Projeto em Sistemas de Automação

Bin picking (Robô UR10 e Visão 2D)

Bruno Oliveira | 93090

Carlos Ribeiro | 98637

Catarina Pereira | 98650

João Neves | 98544

Professor Orientador: Abílio Borges

Índice de Figuras

Figura 1-1 – Planeamento do projeto 2
Figura 2-1 - Esquema de princípio 3
Figura 4-1 - Fluxograma 6
Figura 4-2 - Rexroth EFC 3610 7
Figura 4-3 - Sensor fotoelétrico: PE-R05D 8
Figura 5-1 - Fixação da câmara 12
Figura 5-2 - Modelo CAD da peça de fixação da câmara 13
Figura 5-3 - Objeto binarizado no *Sherlock Embedded* 14
Figura 5-4 - Dados relativos à formação de imagens 15
Figura 6-1 - Digital Outputs no UR10 18
Figura 6-2 - Programa UR10: Parte 1 20
Figura 6-3 Programa UR10: Parte 2 20
Figura 6-4 - Programa UR10: Parte 3 21
Figura 7-1 - Esquema Pneumático 22
Figura 7-2 – Válvula Pneumática SMC SY5220 23
Figura 7-3 - Modelo CAD da peça de fixação do Gripper 23
Figura 7-4 – Gripper SMC MHY2-20D 23
Figura 8-1 – Interface gráfica com o utilizador 24
Figura 11-1 – Título / Frontispício 27
Figura 11-2 – Tabela de conteúdos: /1 - /100 27
Figura 11-3 – Alimentação Geral 28
Figura 11-4 - Alimentação Fonte Alimentação 28
Figura 11-5 – Comando Variador 29
Figura 11-6 – Alimentação PLC 29
Figura 11-7 – Alimentação Dalsa BOA 30
Figura 11-8 – Alimentação UR10 30
Figura 11-9 – Rede Profinet 31
Figura 11-10 – Entradas Digitais 31
Figura 11-11 – Saídas Digitais 32
Figura 11-12 - Código geral em Sherlock Embedded 33
Figura 11-13 – Script de transformação de coordenadas: Parte 1 34
Figura 11-14 - Script de transformação de coordenadas: Parte 35

Índice de Tabelas

Tabela 2-1 - Softwares Utilizados 4
Tabela 3-1 - Páginas do projeto elétrico 5
Tabela 4-1 - Endereços de IP 8
Tabela 4-2 - Entradas Digitais: Robô UR10 9
Tabela 4-3 - Saídas Digitais: Robô UR10 9
Tabela 4-4 - Entradas Digitais: Dalsa BOA 1600 9
Tabela 4-5 - Saídas Digitais: Dalsa BOA 1600 9
Tabela 4-6 - Entradas Digitais 10
Tabela 4-7 - Saídas Digitais 10
Tabela 5-1 - Informações das imagens da câmara 14
Tabela 5-2 - Entradas Digitais - Dalsa BOA 1600 16

Índice

1. Introdução 1
1.1. Planeamento 1
2. Funcionamento do Sistema 3
2.1. Esquema de Princípio 3
2.2. Equipamentos e Softwares utilizados 4
2.3. Metodologia de Desenvolvimento 4
2.4. Posicionamento Utilizando o Robô UR10 4
3. Projeto Elétrico 5
4. PLC Siemens S7-1200 6
4.1 Fluxograma Projeto 6
4.2 Tapete Rolante 6
4.2.1 Variador de Frequência 7
4.2.2 Sensor Fotoelétrico 7
4.3. Protocolos de Comunicação 8
4.4. Entradas e Saídas Digitais 8
4.5. Programação PLC 10
5. Visão 2D 12
5.1. Tratamento de imagem 12
5.2. Transformação das Coordenadas 14
5.3. Protocolo de Comunicação 16
6. Robô UR10 17
6.1. Tipos de Movimento 17
6.2. Entradas e Saídas Digitais 17
6.3. Apresentação do programa 18
7. Atuação Pneumática 22
7.1. Equipamentos 22
8. Interface em Visual Basic 24
9. Conclusão 25
10. Referências 26
11. Anexos 27
A. Projeto Elétrico 27
B. Programação Visão 2D 33

1. Introdução

No âmbito da unidade curricular de Projeto em Sistemas de Automação, foi realizado um projeto para desenvolver um sistema de manipulação de peças com recurso a um robô colaborativo UR10 em conjunto com um sistema de visão 2D. O objetivo principal foi criar um sistema eficiente e preciso, capaz de realizar a manipulação de peças de forma automatizada.

Durante o projeto, foram criados diversos subsistemas interligados entre si de forma a assegurar o correto funcionamento de todo o sistema. Um desses subsistemas é o tapete rolante, responsável pelo transporte das peças. Foi utilizado um variador de frequência de forma a regular a velocidade do tapete, bem como o seu controlar quando é necessário existir movimento do tapete e quando não é.

Além disso, foi implementado um sistema de atuação pneumática, responsável pela abertura e fecho do gripper implementado no robô para agarrar as peças, este foi fundamental para garantir uma manipulação precisa e segura das peças.

Foi ainda utilizado um sistema de visão 2D de forma a adquirir uma imagem em tempo real da peça e com recurso ao software *Sherlock Embedded*, foi possível identificar a posição do centróide de cada peça. Essa informação é necessária para que o robô pudesse efetuar a correção das coordenadas referentes à peça presente no tapete rolante, garantindo assim o correto funcionamento de todo o sistema.

Ao longo deste relatório, serão detalhadas todas as etapas realizadas do projeto desenvolvido, demonstrando a viabilidade e eficiência do sistema realizado para a manipulação de componentes utilizando o robô colaborativo UR10 e o sistema de visão 2D.

1.1. Planeamento

Com intuito de dar a todos os elementos do grupo um maior nível de aprendizagem das competências de projeto em sistemas de automação, foram divididas as tarefas de modo que existisse entreajuda dos membros (ver Figura 1-1). Com a finalidade de perceber se o planeamento estava a ser desempenhado, foi realizada uma atualização do mesmo uma vez por mês.

A realização de certas tarefas demorou mais que o esperado existindo alguns desvios relativamente inicialmente planeado. A comunicação entre hardware, entre o *PLC - UR10* e o *PLC - Dalsa BOA 1600*, exigiu mais tempo que o previsto, isto por não ser tão evidente para quem não tem experiência prévia na área. Da mesma forma, visto que o PLC é o "cérebro" (*Master*) do nosso projeto a programação do mesmo estendeu-se até às últimas semanas.

Contudo, houve tarefas que se conseguiram realizar antes do tempo proposto. A programação em visão 2D e a programação do robô UR10 com recurso ao software "*Sherlock Embedded*" e ao "*Polyscope*", revelaram-se intuitivas o que permitiu um rápido domínio das mesmas.

A programação visão 3D não foi realizada, uma vez que a câmara chegou só no final de maio e não tinha as suas funções operacionais. Além disso, destaca-se o facto de que o desenvolvimento da aplicação ter sido realizada no fim, uma vez que se trabalhou com a finalidade de garantir a operacionalidade de todos os componentes antes de se aplicar uma interface os para controlar.

Finalmente, o planeamento foi cumprido sem grandes percalços e conclui-se que a execução de qualquer projeto complexo requer um planeamento rigoroso, uma vez que envolve a realização de diversas tarefas. Este facto é indispensável para garantir um maior controlo sobre o projeto e

assegurar que todas as tarefas sejam concluídas, desta forma é possível apresentar um projeto final funcional.

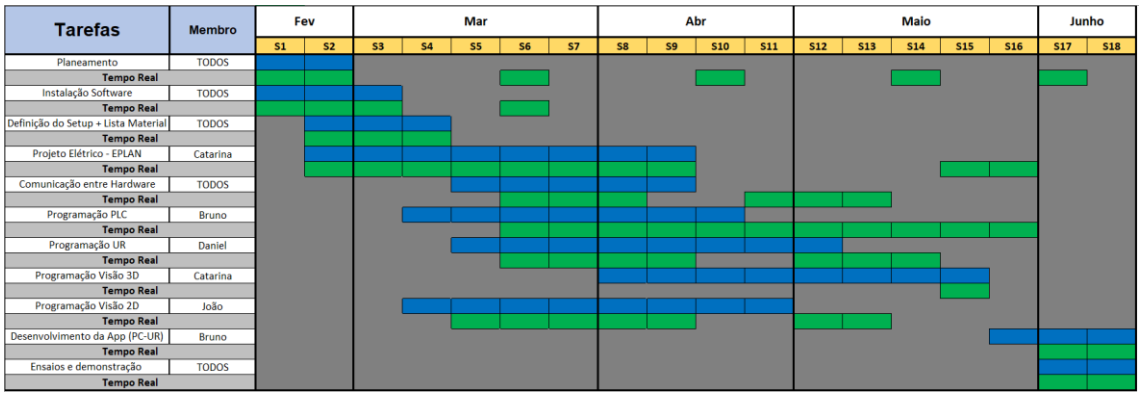


Figura 1-1 – Planeamento do projeto

2. Funcionamento do Sistema

2.1. Esquema de Princípio

Para iniciar o ciclo de operação do robô UR10, basta pressionar o botão “Start” (botão verde). Isso ativará o movimento do tapete rolante, onde a peça estará posicionada. Assim que a célula fotoelétrica do sensor detetar a presença de uma peça, o movimento do tapete rolante será interrompido e o movimento do robô será iniciado.

A câmara será utilizada para reconhecer a peça detetada e identificar a sua posição. O robô inicia então o movimento de recolha da peça e coloca-a num local específico. Após a conclusão dessa etapa, o robô retornará à sua posição inicial para aguardar o próximo ciclo. Esse ciclo será iniciado sempre que o sensor fotoelétrico detetar uma nova peça, reiniciando assim todo o processo.

Com o objetivo de garantir a segurança de todo o sistema e do operador, foi instalado um botão de emergência (botão vermelho). Esse botão pode ser ativado em situações de mau funcionamento ou emergência, interrompendo imediatamente todas as atividades do robô e dos restantes sistemas. Quando o botão de emergência é pressionado é acionado um LED vermelho, indicando assim visualmente o estado de emergência. Além disso, é acionado um LED verde quando o botão “Start” é ativado, sinalizando que o sistema está em operação.

De forma que seja possível a manipulação de objetos, foi implementado um sistema de controlo pneumático, que permite o acionamento do mecanismo de abertura e fecho do *gripper* utilizando ar comprimido. Na Figura 2-1 é possível analisar o esquema de princípio do projeto.

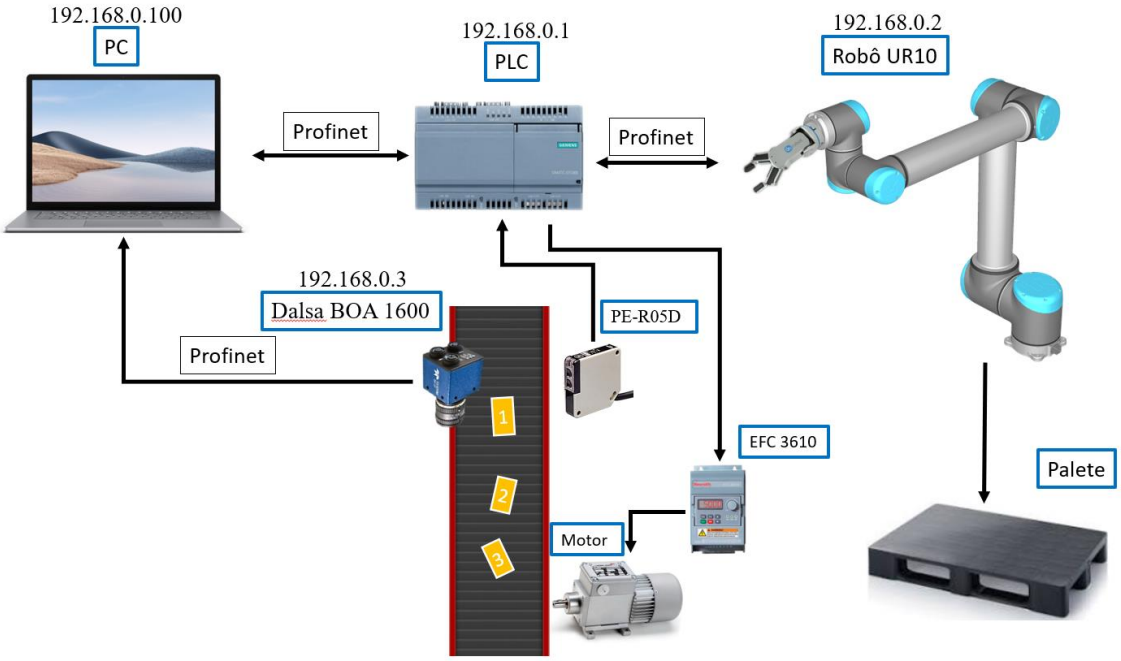


Figura 2-1 - Esquema de princípio

2.2. Equipamentos e Softwares utilizados

No processo de montagem de todo o sistema, foram utilizados diversos equipamentos, conforme mostrado no esquema de princípio. De forma a ser possível configurar e programar alguns desses equipamentos, foram utilizados diferentes softwares (ver Tabela 2-1).

