que se processe o encaixe.

A figura 6.4 traça o setup do primeiro controlo de força:

Além do referencial, escolhe-se o módulo da força bem como a velocidade de movimento.

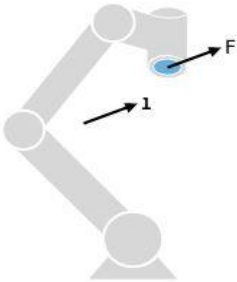
Força

Frame

A parte do programa sob este comando de força será executada em modo de força. No modo de força, o robô estará livre na direção do recurso selecionado para aplicar a força especificada.

Feature

Ferramenta



	Força	Limites de Velocidade
☐ X	N	mm/s
☐ Y	N	mm/s
☒ Z	1 N	150 mm/s
☐ RX	Nm	°/s
☐ RY	Nm	°/s
☐ RZ	Nm	°/s

Use o botão de teste abaixo em combinação com o botão de condução livre no Teach Pendant para testar o modo de força.

Teste

Figura 6.4 - Setup de força unidirecional

A figura 6.5 constitui o setup do segundo controle de força:

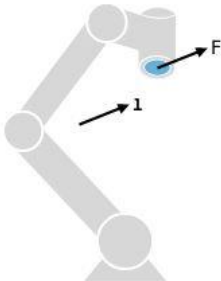
Força

Frame

A parte do programa sob este comando de força será executada em modo de força. No modo de força, o robô estará livre na direção do recurso selecionado para aplicar a força especificada.

Feature

Ferramenta



	Força	Limites de Velocidade
☐ X	N	mm/s
☐ Y	N	mm/s
☒ Z	1 N	150 mm/s
☐ RX	Nm	°/s
☐ RY	Nm	°/s
☒ RZ	-1 Nm	10 °/s

Use o botão de teste abaixo em combinação com o botão de condução livre no Teach Pendant para testar o modo de força.

Teste

Figura 6.5 - Setup de força bidirecional

A única diferença mantém-se na existência de um controlo de binário segundo a direção zz.

O comando teste permite ao utilizador verificar se direção, módulo e velocidade do controlo de força satisfazem os requisitos.

Estabelecido o setup do controlo de força, basta apenas inseri-lo na árvore do programa para que este possa ser efetuado.



Figura 6.6 - Algoritmo de força

A partir deste algoritmo, verifica-se que o controlo de força é realizado de duas formas. Primeiramente, até atingir a base cúbica; seguidamente até se encaixar o cilindro. As ordens presentes indicam que o sistema de força opera até se verificar as condições presentes. A partir desse momento, ele para.

Os módulos de força e as condições de comparação foram todos determinados experimentalmente.

6.4 Rotinas Paralelas

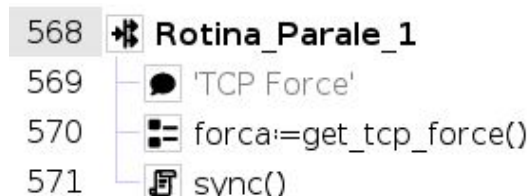


Figura 6.7 - Rotina Paralela de Força

Como foi dito anteriormente, o controlo de força espera que um vetor que possui a força do TCP atinja um determinado valor para deixar de exercer. Este valor é determinado empiricamente, após vários ensaios de modo a que se consiga encaixar o cilindro e, simultaneamente, impedir qualquer desgaste da peça. Este vetor – que recolhe os valores da força do TCP – encontra-se numa rotina paralela, executada simultaneamente com o programa.

6.5 Paletização

A paletização não seguiu o modelo já existente na biblioteca do UR5 uma vez que as embalagens podem possuir alturas diferentes e existe sempre algum desvio de posição aquando da sua paletização. Deste modo, optou-se por utilizar o sensor de força, ainda que não se tenha utilizado nenhum controlo de força. O excerto da figura 6.8 – em conjunto com suporte textual – esclarecerá qualquer dúvida.

464		📍 ➡	Movimento L
465		📍 ▼	Direção: Ferramenta Z+
466		➡	Até ao contacto: Retrair 3 mm
467		📍 ▼	Direção: Ferramenta Y-
468		➡	Até ao contacto: Retrair 0 mm
469		📍 ▼	Direção: Ferramenta Z+
470		➡	Até ao contacto: Retrair 0 mm

Figura 6.8 - Paletização

A paletização realiza-se do seguinte modo: o robô dirige-se para o início da paleta, movimenta-se na vertical até atingir a mesma e retrai 3 mm. O sensor de força deteta quando o robô atingiu a paleta ou outra superfície qualquer que esteja na trajetória do robô. É necessário que ele retraia para se evitar o desgaste de material uma vez que o seguinte passo consiste em movimentá-lo ao longo da paleta. Seguidamente, o robô move a embalagem ao longo da paleta até ter atingido o fim da mesma. Atingindo este ponto, movimenta-se na vertical até atingir a superfície de modo a que a paletização seja o mais fiável possível. Por outras palavras, para se evitar demasiados desvios de posição que irão afetar as sucessivas paletizações.

6.6 Alguns movimentos gerais

Apresenta-se agora a distinção entre os dois tipos de movimentos utilizados ao longo do programa: movimento de juntas e movimento linear. Além disso, apresenta-se o setup de cada movimento de modo a que se perceba o que é mais importante definir em cada um.

6.6.1. Movimento de Juntas (Movimento J)

O movimento de juntas é um movimento adequado quando se pretende que todas as juntas atinjam a respetiva posição o mais rápido possível. Isto é, neste tipo de movimento, o robô não segue nenhuma trajetória específica – linear ou circular – mas movimenta-se da forma que mais rápido lhe permite atingir a posição desejada. Por outras palavras, a trajetória é definida de modo a que cada junta atinja a posição desejada o mais rapidamente possível. Na figura 6.9 , encontra-se o setup de um movimento de juntas:

Movimento Movimento J ▼

Especifica como o robô se moverá entre os pontos.

Os valores abaixo se aplicam a todos os pontos de rota do grupo e dependem do tipo de movimento selecionado acima.

Definir TCP Usar TCP ativo ▼	Velocidade da Articulação 30,0 °/s
Feature Base ▼	Aceleração da Articulação 40,0 °/s²

☐ Usar ângulos de rotação das articulações

Reinício

Figura 6.9 - Setup de Movimento J

Como qualquer outro movimento, é necessário definir o TCP do robô bem como o referencial em relação ao qual se está a mover o mesmo. Como este movimento pretende mover as juntas de modo a se atingir o mais rapidamente uma determinada posição, é necessário definir a velocidade de aceleração de cada articulação.

6.6.2. Movimento Linear (Movimento L)

O movimento linear é um movimento utilizado quando se pretende que o robô se mova segundo um segmento de reta formado entre dois pontos. Este tipo de movimentos é muito utilizado quando se pretende que o robô se mova segundo um eixo ou que se mova segundo uma trajetória rigorosamente linear.

A figura 6.10 apresenta o setup deste movimento:

Movimento

Movimento L ▼

Especifica como o robô se moverá entre os pontos.

Os valores abaixo se aplicam a todos os pontos de rota do grupo e dependem do tipo de movimento selecionado acima.

Definir TCP

Usar TCP ativo ▼

Velocidade da Ferramenta

100,0 mm/s

Feature

Base ▼

Aceleração da Ferramenta

500,0 mm/s

☐ Usar ângulos de rotação das articulações**Reinício**

Figura 6.10 - Setup de Movimento L

Terminando-se com a avaliação destes excertos de programação, dá-se por concluída a análise da célula robótica – a sua constituição, funcionamento, especificidades, etc – e procede-se a uma última fase da dissertação que consiste em verificar se a mesma cumpriu com os requisitos originais e o impacto que tal poderá ter em futuros alunos de robótica colaborativa.

