Faculdade de Engenharia da Universidade do Porto

Automatic Identification Systems for Smart Manufacturing applications

Francisco Cardoso Lima
Gabriel Lima Meireles
José Castro Baptista
Miguel Augusto Marombal Araújo
Pedro Guerner Dias do Carmo Pascoal
Rafael Cardoso Magalhães
Ricardo Jorge Teixeira Tripeça
Tiago Azevedo Rodrigues Silvério Machado

Relatório realizado no âmbito da Unidade Curricular PROJETO INTEGRADOR
da
Licenciatura em Engenharia Eletrotécnica e de Computadores

Orientador: Paulo José Lopes Machado Portugal

julho 2022

Resumo

O trabalho sobre o qual incide este relatório foi realizado no âmbito da unidade curricular Projeto Integrador. O mesmo tinha como objetivo aplicar conceitos aprendidos no decorrer dos 3 anos de formação da L.EEC, para o desenvolvimento de um projeto de EEC envolvendo competências técnicas e não-técnicas de média complexidade.

No final do projeto, os requisitos propostos foram cumpridos culminando num demonstrador funcional capaz de identificar objetos com recurso às tecnologias propostas e manipulá-los segundo as suas caraterísticas com recurso ao braço robótico disponibilizado.

Agradecimentos

A concretização de um projeto com esta natureza não se deve apenas aos seus autores, mas antes, a todos aqueles que de forma direta ou indireta se envolveram. Partilharam-se dúvidas, incertezas e muitas aprendizagens. Agradecemos, em primeiro lugar, ao Prof. Paulo Portugal que ajudou a construir o caminho desde a idealização até à concretização deste projeto e ao técnico Daniel Silva pela prontidão com que se dispunha a ajudar-nos quer em questões teóricas como técnicas.

Índice

Lista de figuras

Lista de tabelas

Abreviaturas

Lista de abreviaturas (ordenadas por ordem alfabética)

FEUP	Faculdade de Engenharia da Universidade do Porto
HF	*High Frequency*
I/O	*Input/Output*
IDE	*Integrated Development Environment*
LED	*Light-emitting diode*
RF	Radio-frequência
RFID	*Radio Frequency Identification*
RGB	*Red, Green, Blue*
RX	Receptor
SPI	*Serial Peripheral Interface*
TX	Transmissor
UHF	*Ultra High Frequency*
UID	*Unique Identifier*

1. Introdução

Este relatório é realizado no âmbito da UC Projeto Integrador e tem como objetivo apresentar, explicar e demonstrar todo o trabalho efetuado durante o desenvolvimento do projeto.

Inicialmente, apresentamos o enunciado do projeto proposto a ser desenvolvido e os requisitos definidos no planeamento do trabalho, sendo estes bastante importantes permitindo manter o resultado final bem definido e dividido em múltiplas partes. A definição de requisitos também nos permite averiguar sobre o sucesso do projeto, comparando-os com as funcionalidades e capacidades do sistema final.

De seguida é apresentado de forma geral o sistema desenvolvido, permitindo perceber que um sistema desta complexidade é simplesmente um conjunto de múltiplos sistemas mais pequenos e interligados, que comunicando entre si originam um sistema robusto e eficiente.

Quando a apresentação geral do sistema final está cumprida, são expostos e explicados ao pormenor os vários subsistemas salientando a respetiva importância de cada um perante os objetivos do projeto. A eficiência e sucesso na implementação de cada subsistema também são demonstrados quando se apresenta os resultados dos mesmos.

Por último, são demonstrados os resultados finais do sistema implementado comparando-os com os objetivos e requisitos definidos inicialmente. Tendo em consideração os resultados, também se chega a conclusões relativas ao trabalho desenvolvido, como abordagens que evitariam certos problemas ou limitações no projeto.

2. Requisitos

Este projeto foi-nos apresentado como uma oportunidade de desenvolvermos um sistema de identificação automática, devido à alta importância que este tipo de sistema tem vindo a assumir no mundo atual, como, por exemplo, no conceito de "*Smart manufacturing – Industry 4.0*". Deste modo, foi-nos proposto o desenvolvimento de um demonstrador, recorrendo a tecnologias de identificação industriais, nomeadamente Visão Industrial e RFID (*Radio Frequency Identification*).

Para além da identificação daquelas tecnologias, o enunciado deste projeto estabelece que o demonstrador a desenvolver deverá ser composto por duas componentes:

- **Módulo Visão industrial**: utilizando câmaras *OpenMV* para suporte de operações de deteção de objetos, formas, cores, contornos, posição, etc. Estas câmaras, programadas em

python, dispõem de um conjunto vasto de bibliotecas para os mais variados tipos de identificação de objetos.

- **Módulo RFID**: utilizando leitores de códigos de RFID HF RC522 ou UHF SparkFun, com tags de variados tipos (HF/UHF) para suporte de operações de leitura, escrita, incluindo validação através de mecanismos de segurança.