Tabela 2-1 - Softwares Utilizados

Equipamentos	Softwares
Projeto Mecânico	SolidWorks
Esquemas elétricos	EPLAN Electrical P8.3
PLC Siemens S7-1200	TIA Portal V16
Robô UR10	PolyScope Robot User Interface
Teledyne Dalsa BOA 1600	Sherlock Embedded
Interface com o utilizador	Microsoft Visual Studio 2022

Alem disso, foi incorporado um tapete rolante para realizar o transporte das peças e um sensor fotoelétrico para que fosse possível a deteção das mesmas.

2.3. Metodologia de Desenvolvimento

A implementação do sistema ocorreu em etapas distintas. Primeiramente, foi realizada a configuração e calibração das câmaras, bem como a definição dos algoritmos de processamento de imagem adequados para a tarefa de identificação de peças. Em seguida, foi necessário realizar a integração do sistema de visão com o robô UR10, garantindo a comunicação eficiente entre os componentes usando o PLC como sistema base de comunicação da informação a ser transmitida.

A programação do robô UR10 foi realizada utilizando a linguagem de programação específica fornecida pelo fabricante, permitindo a configuração das trajetórias de movimento e a definição dos pontos de interesse para o posicionamento das peças.

2.4. Posicionamento Utilizando o Robô UR10

O robô UR10 foi integrado ao sistema como a unidade responsável pelo posicionamento preciso das peças identificadas pela visão 2D. O UR10 é um robô industrial altamente versátil e colaborativo, capaz de executar movimentos precisos e seguros. Através da programação específica, o robô recebe as informações de localização e orientação das peças identificadas pelo sistema de visão e executa as tarefas de manipulação necessárias, como pegar, mover e posicionar as peças nos locais desejados.

3. Projeto Elétrico

Na fase inicial do projeto foi realizado o projeto elétrico utilizando o software *EPLAN* electrical P8.3, englobando todas as conexões e componentes necessárias, desempenhando um papel crucial na integração dos componentes e garantindo o correto funcionamento de todo o sistema.

Foi realizado um esquema de alimentação geral, que inclui a alimentação da fonte de alimentação, do robô UR10 e do *switch Ethernet*. Estes são componentes essenciais para o funcionamento do sistema.

Nas entradas digitais, foram considerados diversos elementos. O botão "Ok" (botão Start) responsável pelo início do sistema. O botão "NOK" (botão de emergência) responsável pelo acionamento de emergência e, consecutivamente, paragem de todo o sistema de forma imediata. O sensor fotoelétrico foi utilizado para detetar a presença de peças no tapete rolante.

As saídas digitais também foram consideradas no projeto. O LED verde é acionado quando o sistema está em funcionamento, o LED vermelho é acionado quando é ativado o botão de emergência, indicando assim uma situação de emergência. A saída do sensor é responsável pela paragem do motor quando deteta a presença de uma peça no tapete rolante. Foram ainda consideradas duas saídas para controlar a abertura e o fecho do gripper, para agarrar as peças.

Para visualização mais detalhada de todo o esquema elétrico realizado o mesmo encontra-se no Anexo 11-A e no repositório do GitHub do projeto [6]. Na Tabela 3-1 é possível analisar todas as páginas desenvolvidas para os esquemas elétricos do projeto.

Tabela 3-1 - Páginas do projeto elétrico

Página	Descrição da página
/1	Título / Frontispício
/2	Tabela de conteúdos: /1-/100
/5	Alimentação Geral
/6	Alimentação Fonte de Alimentação
/20	Comando do Variador
/40	Alimentação PLC
/41	Alimentação Dalsa BOA
/42	Alimentação UR10
/43	Rede Profinet
/50	Entradas Digitais
/100	Saídas Digitais

4. PLC Siemens S7-1200

Foi utilizado o PLC (*Programmable Logic Controller*) Siemens S7-1200, com oito entradas e saídas digitais, para realizar a comunicação e controlo de todos os equipamentos referidos anteriormente. Essa comunicação é realizada através do protocolo de comunicação PROFINET. O PLC Siemens S7-1200 será programado através do software TIA Portal.

4.1 Fluxograma Projeto

Na Figura 4-1 é possível observar o fluxograma do projeto que reúne, de forma sequencial, todas as etapas do funcionamento do mesmo. Deste modo, consegue-se ter uma perceção geral de como foi programado todo o projeto.

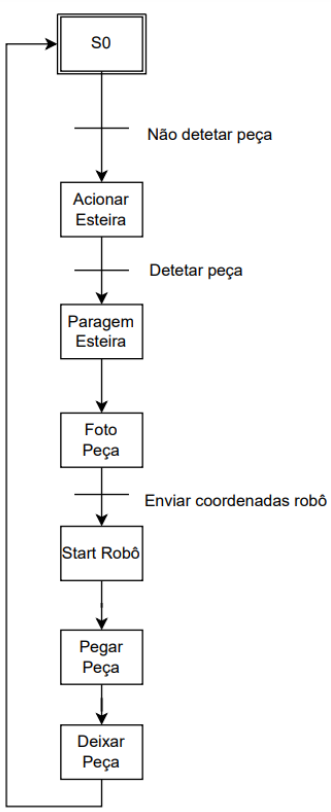


Figura 4-1 - Fluxograma

4.2 Tapete Rolante

O tapete rolante é um sistema composto por um motor trifásico, um variador de frequência, um relé interruptor e um sensor fotoelétrico. Todos estes equipamentos juntamente com a programação no PLC são responsáveis pelo movimento do tapete, permitindo assim controlar e regular a velocidade, e a paragem do tapete rolante, assegurando o transporte seguro das peças.

4.2.1 Variador de Frequência

O motor utilizado foi um motor trifásico de velocidade síncrona, o que significa que possui uma velocidade fixa determinada pela frequência (f) e o número de polos (n_p).

$$v = 120 \times \frac{f}{n_p} \tag{3.1}$$

De forma a permitir a variação de velocidade de rotação do motor e, consequentemente, a velocidade de avanço do tapete, foi necessário utilizar um variador de frequência. O variador de frequência *Bosch Rexroth EFC3610*[3] (ver Figura 4-2) é responsável por controlar a frequência de alimentação do motor, permitindo assim ajustar a velocidade do tapete para a velocidade desejada. O motor possui 4 polos e foi selecionada uma frequência de 11,5Hz, o que resulta numa velocidade de rotação de 345 rad/s. Para acionar o motor através do PLC, foi utilizado um relé de contato normalmente aberto, atuando como um interruptor, em conjunto com um sensor fotoelétrico. Assim é possível efetuar a ativação e desativação automática do motor, de acordo com a deteção de objetos pelo sensor fotoelétrico.

Figura 4-2 - Rexroth EFC 3610

4.2.2 Sensor Fotoelétrico

O sensor fotoelétrico utilizado foi o *beam sensor PE-R05D hanyoling nux korea* [4], que se trata de um sensor de deteção por reflecção difusa através de uma fonte de luz infravermelha (ver Figura 4-3).

O sensor fotoelétrico foi colocado na direção da câmara 2D, na lateral do tapete rolante. Este tem como principal função detetar a presença de objetos, ativando ou desativando o movimento do tapete rolante. As ligações elétricas entre o sensor fotoelétrico e o PLC foram estabelecidas de acordo com o esquema elétrico (Anexo 11-A).

Figura 4-3 - Sensor fotoelétrico: PE-R05D

4.3. Protocolos de Comunicação

PROFINET é um protocolo de comunicação industrial que usa o protocolo Ethernet, rede local, para transferir as suas mensagens. Optou-se por este protocolo de comunicação, em oposição ao protocolo TCP/IP, por ser mais rápido nas comunicações em tempo real. Este protocolo implica que todos os equipamentos devem estar na mesma rede, de forma a se reconhecerem uns aos outros. Para isso foi necessário definir os endereços IP dos diversos equipamentos (ver Tabela 4-1).

Tabela 4-1 - Endereços de IP

	Endereços IP
PLC	192.168.0.1
Robô UR10	192.168.0.2
Dalsa BOA 1600	192.168.0.3

Para o computador pessoal utilizado durante a fase de programação, definiu-se o endereço de IP 192.168.0.100 de forma a não existir nenhum tipo de conflito com os endereços acima indicados. Os restantes componentes do projeto, por exemplo, o tapete rolante, são controlados através das entradas e saídas digitais do PLC.

4.4. Entradas e Saídas Digitais

Estabelecidos os endereços de IP dos diversos componentes do projeto foi necessário instalar os *GSD* (General Station Description) de cada equipamento, de modo a ser possível comunicar com o robô *UR10* e a câmara *Dalsa BOA 1600*, via PROFINET, através do TIA Portal. O ficheiro *GSD* é fornecido pelo fabricante do equipamento e contém uma descrição do dispositivo PROFINET. Deste modo, é possível definir as entradas e saídas digitais correspondentes a cada equipamento. De seguida apresentam-se as entradas e saídas digitais de cada equipamento (ver Tabela 4-2, Tabela 4-3, Tabela 4-4 e Tabela 4-5).

Tabela 4-2 - Entradas Digitais: Robô UR10

Robô UR10	**I address**	**Comentário**
T2O_State	[2…33]	Estado do robô
T2O_IO	[68…135]	Entradas digitais do robô
T2O Joints	[136…239]	Juntas do robô
T2O TCP	[240…315]	TCP do robô
T2O General Purpose Bit Registers	[34…41]	Registo de bits
T2O General Purpose Int Registers	[316…411]	Registo de números inteiros
T2O General Purpose Float Registers	[507…412]	Registo de números reais

Tabela 4-3 - Saídas Digitais: Robô UR10

Robô UR10	**Q address**	**Comentário**
O2T Robot IO	[2…25]	Saídas digitais
O2T General Purpose Registers 1	[26…125]	Registo de números inteiros/reais 1
O2T General Purpose Registers 2	[126…225]	Registo de números inteiros/reais 1

Tabela 4-4 - Entradas Digitais: Dalsa BOA 1600

Dalsa BOA 1600	**I address**	**Comentário**
Input module 254 byte	[508…761]	Entradas digitais

Tabela 4-5 - Saídas Digitais: Dalsa BOA 1600

Dalsa BOA 1600	**Q address**	**Comentário**
Input module 254 byte	[226…479]	Saídas digitais

Além das entradas e saídas digitais dos equipamentos que o PLC irá comandar, definiu-se ainda as entradas e saídas digitais de todo o hardware instalado (ver Tabela 4-6 e Tabela 4-7).

Tabela 4-6 - Entradas Digitais

Entradas Digitais	I address
Start	%I0.0
Paragem Emergência	%I0.1
Sensor Peça	%I0.2

Tabela 4-7 - Saídas Digitais

Saídas Digitais	Q address
LED Verde	%Q0.0
LED Vermelho	%Q0.1
Variador Velocidade	%Q0.2
Abrir Gripper	%Q0.4
Fechar Gripper	%Q0.5

Além das entradas e saídas digitais, foram naturalmente definidas algumas variáveis de memória internas ao PLC, variáveis do tipo M nos mais diversos tipo de dados. Na secção seguinte irá explicar-se a estrutura da programação do PLC efetuada.

4.5. Programação PLC

A estratégia de programação utilizada para o PLC passa por criar as seguintes funções que são declaradas no *main* do programa:

Start/Emergência

Responsável pelo controlo do início de ciclo e respetivo acionamento de emergência. Trata sobretudo da segurança do sistema, dado que, a paragem de emergência terá de parar qualquer movimento que o sistema apresente.

Controlo esteira

Responsável pelo controlo do movimento do tapete rolante. Nesta função, consoante a verificação de algumas condições, por exemplo, o estado do sensor fotoelétrico, é ativada a saída digital que aciona o movimento do tapete rolante. Se, porventura, as condições deixaram de ser verificadas a saída digital deixa de ser ativada e, assim, o tapete rolante para.