7 Conclusões

Nesta dissertação, desenvolveu-se uma célula robótica colaborativa compacta, organizada, flexível e apropriada à formação de quem se pretende integrar no crescente mundo da robótica colaborativa. Foram incluídas características próprias dos robôs colaborativos de modo a se realçar a sua especificidade em relação aos robôs tradicionais e para aprofundar o conhecimento e estudo dos primeiros. As peças desenvolvidas por impressão 3D foram todas desenhadas pelo mestrando – ora mediante conselho dos orientadores ora mediante iniciativa própria. Para cada um dos tópicos apresentados a seguir, traçam-se as considerações finais:

7.1 Complexidade e Inovação da célula

Considera-se que a célula robótica cumpre os requisitos de complexidade e inovação que foram estabelecidos entre o mestrando e os orientadores no início da dissertação. Ainda que se pretendesse introduzir outras tecnologias – em particular a leitura de código de barras ou por RFID, conclui-se que as demais funções e a flexibilidade do interface em C# e do sistema de visão são suficientes para responder ao problema inicial. Contudo, devido à situação pandémica a operação de *Conveyor Tracking* não pôde ser realizada uma vez que o prazo de entrega dos encoders superava 6 meses. Apesar disso, entende-se que a célula introduz o utilizador a uma série de conceitos e operações típicas de robôs colaborativos, constituindo uma boa plataforma de aprendizagem.

7.2 Eficácia do Sistema de Visão

Já no capítulo 5 se mencionou que o sistema de visão não é idílico. A conversão de pixels para milímetros bem como o correto posicionamento do centro do cilindro afetam a precisão do mesmo. Ainda que o sistema possua as suas falhas, o seu funcionamento comprovou-se pelo que nada leva a crer que este não seja eficaz. Foram realizados vários ciclos, com cilindros em diferentes posições e nenhum erro grave surgiu. Com grave, identifica-se um erro que impedisse a realização do ciclo. Como nenhum surgiu, infere-se que o sistema de visão é eficaz.

7.3 Dificuldades Experienciadas

A conceção dos elementos da célula (alimentadores, *conveyores*, rampas, etc) foi uma das principais dificuldades sentidas ao longo do projeto – tendo ocupado grande parte do mesmo. Esta dificuldade prendeu-se, sobretudo, com a harmonia entre os elementos obtidos por impressão 3D e os elementos normalizados. Nos primeiros, surgiram dificuldades aquando da projecção de um alimentador que permitisse a circulação de bases cúbicas; quanto aos elementos

normalizados, as dificuldades foram sentidas ao se ter de contactar com diversas empresas e fornecedores que pudessem responder aos pedidos. Além disso, a necessidade de um encoder com os terminais alimentados a 24V impediu que o mesmo fosse incorporado no motor DC – o que resultou em novas procuras por encoders à parte e, consequentemente, em alterações na conceção dos *conveyors*. A elaboração de um esquema elétrico provou também ser uma dificuldade uma vez que a definição do número de entradas e saídas do sistema só ficou resolvido perto do final do projeto. A programação em Python do sistema de visão e a constituição da interface em C# também constituíram um problema devido à inexperiência e dificuldade de programação.

7.4 Trabalhos Futuros

Existem várias perspetivas que visam melhorar o aspeto e a funcionalidade da célula colaborativa. Naturalmente que um trabalho futuro seria aplicar o *Conveyor Tracking*. Além disso, poder-se-ia aplicar uma leitura RFID e talvez até código de barras nas bases cúbicas, sendo distinguidas não somente pela cor, mas também por esta identificação. Além disso, uma tecnologia que poderia ter sido aprofundada era o *Machine Learning* como substituto da deteção de contornos. Outra adição à célula poderia ser a introdução de um autómato – um PLC – além de outros sistemas de segurança – botoneiras de emergência, sinalizadores luminosos, etc..Por fim, a interface poderia ainda ser melhorado para impedir que os botões deixassem de funcionar quando um ciclo está a decorrer. Desta limitação resulta que o programa – para ser parado durante o ciclo – apenas o pode ser com recurso à consola.