O demonstrador deve ser composto por um tapete, de aproximadamente 1 metro, onde circularão peças com diferentes características, cada uma com uma *tag* RFID (conjunto de antena e *chip* com memória onde pode ser guardada informação). Pretende-se que quando uma peça passar num determinado ponto do tapete, esta seja analisada e identificada pela câmara associada ao Módulo Visão Industrial, recorrendo-se à tecnologia RFID para guardar a informação de identificação. Os módulos devem estar associados a plataformas computacionais de baixo custo. Por último, o enunciado sugere a utilização de um braço robótico para retirar as peças do tapete.

Tendo em conta estes requisitos estabelecidos no enunciado do projeto, no início do seu desenvolvimento, foram, por nós, definidos requisitos adicionais que o sistema final teria de cumprir, levando a que todo o trabalho efetuado fosse executado de forma a respeitá-los. Deste modo, consegue-se resumir todo o planeamento de um projeto complexo, em múltiplas funções que o sistema final tem de ser capaz de cumprir. Sendo assim, tendo os requisitos definidos, o desenvolvimento do projeto pode ser concretizado com várias tarefas em paralelo, tendo sempre noção do ponto em que se irão interligar. Os requisitos que definimos foram:

- **Módulo Visão Industrial** capaz de identificar, através de análise da imagem, o objeto que está a ser visualizado, tendo em conta a sua cor e a sua forma;
- **Escrita com sucesso na *tag* RFID associada a cada peça quando este é identificado pela câmara**, sendo a informação guardada em cada peça constituída por um número único e a forma e cor do respetivo objeto;
- **Braço robótico** que, recorrendo ao módulo RFID, seja capaz de ler a *tag* associada à peça em questão e consiga armazená-la da forma correta, tendo em conta a informação de identificação lida;
- **Controlo de tapete** responsável pelo transporte da peça permitindo a sua paragem na zona onde a peça é analisada pela câmara do Módulo Visão Industrial e também na zona onde será retirada do tapete;
- **Comunicação entre Módulo Visão Industrial e Módulo RFID**, de modo a que o resultado da análise da câmara seja transmitida de forma segura, ou seja, sem danificação da informação;
- **Apresentação da informação guardada na *tag* de cada peça**, no momento em que o módulo Visão Industrial chega a um resultado de identificação;

3. Visão geral do sistema

Estando definido o contexto do projeto e os respetivos requisitos, estamos em condições de apresentar o sistema desenvolvido.

Perante a complexidade de todo o sistema implementado, optamos por fazer uma apresentação geral do funcionamento e interligação entre as várias partes do projeto, e nos capítulos 4.1, 4.2 e 4.3 explicaremos pormenorizadamente cada subsistema.

O sistema é composto por 3 subsistemas: Sistema de Visão Industrial, Sistema de Classificação das Peças e Sistema de Armazenamento das Peças.

3.1. Apresentação dos subsistemas

O Sistema de Visão Industrial é constituído pelo Módulo Visão Industrial, ou seja, um microcontrolador, chamado *OpenMV Cam*, de baixo custo e de baixo consumo energético, com uma câmara associada. Devido à sua simplicidade e de apresentar uma programação *user-friendly*, permite a fácil implementação de projetos relacionados com visão por computador. Para além disso, recorrendo aos *I/O pins*, é possível comunicar facilmente com plataformas externas. Este sistema será responsável pela análise da imagem da peça, tendo como *output* a identificação da mesma. O Sistema de Visão Industrial também tem como responsabilidade, o controlo do tapete do sistema, utilizando os *I/O pins* para controlar o funcionamento do motor do tapete.

O Sistema de Classificação das Peças é composto pelo Módulo RFID, ou seja, tem como plataforma computacional um Arduino UNO com um leitor RFID ligado. Para além disso também comunica com um *display* de apresentação que terá como função apresentar a informação da peça analisada. Este sistema será responsável por guardar informação de identificação na *tag* associada à peça em questão e também apresentar essa informação no display.

O Sistema de Armazenamento de Peças é uma terceira parte do sistema desenvolvido, composto por um módulo RFID para ser possível a leitura das *tags* das peças e um braço robótico que armazenará as mesmas tendo em conta a informação lida. Tanto o braço robótico como o leitor RFID estão ligados e executam as suas funções através de um Arduino UNO.

3.2. Interação entre subsistemas

A interação entre os sistemas é o que permite transformar diversas partes distintas, cada uma com as suas responsabilidades, num sistema final complexo, capaz de cumprir os requisitos definidos para o projeto de forma simples e estruturada.

O Sistema de Classificação das Peças e o Sistema de Visão Industrial têm uma forte interação, existindo uma troca de dados constante entre estes. Esta ligação é fundamental para o sucesso do sistema final, pois é deste modo que o resultado de identificação do Sistema de Visão Industrial é transmitido para o Sistema de Classificação das Peças, permitindo assim a

este último armazenar na *tag* RFID de cada peça a respetiva identificação. O protocolo de comunicação escolhido para o estabelecimento desta ligação foi SPI (*Serial Peripheral Interface*). Devido à importância desta ligação consideramos fundamental aprofundar o funcionamento deste protocolo.

SPI é uma interface de comunicação síncrona e *full-duplex* (devido aos canais MISO e MOSI referidos mais à frente), que utiliza uma arquitetura *master-slave*, permitindo que um dispositivo definido como *master* consiga comunicar com múltiplos *slaves* de forma sincronizada. Este protocolo é composto por 4 fios: SCK, MOSI, MISO, SS.