Robô UR

Responsável pelo *Start*, *Stop* e *Pause* dos movimentos do UR10 e responsável também pela correção das coordenadas enviadas pela câmara. Para isso, com base num par de coordenadas de referência, previamente guardadas através do robô UR10, realizam-se operações matemáticas de

subtração, de forma que os valores das coordenadas X e Y possam ser corrigidos em relação à referência. No que diz respeito à correção do valor do ângulo (R_Z), como a referência corresponde a, aproximadamente zero radianos, realiza-se a conversão de graus (unidades do valor do ângulo enviadas pela câmara de visão 2D) para radianos.

Nesta função é também realizada a conversão do valor das coordenadas corrigidas, variáveis do tipo Real, para variáveis do tipo Inteiro, que posteriormente serão usadas na interface desenvolvida em Visual Basic (ver secção 8).

Saídas

Responsável pelo acionamento das saídas digitais (ver Tabela 4-7).

É possível analisar todo o código desenvolvido para a programação do PLC consultando o repositório do projeto no GitHub [6].

5. Visão 2D

A visão artificial desempenha um papel crucial no mercado de trabalho, especialmente no setor industrial visto que permite um maior controlo de qualidade, inovação, e identificação de componentes de forma mais eficiente, impulsionando o desenvolvimento industrial.

Com base da sua importância, a implementação de visão 2D tem vindo a crescer no que toca a aplicações de indústria pois diminui os custos do hardware, fornece maior flexibilidade ao sistema e possibilita um maior controlo de qualidade dos processos.

Deste modo, foi utilizado um sistema de visão 2D utilizando uma câmara industrial Teledyne Dalsa BOA 1600 e os algoritmos de processamento de imagem incorporados no software Sherlock Embedded.

A implementação de visão no projeto tem como principal objetivo detetar a posição e a orientação da peça que vem no tapete de forma que o robô pegue a peça na posição correta e posteriormente coloque a mesma em uma palete, simulando um processo industrial de *Pick&Place* inserido numa linha de produção.

Em sistemas de visão artificial, a iluminação é bastante importante pelo que é um dos fatores essenciais a caracterizar no processo de processamento no Sherlock. Considerando a localização de desenvolvimento do projeto (Laboratório de Eletrotecnia, Instrumentação e Controlo Automático – DEM) e a alteração de luz natural, o fator de iluminação foi ajustado no *threshold* utilizado para binarizar a imagem captada pela câmara. Atendendo ao requisito de projeto com menor nível de precisão, não foi necessário utilizar iluminação artificial.

5.1. Tratamento de imagem

A câmara encontra-se fixa perpendicularmente ao plano da esteira a uma cota de 550 mm acima. A fixação é feita a partir de uma peça impressa por impressão 3D que realiza a ligação da câmara ao perfil com um ajuste no plano Z (plano perpendicular à esteira) de modo a corrigir possíveis erros de foco aquando da posição das peças, Figura 5-1 e Figura 5-2.

Figura 5-1 - Fixação da câmara

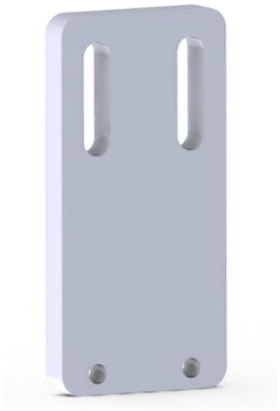


Figura 5-2 - Modelo CAD da peça de fixação da câmara

Após o processo de fixação da câmara e posterior ajuste do foco e abertura de luz (fatores para a distância de 550 mm), realizou-se o programa no *software Sherlock Embedded* para processar as imagens da câmara.

Foi definida uma região de interesse (*ROI*) que contém o tamanho total da imagem necessário para identificar peças em diversas posições. Foram aplicados os pré-processadores:

- ***Threshold*** – Pré-processador de binarização, definindo manualmente a binarização, como referido anteriormente, de modo a ajustar a luminosidade natural.
- ***Close*** – Pré-processador que realiza uma dilatação seguida de uma erosão com o intuito de fechar pixels que estejam isolados para que a imagem final tenha apenas um objeto ("*blob*") a analisar.

De seguida, aplicou-se o algoritmo "***connectivity binary***" do qual podemos definir vários parâmetros, sendo os de maior relevância para esta aplicação:

- **Cor dos objetos** – Selecionou-se objetos de cor branca.
- **Área** – Parametrizou-se a área mínima para 10000 pixels
- **Número de objetos a encontrar e a sua ordem** - Definiu-se encontrar apenas o objeto com maior área.

Embora, este algoritmo forneça várias informações à cerca dos objetos encontrados, neste caso, apenas serão utilizadas as coordenadas do centróide do objeto e o ângulo de orientação através do cálculo do ângulo da elipse. É necessário ser selecionado a opção de cálculo de momentos de 2ª ordem que vem por predefinição desativada, uma vez que exige mais tempo de ciclo e em muitas aplicações não será necessário.

O resultado visualizado no software *Sherlock Embedded* após o processamento dos dados é evidenciado na Figura 5-3.

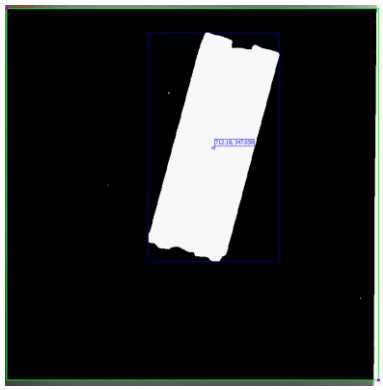


Figura 5-3 - Objeto binarizado no *Sherlock Embedded*

Após extrair os parâmetros necessários da imagem, é necessário transformar o valor das coordenadas do centróide do objeto, em coordenadas do mundo real. Para isso, apresenta-se a formulação utilizada na secção seguinte.

5.2. Transformação das Coordenadas

Inicialmente, foram recolhidas as informações sobre as imagens fornecidas pela câmara expostas na Tabela 5-1.

Tabela 5-1 - Informações das imagens da câmara

Tamanho da imagem	1280 x 960
Distância focal (f) [mm]	12
Tamanho de um pixel [mm]	$3{,}7 * 10^{-3}$
CCD [pixels/mm]	270,27

Com os dados calculou-se a matriz intrínseca necessária para os cálculos posteriores.

$$K = \begin{bmatrix} f \times CCD & 0 & x/2 \\ 0 & f \times CCD & y/2 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

$$K = \begin{bmatrix} 3243 & 0 & 640 \\ 0 & 3243 & 480 \\ 0 & 0 & 1 \end{bmatrix}$$

De modo em saber a matriz de rotação da câmara, colocou-se o robô com a pinça em contacto com a esteira e retirou-se os dados da localização do mesmo.

$$Posição\ da\ câmara = (-0.615, -0.352, 0.172, 3.100, 0.032, -3{,}165)$$

Retirou-se os valores das rotações (últimos três valores da Posição da câmara: R_X, R_Y, R_Z, respetivamente) e armazenou-se numa variável T, necessária em processos seguintes. De seguida, com base nos valores das rotações foi possível calcular a matriz de rotação da câmara R_{cr}, com base num conversor online [1].

$$R_{cr} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

Após detetadas as coordenadas do pixel do centroide do objeto (u,v), o próximo passo é então passar essas coordenadas para o mundo real (X_c,Y_c,Z_c). Primeiramente é necessário analisar o processo de formação de imagens a partir da Figura 5-4 [2].

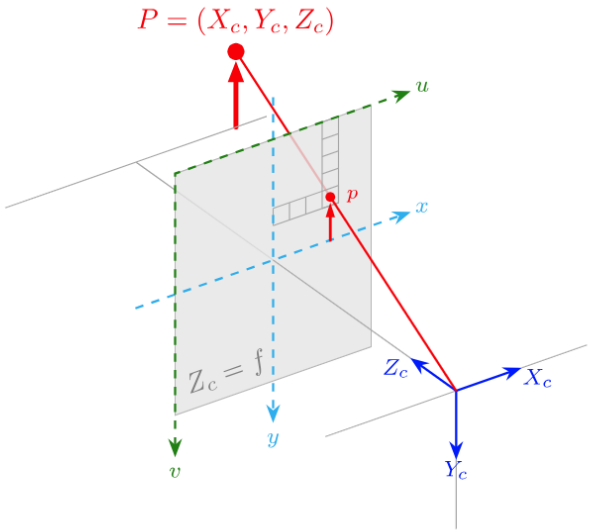


Figura 5-4 - Dados relativos à formação de imagens

Considerando a seguinte equação, conseguiríamos calcular as coordenadas do centróide no mundo real, mas é necessário saber o valor de λ, que é desconhecido neste momento.

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = \lambda K^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{5.2}$$

Assumindo que a esteira seja plana, o plano da mesma pode ser caracterizado por um vetor normal (n) que assume o valor $n = (0,0,1)$.

Para caracterizar a normal do plano da câmara foi utilizada a seguinte equação.

$$n_c = R_{cr} * n^T \tag{5.3}$$

Onde,

n_c – Normal do plano da câmara

R_{cr} – Matriz de rotação

n^T – Normal do plano da esteira transposto

De seguida, tendo em conta a equação (5.4), foi deduzida a equação para o valor de λ, equação (5.5).

$$h = n_c^T \lambda K^{-1}(u, v, 1)^T \tag{5.4}$$

$$\lambda = \frac{h}{n_c^T K^{-1}(u, v, 1)^T} \tag{5.5}$$

Onde,

h – Altura da câmara em relação à esteira

Por fim, após saber como calcular o valor de λ, efetuou-se o cálculo das coordenadas do centroide da peça em análise no mundo real a partir da seguinte equação.

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = \frac{h}{n_c^T K^{-1}(u, v, 1)^T} * K^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{5.6}$$

Para se saber a posição da peça, foi utilizada a seguinte expressão:

$$Posição = R_{cr} * \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} + T^T \tag{5.7}$$

Toda esta formulação foi transcrita para JavaScript e colocada dentro do programa da câmara de visão 2D. É possível analisar o código desenvolvido no Anexo 11- B.

5.3. Protocolo de Comunicação

Concluída a etapa de tratamento de imagem e cálculo da transformação das coordenadas do centróide do objeto para o mundo real é necessário enviar essa informação para o PLC. De seguida, o PLC é responsável pela comunicação desses valores ao robô.

Nesse sentido, utilizou-se o protocolo de comunicação PROFINET para realizar a comunicação entre a câmara de visão 2D e o PLC. Deste modo, com recurso à instrução *IO:Profinet – Profinet Write*, disponível no software, escreveu-se os valores acima descritos para o PLC (ver Tabela 5-2).

Tabela 5-2 - Entradas Digitais - Dalsa BOA 1600

Coordenada X	ID508
Coordenada Y	ID512
Ângulo	ID516

6. Robô UR10

Como referido anteriormente, o robô utilizado foi um UR 10 (Universal Robots). Este robô é bastante polivalente, uma vez que apresenta 6 graus de liberdade, também pode ser utilizado para movimentar cargas de 12.5 kg e, um curso de 1300 mm na posição máxima. Este equipamento é bastante utilizado em aplicações industriais em que os movimentos sejam repetitivos e desgastantes. Por exemplo, em linhas de montagem, paletizações, manuseio de materiais, inspeção e controlo, entre outros.

6.1. Tipos de Movimento

Neste subcapítulo serão apresentados os diferentes tipos de movimentos possíveis a ser realizados no UR10. A compreensão destes movimentos tornou-se crucial para a aplicação das coordenadas variáveis provenientes do sistema de visão 2D referido na secção 5.

O UR10 pode realizar 3 tipos de movimentos diferentes que, com base na aplicação em questão uns poderão ser mais indicados em detrimento dos outros. Os movimentos são:

- **Movimento J (*Move Joint*)**: usado para controlar os movimentos das juntas do robô. Define os ângulos de cada junta do robô para posicionar e orientar o braço robótico determinada configuração de junta. Isso permite que o UR10 execute movimentos precisos e complexos, ajustando as juntas individualmente. Este tipo de movimento é preferível para movimentos rápidos entre pontos de percurso, desconsiderando o movimento da ferramenta.
- **Movimento P (*Move Point to Point*):** é usado para mover o robô em uma sequência de pontos individuais. Ao especificar uma série de pontos de destino no espaço de trabalho o UR10 move-se de um ponto para o próximo em linha reta, em trajetórias discretas de ponto a ponto.
- **Movimento L (*Move Linear*)**: é usado para especificar um movimento linear suave e contínuo do robô de uma posição inicial para uma posição final no espaço de trabalho. O movimento é realizado em uma trajetória reta e a velocidade é ajustada para garantir uma transição suave entre as posições.