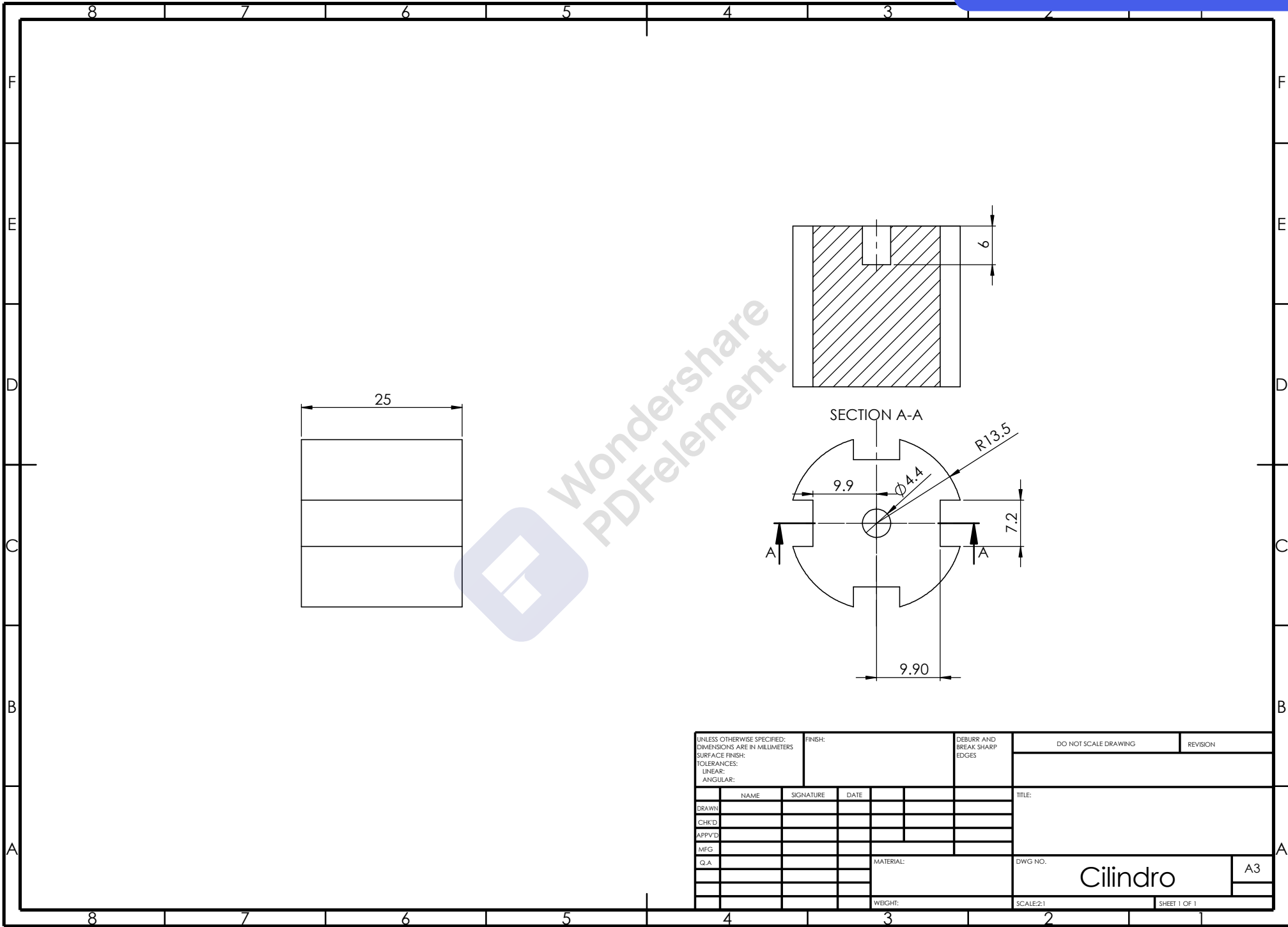
Referências

- Campbell, J. 2020, “How to choose between Collaborative and Industrial Robots”, 13 março 2020, último acesso: março 2021, <https://www.universal-robots.com/blog/how-to-choose-between-collaborative-and-traditional-industrial-robots/>
- Cobot Trend Staff, “Cobots vs industrial robots: what are the differences?”, último acesso: março 2021, <https://www.cobottrends.com/cobots-vs-industrial-robots-what-are-differences/>
- Mccrea, N. 2018, “An Introductory Robot Programming Tutorial”, último acesso: fevereiro 2021, <https://www.toptal.com/robotics/programming-a-robot-an-introductory-tutorial>
- Robots, U. “UR10 Cobot Solves Labor Shortage and Automates Production”, último acesso: fevereiro 2021, <https://www.universal-robots.com/case-stories/2k-trend/>
- Robots, U. “Maintenance free UR cobots operate continuously in harsh environment”, último acesso: fevereiro 2021, <https://www.universal-robots.com/case-stories/aircraft-tooling-inc/>
- Robots, U. “UR5 Allows Employees at Schneider Electric to Focus on Creative Tasks Through Automating Repetitive Processes”, último acesso: fevereiro 2021, <https://www.universal-robots.com/case-stories/schneider-electric/>
- Robotic, S. “Best Robotics Simulation Software”, último acesso: fevereiro 2021, <https://roboticsshop.net/best-robotics-simulation-software/#RoboDK>
- Buskirk, B. “An Overview of Robotic Simulation Software”, último acesso: março 2021, <https://thinkcomputers.org/an-overview-of-robotic-simulation-software/>
- Robotics, S. 2012, “Most Advanced Robotics Simulation Software Overview”, 1 junho 2012, último acesso: março 2021, <https://www.smashingrobotics.com/most-advanced-and-used-robotics-simulation-software/>
- Owen-Hill, A. 2016, “What Are The Different Programming Methods for Robots?”, 24 março 2016, último acesso: fevereiro 2021, <https://blog.robotiq.com/what-are-the-different-programming-methods-for-robots>
- RobotIQ, “Collaborative Robots in Global Companies”, último acesso: fevereiro 2021
- RobotIQ, “Getting Started with Collaborative Robots”, último acesso: fevereiro 2021
- Robots, U. “Get Started with Cobots”, último acesso: fevereiro 2021
- Abreu, P. 2020, “Programação de robôs industriais, Apresentação de *software* RobotStudio e Linguagem Rapid”, último acesso: março 2021
- Woodford, C. “3D Printers”, 24 de abril de 2021, último acesso: maio de 2021, <https://www.explainthatstuff.com/how-3d-printers-work.html>
- AMG 2020, último acesso: maio de 2021 <https://amfg.ai/industrial-applications-of-3d-printing-the-ultimate-guide/>
- Oriental Motor USA Group 2020, último acesso: maio de 2021 <https://www.orientalmotor.com/motor-sizing/beltConveyor-sizing.html>
- MPS Robot Assembly, ultimo acesso: março de 2021

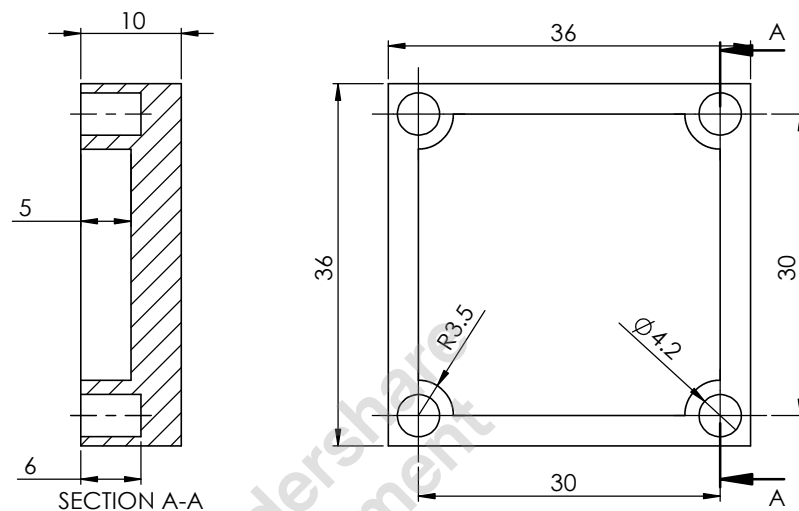
<https://www.youtube.com/watch?v=NxRrgJMbbm0>

UBIFactory, último acesso: março de 2021 <https://www.youtube.com/watch?v=etpSCNqXpV4>

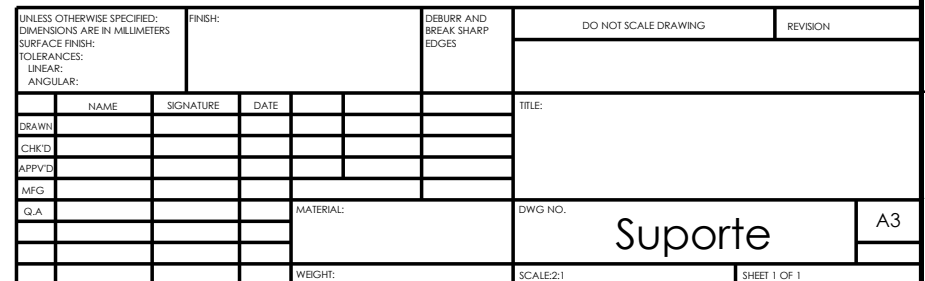
ANEXO A: Vistas 2D das peças concebidas por impressão 3D

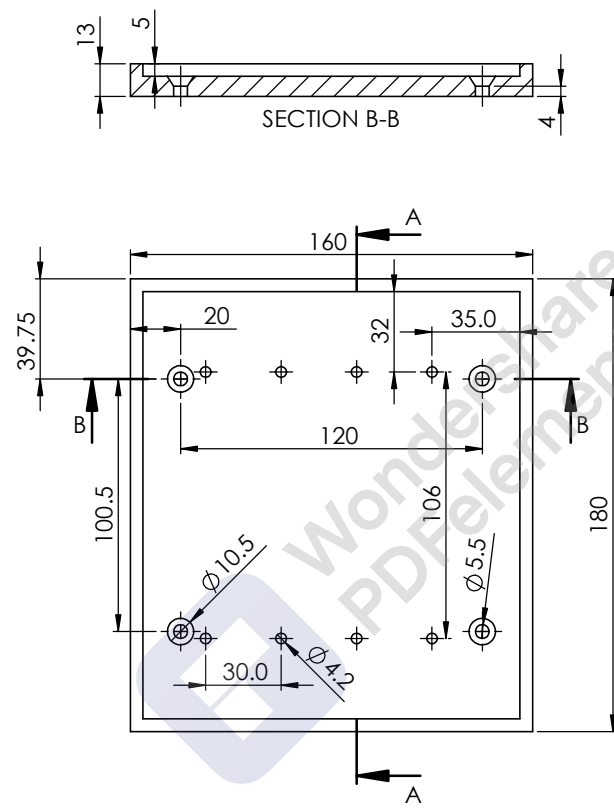


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:				FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
DRAWN				NAME	SIGNATURE	DATE		TITLE:	
CHKD									
APPVD									
MFG									
G.A.									
				MATERIAL:				DWG NO.	Cilindro
				WEIGHT:				SCALE:2:1	
								SHEET 1 OF 1	A3