- **SCK (*Serial Clock*)** é responsável pela transmissão do sinal de *clock* que permite a sincronização da informação transmitida, sendo este um sinal definido pelo *master*.
- **O MOSI (*Master Out Slave In*)** é o fio por onde a informação é transmitida no sentido *Master-Slave*, tal como indica o significado do nome MOSI.
- **O MISO (*Master In Slave Out*)** é o fio por onde a informação é transmitida no sentido *Slave-Master*, tal como indica o significado do nome MISO.
- **SS (*Slave Select*)** permite a definição do slave com que o *master* comunica. O SS é feito em lógica negativa, ou seja, quando o sinal transmitido pelo master para um certo *slave* for um 0 lógico, então significa que esse é o *slave* selecionado. Este é o único fio que numa rede de comunicação por SPI envolvendo vários dispositivos, não pode ser partilhado entre os vários *slaves*, pois é o que define qual será o dispositivo com que o *master* irá trocar informação.

Na nossa implementação, definimos como *master* o Arduino constituinte do Sistema de Classificação das Peças, sendo que o leitor RFID (desse sistema), o *display* e o microcontrolador *OpenMV Cam* do Sistema de Visão Industrial foram definidos como *slaves*.

Deste modo, o Arduino é o responsável por intercalar a permissão de transferência de informação proveniente dos três dispositivos definidos como *slaves*, através do *Slave Select*.

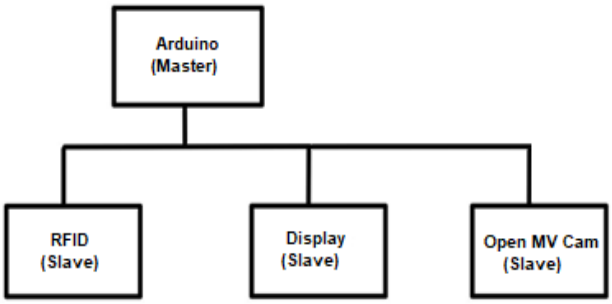

Figura 1- Ligação SPI entre dispositivos

Relativamente ao Sistema de Armazenamento de Peças, este não interage diretamente com nenhum dos outros subsistemas. Este simplesmente recorre à informação guardada nas *tags* das peças, que foi resultado de todo um trabalho conjunto entre os outros dois subsistemas, para retirar as peças do tapete e as armazenar tendo em conta a sua identificação.

3.3. Funcionamento

Estando apresentado o sistema desenvolvido, de uma forma geral, para se compreender exatamente o funcionamento da solução concebida, expomos o percurso de uma peça, desde o momento em que é colocada no tapete até ao seu armazenamento.

O nosso sistema final foi produzido tendo em mente 8 tipos de peças, sendo que os dois fatores que influenciam são a cor (vermelho ou verde) e a forma (retângulo, círculo, triângulo equilátero ou estrela regular).

Inicialmente, o tapete está em andamento e o sistema aguarda a colocação de uma peça no mesmo. Após a peça ser colocada, esta desloca-se e quando a câmara deteta a sua chegada à zona de identificação, o microcontrolador *OpenMV Cam* interrompe a alimentação do motor do tapete, originando a paragem da peça. Após a paragem do tapete, a câmara executa o processo de identificação da peça, o que poderá demorar alguns segundos (*benchmarks* apresentadas no capítulo 4.2.5).

Quando o algoritmo de identificação finalmente gera resultados, envia a informação de identificação (por exemplo retângulo vermelho) para o Sistema de Classificação das Peças. Este sistema recebe a informação e recorre ao preenchimento da *tag* da peça avaliada, guardando a informação de identificação.

Após o armazenamento da informação na *tag*, o motor do tapete é novamente alimentado e assim a peça é deslocada em direção ao fim do tapete com o intuito de ser armazenada. Durante este deslocamento desde a zona de identificação até à zona de armazenamento, a *tag* da peça é novamente lida, em andamento, pelo leitor RFID associado ao Sistema de Armazenamento de Peças.

Antes de chegar ao fim do tapete, o motor é desligado novamente na zona de armazenamento de peças, permitindo ao braço robótico pegar na peça de forma segura. A leitura da *tag* feita durante o deslocamento da peça é efetuada para que o *robot* tenha conhecimento de que peça (dentro dos 8 tipos possíveis) se trata e perante essa informação recolhida, armazena a peça no local adequado. Após o armazenamento, o braço robótico volta à posição de partida pré-definida.