Desta forma, foram utilizados movimentos lineares onde é requerido precisão e transições suaves. Por fim, movimentos de junta para os restantes em que não havia necessidade de controlar a posição da ferramenta.

6.2. Entradas e Saídas Digitais

A definição das entradas e saídas foram definidas à medida que o programa foi realizado e os componentes foram montados no projeto. Como por exemplo, a válvula pneumática acoplada à pinça deu origem à criação de 2 saídas digitais.

Neste projeto, não houve necessidade de usar entradas digitais. Assim, apresenta-se as “Digital Outputs” do programa (ver Figura 6-1).

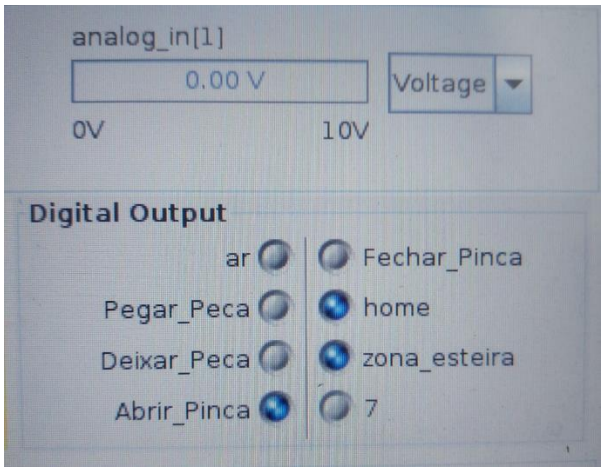


Figura 6-1 - Digital Outputs no UR10

6.3. Apresentação do programa

O programa foi estruturado de uma forma lógica e concisa para rápida perceção e, se necessário, alteração. Ao longo da explicação do código, é assinalado o passo correspondente indicado nas, Figura 6-1 Figura 6-2, Figura 6-3 e Figura 6-4. Foram criados dois subprogramas (1) em que o primeiro corresponde à inicialização das variáveis, uma vez que antes de o robô estar operacional é importante definir o valor de algumas variáveis de forma a garantir uma correta inicialização, já o segundo corresponde aos movimentos efetivamente realizados pelo robô.

No subprograma "Cond_Iniciais" foi realizada a inicialização das variáveis, em que apenas se deixou ligada a variável "Abrir_Pinca" com o intuito de a pinça permanecer aberta antes de realizar qualquer tipo de movimento. (2)

"Set Deixar_Peca = Off"

"Set Pegar_Peca = Off"

"Set Abrir_Pinca = On"

"Set Fechar_Pinca = Off"

"Set Zona_Esteira = Off"

No subprograma "Movimentos", como referido anteriormente o robô realiza todos os movimentos em sequência. Estes movimentos, poderão ser relativos à base ou à ferramenta, que serão referenciados adiante com mais detalhe.

Inicialmente definiu-se a casa do robô como "P_home", este ponto corresponde à posição de segurança, isto é, afastada da zona de trabalho. Depois de o robô estar efetivamente neste ponto, a saída digital "home" é ativada, alertando o PLC. (3)

De seguida, o robô movimenta-se para um ponto de aproximação da esteira ("P1") e realiza-se uma rotação em torno da ferramenta em que o eixo X fica alinhado com a esteira e o eixo Y, perpendicular à mesma ("Ptest"). Da mesma forma, a saída digital zona_esteira é ativada, uma vez que o robô se encontra próximo da zona da esteira.

"Set Zona_Esteira=On"

Após estarem definidos os eixos, realiza-se um movimento linear relativamente à ferramenta, para as coordenadas "X" e "Y" provenientes do PLC que correspondem ao centróide da peça. Ao

mesmo tempo, através da variável “Angle”, realiza-se uma rotação da ferramenta de forma a ter as pinças paralelas ao perfil.

O movimento foi realizado com recurso à ferramenta “script”, em que a primeira linha corresponde à definição do ponto aonde se quer mover o robô, e a segunda linha corresponde à transformação de coordenadas da ferramenta relativamente à base. Depois, o robô movimenta-se para o ponto com uma aceleração de 0.6 [m/s^2] e uma velocidade de 0.1 [m/s]. (4)

“global pose_wrt_tool = p [X, Y, 0, 0, 0, Angle]

“global pose_wrt_base = pose_trans(get_forward_kin(), pose_wrt_tool)”

“movel(pose_wrt_base, a=0.6, v=0.1)

De seguida realizou-se um processo semelhante no movimento para o ponto de pegar a peça. Isto é, realizou-se um movimento linear de aproximação ao perfil na direção Z. Neste movimento, reduziu-se a aceleração e a velocidade por ser um movimento de contacto com a peça. (5)

“global pose_wrt_tool = p [0, 0, 0,184 0, 0, 0]

“global pose_wrt_base = pose_trans(get_forward_kin(), pose_wrt_tool)”

“movel(pose_wrt_base, a=0.2, v=0.05)

Depois de chegar à posição requerida, fecha-se a pinça e o robô espera 1.5 segundos até realizar o próximo movimento. Para além disso, ativa-se variáveis para informar o PLC que a peça já foi retirada. (6)

“Set Fechar_pinca=On”

“Set Abrir_pinca=Off”

“Set Pegar_Peca=On”

“Wait:1.5”

Efetua-se um movimento de retorno ao “Ptest” e uma vez que o robô já não se encontra na zona da esteira a variável é alterada.

“Set Zona_Esteira=Off”

Os pontos P4, P5 e P6 correspondem aos pontos em que é realizado a movimentação do perfil para a zona de paletização (7). Depois disto, inicializou-se uma variável incremental “Peca”, com o intuito de haver 4 diferentes sítios de colocação dos perfis. Com base, na variável incremental “Peca” o robô realiza os movimentos (8). Sendo que, na última posição a variável é colocada a 0. (10).

“Set Peca=Peca+1”

“If Peca=1”

Para deixar as peças foi realizado um movimento linear, em que primeiramente se realiza um movimento de aproximação “Pal1_up”, depois deixa-se o perfil no ponto “Pal1” (9). Posteriormente, é acionada a abertura da pinça. Depois de deixada a peça, o robô espera 1.5 segundos até retornar à posição de aproximação. O PLC é informado que a peça foi deixada através das variáveis apresentadas a seguir. (11)

“Set Fechar_pinca=Off”

“Set Abrir_pinca=On”

“Set Pegar_Peca=Off”

“Set Deixar_Peca=On”

Por fim, o robô volta à posição de segurança “P_home” e aguarda nova ordem do PLC para inicializar novo ciclo. (12)

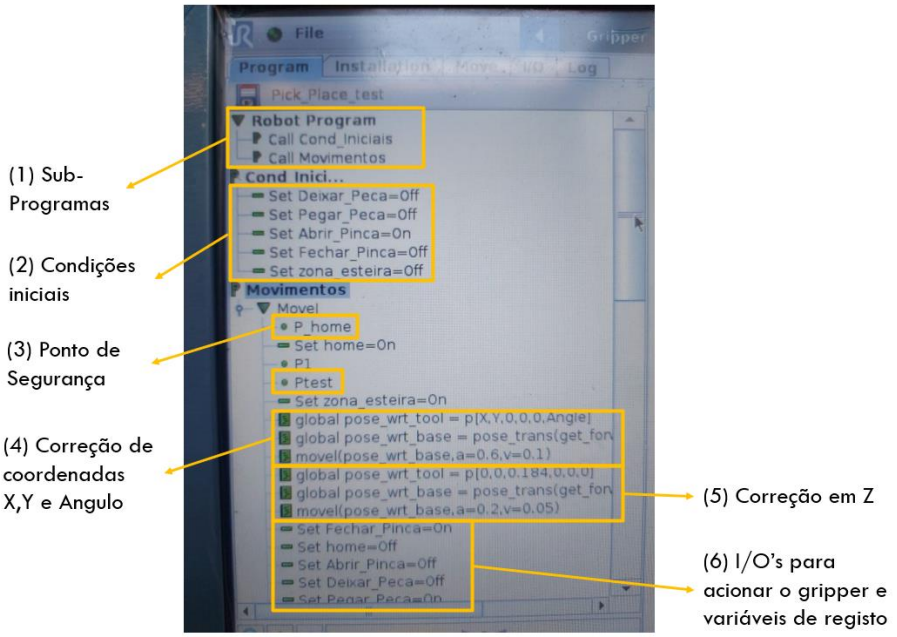


Figura 6-2 - Programa UR10: Parte 1

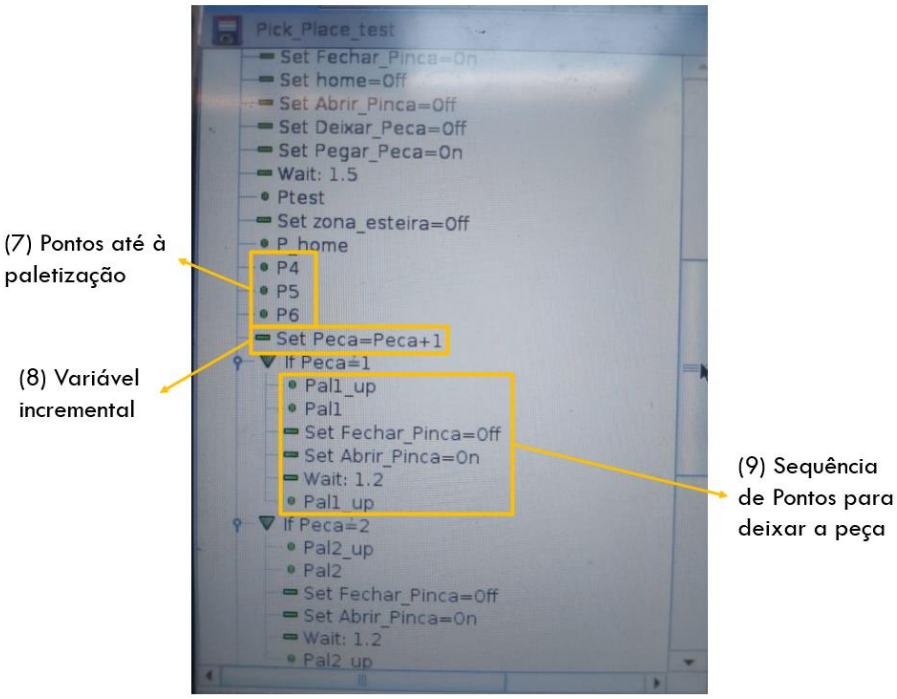


Figura 6-3 Programa UR10: Parte 2

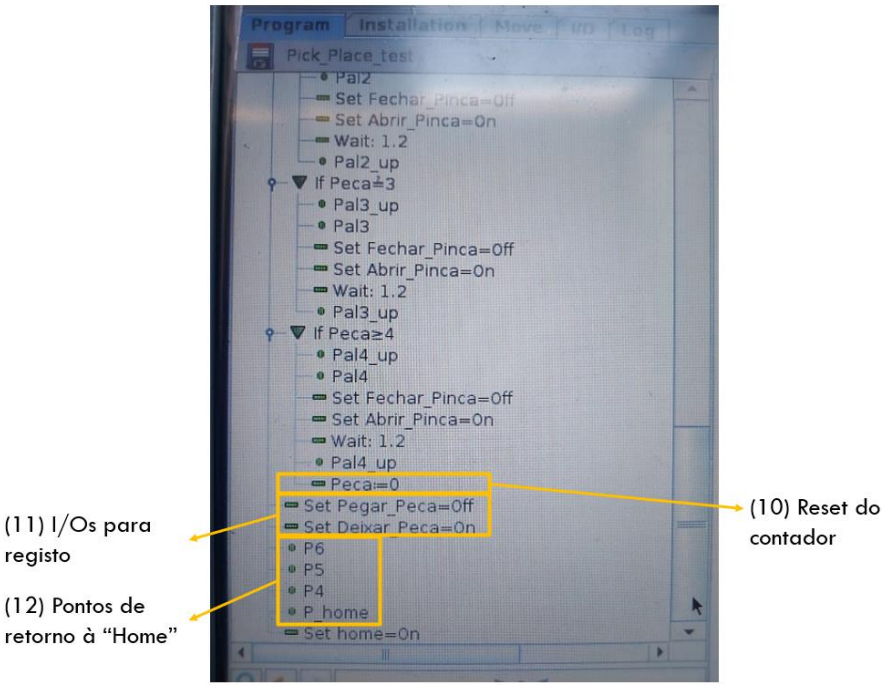


Figura 6-4 - Programa UR10: Parte 3

7. Atuação Pneumática

A atuação pneumática é um método de controle e movimentação de dispositivos mecânicos que utiliza o ar comprimido como meio de energia. É amplamente utilizada na indústria devido a várias vantagens, como simplicidade, confiabilidade, segurança e baixo custo.

