Universidade do Porto

Faculdade de Engenharia

FEUP

Projeto Final

Braço Robótico

Gonçalo Nunes Rodrigues
Guilherme Miguel Monteiro da Costa

Relatório do Projeto Final
Arquitetura de Computação Embarcada
Mestrado em Engenharia Eletrotécnica e de Computadores

02/02/2025

Conteúdo

1 Introdução

O presente relatório descreve o desenvolvimento de um projeto de arquiteturas de computação embarcada, onde foi implementado um sistema de manipulação e classificação de peças utilizando um braço robótico.

O objetivo principal é recolher peças de cores distintas, classificar cada uma através de um sensor de cor e depositá-las nos respetivos locais de depósitos pré-definidos.

O relatório detalha a implementação de três cenários distintos:

- Recolha e classificação de uma peça posicionada num local fixo.
- Recolha e classificação de nove peças posicionadas numa grelha 3x3.
- Recolha e classificação de uma peça posicionada numa localização aleatória, detetada através de um sensor de distância.

Um dos principais desafios foi garantir a comunicação entre o microcontrolador e os sensores através do barramento I2C, evitando conflitos e assegurando leituras corretas. A integração dos servos também exigiu especial atenção à geração dos sinais PWM para garantir o controlo dos movimentos.

Outro aspeto crítico foi a organização do código e a implementação de um algoritmo que permitisse o funcionamento simultâneo dos sensores e atuadores, garantindo que o sistema respondesse de forma correta às leituras obtidas. A sincronização, organização e otimização do tempo de resposta foram aspetos a considerar com algum cuidado de modo a garantir um desempenho adequado.

2 Material e Ferramentas Utilizadas

2.1 Hardware

Os componentes utilizados no projeto incluem:

- **Microcontrolador**: Raspberry Pi Pico W.
- **Motherboard**: Board com headers para o Raspberry Pi Pico.
- **Servos**: 4 unidades de servomotores MG996R (180 graus).
- **Placa de Controlo de Servos**: PCA9685PW.
- **Sensor RGB**: TCS34725 para deteção de cores.
- **Sensor de Distância**: Sensor ToF (Time-of-Flight).
- **Cabos de Ligação**: Conjunto de cabos fêmea-fêmea de 20 cm.

2.2 Software

- IDE: VS Code.
- Bibliotecas:
  - `Adafruit_TCS34725`: Para a comunicação com o sensor de cor.
  - `Adafruit_PCA9685`: Para o controlo dos servos.
  - `VL53L0X`: Para a comunicação com o sensor de distância.

2.3 Ligações realizadas

O circuito liga o *Raspberry Pi Pico* a quatro servos *MG996R* e a dois sensores, um de distância *VL53L0X ToF* e um de cor *TCS34725 RGB*. Além disso, é utilizado um controlador de servos *PCA9685*.

- Os servos são controlados pela *PCA9685*, que recebe comandos do *Pico* através do barramento *I2C*. O módulo é alimentado a 5V e liga-se aos servos, fornecendo os sinais PWM necessários para o seu funcionamento.
- Os sensores comunicam com o microcontrolador através do barramento *I2C*. O *VL53L0X* mede distâncias e liga-se ao *Pico* através dos pinos *SDA* e *SCL*, além da alimentação a 5V e *GND*. O pino *XSHUT* pode ser utilizado para ativação/desativação. O sensor *TCS34725*, que deteta cores, também usa o barramento *I2C*, mas é alimentado a 3.3V, partilhando as linhas de dados com o *VL53L0X*.
- O *Raspberry Pi Pico* tem ligações diretas para comunicação com o barramento *I2C* e alimentação dos sensores. O circuito garante uma ligação estável entre todos os componentes, permitindo um controlo preciso dos atuadores e uma correta leitura dos sensores.

A utilização do barramento *I2C* permite que os sensores e o controlador de servos partilhem as mesmas ligações sem interferências, sendo identificados pelos seus endereços únicos. Com esta configuração, o *Raspberry Pi Pico* pode controlar os servos e obter leituras precisas dos sensores de distância e cor.

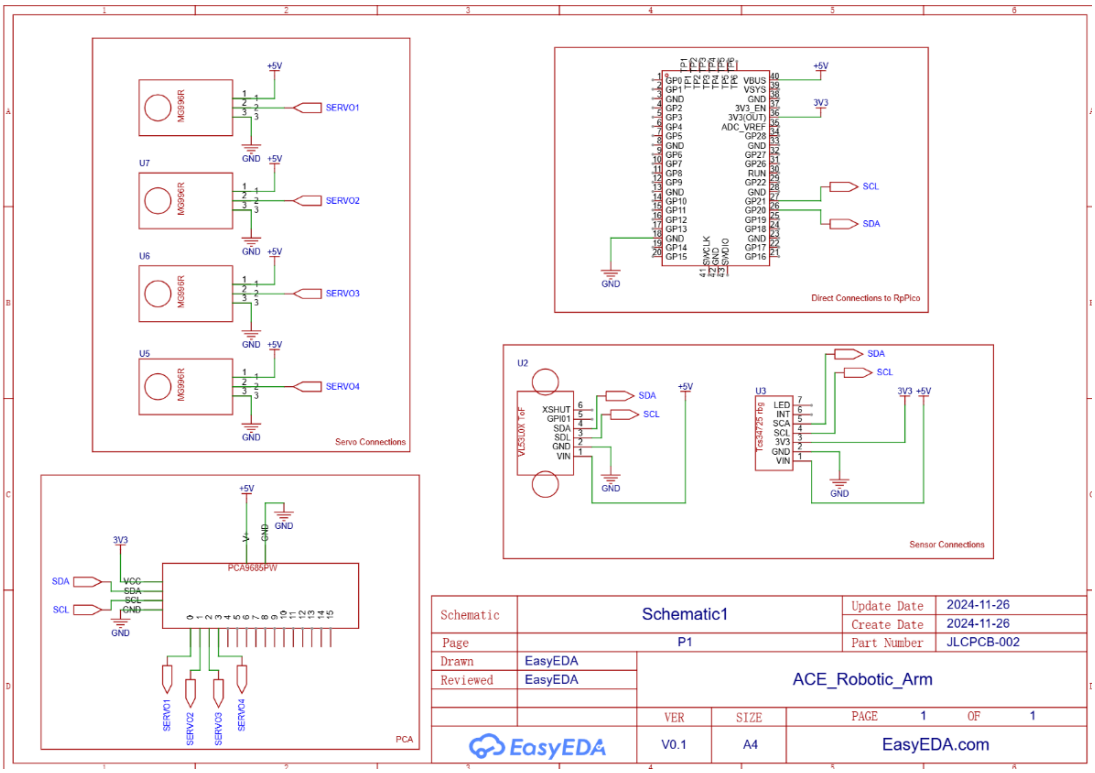

Figura 1: Esquema de ligações realizadas

2.4 Montagem

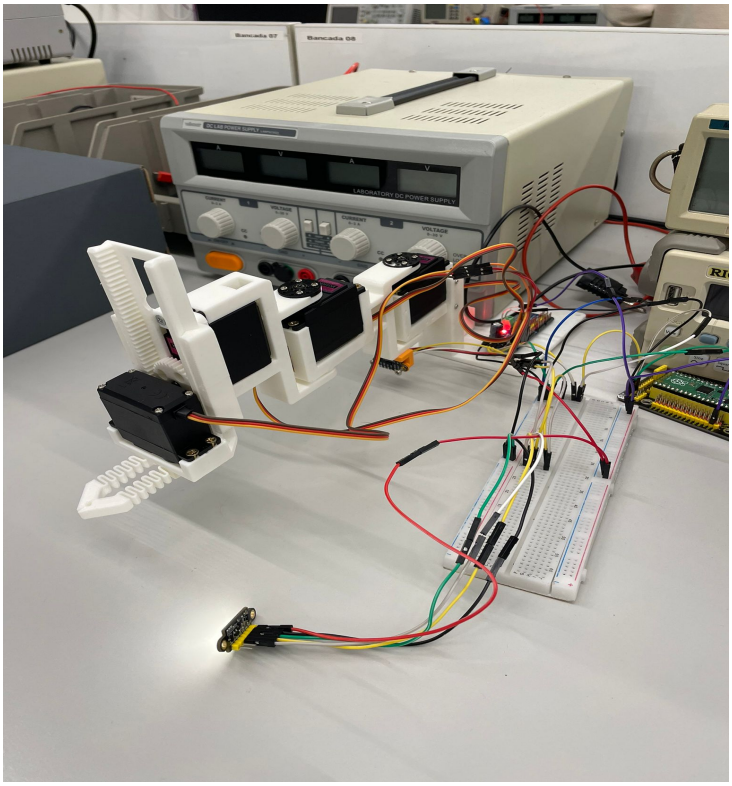

Figura 2: Montagem física do sistema.

Como é possível ver na imagem, o braço define-se principalmente pelos quatro *Servo Motors*. Existe um servo responsável pela rotação da base, outro servo responsável pela rotação do segundo segmento do braço, seguido de um terceiro servo responsável por definir a altura da garra e por último, o quarto servo responsável por abrir e fechar a garra. Debaixo do *Servo Motor* de base está um sensor de distância *ToF VL53L0X*. Existe também uma base que acomoda a PCA para facilitar a organização de cabos. Externamente a esta estrutura do braço, há também a *Breadboard* que acomoda as ligações entre o *Pico*, definindo o barramento I2C que os sensores e a PCA utilizam para comunicar.