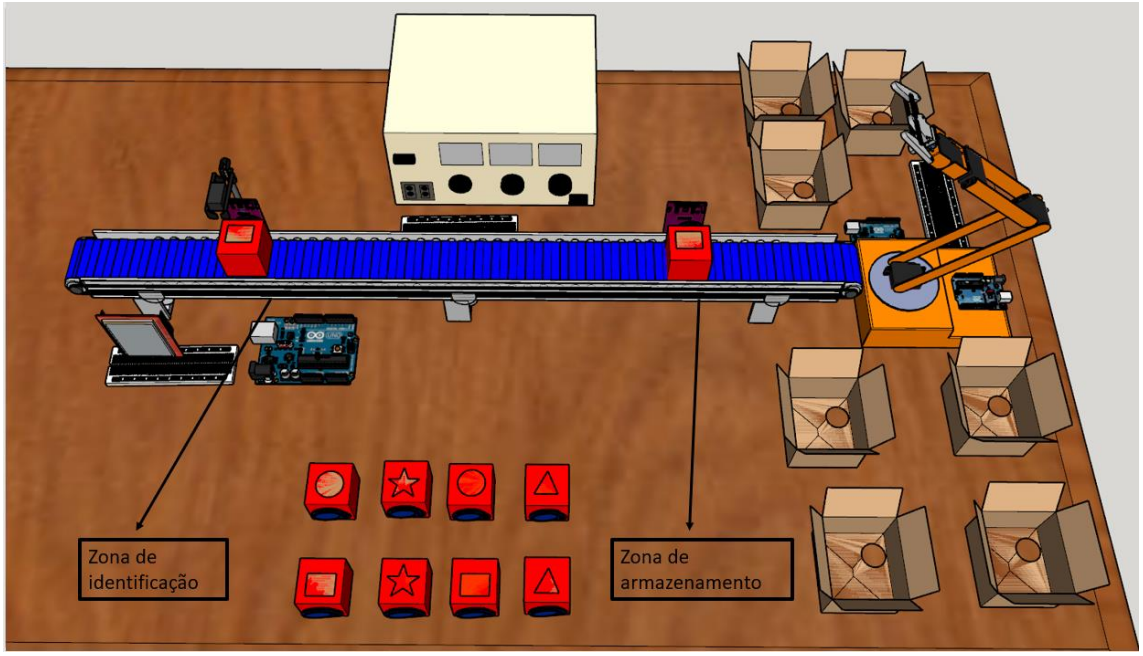

Figura 2- Zona de identificação e armazenamento de peças

4. Subsistemas

4.1 Sistema de Classificação das Peças

4.1.1. Contextualização e princípio de funcionamento

Uma das peças principais para a concretização deste trabalho foi a utilização de um sistema de identificação que permite a leitura e a escrita de dados utilizando comunicações sem-fios, sem contacto e sem a necessidade de campo visual direto, o RFID. Esta relação só é conseguida através da interligação entre duas componentes:

- as etiquetas ou "*tags*" em inglês, que têm no seu interior uma antena de radiofrequência e um *chip* digital minúsculo com memória, que lhes permite responder a pedidos de um módulo emissor-recetor RFID. Existem dois tipos de etiquetas RFID, as passivas e as ativas. As passivas podem responder a pedidos sem necessidade de alimentação externa, enquanto que as ativas precisam de uma fonte de alimentação como por exemplo uma pilha ou uma bateria. As *tags* que utilizámos no projeto são passivas, portanto não contêm uma bateria e têm um *microchip* que guarda e processa informação, e uma antena para receber e transmitir o sinal.

- o módulo emissor-recetor RFID, o chamado leitor de RFID, gera um campo eletromagnético de alta frequência que provoca o movimento dos eletrões através da antena da *tag* e consequentemente alimenta o *chip*. O leitor RFID controla-se através do protocolo SPI e é compatível com Arduino e com quase todos microcontroladores.

Interligando estas duas componentes, o *chip* alimentado dentro da *tag*, responde ao campo eletromagnético gerado pelo leitor RFID enviando a sua informação guardada de volta para o leitor sobre a forma de outro sinal de rádio. O leitor transmite sempre um sinal como uma onda contínua sinusoidal, independentemente de a *tag* ter bits para enviar ou não.

A *tag* RFID segue o padrão MIFARE Classic®EV1 1k: 1 kB. A memória de 1K da *tag* está organizada em 16 setores (de 0 a 15), cada um é dividido em 4 blocos (bloco 0 a 3), e cada bloco pode armazenar 16 bytes de dados (de 0 a 15).

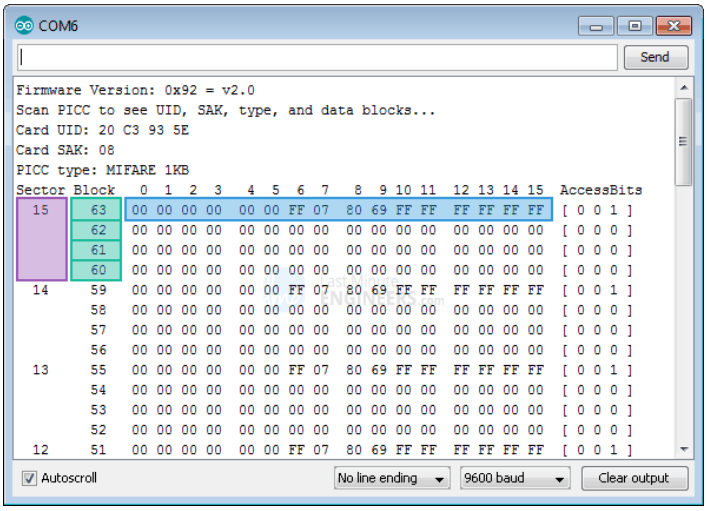

Figura 3 – Estrutura de memória da tag através da plataforma computacional Arduino

Assim, conhecendo o modo como as *tags* estão estruturadas, conseguimos ter perceção de como podemos efetuar a escrita, leitura e manipulação da informação guardada de forma segura, não correndo o risco de danificar as *tags*.