Nos projetos de automação industrial é utilizada para acionar válvulas de controle, cilindros pneumáticos e atuadores, permitindo o controle de processos e movimentação de equipamentos.

Neste projeto, recorreu-se à atuação pneumática para realizar o fecho da pinça com recurso a uma electroválvula. Estes componentes desempenharam um papel crucial no transporte das peças em segurança para os pontos pretendidos. As funções de todos estes componentes serão abordadas com mais detalhe nas secções seguintes.

7.1. Equipamentos

1. Electroválvula de Atuador Pneumático

Utilizou-se uma electroválvula de cinco vias, duas posições e ativação por solenoide (ver Figura 7-2). O objetivo desta electroválvula é de regular a direção do fluxo de ar que entrará no atuador: numa posição, o fluxo faz a haste do atuador avançar e desta forma fixar a peça, na outra posição, o fluxo faz a haste do atuador recuar e desta forma tornar a peça livre. A sua ativação é realizada a partir de saídas digitais do PLC.

O esquema pneumático é apresentado na Figura 7-1, nele estão representados os componentes do sistema. Para o realizar, utilizou-se um software de desenho de esquemas pneumáticos da SMC.

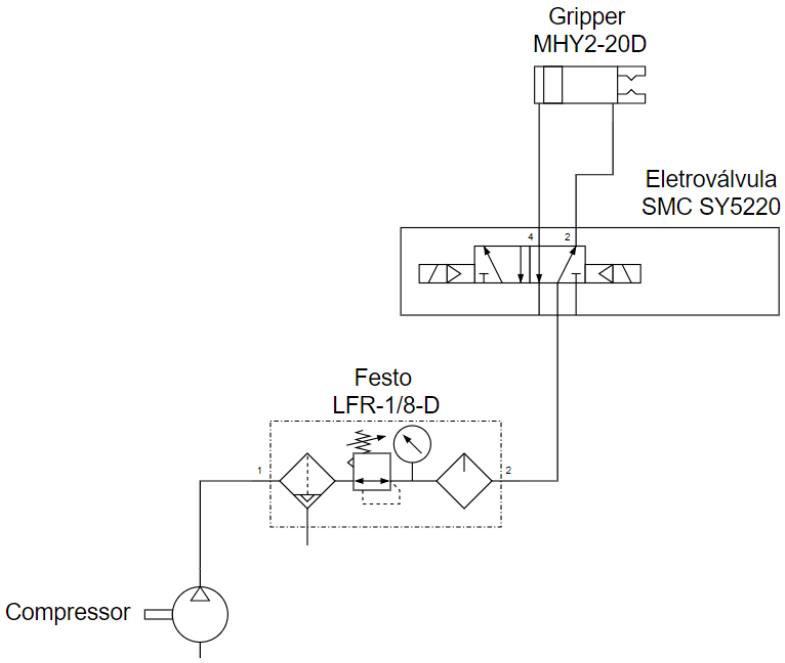


Figura 7-1 - Esquema Pneumático

Figura 7-2 – Válvula Pneumática SMC SY5220

2. Gripper

O sistema de aperto e transporte das peças foi utilizado um *gripper* da SMC (MHY2-20D)[5] acionado por atuação pneumática (Figura 7-4). Este *gripper* apresenta como pressão de trabalho 1 - 8 [bar], uma vez que o sistema está pressurizado a 4 - 5 [bar] o gripper está a operar no devido intervalo de trabalho.

Para este equipamento foi impressa em PLA uma peça para fazer o acoplamento do mesmo com o robô, em que foi fixo com recurso a parafusos representado na Figura 7-3 e na Figura 7-4.

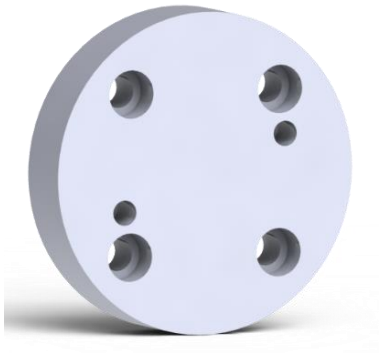


Figura 7-3 - Modelo CAD da peça de fixação do Gripper

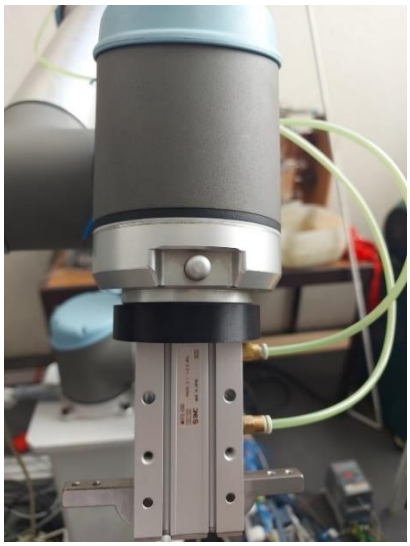


Figura 7-4 – Gripper SMC MHY2-20D

8. Interface em Visual Basic

Visual Basic (VB) é uma linguagem de programação amplamente utilizada para desenvolver aplicativos de desktop e interfaces gráficas de utilizador (GUI). Uma das principais características do VB é a sua capacidade de criar interfaces interativas e intuitivas por meio de recursos poderosos e fáceis de usar. Para criar interfaces em VB, utilizou-se o ambiente de desenvolvimento integrado (IDE) fornecido pelo *Microsoft Visual Studio*.

O desenvolvimento de uma interface com o utilizador permite ao utilizador o controlo do sistema e a visualização de algumas informações, em tempo real, do mesmo. Nesse sentido, procurou-se desenvolver esta interface com o intuito do controlo do sistema e também da visualização de alguns parâmetros.

Relativamente ao controlo será possível:

- Ativar o modo automático ou o modo manual do sistema e ter feedback de qual modo de operação do sistema é que se encontra ativo.
- Realizar *o Start, Stop e Pause* do robô UR10 e ter feedback do estado do robô.
- Ligar/Desligar o PLC e verificar o seu estado.

Relativamente à visualização de parâmetros do sistema é possível:

- Visualizar a quantidade de peças que o robô UR10 já colocou na palete.
- Visualizar, em tempo real, o valor da correção das coordenadas X e Y e o valor do ângulo (R_Z), em graus, que o robô UR10 está a executar.
- Visualizar, através do botão de "*Diag*" (diagnóstico) os valores de qualquer variável do PLC. Para isso recorre-se também aos botões "*DB Read*" e "*MW Read*"

Na Figura 8-1 encontra-se a interface gráfica desenvolvida para este projeto. No repositório do GitHub[6] e no vídeo de demonstração[7] deste projeto é possível encontrar o ficheiro da interface com o utilizador em VB que contém o código desenvolvido para a mesma.

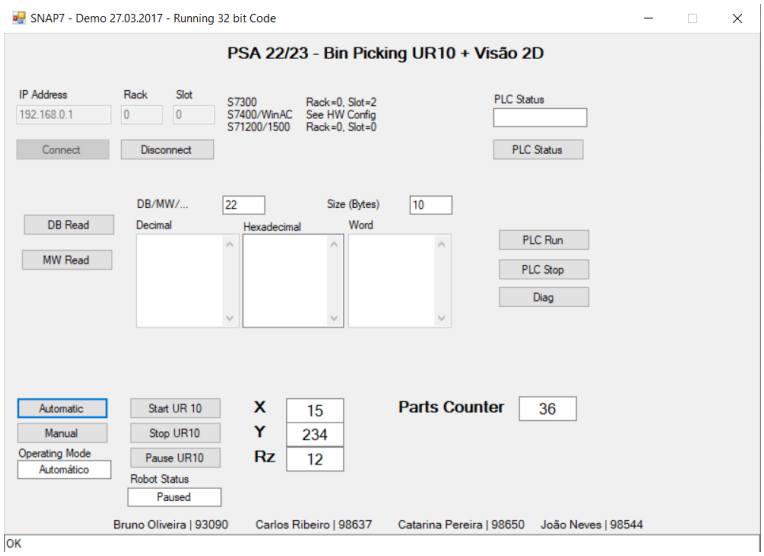


Figura 8-1 – Interface gráfica com o utilizador

9. Conclusão

Após a conclusão do desenvolvimento, o sistema mostrou-se altamente eficiente na identificação e manipulação das peças. Através do uso da visão 2D, as peças foram reconhecidas de forma precisa e confiável, permitindo a execução das tarefas de posicionamento pelo robô UR10 de forma rápida e precisa.

No entanto, o projeto teve certas etapas em que se verificou uma dificuldade adicional. Nestas partes, houve momentos de intensa pesquisa e esforço por parte dos membros do grupo para realizar e superar esses obstáculos.

Os principais problemas que enfrentamos foi na parte de comunicação com os diferentes equipamentos integrantes do projeto. A comunicação PLC-UR10 foi desafiante no sentido em que se teve de procurar escrever nas entradas do UR10 com os tipos de variáveis corretas, que nos fez pesquisar mais relativamente ao funcionamento do robô. A comunicação PLC-Dalsa Boa também foi difícil de se realizar, neste campo tivemos de explorar mais competências relativamente ao software *Sherlock Embedded*. Por fim, a comunicação revelou-se numa etapa difícil, mas permitiu nos desenvolver mais competências neste âmbito.

Para além disso, o conjunto de sistemas utilizado, isto é, o sistema de visão e a programação do robô permitiu estar em contacto com grande parte de áreas utilizadas em projetos de automação industrial.

Assim, o sistema desenvolvido para manipulação de componentes utilizando recursos de visão 2D e o robô UR10 mostrou-se uma solução promissora para a automação de processos industriais. A capacidade de identificar e posicionar as peças de forma precisa e eficiente abre portas para a otimização de processos, redução de erros e aumento da produtividade. Este projeto representa um avanço significativo no campo da automação industrial e demonstra o potencial da combinação de visão computacional e robótica colaborativa para o desenvolvimento de sistemas inteligentes e adaptáveis.

10. Referências

[1] **Conversor da matriz de rotação**, disponível em URL:<https://www.andre-gaschler.com/rotationconverter/>.[Acessed: 10/06/2023]

[2] **Transformação de coordenadas**, disponível em URL:

https://thomasfermi.github.io/Algorithms-for-Automated-Driving/Introduction/intro.html

[Acessed: 10/06/2023]

[3] **Bosch-Rexroth-EFC3610**, disponível em URL:<https://inverterdrive.com/file/Bosch-Rexroth-EFC3610-Easy-Start-Guide>.[Acessed: 10/06/2023]

[4] **HNYOUNG-NUX-PE-R05D**, disponível em URL:

http://hanyoungnux.com/mobile/product/product-info02.php?lcode=05&mcode=&pcode=1305100080

[Acessed: 10/06/2023]

[5] **SMC gripper MHY2-20D**, disponível em URL:

https://www.smcpneumatics.com/pdfs/MHY_MHW.pdf

[Acessed: 10/06/2023]

[6] **Repositório do github**, disponível em URL:

https://github.com/BrunoO26/PSA2023_UR/tree/main

[Acessed: 10/06/2023]