3 Desafios de implementação

Definidos os requisitos físicos e as dependências do projeto, enfrentámos alguns desafios no que toca ao desenvolvimento do software

3.1 Cinemática Inversa

Os *Servos* utilizam-se devido à sua precisão no que toca a ângulo de rotação, no entanto, a operação desejada exige coordenadas, logo, foi necessário desenvolver uma solução de Cinemática Inversa para, a partir de uma posição conhecida, determinar os ângulos que cada servo deve ter para atingir esse ponto.

Figura 3: Referêncial para cálculo da solução de cinemática inversa.

Considerando a posição de repouso como o eixo Oy, o primeiro *Servo*, ao qual chamámos *BASE*, definiu-se como o responsável pela rotação do primeiro segmento, e o segundo *Servo Motor*, ao qual chamámos *ELBOW*, definiu-se como o responsável pela rotação do segundo segmento. Para, sabendo X e Y do ponto desejado, determinar os ângulos que cada *Servo Motor* deve ter determinámos que:

$$\theta_1 = \alpha + \Omega \tag{1}$$

$$\alpha = \arctan\left(\frac{y}{x}\right) \tag{2}$$

Aqui surgiu o primeiro problema, pois não existe uma razão trigonométrica que nos permitisse obter Ω diretamente. Por isso, foi utilizada a lei dos cossenos, pois temos conhecimento dos valores de ambos os segmentos e da hipotenusa, podendo assim definir um triângulo e obter Ω a partir da seguinte equação:

$$\Omega = \arccos\left(\frac{Seg1L^2 + Hipotenuse^2 - Seg2L^2}{2 \cdot Seg1L \cdot Hipotenuse}\right) \tag{3}$$

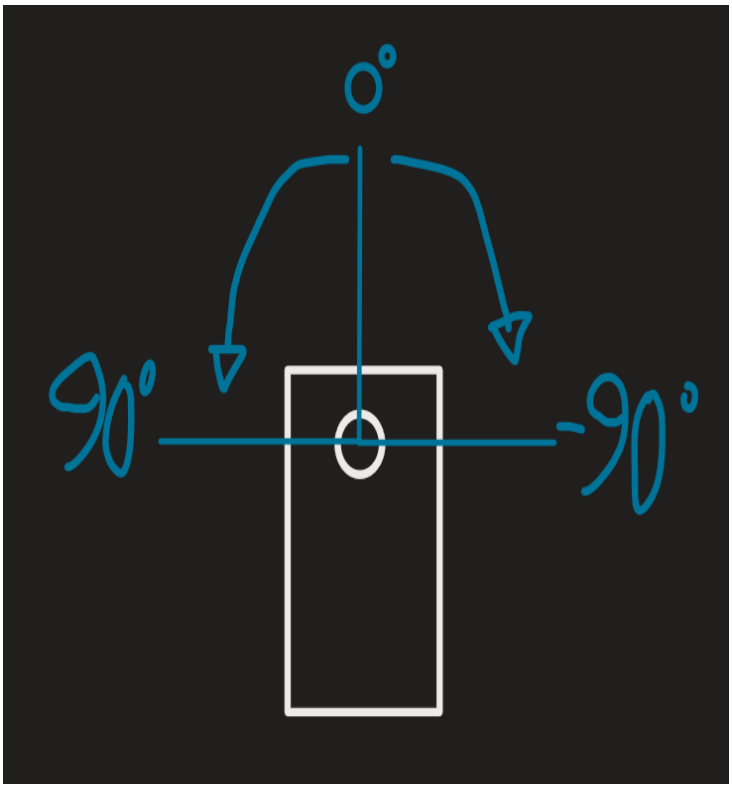

Figura 4: Orientação de ângulos para os servos utilizados.

Com isto, apenas se torna necessário aplicar um offset ao θ_1 de forma a obter o ângulo "absoluto"para o servo. Como a posição de repouso é o eixo Oy, considerámos $\theta_1 = 0°$ para essa posição, logo, de acordo com a imagem acima, obtém-se o ângulo para $BASE$ a partir da seguinte equação:

$$\theta_{s1} = \theta_1 - 90° = \Omega + \alpha - 90° \tag{4}$$

$$\theta_{s1} = \arccos\left(\frac{Seg1L^2 + Hipotenuse^2 - Seg2L^2}{2 \cdot Seg1L \cdot Hipotenuse}\right) + \arctan\left(\frac{y}{x}\right) - 90° \tag{5}$$

Passando agora para o ângulo de $ELBOW$, considerando também que $\theta_{s2} = 0°$ no eixo Oy, podemos concluir que o ângulo $-\phi$ corresponderá diretamente a θ_{s2}. ϕ, por sua vez, corresponde a subtrair de 180 o valor de θ_2. Novamente, a partir da lei dos cossenos com o triângulo *Seg1-Seg2-Hipotenuse*, podemos definir que:

$$\theta_2 = \arccos\left(\frac{Seg2L^2 + Seg1L^2 - Hipotenuse^2}{2 \cdot Seg1L \cdot Seg2L}\right) \tag{6}$$

E, consequentemente, que:

$$\theta_{s2} = -\phi = \theta_2 - 180° \tag{7}$$

Em suma, a nossa solução de cinemática inversa corresponde a:

$$\text{BASE} = \theta_{s1} = \arccos\left(\frac{Seg1L^2 + Hipotenuse^2 - Seg2L^2}{2 \cdot Seg1L \cdot Hipotenuse}\right) + \arctan\left(\frac{y}{x}\right) - 90° \quad (8)$$

$$\text{ELBOW} = \theta_{s2} = \arccos\left(\frac{Seg2L^2 + Seg1L^2 - Hipotenuse^2}{2 \cdot Seg1L \cdot Seg2L}\right) - 180° \quad (9)$$

NOTA: Hipotenuse $= \sqrt{x^2 + y^2}$

Com estas equações adquirimos a possibilidade de, para qualquer coordenada x e y, calcular os ângulos necessários para a garra do nosso braço robô se deslocar para essa coordenada corretamente.

3.2 Determinação da posição de uma peça detetada pelo sensor de distância

Outro desafio técnico a enfrentar correspondeu à determinação correta da posição de uma peça detetada pelo sensor de distância. Este apenas retorna a distância, no entanto, devido ao movimento do braço ser definido pela cinemática inversa, seriam precisas coordenadas X e Y. A solução adotada para este problema foi o método ao qual chamámos *scan*. Este método consiste em "esticar"o braço, enviando-o para um dos extremos de amplitude do braço e, com incrementos de 1º, rodar o braço e ler a distância através de um ciclo *FOR*. A partir da definição de um limite de alcance (a soma dos comprimentos dos segmentos do braço) foi possível implementar uma lógica em que: para cada iteração (para cada ângulo da base) caso o sensor de distância detete algo que se encontre a uma distância inferior ao limite definido, é realizado um cálculo de acordo com a seguinte imagem:

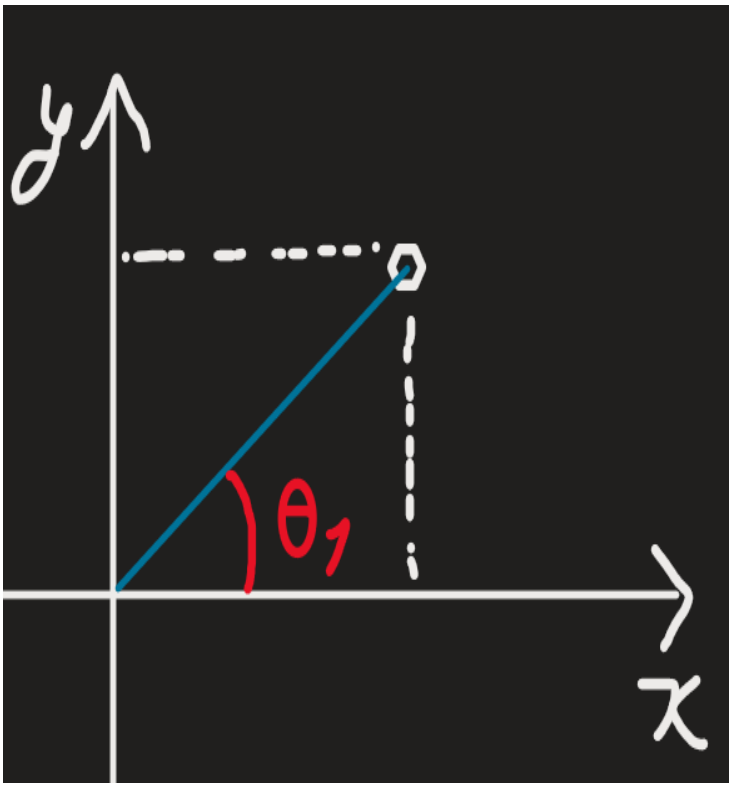

Figura 5: Representação do processo de deteção de uma peça.