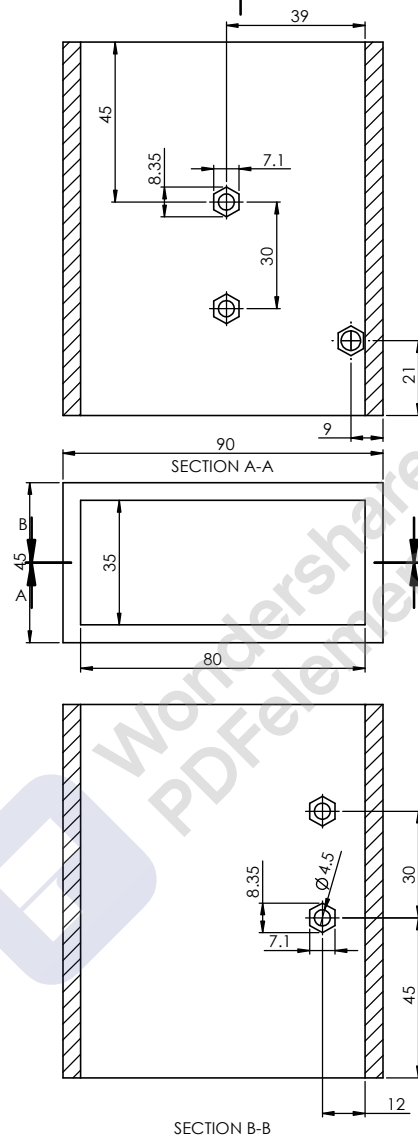


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:						FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING		REVISION	
DRAWN						SIGNATURE		DATE		TITLE:			
CHK'D													
APP'D													
MFG													
Q.A.								MATERIAL:		DWG NO.		A3	
										Tampa			
								WEIGHT:		SCALE:2:1		SHEET 1 OF 1	

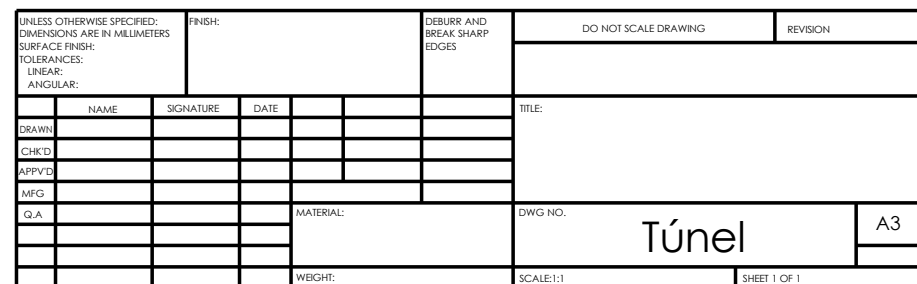


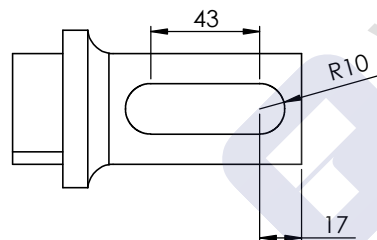


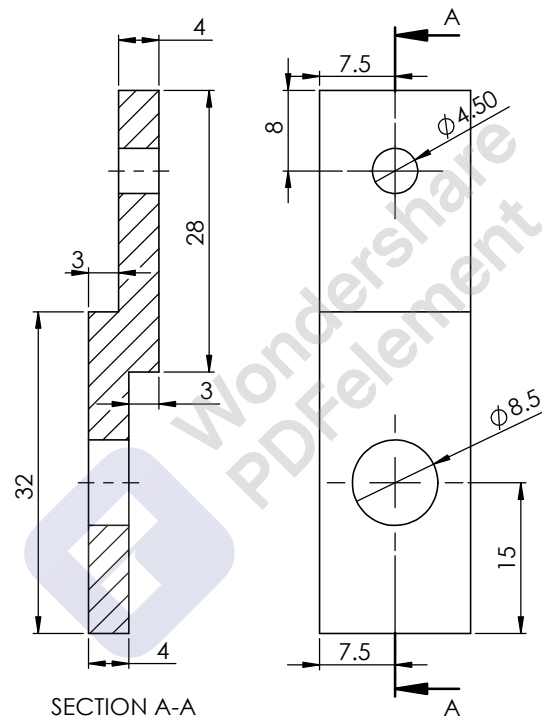
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:							FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING		REVISION					
											TITLE:							
DRAWN	NAME		SIGNATURE		DATE													
CHK'D																		
APP'VD																		
MFG																		
Q.A.																		
							MATERIAL:		DWG NO.				Reservatório				A3	
									SCALE:1:2				SHEET 1 OF 1					



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH TOLERANCES: LINEAR: ANGULAR:				FINISH:		DEBURY AND BREAK SHARP EDGES		DO NOT SCALE DRAWING		REVISIONS	
DRAWN:		NAME		SIGNATURE		DATE		TITLE:			
CHECKED:											
APPROVED:											
MFG:											
Q.A.:											
				MATERIAL:				DWG NO.		Base	
				WEIGHT:				SCALE:1:1		A2	
								SHEET 1 OF 1			



[illegible]

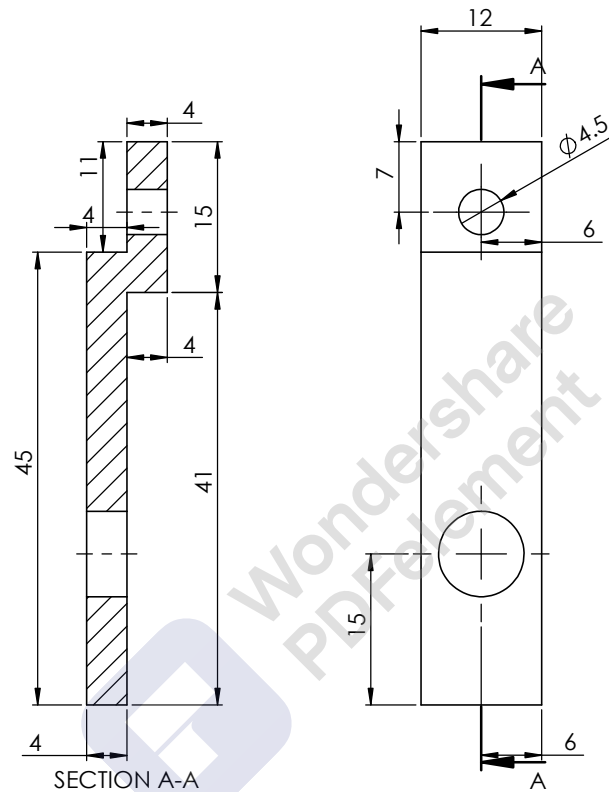


SECTION A-A

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS						FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING				REVISION	
SURFACE FINISH:															
TOLERANCES:															
LINEAR:															
ANGULAR:															
NAME		SIGNATURE		DATE						TITLE:					
DRAWN															
CHK'D															
APP'D															
MFG															
Q.A						MATERIAL:				DWG NO.				A3	
										Conector 1					
						WEIGHT:				SCALE:2:1				SHEET 1 OF 1	