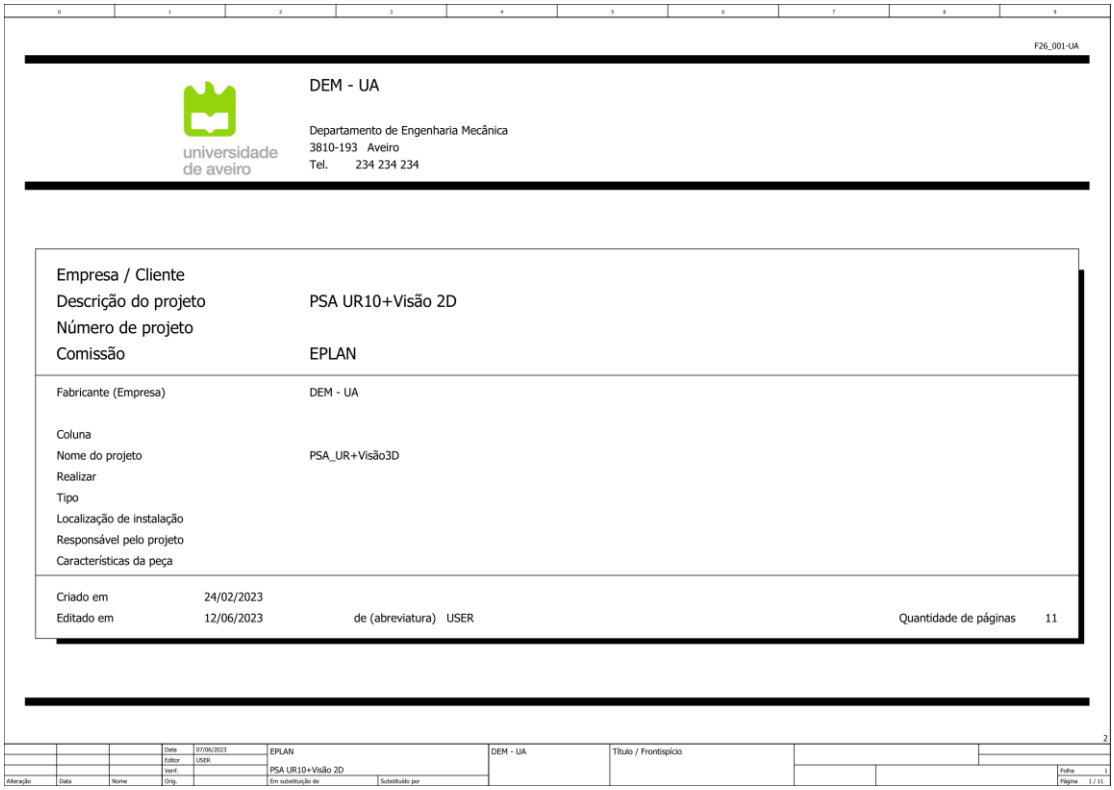
[7] **Vídeo de demonstração**, disponível em URL:

https://www.youtube.com/watch?v=vr1dsS1yN4A&ab_channel=BrunoOliveira

[Acessed: 12/06/2023]

11. Anexos

A. Projeto Elétrico



F26_001-UA

DEM - UA

Departamento de Engenharia Mecânica
3810-193 Aveiro
Tel. 234 234 234

Empresa / Cliente	
Descrição do projeto	PSA UR10+Visão 2D
Número de projeto	
Comissão	EPLAN
Fabricante (Empresa)	DEM - UA
Coluna	
Nome do projeto	PSA_UR+Visão3D
Realizar	
Tipo	
Localização de instalação	
Responsável pelo projeto	
Características da peça	

Criado em 24/02/2023
Editado em 12/06/2023 de (abreviatura) USER Quantidade de páginas 11

			Data	07/06/2023	EPLAN	DEM - UA	Título / Frontispício
			Editor	USER	PSA UR10+Visão 2D		
			Verif.				Folha 1
Alteração	Data	Nome	Orig.		Em substituição de	Substituído por	Página 1 / 11

Figura 11-1 – Título / Frontispício

Tabela de conteúdos

Coluna X: uma página gerada automaticamente foi alterada manualmente F06_001

Página	Descrição da página	Campo suplementar da página	Data	Editor	X
/1	Título / Frontispício		07/06/2023	USER	X
/2	Tabela de conteúdos : /1 - /100		07/06/2023	USER	
/5	Alimentação Geral		07/06/2023	USER	
/6	Alimentação Fonte Alimentação		07/06/2023	USER	
/20	Comando Variador		07/06/2023	USER	
/40	Alimentação PLC		07/06/2023	USER	
/41	Alimentação Dalsa BOA		07/06/2023	USER	
/42	Alimentação UR10		07/06/2023	USER	
/43	Rede Profinet		07/06/2023	USER	
/50	Entradas Digitais		07/06/2023	USER	
/100	Saídas Digitais		07/06/2023	USER	

			Data	07/06/2023	EPLAN	DEM - UA	Tabela de conteúdos : /1 - /100
			Editor	USER	PSA UR10+Visão 2D		
			Verif.				Folha 2
Alteração	Data	Nome	Orig.		Em substituição de	Substituído por	Página 2 / 11