$\theta 1$ conhece-se pois o ângulo da iteração $\theta s1$ corresponde apenas à "tradução"de $\theta 1$ para o referencial do *Servo Motor*. Ora, como na solução de cinemática inversa $\theta s1 = \theta 1 - 90$, para obter $\theta 1$ apenas necessitamos de somar 90^{o} ao valor do ângulo da iteração atual do ciclo *FOR*. Após obtido este ângulo, como se sabe a distância a partir da leitura do sensor, pode-se simplesmente definir que:

$$x = cos(\theta 1) * distance \tag{10}$$

$$y = sin(\theta 1) * distance \tag{11}$$

Com estas equações em mente, é possível definir o ponto onde se detetou uma peça a partir da leitura do sensor de distância.

3.3 Tradução de posição de *Grid*

Um último desafio técnico a enfrentar foi a tradução de uma posição de uma grelha (valores de 1 a 9) para coordenadas X e Y. A grelha, visto que é definida por nós, alunos, pode ter as suas características *Hard Coded* (distância entre pontos da grelha e posição de origem) no entanto, uma solução mais robusta seria tornar estas posições configuráveis, permitindo assim evitar recompilação do código caso a grelha mude de posição. Para tal, foi definida uma classe responsável por guardar, alterar e "traduzir"as posições da grelha para coordenadas X e Y. Esta classe, para além de oferecer funções para configurar as características da grelha, oferece também um método para obter as coordenadas de uma dada posição da grelha a partir do seguinte processo:

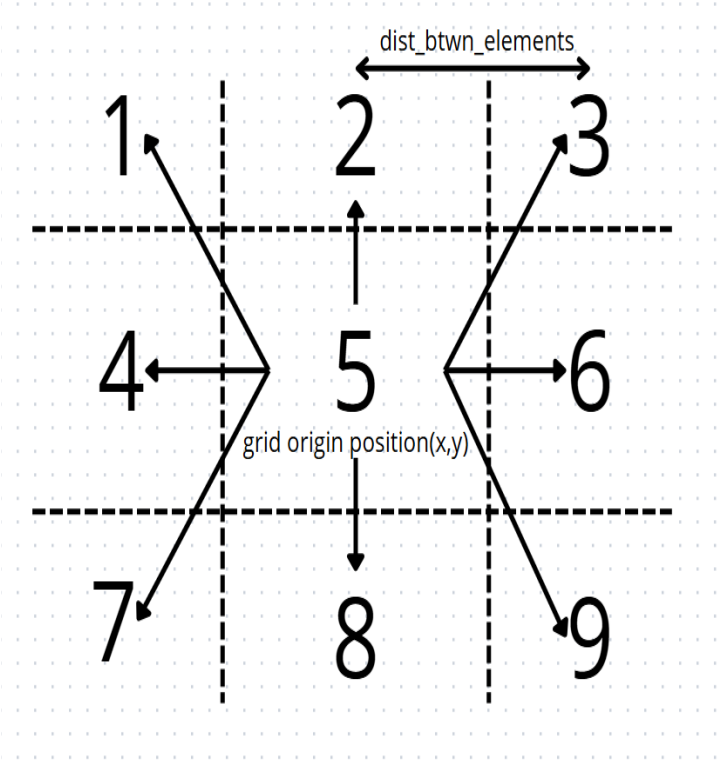

Figura 6: Representação da grelha: origem, distância e suas posições.

A posição 5 é considerada a origem do referencial e todas as restantes posições são calculadas a partir dessa origem, garantido assim maior flexibilidade no que toca à atualização da origem da grelha. As posições são calculadas da seguinte forma:

$$x_1 = x_4 = x_7 = (x_5 - dist) \tag{12}$$

$$x_3 = x_6 = x_9 = (x_5 + dist) \tag{13}$$

$$y_1 = y_2 = y_3 = (y_5 + dist) \tag{14}$$

$$y_7 = y_8 = y_9 = (y_5 - dist) \tag{15}$$

NOTA: CASOS NÃO DEFINIDOS ACIMA SÃO CONSIDERADOS COMO AS COORDENADAS DE 5 i.e $x_2 = x_5$ Com esta solução garante-se alta modularidade no código visto que quaisquer métodos relacionados com a grelha necessitam de ser adicionados apenas na classe que se encontra numa biblioteca separada. Assim obtém-se uma solução robusta para este problema que, como bónus, permite a definição de mais do que uma grelha, desde que o suporte para tal seja adicionado no código principal.

4 Organização do Trabalho

Antes de iniciarmos a implementação do código, definimos uma estratégia detalhada para organizar as atividades e garantir que todos os objetivos fossem alcançados de forma sistemática. O processo adotado foi o seguinte.

4.1 Reunião de Requisitos e Material

O primeiro passo consistiu em analisar atentamente os requisitos do trabalho e reunir todo o material necessário, incluindo componentes de hardware e referências técnicas. Esta fase foi crucial para assegurar que dispúnhamos dos recursos necessários para a execução do projeto.

4.2 Calibração dos Servos e Montagem do Robô

Antes de termos em nossa posse o chassi do braço robótico, procedemos à calibração dos servos através de sinais PWM. Este procedimento foi fundamental para evitar movimentos bruscos que pudessem danificar o chassi, garantindo assim um funcionamento mais seguro do nosso sistema, bem como aumentar a precisão posicional dos servos, já que cada servo possui uma função que determina o PWM correspondente para cada ângulo. Após a calibração, efetuámos a montagem do robô, integrando os diversos componentes consoante planeado na montagem e no esquemático elétrico.

4.3 Definição da Lógica e Criação da Máquina de Estados

Com o robô já montado, passou-se à definição da lógica de operação deste. Para facilitar a implementação, desenvolvemos inicialmente uma máquina de estados que serviu de guia para a sequência de operações do sistema. Esta abordagem permitiu uma transição mais simples e organizada da lógica para a implementação em código.

4.4 Organização do Código

A etapa seguinte consistiu na organização do código. Optámos por dividir o projeto num código principal e em várias bibliotecas (ficheiros `.h`), com o objetivo de manter o código o mais organizado possível. As bibliotecas criadas foram:

- **ColorPositions**: Gestão das posições relativas à identificação e classificação das cores.
- **gridTranslation**: Tradução das posições detetadas para coordenadas utilizáveis pelo sistema.
- **Kinematics**: Cálculos da cinemática necessários para o controlo correto dos servos.
- **Sensors**: Interface para a comunicação e análise de dados provenientes dos sensores.

4.5 Implementação e Testes

Após definir a estrutura do código, implementámos as funcionalidades desejadas de forma incremental. Em cada etapa, procedemos à realização de testes que permitiram validar o funcionamento da lógica, facilitando a identificação e correção de eventuais problemas.

5 Lógica implementada

5.1 Máquina de estados

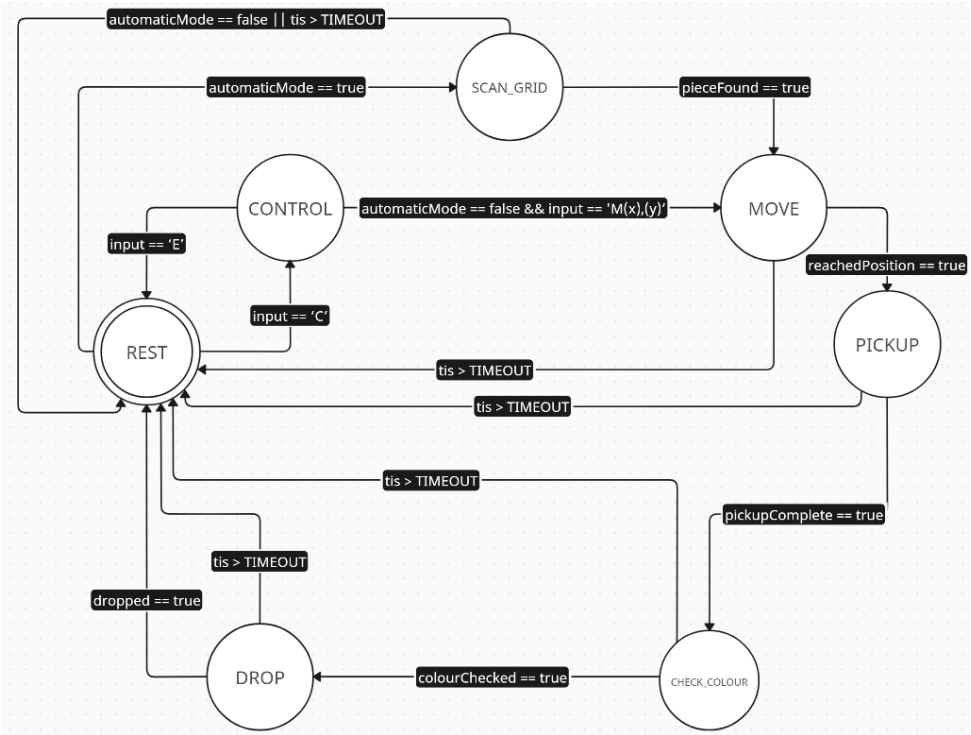

Figura 7: Máquina de Estados

O código implementa a máquina de estados responsável pelo controlo do robô, permitindo a operação tanto em modo automático como manual. No modo automático, o sistema segue uma sequência de estados: **SCAN_GRID**, onde o robô procura por peças, **MOVE** para se deslocar até à posição da peça, **PICKUP** para agarrá-la, **CHECK_COLOR** para identificar a cor e **DROP** para depositá-la no local correto. O estado **CONTROL** permite a operação manual através de comandos enviados via comunicação serial.