4.1.2. Requisitos

Relativamente aos requisitos definidos para este subsistema, pretende-se que este seja capaz de armazenar dados dentro das *tags*, no momento em que estas são detetadas pelo leitor RFID, de forma suficientemente rápida para que toda a informação consiga ser escrita independentemente da velocidade a que as peças se deslocam no tapete.

Esta informação tem de ser guardada de modo a que posteriormente os seus dados possam ser lidos e interpretados. Para além disso, os Sistema de Classificação das Peças também é responsável pela apresentação da informação guardada num *display*. Para que este subsistema tenha acesso à informação que será armazenada também é necessário o estabelecimento de uma ligação segura e fiável com o Sistema de Visão Industrial.

4.1.3. Arquitetura

De modo a cumprir os requisitos mencionados foi estruturado um código dividido em 8 partes:

- ***Setup*** – Este segmento é responsável por inicializar a comunicação SPI necessária entre os subsistemas e pela inicialização das funções da biblioteca MRFC522 que permite fazer o *scan* das *tags* RFID. A inicialização do *display*, definição da cor de fundo, e do tamanho da letra a usar, também são elementos desta parte.
- **Comunicação com *OpenMV Cam*** – Nesta parte estabelece-se uma ligação com o Sistema de Visão Industrial, permitindo troca de informações entre os dois subsistemas, isto é, possibilitando que o *OpenMV Cam* envie informações sobre as características da peça que identificou para posteriormente serem gravadas na *tag* respetiva.
- **Validação da informação transmitida pelo *OpenMV Cam*** – Nesta parte verifica-se se os dados recebidos são realmente importantes para o pretendido. Se a mensagem transmitida não corresponder ao conjunto de possibilidades esperadas, então será descartada e o algoritmo esperará por um novo envio, sempre verificando se é significante ou não. Desta forma é garantido que será sempre escrita informação útil na *tag* associada a cada peça.
- **Deteção da *tag*** – Parte responsável por permitir que o leitor RFID consiga detetar a presença de uma *tag*, sempre que houver uma presente. Este segmento é então de extrema importância pois o algoritmo só avança a partir do momento em que uma *tag* é detetada.
- **Verificação de *tag* escrita** – Devido à possibilidade de uma *tag* ser escrita mais do que uma vez, foi necessário a elaboração deste segmento, pois era uma situação não favorável ao nosso projeto. Nesta parte, verifica-se, então, se uma *tag* já foi escrita, impossibilitando uma segunda escrita.
- **Escrita de dados na *tag*** – É nesta parte que escrevemos nos blocos das *tags* toda a informação que queremos, nomeadamente o número da peça, a sua forma e cor.
- **Leitura de dados presentes na *tag*** – Esta parte é responsável por ler toda a informação contida na *tag*, para, posteriormente, ser projetada no *display*.
- **Apresentação da informação no *display*** – Este último segmento tem como objetivo apresentar a informação contida nas *tags* num pequeno *display*, ou seja, o número da peça, a forma e a cor, e também uma figura que representa a respetiva peça analisada.

4.1.4. Implementação

O código começa com a inclusão das bibliotecas **MFRC522** e **SPI**, definindo os pinos do Arduino aos quais o RC522 (nome do leitor RFID) está conectado. Integrando o *display*, incluímos também a biblioteca **Adafruit** e enunciámos os pinos das suas entradas, nomeadamente o *slave select* para a ligação SPI, um pino para *reset*, ambos diferentes daqueles utilizados pelo leitor RFID, e outro para a alimentação (+3.3V) tendo que montar um divisor de tensão uma vez que utilizámos a saída de +5V do Arduíno.

Inicialmente, definimos também os códigos hexadecimais das cores a utilizar no *display*. Em seguida, definimos os blocos nos quais armazenamos os dados, nunca podendo selecionar o terceiro bloco de qualquer setor (*Sector Trailer*) uma vez que este contém informações para conceder acesso de leitura e gravação aos blocos restantes num setor. Isso significa que apenas os 3 blocos inferiores (bloco 0, 1 e 2) de cada setor estão realmente disponíveis para armazenamento de dados, com exceção do bloco 0 do setor 0 que contém os dados do fabricante e o identificador exclusivo (UID) e, portanto, também não pode ser utilizado para fins de escrita. Deste modo, após alguns testes verificámos a necessidade de utilizar 3 blocos para a escrita de toda a informação a apresentar no *display*: o número da peça no bloco 1 (setor 0), forma no bloco 2 (setor 0) e cor no bloco 5 (bloco 1 do setor 1).