Figura 11-2 – Tabela de conteúdos: /1 - /100

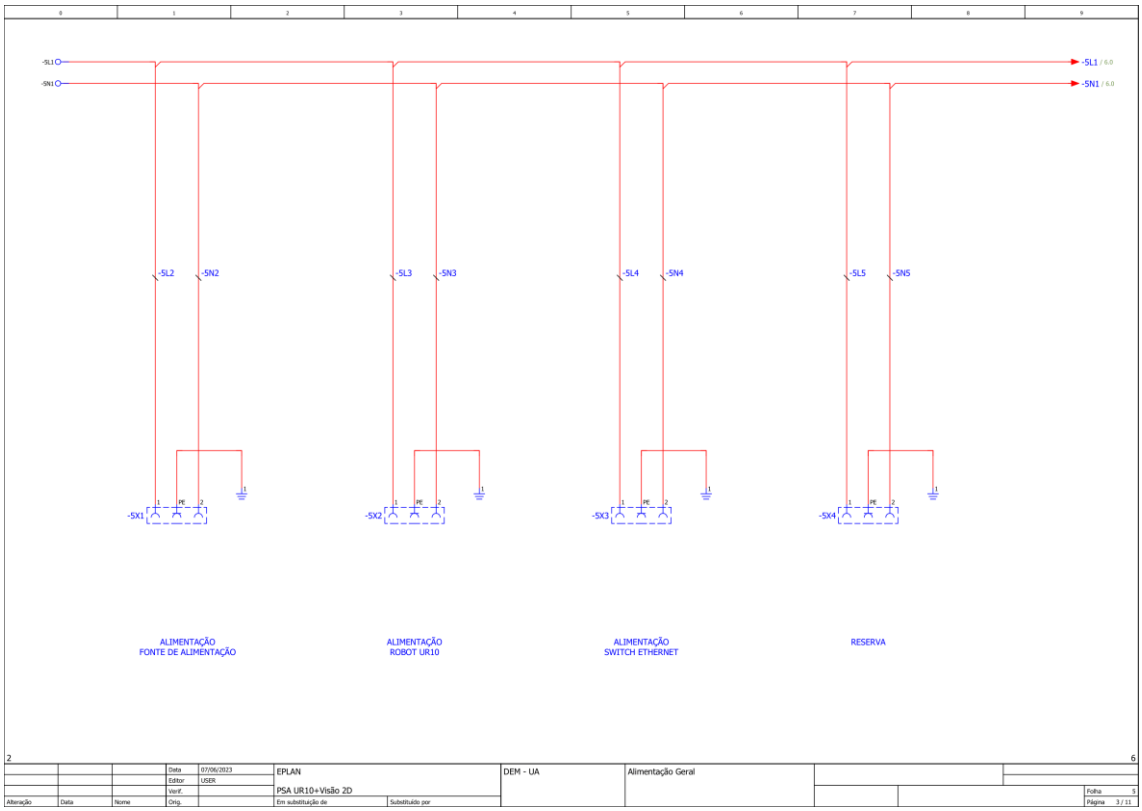


Figura 11-3 – Alimentação Geral

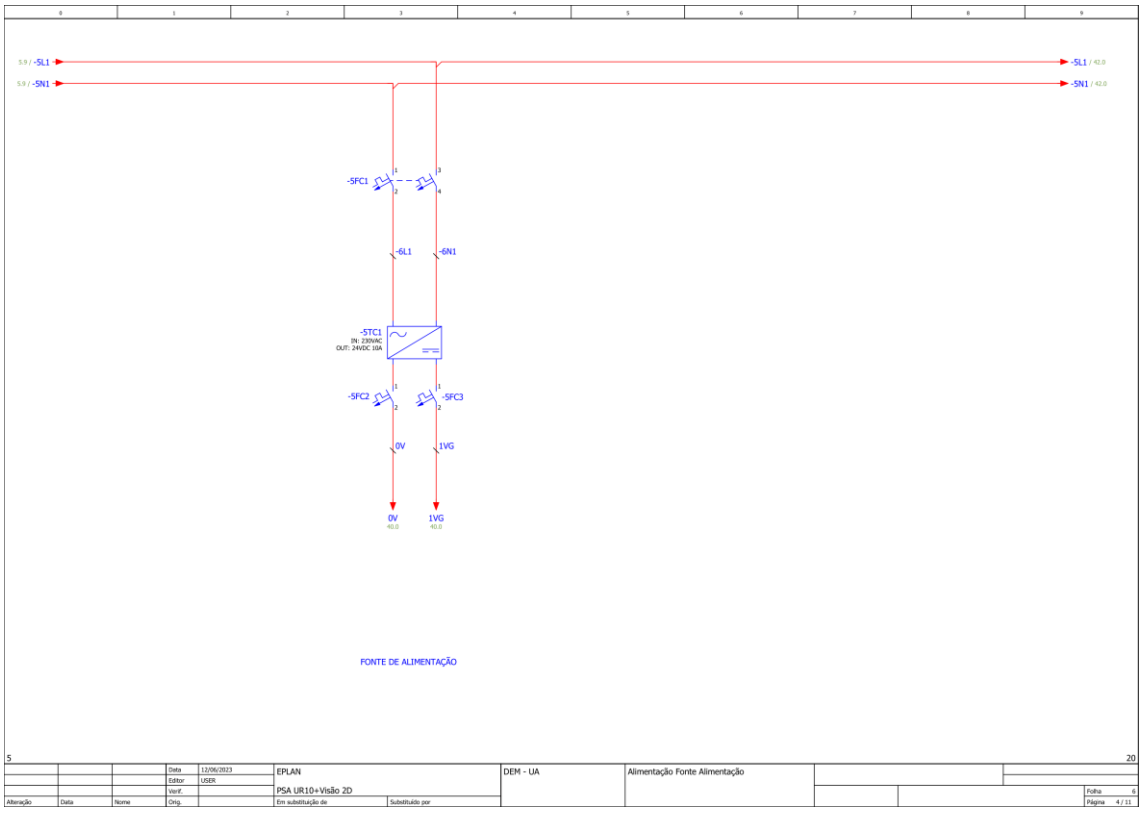


Figura 11-4 - Alimentação Fonte Alimentação

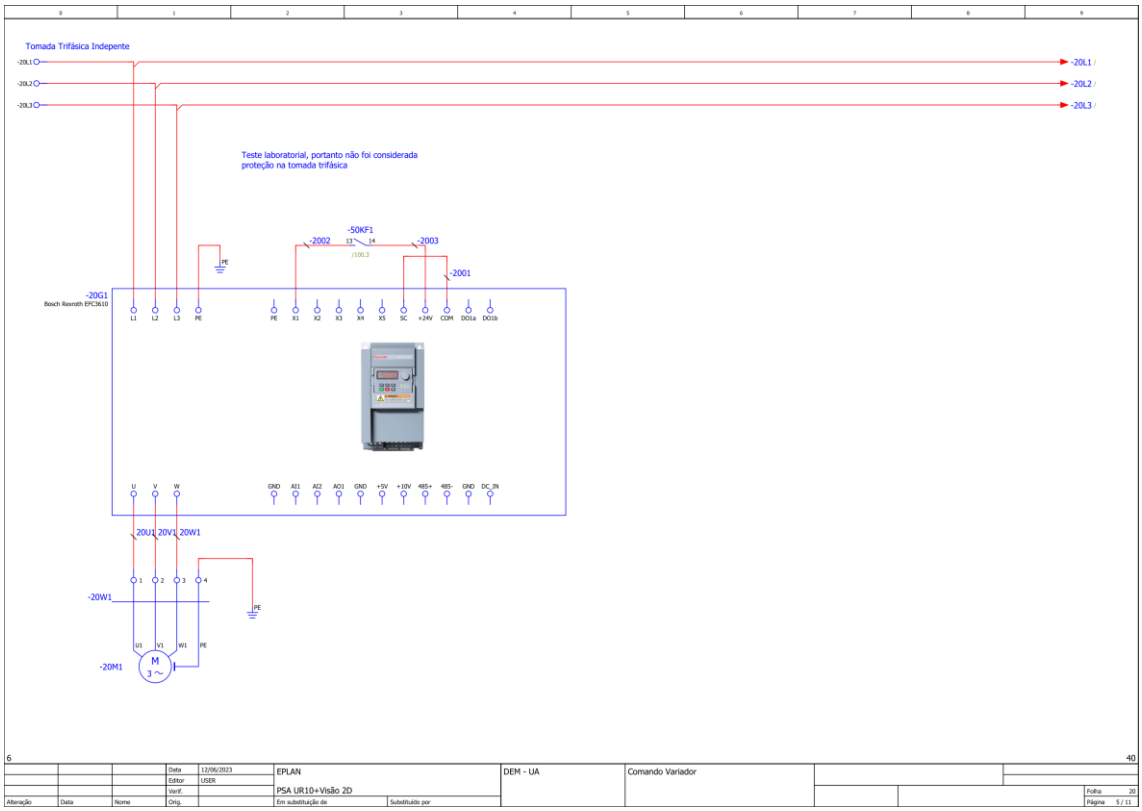


Figura 11-5 – Comando Variador

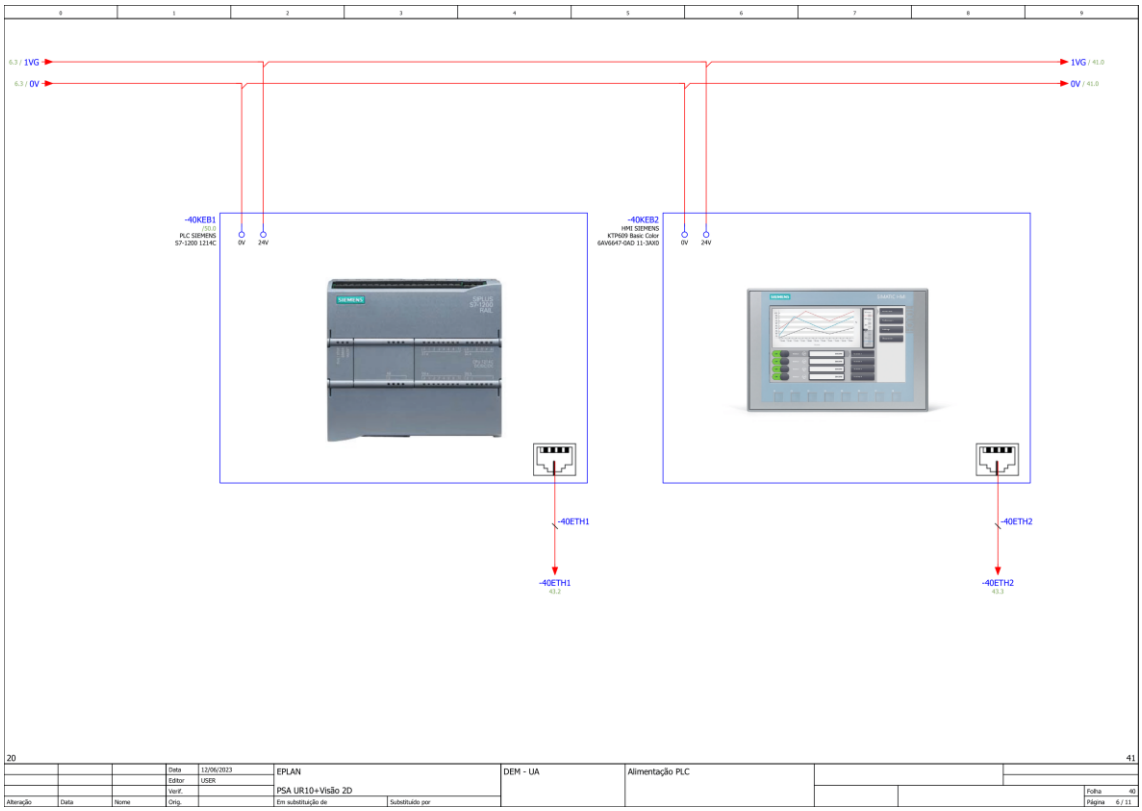


Figura 11-6 – Alimentação PLC

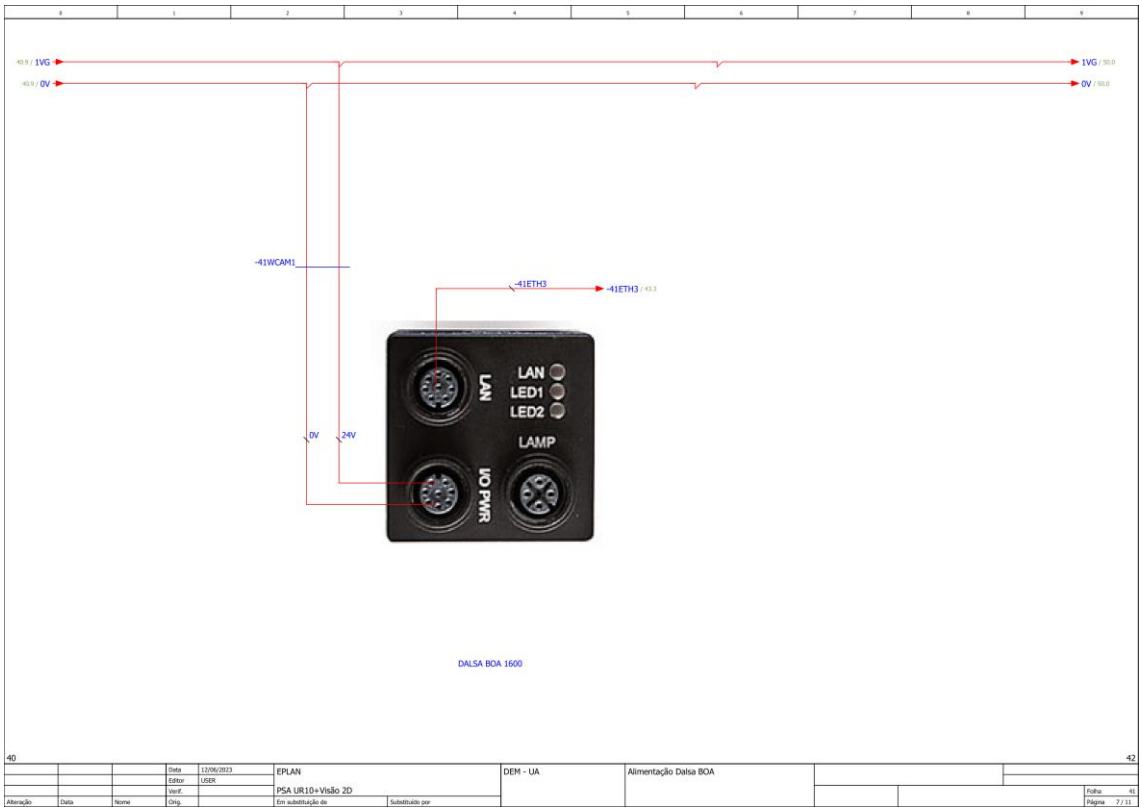


Figura 11-7 – Alimentação Dalsa BOA

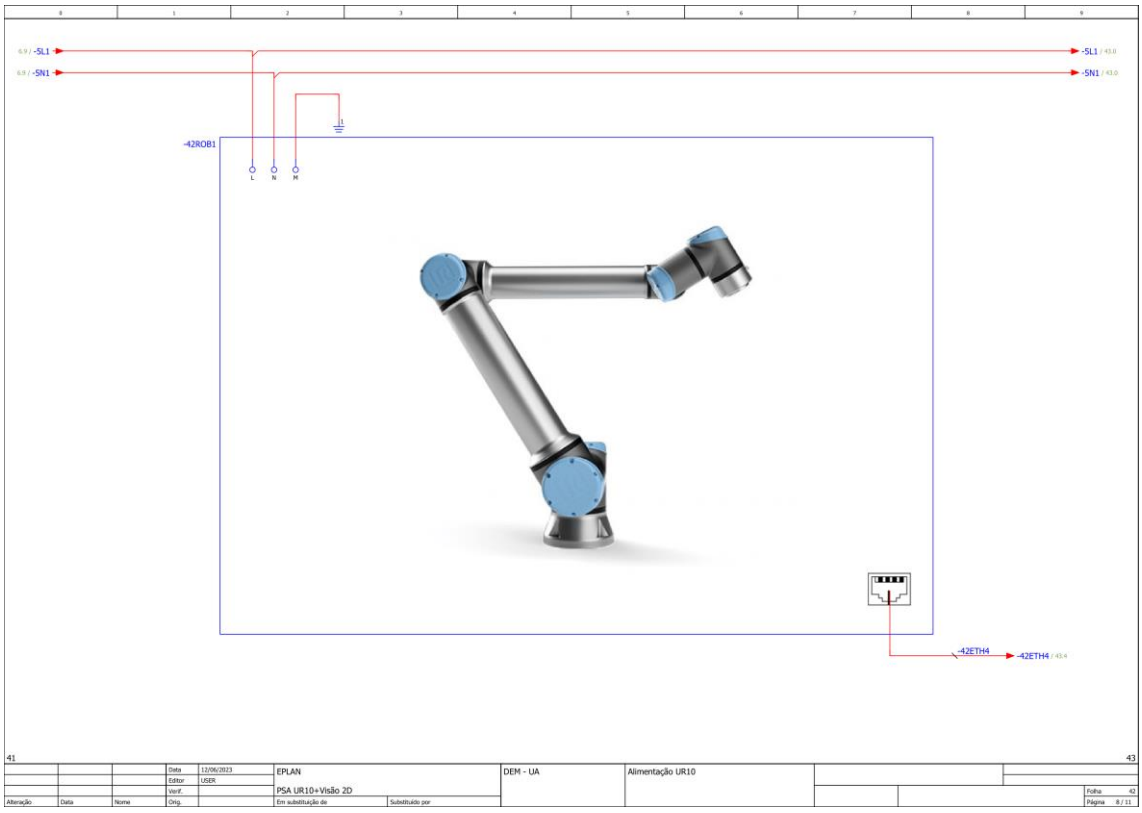


Figura 11-8 – Alimentação UR10

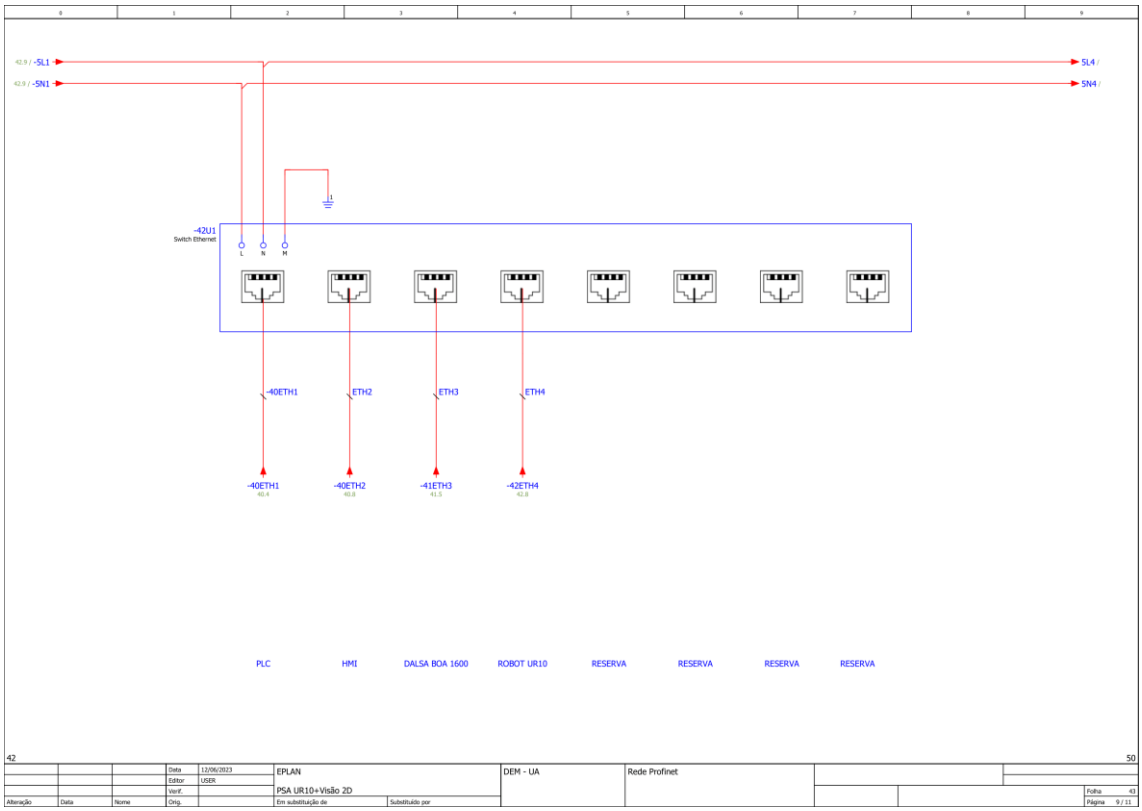


Figura 11-9 – Rede Profinet

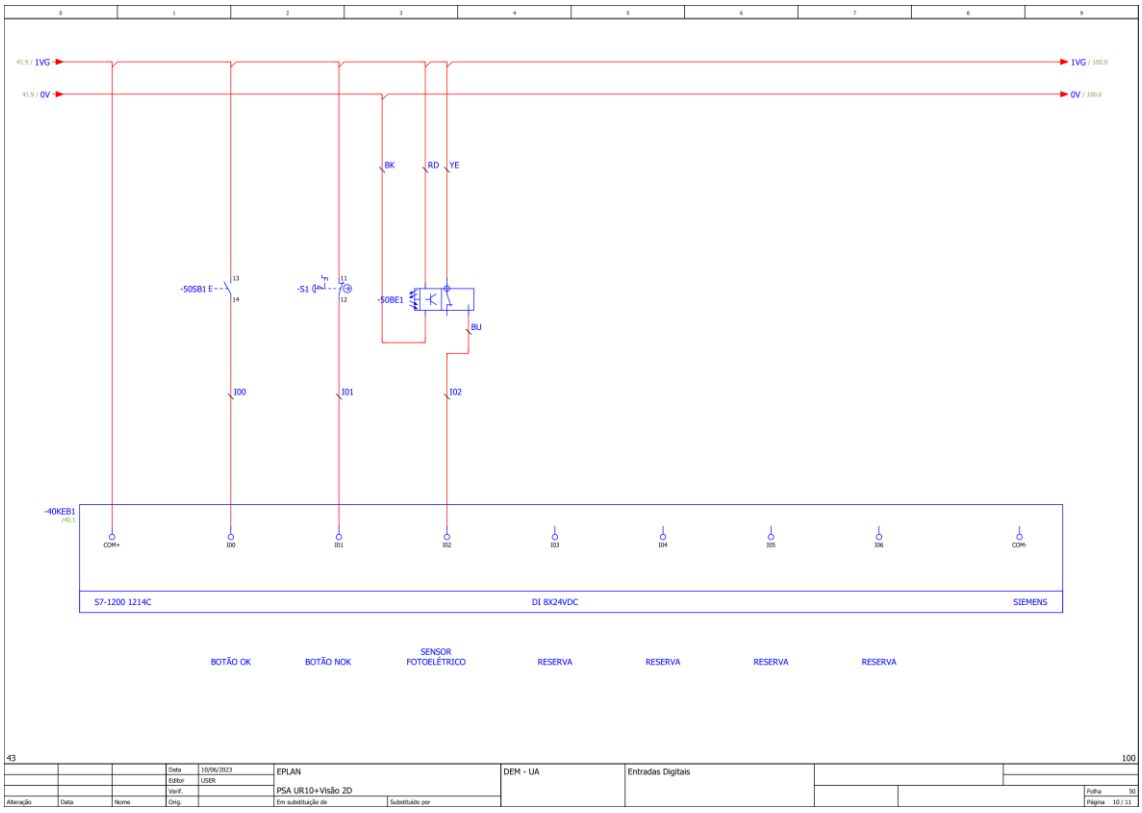


Figura 11-10 – Entradas Digitais

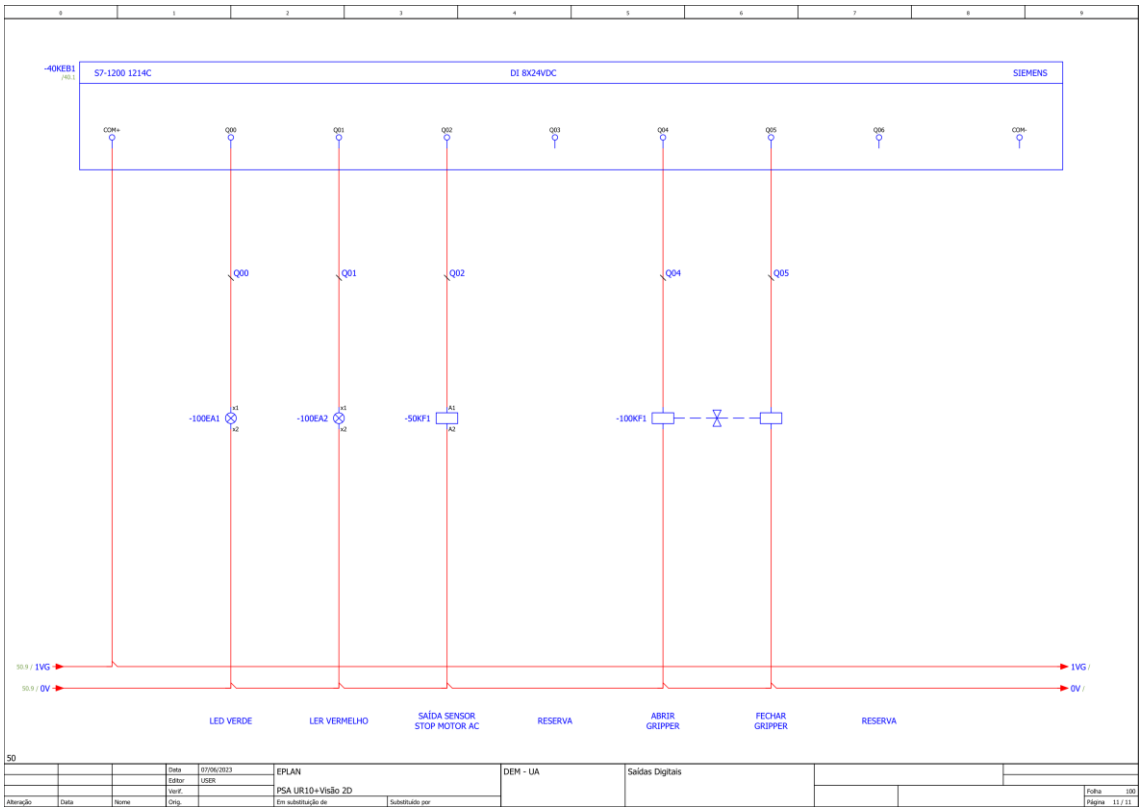


Figura 11-11 – Saídas Digitais

B. Programação Visão 2D

Programa desenvolvido no software *Sherlock Embedded* para a câmara de visão 2D.

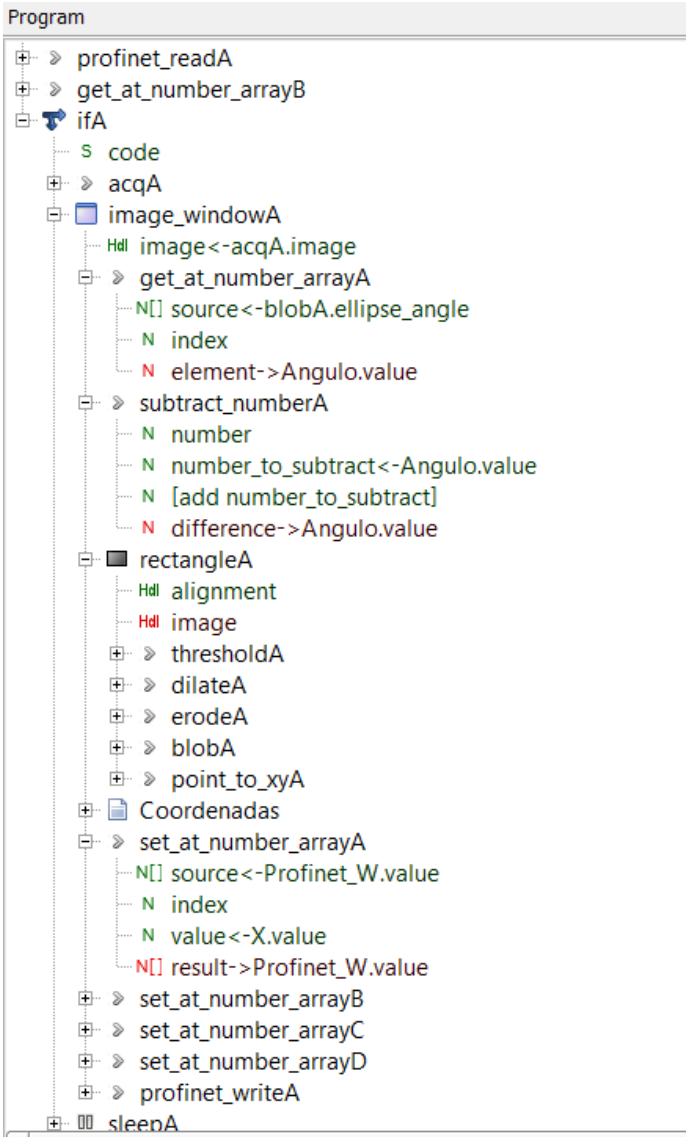


Figura 11-12 - Código geral em Sherlock Embedded

Script da transformação de coordenadas em *JavaScrip*:

Scrip_Sherlock.txt

```
const K = [
  [3243.0, 0.0, 640.0],
  [0.0, 3243.0, 480.0],
  [0.0, 0.0, 1.0]
];


const y = -0.055; // distância camera pinça
const Pos_cam = [-0.61495, -0.352, 0.17121, 3.10, 0.032, -0.023 - 3.1415];
const T = [[Pos_cam[0], Pos_cam[1], Pos_cam[2]]];
const TT = [
  [[Pos_cam[0]],
  [Pos_cam[1]],
  [Pos_cam[2]]]
];
const R = [
  [1, 0, 0],
  [0, -1, 0],
  [0, 0, -1]
];

const h = -0.55; // altura
const nw = [[0, 0, 1]]; // normal em relação ao mundo
const nwT = [
  [0],
  [0],
  [1]
];


var nc = multiply(R, nwT); // normal em relação à camera
const c = [[VEngine.point_to_xyA.x], [VEngine.point_to_xyA.y], [1]]; // centroide
const cT = [
  [VEngine.point_to_xyA.x],
  [VEngine.point_to_xyA.y],
  [1]
];


const Kinv = [
  [0.00030836, 0, -0.1973],
  [0, 0.00030836, -0.1480],
  [0, 0, 1]
];
const ncT = [[0, 0, 1]];
var passo1 = multiply(ncT,Kinv);
var passo2 = multiply(passo1,cT);
const passo3 = h/passo2;
var passo4 =multiply(Kinv,cT);
const elemento1 = passo3*passo4[0][0];
const elemento2 = passo3*passo4[1][0];
const elemento3 = passo3*passo4[2][0];