Tudo isto acontece de forma estruturada, garantindo que o robô só avança para o próximo passo quando a respetiva variável de estado fica verdadeira, seja porque encontrou uma peça, completou uma ação ou recebeu uma nova instrução. Desta forma, adiciona-se também modularidade à maquina de estados e fica mais fácil a determinação de um erro, já que cada estado tem um *TIMEOUT* caso ocorra algum erro que o retorna a *REST* caso esse tempo passe.

5.1.1 Estado REST

O estado **REST** representa o ponto inicial e de repouso da máquina de estados. Sempre que o sistema não estiver a executar tarefas ativas, ele permanece neste estado. Aqui, o braço robótico coloca-se numa posição de repouso e todas as variáveis de estado são reinicializadas para garantir que não há transições erradas com base em ciclos anteriores.

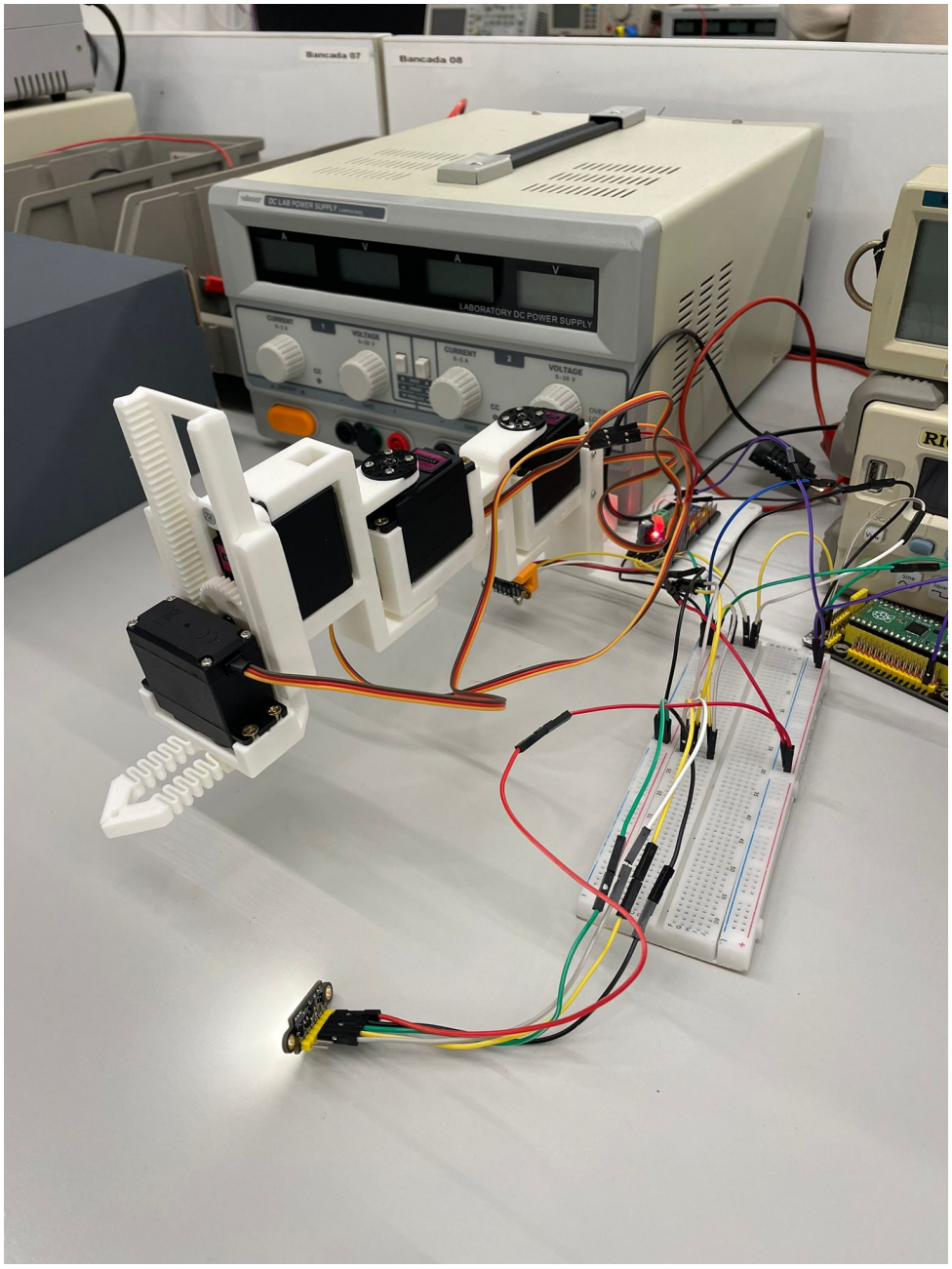

Figura 8: Braço em repouso

Transições:

- Se o modo automático estiver ativado (`automaticMode == true`), a máquina transita para **SCAN_GRID**.
- Se o utilizador inserir o comando `'C'` via série, transita para o estado **CONTROL**, onde pode receber comandos manuais.

5.1.2 Estado SCAN_GRID

Neste estado, o braço inicia um processo de scan da área de trabalho para detetar a presença de uma peça. O servo da base gira gradualmente, e o sensor de distância ToF (Time of Flight) mede continuamente a presença de objetos. Se for encontrada uma peça dentro do alcance máximo permitido, a posição da peça é calculada e guardada para ser usada na próxima fase.

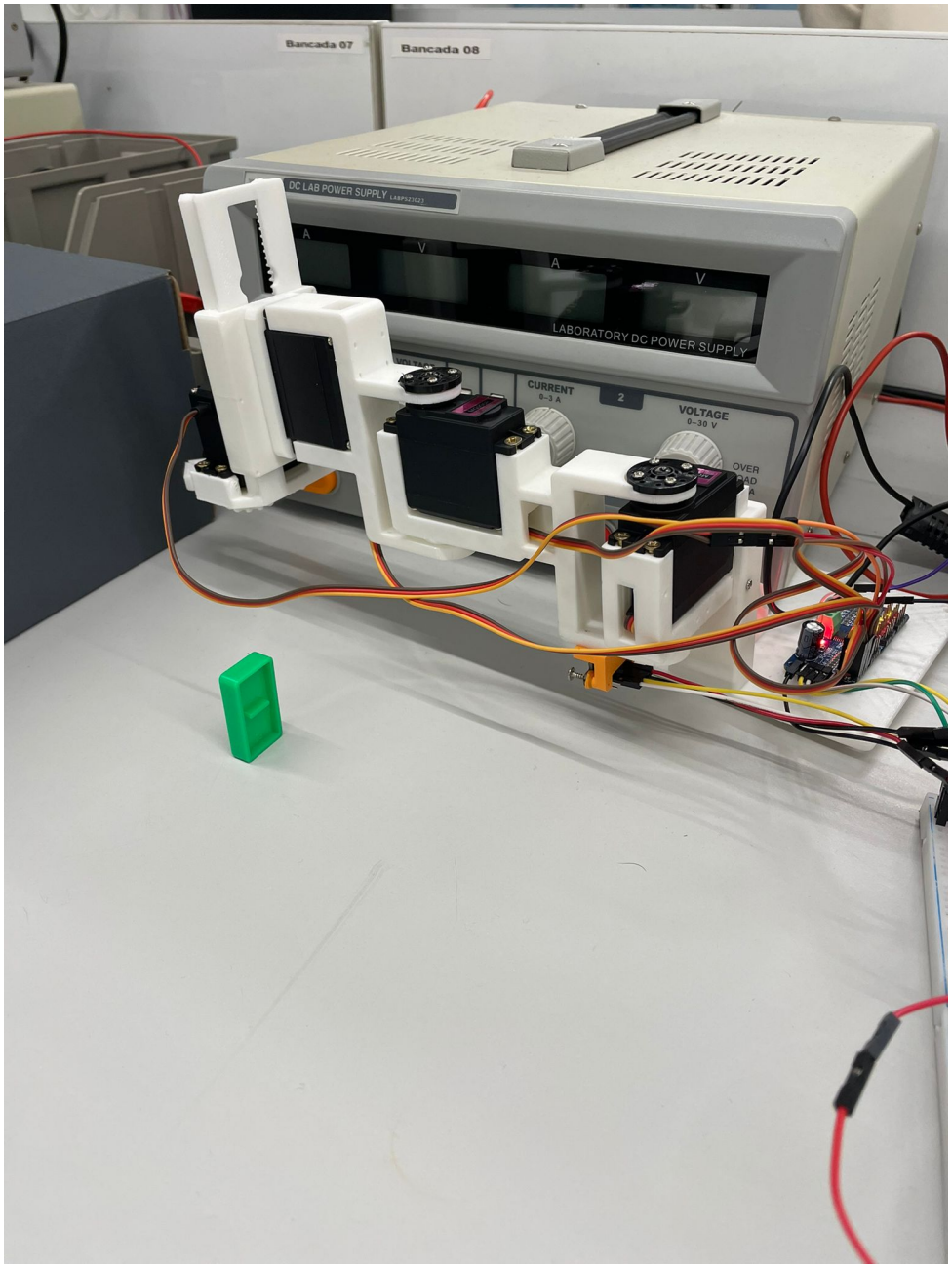

Figura 9: Braço realizando scan

Transições:

- Se for encontrada uma peça (`pieceFound == true`), o sistema avança para **MOVE**.
- Caso o tempo limite (`TIMEOUT`) seja atingido sem encontrar peças ou o modo automático seja desativado, o sistema regressa ao estado **REST**.