Conector 2



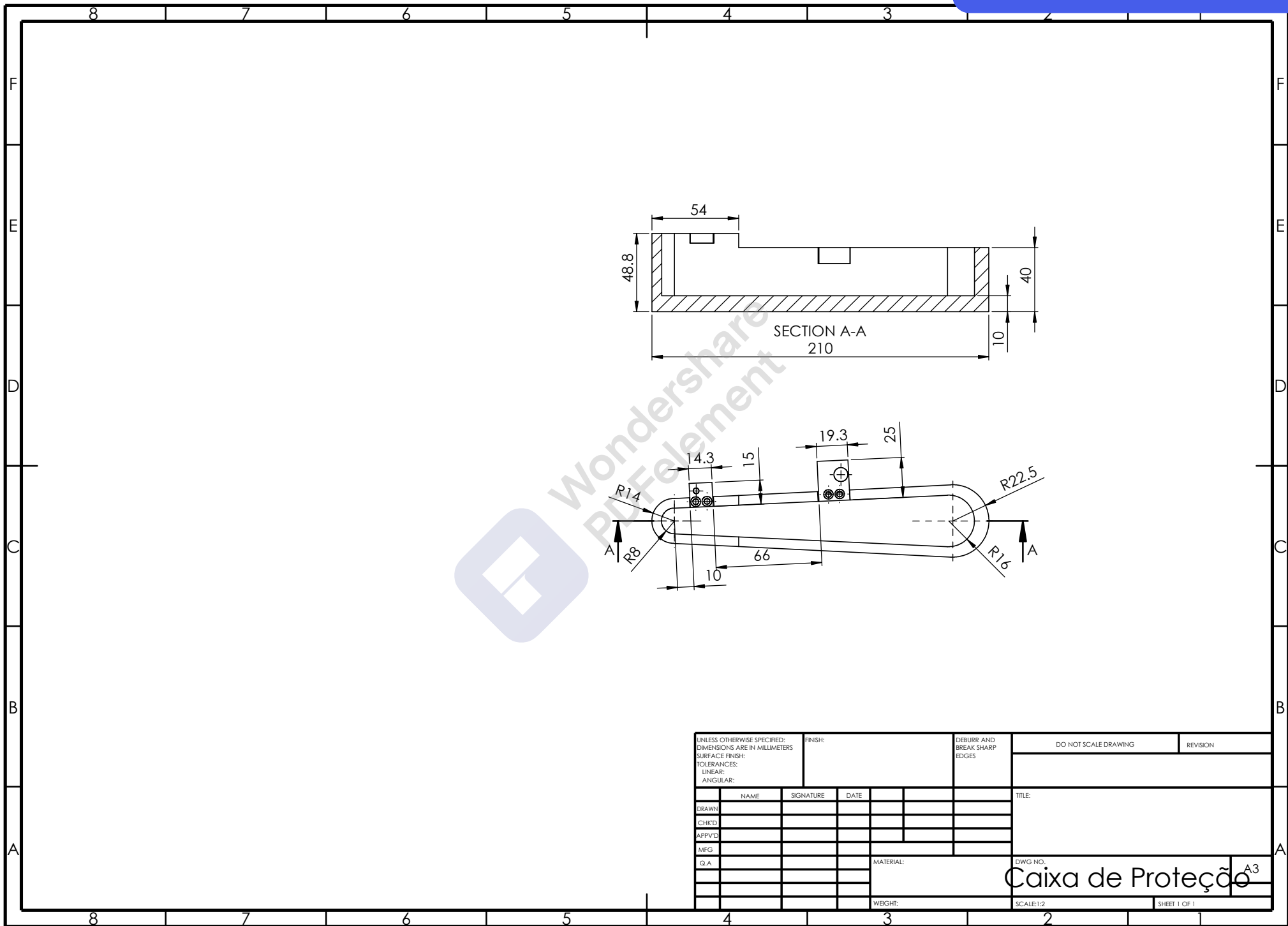
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:						FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING				REVISION	
NAME		SIGNATURE		DATE						TITLE:					
DRAWN															
CHK'D															
APP'D															
MFG															
Q.A						MATERIAL:				DWG NO.				A3	
										Conector 3					
						WEIGHT:				SCALE:2:1				SHEET 1 OF 1	



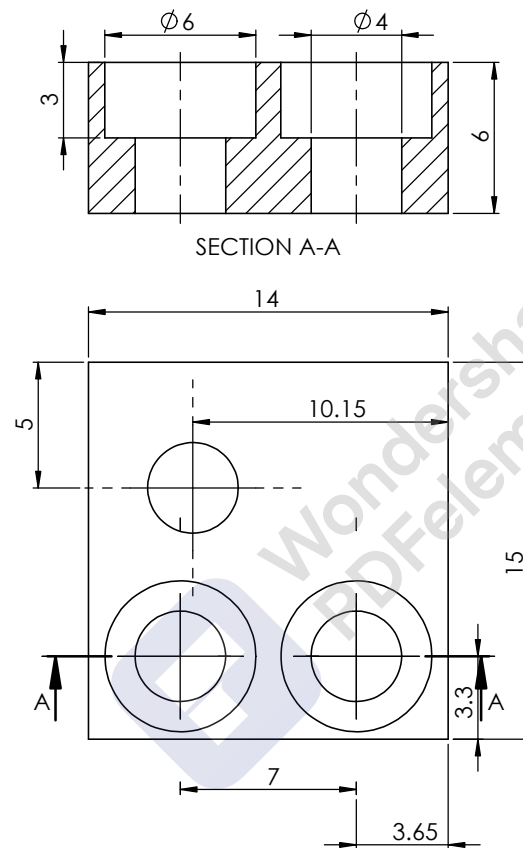
DWG NO. Placa_encoder



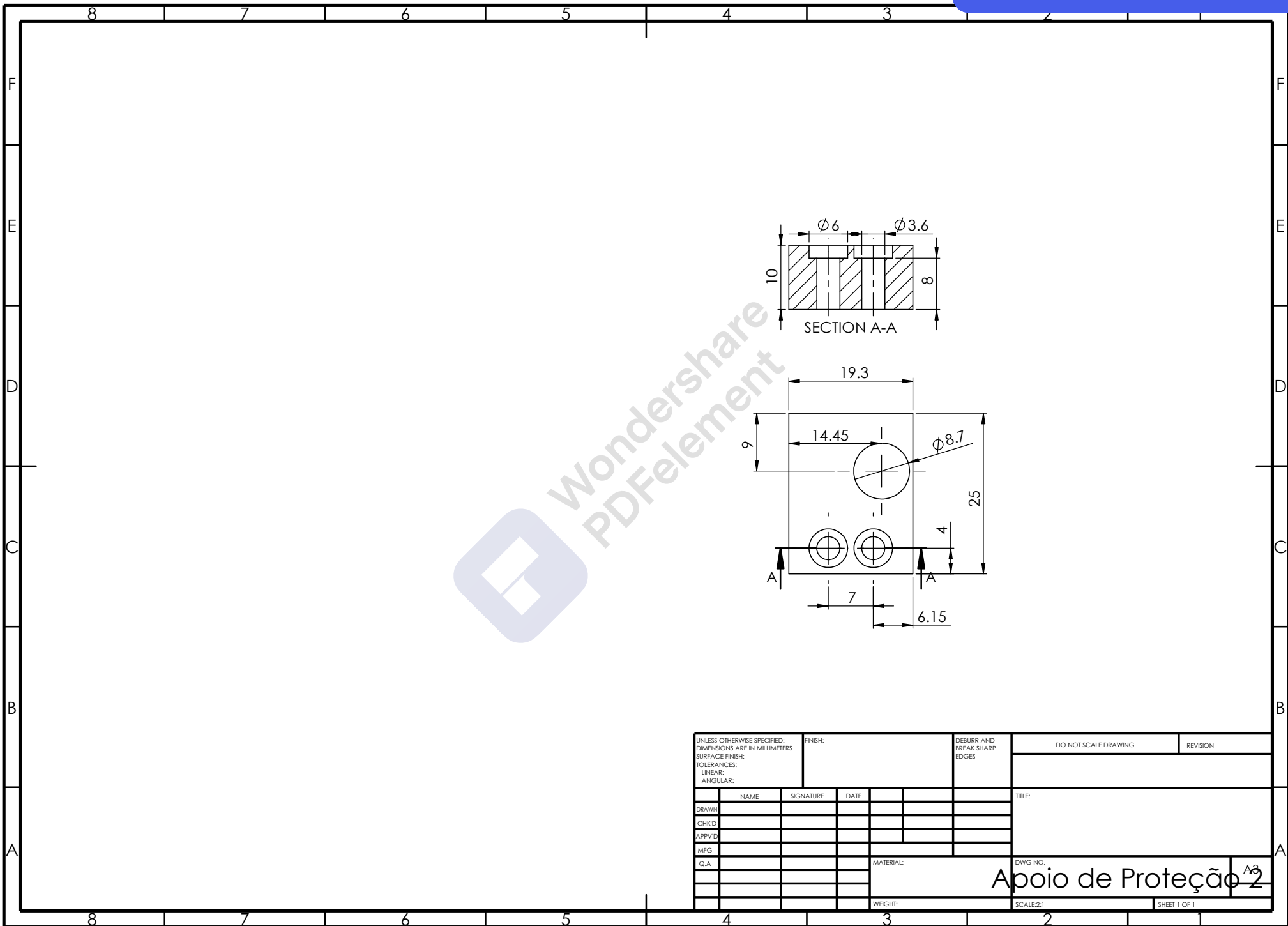
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS						FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING				REVISION	
SURFACE FINISH:															
TOLERANCES:															
LINEAR:															
ANGULAR:															
NAME		SIGNATURE		DATE						TITLE:					
DRAWN															
CHK'D															
APP'D															
MFG															
Q.A						MATERIAL:				DWG NO.				A3	
										Placa do Motor					
						WEIGHT:				SCALE:1:1				SHEET 1 OF 1	