const test = [
  [elemento1],
  [elemento2],
  [elemento3]
];

var posicao =multiply(R,test);
const X = (TT[0][0]) + (posicao[0][0]);
const Y = TT[0][1] + posicao[1][0];
const Z = posicao[2][0] + TT[0][2];

VEngine.varF.value = TT[0][0];
VEngine.varG.value = posicao[0][0];
VEngine.X.value = (VEngine.varG.value + VEngine.varF.value)

VEngine.varF.value = TT[0][1];
VEngine.varG.value = posicao[1][0];
VEngine.Y.value = (VEngine.varG.value + VEngine.varF.value)

VEngine.varF.value = TT[0][2];
VEngine.varG.value = posicao[2][0];
VEngine.Z.value = (VEngine.varG.value + VEngine.varF.value)


const pos = [
  [VEngine.X.value],
  [VEngine.Y.value],
  [VEngine.Z.value]
];

VEngine.varF.value = TT[0][0];
 VEngine.varG.value = pos[0][0];
 VEngine.difX.value = VEngine.varG.value - VEngine.varF.value

VEngine.varF.value = TT[0][1];
VEngine.varG.value = pos[1][0];
VEngine.difY.value = VEngine.varG.value - VEngine.varF.value

VEngine.varF.value = TT[0][2];
VEngine.varG.value = pos[2][0];
VEngine.difZ.value = VEngine.varG.value - VEngine.varF.value
```

Figura 11-13 – Script de transformação de coordenadas: Parte 1

```
99
100   function multiply(m1, m2) {
101   var result = [];
102   for (var i = 0; i < m1.length; i++) {
103     result[i] = [];
104     for (var j = 0; j < m2[0].length; j++) {
105       var sum = 0;
106       for (var k = 0; k < m1[0].length; k++) {
107         sum += m1[i][k] * m2[k][j];
108       }
109       result[i][j] = sum;
110     }
111   }
112   return result;
113 }
114   function multiplyElement(m1, m2) {
115   var result = [];
116   for (var i = 0; i < m1.length; i++) {
117     result[i] = [];
118     for (var j = 0; j < m2[0].length; j++) {
119       var sum = 0;
120       for (var k = 0; k < m1[0].length; k++) {
121         sum += m1 * m2[k][j];
122       }
123       result[i][j] = sum;
124     }
125   }
126   return result;
127 }
128
129
130
131
132   VEngine.varH.value = pos;
133
134
135   // VEngine.varC.value = VEngine.varH.value [0][0]
136
```

Figura 11-14 - Script de transformação de coordenadas: Parte