5.1.3 Estado MOVE

No estado **MOVE**, o sistema calcula a trajetória para a posição onde a peça foi detetada e movimenta-se até lá. A posição alvo foi guardada anteriormente no estado **SCAN_GRID** e agora os servos do braço ajustam-se para alcançar as coordenadas corretas.

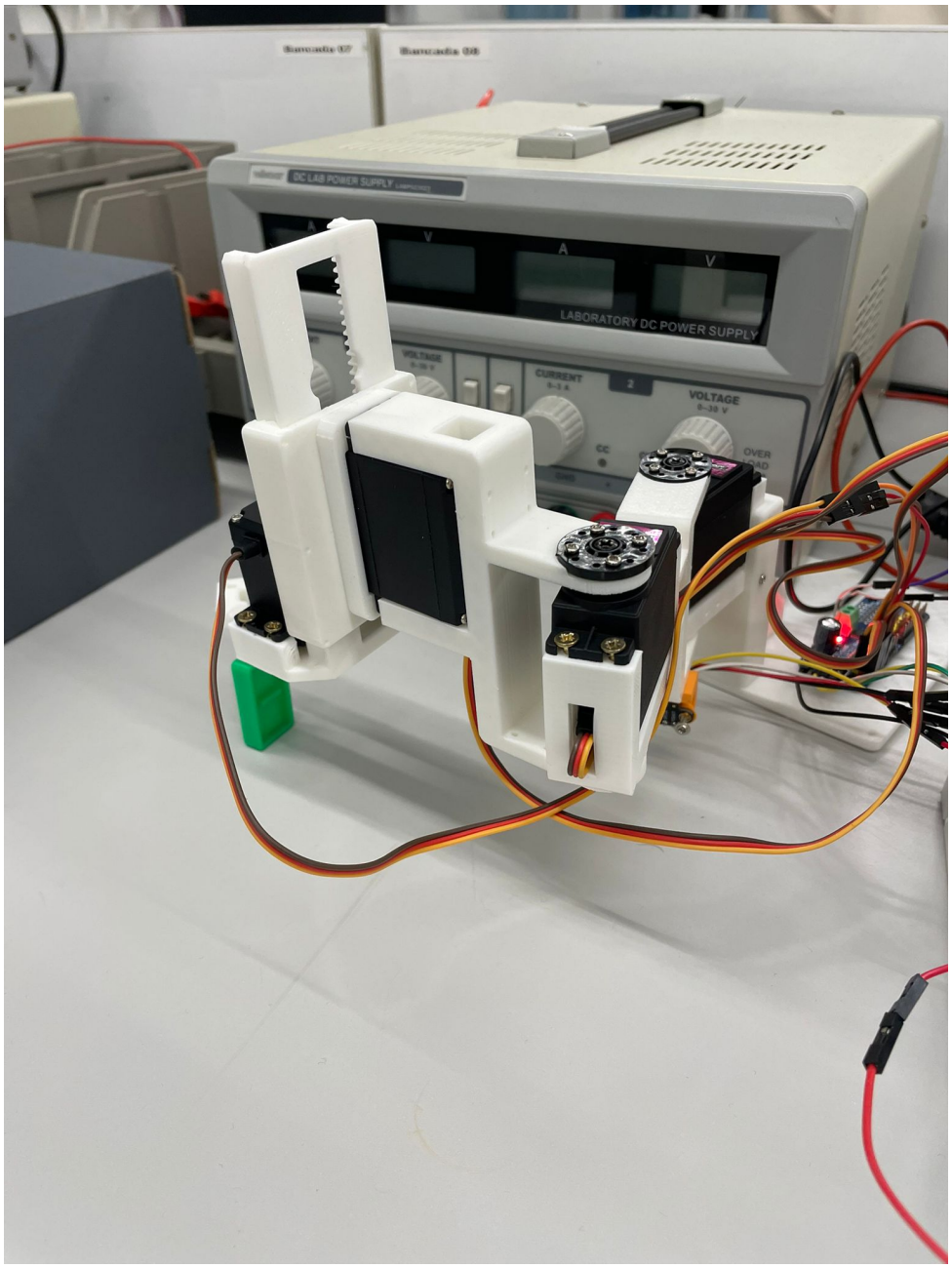

Figura 10: Braço em movimento

Transições:

- Quando a posição é atingida (`reachedPosition == true`), o estado atual do sistema avança para **PICKUP**.
- Se o tempo limite for atingido antes de alcançar a posição desejada, o sistema assume um erro e volta para **REST**.

5.1.4 Estado PICKUP

Agora que o braço está na posição correta, ativa-se o mecanismo para agarrar a peça. A garra fecha-se para segurar a peça.

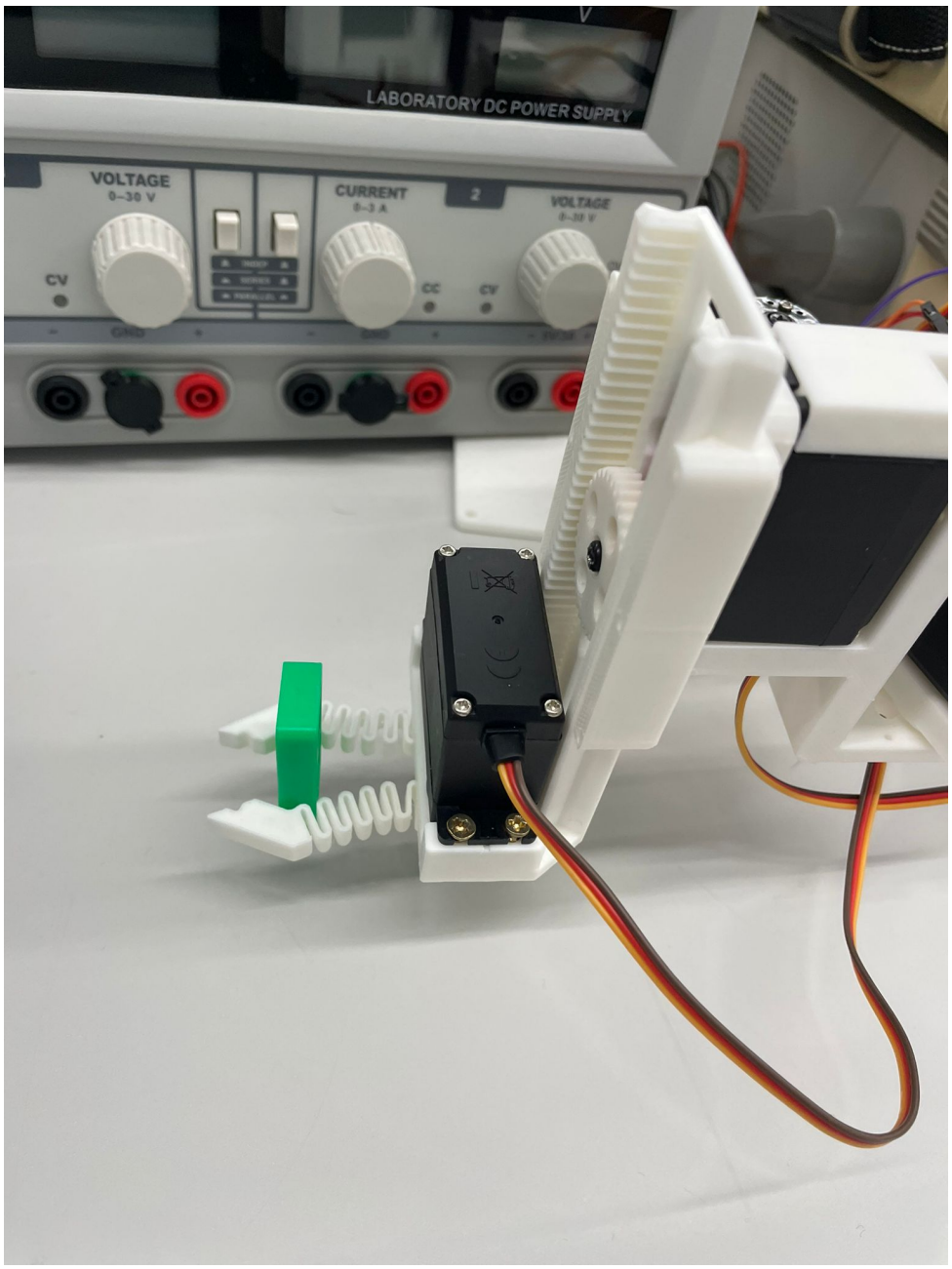

Figura 11: Braço a segurar peça

Transições:

- Se a operação for bem-sucedida (`pickupComplete == true`), o sistema avança para **CHECK_COLOR**.
- Se o tempo limite for atingido sem sucesso, volta para o estado **REST**.