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:						FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING		REVISION	
NAME		SIGNATURE		DATE						TITLE:			
DRAWN										Caixa de Proteção ^{A3}			
CHKD													
APP'D													
MFG													
Q.A.													
						MATERIAL:				DWG NO.			
						WEIGHT:				SCALE:1:2			
										SHEET 1 OF 1			

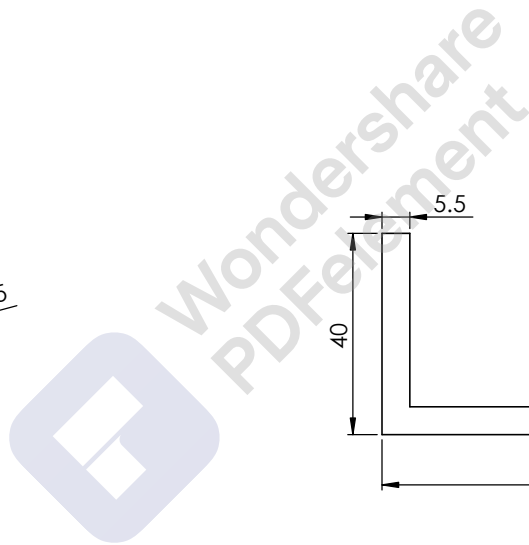


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS						FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING		REVISION	
SURFACE FINISH:													
TOLERANCES:													
LINEAR:													
ANGULAR:													
	NAME		SIGNATURE	DATE				TITLE:					
DRAWN													
CHK'D													
APP'D													
MFG													
Q.A.						MATERIAL:		DWG NO.					
								Apoio de Proteção I A3					
						WEIGHT:		SCALE:5:1				SHEET 1 OF 1	

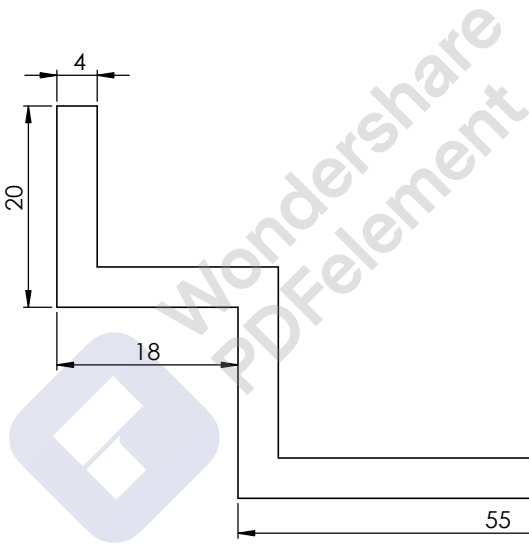


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:				FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
DRAWN	NAME	SIGNATURE	DATE					TITLE:	
CHKD									
APPVD									
MFG									
G.A.									
						MATERIAL:		DWG NO.	A3
						WEIGHT:		SCALE:2:1	SHEET 1 OF 1

Apoio de Proteção 2



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS						FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING				REVISION			
SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:																	
	NAME		SIGNATURE		DATE					TITLE:							
DRAWN																	
CHK'D																	
APP'D																	
MFG																	
Q.A																	
						MATERIAL:				DWG NO. Elevação de Rampa							
										A3							
						WEIGHT:				SCALE:1:1				SHEET 1 OF 1			