Em seguida, definimos três arrays de 16 bytes que iriam conter a informação a escrever nos blocos (número da peça, forma e cor) e outros 3 arrays para posteriormente guardar o conteúdo escrito na *tag*, no momento em que fosse executada a sua leitura.
A seguir, na função ***setup()*** inicializamos a comunicação SPI necessária para haver comunicação entre os subsistemas e as funções da biblioteca **MRFC522**. Nesta parte também se inicializa o *display*, definindo a cor de fundo preta através da função ***tft.fillScreen()***, e o tamanho da letra utilizando a função ***tft.setTextSize()***, ambas disponiveis na biblioteca **Adafruit**.

Após a definição e inicialização do que é necessário para o desenvolvimento deste algoritmo, resta apenas o estabelecimento da comunicação com o Sistema de Visão Industrial para receber os dados sobre a peça identificada.

Para esse efeito estabelece-se uma ligação com o *OpenMV Cam* através de SPI e com recurso à função ***SPI.transfer()*** (da biblioteca **SPI**), recebe-se um *array*, proveniente do *OpenMV*, contendo as informações relativas às características da peça identificada, nomeadamente a sua forma e cor. Este *array* é, posteriormente, dividido em dois: um com informação relativa à forma e outra relativa à cor da peça. Por fim, ambos serão copiados para os blocos das *tags*.

Antes de passarmos para a escrita da informação nos blocos da *tag*, garantimos que a mensagem transmitida pelo Sistema de Visão Industrial é relevante. Para tal, efetua-se a comparação do *array* que contém a informação sobre a forma da peça com as *strings*: “RETANGULO”, “TRIANGULO”, “ESTRELA” e “CIRCULO”, correspondendo às figuras das peças que esperamos identificar. Após ser efetuada a comparação com essas 4 *strings*, caso nenhuma dessas comparações seja concluída com sucesso significa que a mensagem transmitida pelo Sistema de Visão Industrial não é relevante e, portanto, será descartada, ficando à espera que uma nova mensagem seja recebida.

Após verificar que a informação obtida é útil verifica-se se existe um cartão nas periferias que possa ser escrito e, em caso positivo, esse cartão é selecionado para fins de

escrita e leitura. Para esse efeito são utilizadas funções da biblioteca **MFRC522**, nomeadamente a função ***mfrc522.PICC_IsNewCardPresent()*** num *if statement*, em que o programa fica retido até detetar a presença de uma *tag*.

Numa fase do desenvolvimento do código apercebemo-nos que não poderíamos permitir que o leitor RFID escrevesse duas vezes na mesma *tag* pois poderia levar à danificação ou corrompimento da informação. Este problema pode surgir devido ao tempo que a peça fica à frente do leitor enquanto espera pela sua identificação por parte do Sistema de Visão Industrial. De forma a identificar se a peça presente é a mesma utiliza-se um sistema de numeração de peças, sendo atribuído um número a cada uma. Para evitar a escrita consecutiva na mesma *tag*, a solução que implementada baseia-se um sistema de *flags*.

Este sistema associa uma variável binaria (0 ou 1) a cada peça, sendo que essa associação é feita a partir do número único característico de cada peça que está guardado na respetiva *tag*. Sempre que se escreve na *tag*, associando-se o respetivo número, a *flag* toma o valor de 1.

Deste modo, tem se sempre o histórico das *tags* escritas, portanto antes de efetuarmos uma escrita lemos a informação guardada e, caso essa informação corresponda a alguma *flag* sinalizada com o valor 1 significa que a *tag* em questão já foi escrita anteriormente interrompendo assim o processo de escrita.

Posteriormente, também desenvolvemos um código à parte com o objetivo de limpar o conteúdo das *tags* permitindo escrever novamente nestas. Este código é simples, consistindo apenas em preencher o *array* com a identificação da peça com zeros. Finalmente, passa a ser apenas necessário a escrita da informação nos blocos das *tags* recorrendo a funções da biblioteca **MFRC522**, nomeadamente ***mfrc522.PCD_Authenticate()*** que autoriza a escrita no bloco desejado e ***mfrc522.MIFARE_Write()*** que escreve no bloco, tendo como argumentos o número do bloco a ser escrito, o *array* com a informação que se pretende escrever e o tamanho do respetivo *array*.