5.1.5 Estado CHECK_COLOR

Neste estado, o braço desloca-se para uma posição predefinida, onde o sensor de cor realiza a análise da peça. O sensor **TCS34725** determina a cor do objeto, e essa informação será usada para decidir onde largar a peça.

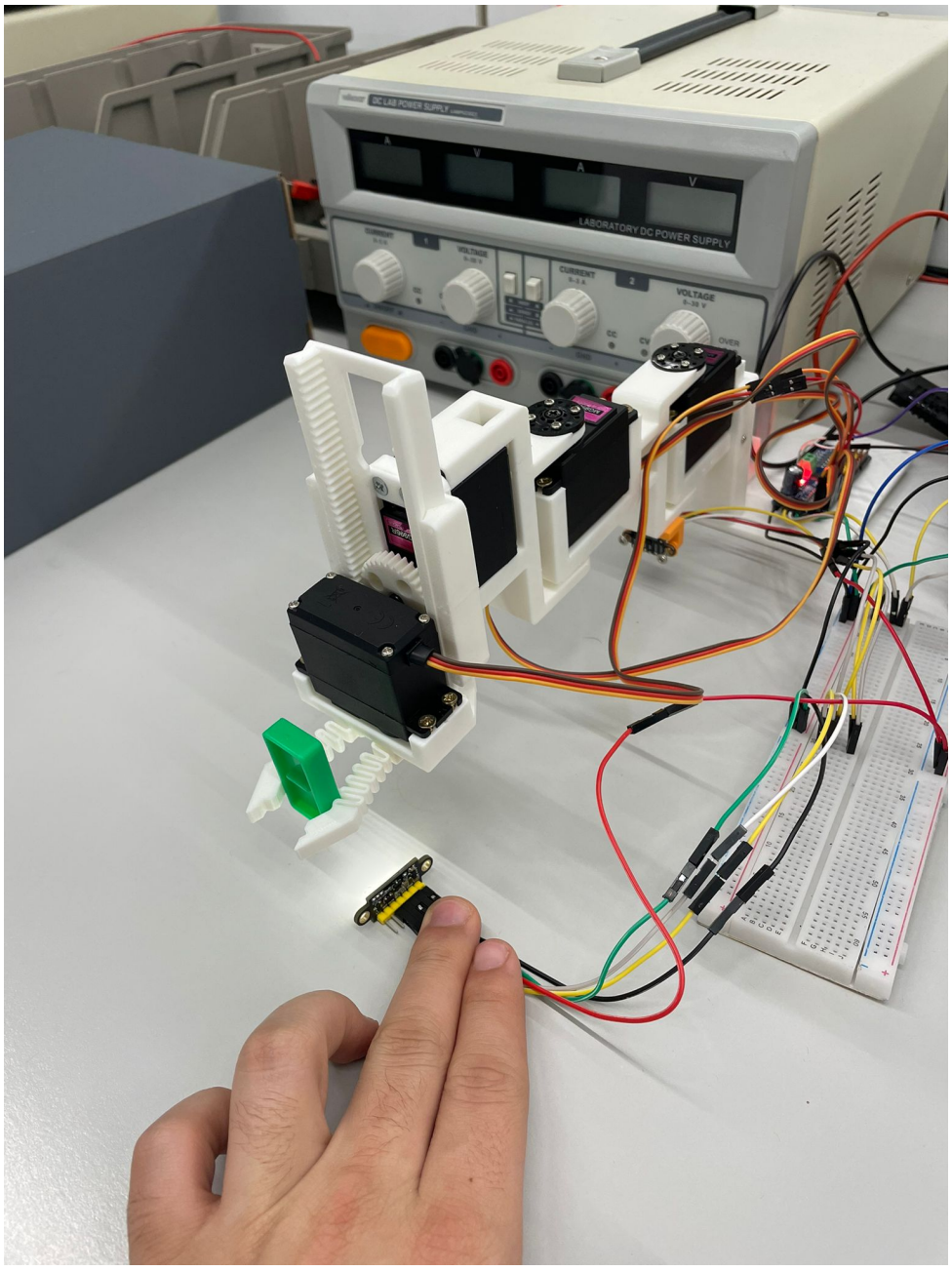

Figura 12: Braço a verificar a cor

Transições:

- Quando a cor é identificada (`colorChecked == true`), a máquina de estados avança para **DROP**.
- Se o tempo limite for atingido sem sucesso, volta para o estado **REST**.

5.1.6 Estado DROP

Com base na cor identificada, o sistema movimenta-se para a posição de depósito correspondente e larga a peça. A garra abre-se para largar a peça no local correto.

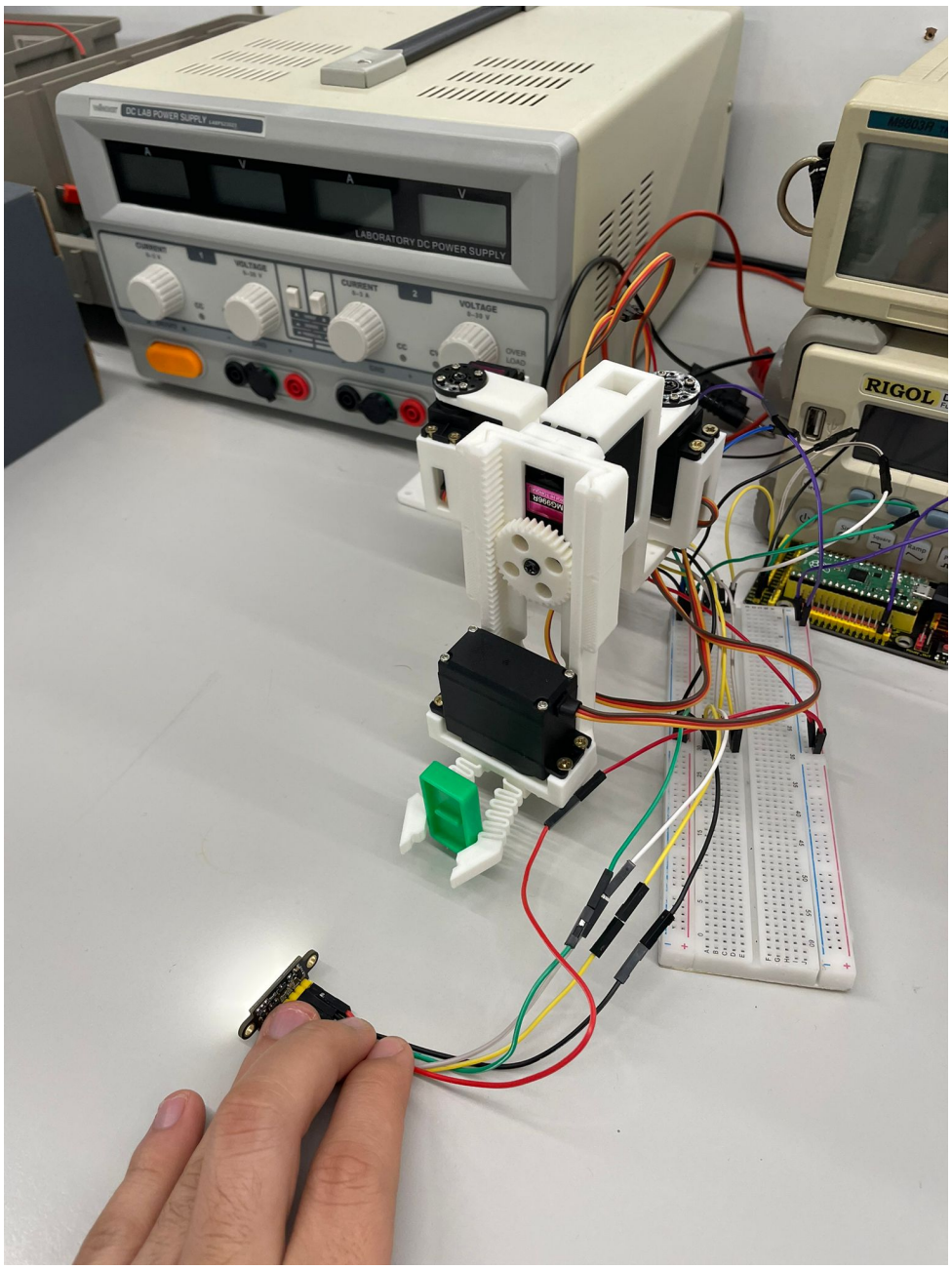

Figura 13: Braço a largar a peça

Transições:

- Quando o objeto é largado com sucesso (`dropped == true`), o sistema retorna ao estado **REST**.
- Se o tempo limite for atingido sem sucesso, também volta para o estado **REST**.

5.1.7 Estado CONTROL

Este estado é especial, pois permite que o utilizador assuma o controlo manual do braço através de comandos enviados via Serial Monitor. O utilizador pode movimentar o braço para coordenadas específicas, abrir ou fechar a garra e até realizar um scan manual da área.