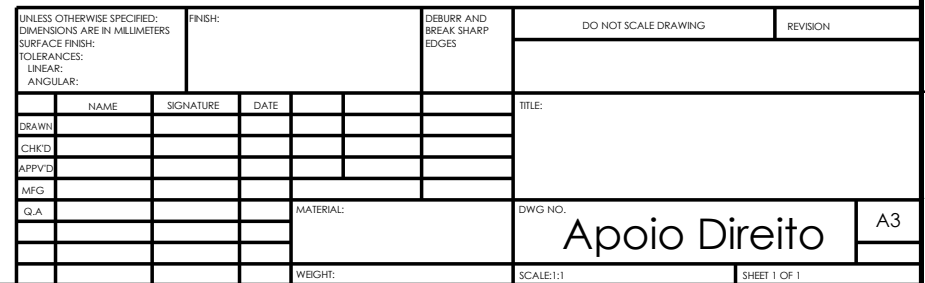


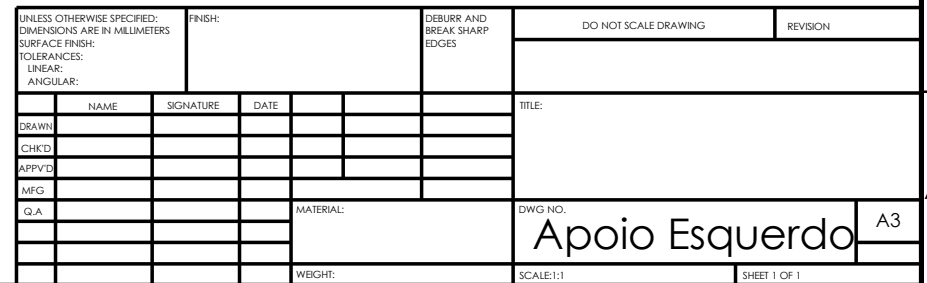
DWG NO. A3

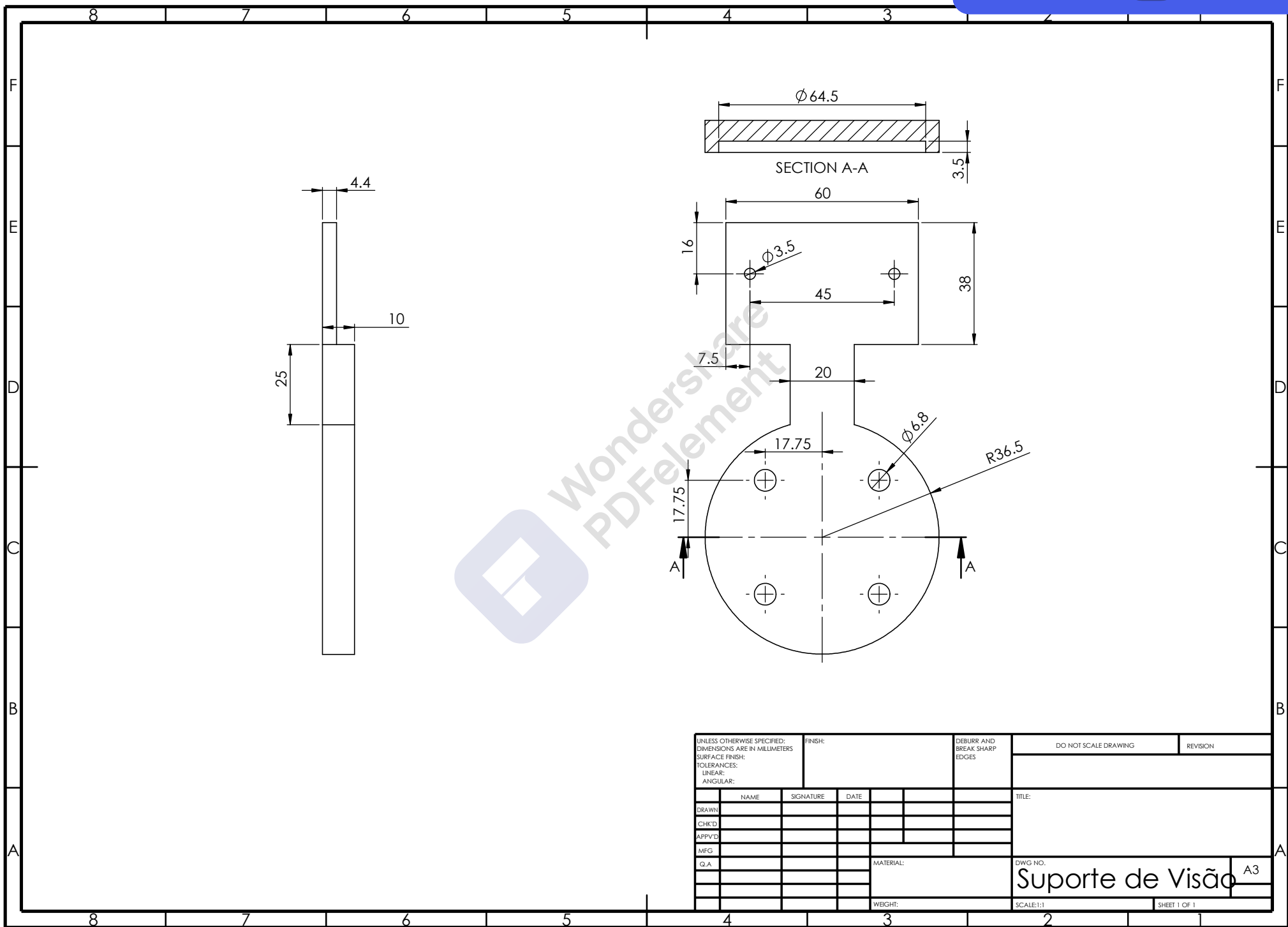
Conector de Rampa



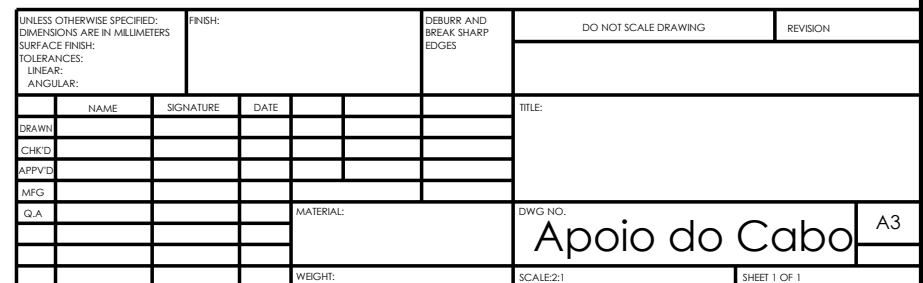
Plataforma

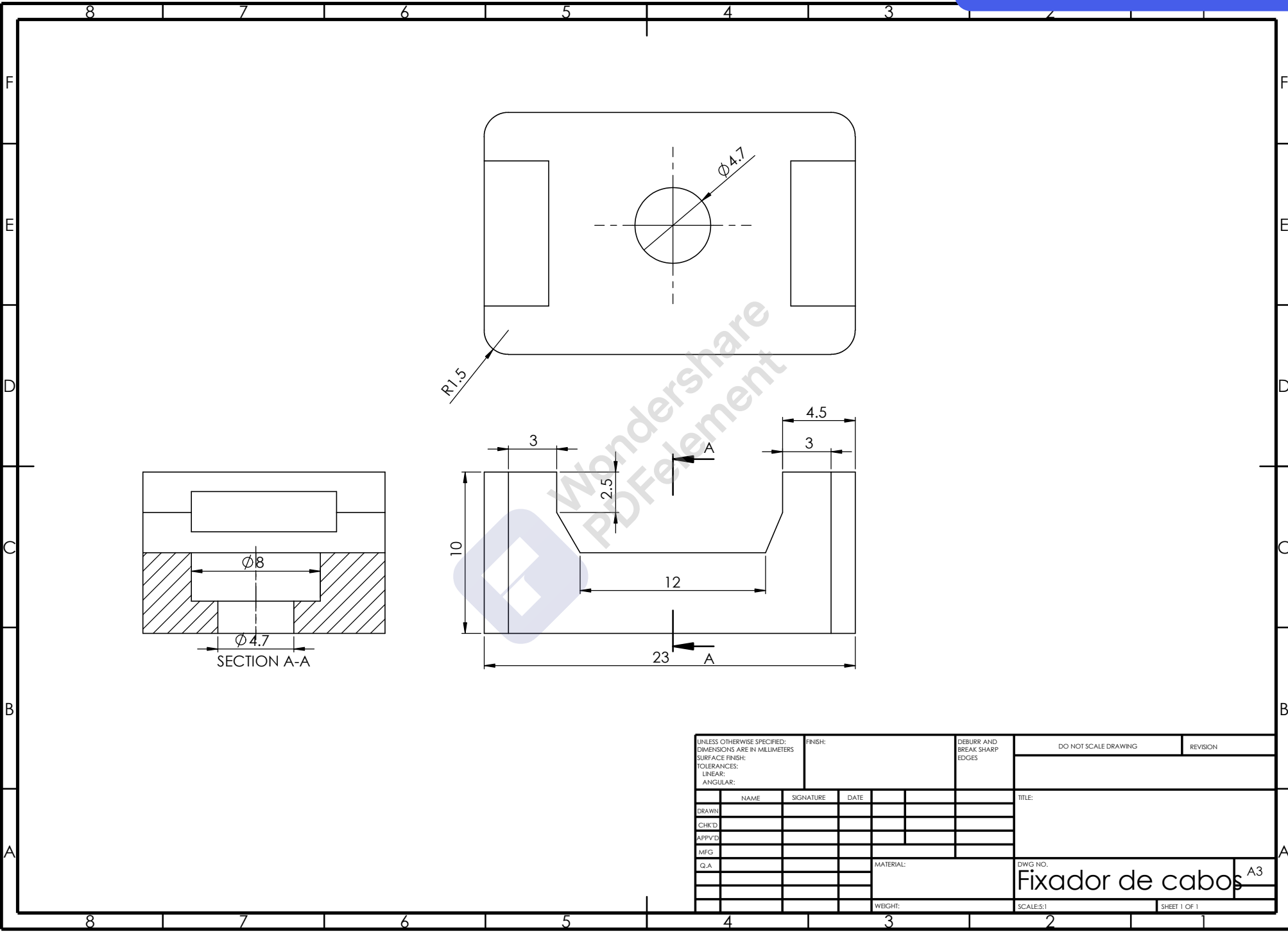




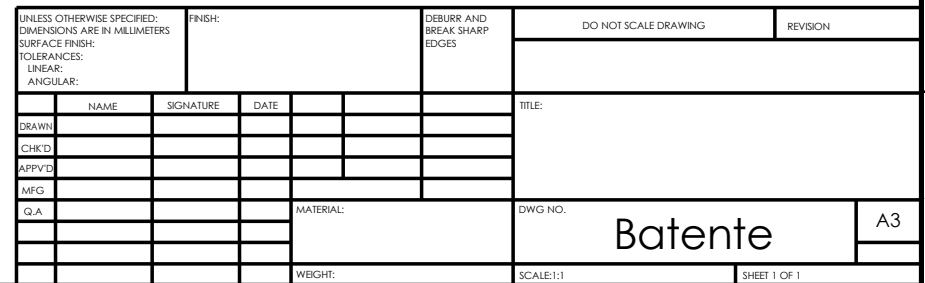


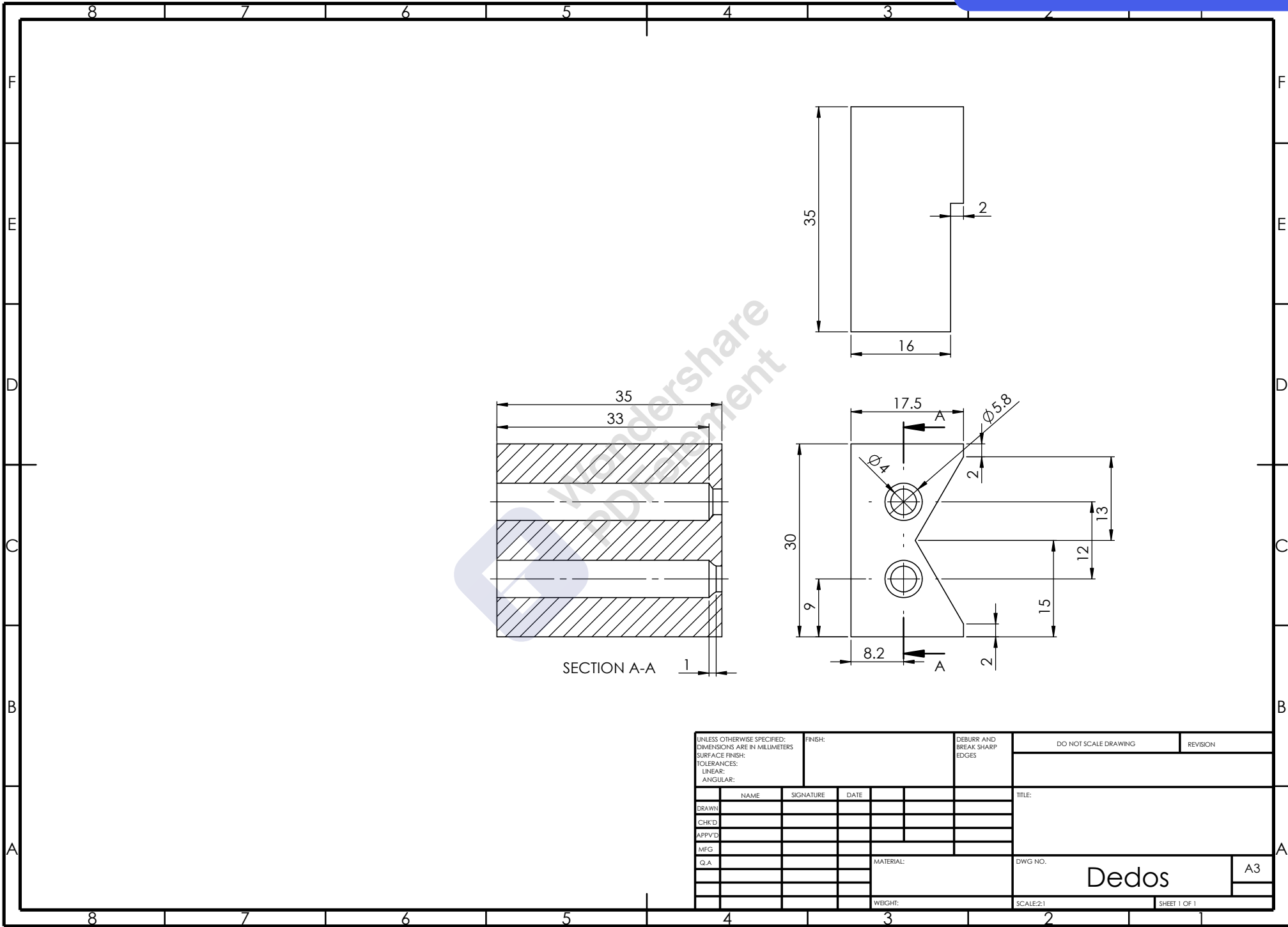
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:							FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING		REVISION		
DRAWN	NAME	SIGNATURE	DATE												
MFG															
Q.A.															



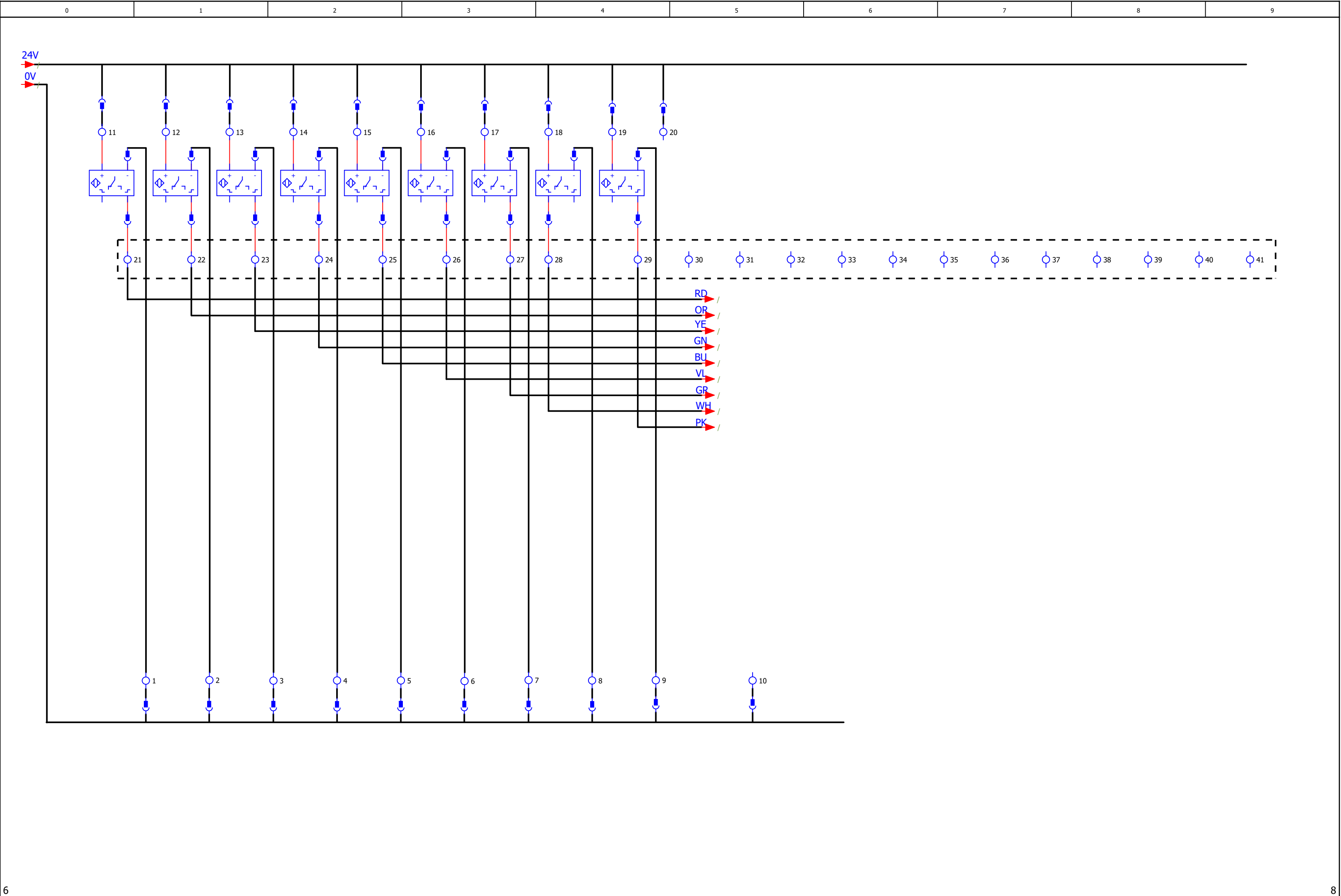


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:				FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING		REVISION
DRAWN				NAME	SIGNATURE		TITLE:		
CHKD							Fixador de cabos		A3
APPVD									
MFG									
G.A.									
				MATERIAL:		DWG NO.		SCALE: 1:1	
				WEIGHT:		SHEET 1 OF 1			





ANEXO B: Esquema Elétrico em EPLAN





13

		Date	8/28/2021	EPLAN	EPLAN Software & Service GmbH & Co. KG		=	
		Ed.	PEDRO MONFORTE				+	
		Appr		Project template with identification structure in accordance with IEC standard:	Page structure with higher-level function and mounting location			
Modification	Date	Name	Original	Replacement of	Replaced by		IEC_tpI001	Page 12 9 / 29