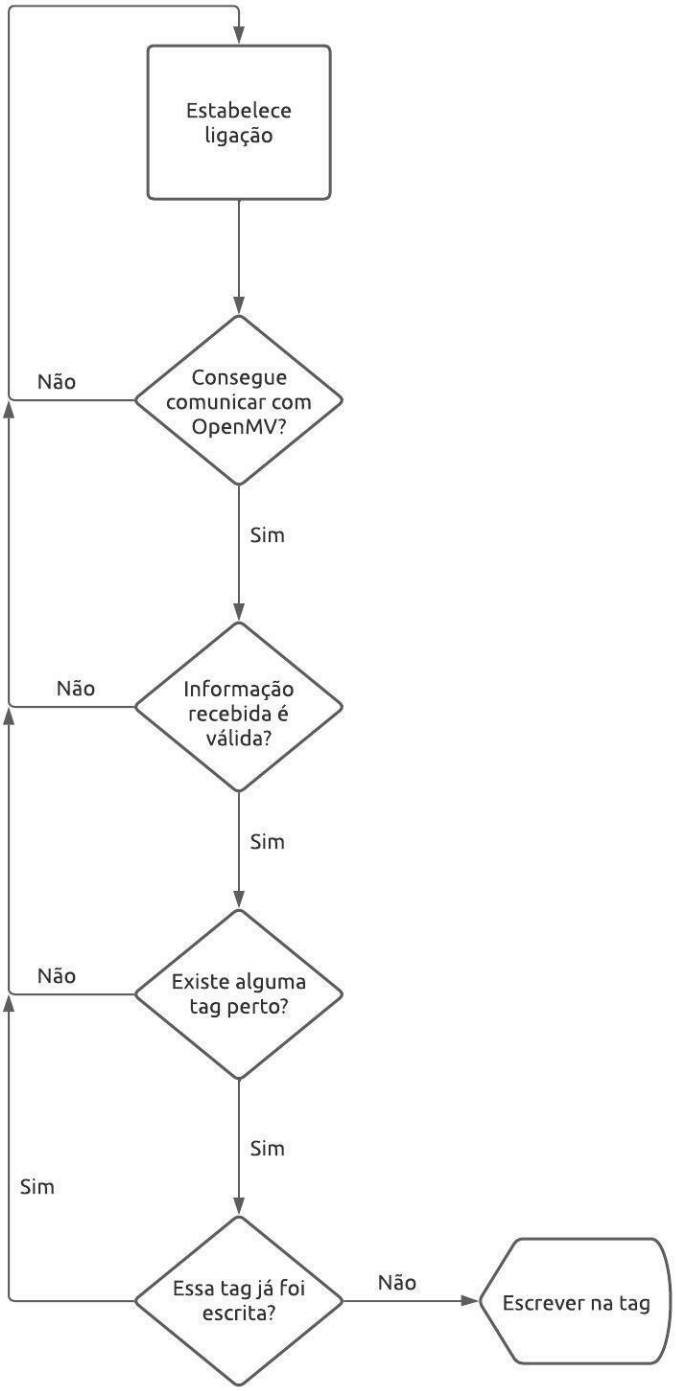

Figura 4 - Sistema de Classificação de Peças – Armazenamento de informação

De seguida, no sentido de permitir a projeção no Display da informação contida na *tag* é realizada a sua leitura recorrendo a funções da biblioteca **MFRC522**, nomeadamente a função ***mfrc522.PCD_Authenticate()*** anteriormente referida e a função ***mfrc522.MIFARE_Read()*** que permite a leitura do conteúdo presente no bloco, tendo como argumentos o número do bloco a ser lido, o *array* onde fica gravada a informação lida e o tamanho desse mesmo *array*.

Finalmente, exibimos a informação da peça (armazenada na *tag*) no d*isplay*. Para isso, recorremos a algumas funções disponibilizadas pela biblioteca do display (**Adafruit***)*, como a função ***tft.setCursor()*** que permite definir a posição do display onde a escrita é realizada, tendo como parâmetros as coordenadas cartesianas dos *pixeis*, e as funções ***tft.println()*** e ***tft.write()*** que permitem escrever no ecrã.

Por fim, adicionámos ainda uma figura para representar a forma e cor da peça, utilizando também funções da biblioteca, nomeadamente a função ***tft.fillCircle()*** para desenhar círculos, ***tft.fillRect()*** para desenhar retângulos e ***tft.fillTriangle()*** para desenhar triângulos, tendo esta também sido útil para o desenho das estrelas, já que não havia nenhuma função especifica para esse efeito. Estas funções têm como argumentos as coordenadas dos vértices, bem como a cor representada por um número hexadecimal, exceto a função para desenhar círculos que tem como argumentos o raio e as coordenadas do centro.

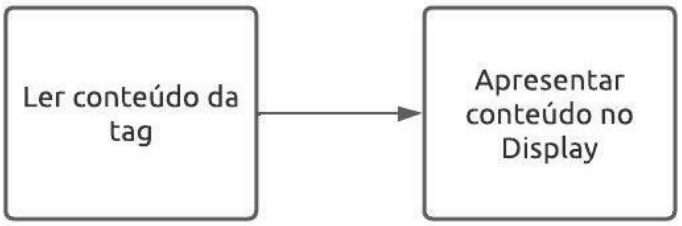

Figura 5 - Sistema de Classificação de Peças- Apresentação no *Display*

4.1.5. Resultados

No final, o Sistema de Classificação das Peças cumpriu os requisitos definidos, sendo capaz de receber e interpretar informação do Sistema de Visão Industrial, armazenar a informação nas *tags* e apresentá-la no *display*.

O Sistema de Armazenamento de Peças teve de lidar com a limitação do alcance das *tags* disponibilizadas ser muito reduzido o que condiciona a distância a que o leitor de RFID pode estar da *tag* para uma leitura e escrita bem-sucedida. Isso, como esperado, gerou alguns constrangimentos, visto que exigia a colocação rigorosa da peça no tapete para ser possível a deteção da *tag*.

4.2. Sistema de Visão Industrial

4.2.1. Contextualização e Princípio de Funcionamento

Para o Sistema de Visão Industrial foi utilizado o microcontrolador ***OpenMV Cam H7 R2*** para suporte das funcionalidades que permitiram a identificação das peças e das suas caraterísticas.