Transições:

- Se o utilizador enviar o comando `'E'`, o sistema volta ao estado **REST**.

5.2 Funcionalidades do estado Control

De forma ao estado *CONTROL* permitir total controlo sobre o braço, era necessário realizar leitura da Porta Série, e definir comandos específicos a escrever dependendo do que se desejava fazer. Como tal foi definida a seguinte lista de *Binds* de controlo:

- R : enviar o braço para a sua posição "esticada"de repouso
- O : abrir a garra
- C : fechar a garra
- + : subir o braço
- - : descer o braço
- P : rotina de apanhar uma peça
- D : rotina para pousar uma peça
- S : obter cor lida pelo sensor de cor
- T : obter distância lida pelo sensor de distância
- Z : realizar método de scan
- M : entra num modo de leitura para inserção de coordenadas manualmente
- G : entra num modo de leitura para inserção de coordenadas de origem do *GRID*
- B : entra num modo de leitura para inserção de posição do *GRID* para a qual se deseja o braço ir
- X : entra num modo de leitura para inserção de nova distância entre posições do *GRID*
- U : entra num modo de leitura para atualização de posições de um depósito (Verde, Amarelo, Azul ou Vermelho são escolhidos depois de iniciar o modo de leitura)
- J : entra num modo de leitura para atualização das coordenadas do sensor de cor

A partir desta implementação, foi possível obter um modo manual que abranja totalmente as funcionalidades do robô, permitindo assim um modo manual com controlo total, bem como uma forte ferramenta de identificação de problemas com outros métodos implementados.

```
case CONTROL:
    if (input == 'R') {
        kinematics.moveToPos(0, SEGMENT_1_LENGTH + SEGMENT_2_LENGTH);
    }

    else if (input == 'O') {
        kinematics.OpenClaw();
    }

    else if (input == 'C') {
        kinematics.CloseClaw();
    }

    else if (input == '+') {···

    else if (input == '-') {···

    else if (input == 'P') {···

    else if (input == 'D') {···

    else if (input == 'S') {···
```

Figura 14: Excerto da implementação em código das *Binds* do estado *CONTROL*

5.3 Workspace do braço

Como mencionado na secção 3.1 (Cinemática Inversa), foi definida a solução de cinemática inversa para corretamente definir os ângulos para um ponto desejado. Durante a fase de testes, no entanto, verificou-se que o espaço de trabalho real do robô não correspondia ao de um robô da tipologia *SCARA* como inicialmente proposto.

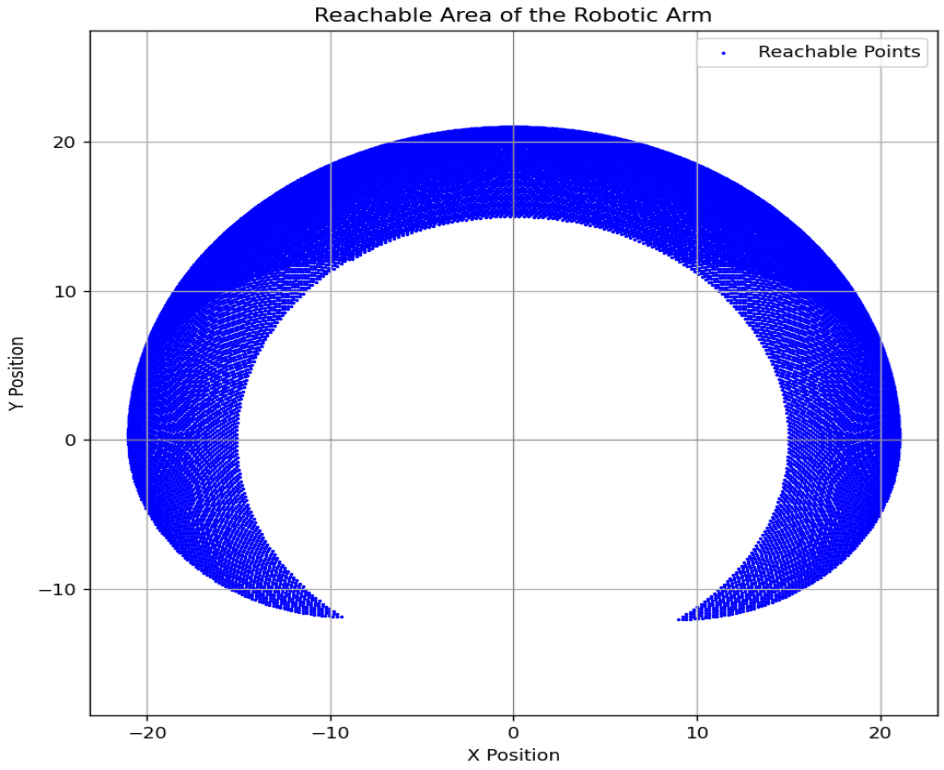

Figura 15: Workspace inicial (vista de cima

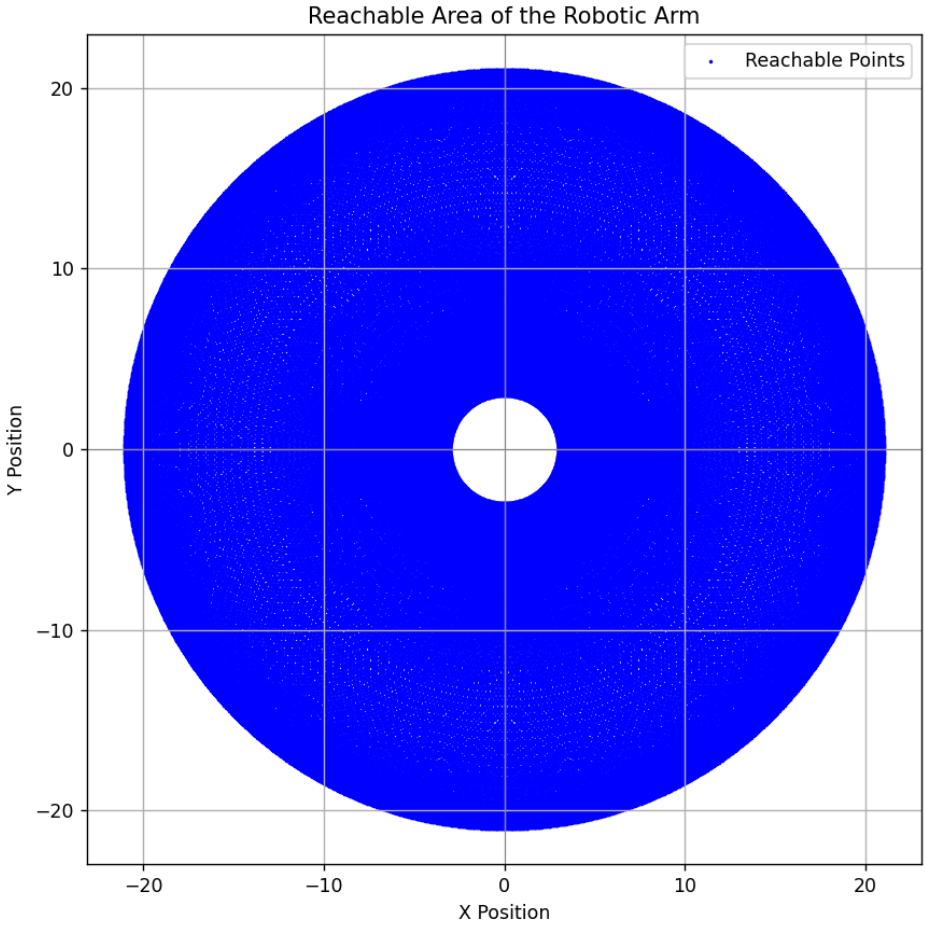

Figura 16: Workspace SCARA (o desejado para o caso seria um semi-circular)

Devido à significante diferença entre o *Workspace* desejado e o que se verificava com os nossos testes, foi realizada uma pequena pesquisa pelo motivo. Esta pesquisa chegou à conclusão de que o motivo para tal se reduzia à amplitude dos *Servo Motors*. Os *Servos* utilizados possuem uma amplitude de 180^{o}, o que torna fisicamente impossível alcançar o workspace circular (visto de cima) desejado. De forma a combater esta limitação de workspace, entrámos em contacto com os professores que sugeriram uma mudança na montagem. Esta mudança consistia em alinhar os 90^{o} do *ELBOW Servo* com o que anteriormente era o eixo dos 0^{o}.

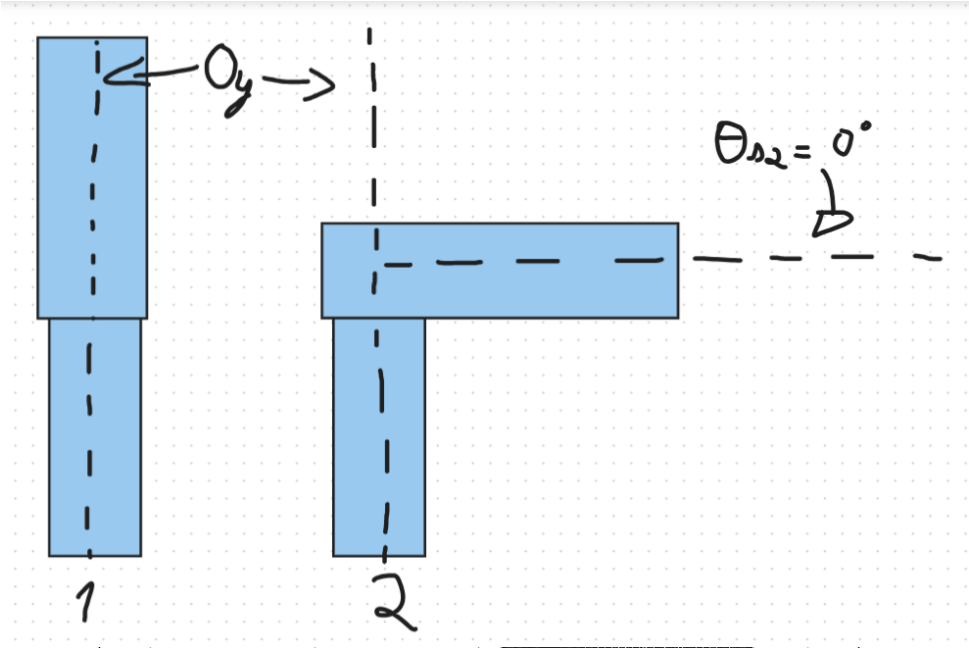

Figura 17: Comparação de montagens (1- montagem inicial, 2- montagem proposta)

A realização desta montagem permitia o seguinte *Workspace*:

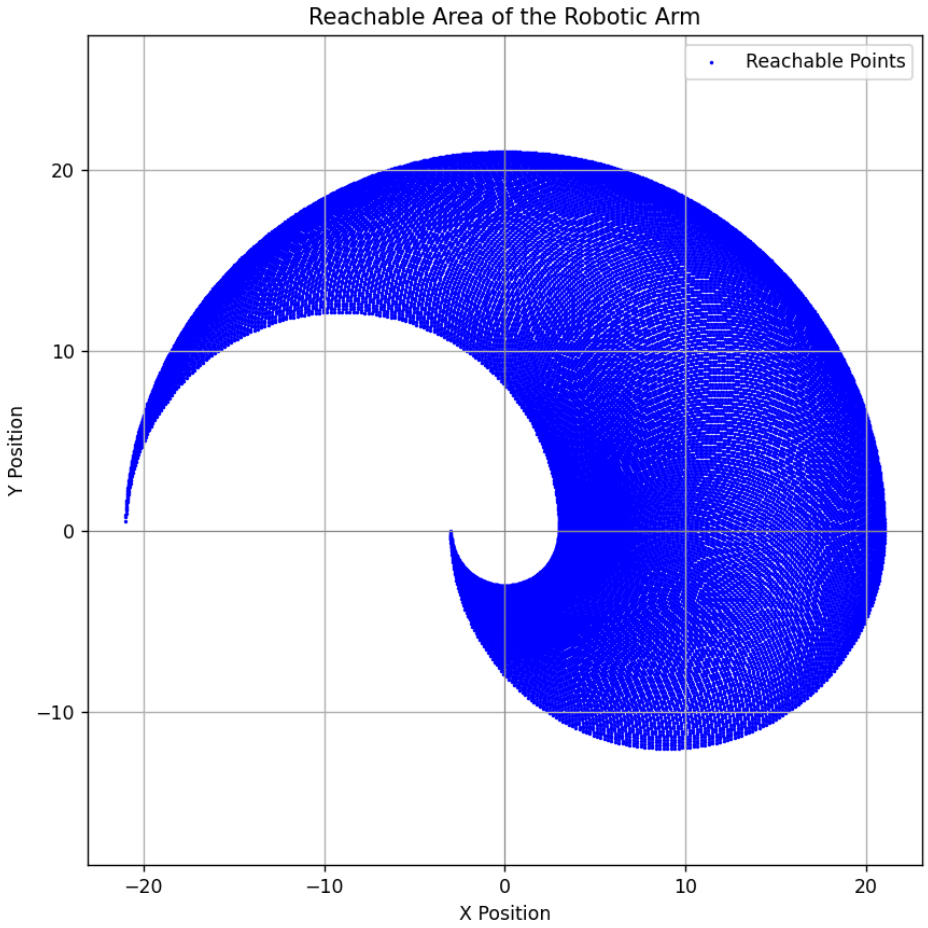

Figura 18: Workspace após mudança de montagem

Visto que este *Workspace* se assemelhava mais ao desejado, optámos por esta mu-

dança na montagem mas que, consequentemente obrigava a uma alteração na solução de cinemática inversa pois com esta mudança $\theta s2 \neq \phi$. Como quando $\theta 2 = 90 <=> \theta s2 = 0$ e $\theta 2 = 0 <=> \theta s2 = -90$, a alteração feita à solução de cinemática inversa foi a seguinte:

$$\theta s2 = \theta 2 - 180 => \theta s2 = \theta 2 - 90 \tag{16}$$

Assim alargámos o *Workspace* do robô, aproximando este do de um robô da tipologia *SCARA* como inicialmente planeado.

6 Resultados Finais e Análise Crítica

No geral, os objetivos propostos foram praticamente todos atingidos, tendo o sistema demonstrado um desempenho satisfatório na grande maioria dos aspetos. A seguir, descrevemos os pontos positivos e as limitações identificadas durante os testes, bem como sugestões de melhorias futuras.

6.1 Objetivos e Desempenho Geral

- **Objetivos Atingidos:** Todos os objetivos estabelecidos foram alcançados com sucesso. O braço robótico demonstrou movimentos suaves e precisos, resultado de uma correta calibração dos servos, e a lógica implementada, baseada na máquina de estados, permitiu a realização das tarefas de manipulação e classificação das peças.
- **Desempenho dos Movimentos:** O movimento do braço foi suave e conforme o esperado, uma vez que os servos se encontravam devidamente calibrados, proporcionaram uma operação precisa.

6.2 Funcionamento dos Sensores

- **Sensor de Cor:** O sensor de cor funcionou na perfeição, identificando corretamente todas as cores das peças.
- **Sensor ToF:** O sensor ToF apresentou um pequeno problema relacionado com a sua montagem: encontrava-se ligeiramente inclinado para baixo, o que, por vezes, detetando a bancada, fazia com que o robô considerasse a existência de uma peça nessa posição, prejudicando assim o funcionamento do modo automático.

6.3 Limitações e Áreas de Melhoria

- **Posicionamento do Sensor ToF:** A inclinação do sensor ToF é a principal limitação identificada. Este pequeno desvio resulta numa interpretação errada da posição de uma peça visto que o sensor por muitas vezes deteta a bancada, o que compromete a operação em determinadas situações.
- **Sugestão de Melhoria:** A partir de um ajuste da inclinação do sensor, de forma à mesma ficar corretamente alinhada com o plano da bancada seria possível remover totalmente este problema de má deteção visto que vários testes com o braço no limite da bancada (inclusive durante a apresentação final) demonstraram que o braço determinava corretamente a posição do objeto detetado.

7 Conclusão

A realização deste projeto levou-nos até ao presente relatório, o qual contribuiu significativamente para o nosso desenvolvimento técnico e académico. Ao longo da implementação, aprofundámos o nosso conhecimento na utilização de diferentes sensores e servos, além de reforçarmos a importância de manter um raciocínio estruturado para gerir os diversos estados do sistema.

Durante a atividade prática, seguimos as etapas propostas para o desenvolvimento do braço robótico e realizámos múltiplos testes para garantir a sua funcionalidade. Os resultados obtidos validam o projeto, demonstrando que a lógica implementada e os ajustes efetuados permitiram alcançar um desempenho global muito positivo. No entanto, identificámos pequenos aspetos que podem ser melhorados de modo a reduzir eventuais margens de erro.

Embora a limitação de montagem do sensor de distância tenha prejudicado no momento da avaliação prática final, seguimos confiantes de que o robô implementa todas as funcionalidades desejadas corretamente, incluindo a questão de configuração de posições dos depósitos, sensor de cor e grelha. Com todas estas funcionalidades corretamente implementadas surge um grande sentimento de orgulho e de que as nossas capacidades foram devidamente postas à prova de forma triunfante.

Além dos desafios técnicos, este projeto destacou a importância da documentação e da elaboração de relatórios técnicos, uma competência essencial para qualquer futuro engenheiro. Aprender a reportar de forma clara e precisa os processos e resultados obtidos fortalece a capacidade de análise crítica e contribui para a evolução contínua na área da engenharia.

Em suma, o projeto cumpriu os objetivos estabelecidos e permitiu-nos consolidar conhecimentos essenciais na área da robótica e automação. As melhorias sugeridas centram-se essencialmente no aperfeiçoamento do posicionamento do sensor ToF. Com este ajuste, o sistema poderá atingir um nível de desempenho superior.

Referências

[1] Francisco Vieira, *Servo Motors*, slides das aulas teóricas do Moodle.

[2] Texas Advanced Optoelectronic Solutions Inc., *TCS34725*, Ficha técnica do produto.

[3] STMicroelectronics, *VL53L0X*, Ficha técnica do produto.

O *Source Code* do projeto está disponível em: GitHub - ACE Robot Arm