A plataforma computacional *OpenMV Cam* possui caraterísticas que a tornam de especial interesse para este projeto. O seu reduzido tamanho e baixo consumo de potência permitiu que fosse facilmente incorporada no ambiente pretendido, o facto de ser programável numa linguagem de alto nível (*Python*) e de dispor de um conjunto basto de bibliotecas para os diversos tipos de identificação de formas e cores permitiu um maior nível de abstração facilitando assim o desenvolvimento do algoritmo de identificação das peças e suas caraterísticas num **IDE** intuitivo que disponibiliza uma série de funcionalidades úteis. Para além do mencionado o *OpenMV Cam* dispõe também de um conjunto de interfaces I/O fundamentais para a comunicação entre dispositivos.

Figura 6- OpenMV Cam H7 R2

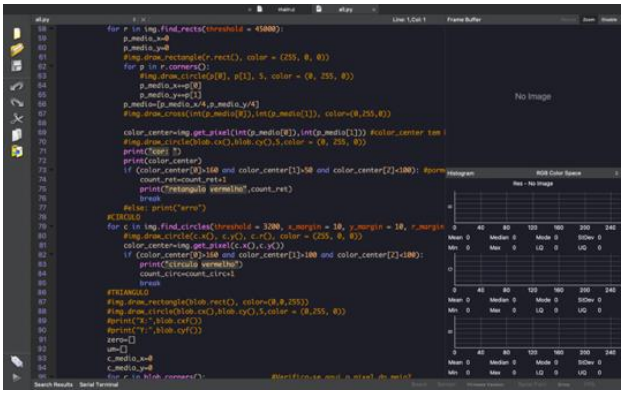

Figura 7- *OpenMV Cam IDE*

4.2.2. Requisitos

No enquadramento do trabalho, pretendia-se que o Sistema de Visão Industrial suportado no *OpenMV Cam*, previamente descrito, fosse capaz de identificar e distinguir entre as diferentes formas possíveis de circular no tapete (retângulos, triângulos equiláteros, círculos e estrelas regulares) bem como as suas respetivas cores (vermelho e verde).

Exigia-se também que este sistema fosse capaz de comunicar a peça identificada com o leitor de RFID. Para além dos requisitos já mencionados, também é da responsabilidade do Sistema de Visão Industrial o suporte necessário para o controlo do tapete, parando-o no momento de identificação da peça e no momento de armazenamento.

4.2.3. Arquitetura

Visando cumprir os requisitos mencionados foi estruturado um código divido em 5 partes:

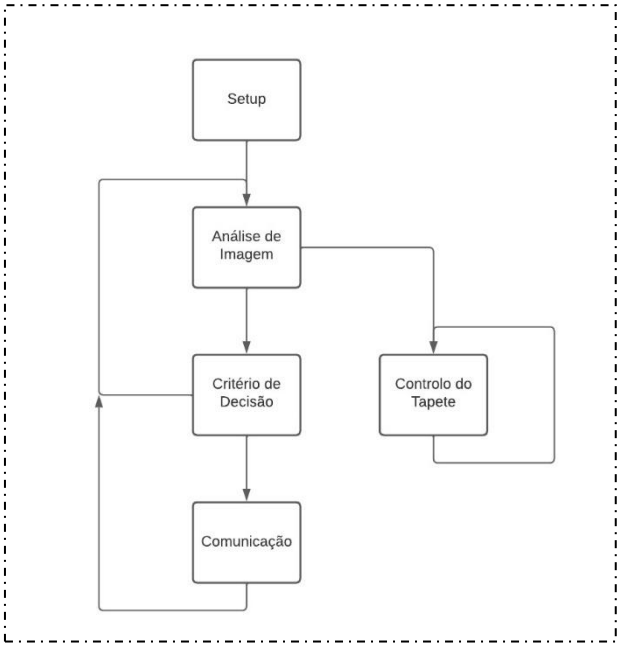

Figura 8- Arquitetura do Sistema de Visão Industrial

- **Setup** – Parte responsável pelas configurações iniciais do *OpenMV Cam*, isto inclui quer o sensor da camara, quer os pinos e funções necessárias para o envio de dados para o leitor RFID. Ainda no ***Setup*** são definidas e inicializadas variáveis de apoio que serão usadas pelo algoritmo, como por exemplo, os contadores.
- **Análise da Imagem** – É nesta parte que se concentra todo o algoritmo responsável por identificar e interpretar o que está a ser capturado pelo sensor da camara. A cada ciclo neste bloco espera-se que, caso alguma peça seja identificada, seja incrementado o respetivo contador associado.
- **Critério de Decisão** – Devido à velocidade com que a Análise da Imagem é feita e ao facto da mesma estar a ser executada continuamente (mesmo quando a peça ainda está em movimento e ainda não se encontra perfeitamente enquadrado com a câmara) foi decidido aumentar a resiliência da leitura com recurso a um "critério de decisão" recorrendo ao número de vezes que uma peça já foi identificada na análise de imagem, isto é, apenas quando a peça em questão já foi detetado pela câmara um determinado número de vezes é que é autorizada a comunicação. Este critério permite que uma eventual análise errada que aconteça esporadicamente (por exemplo, enquanto a peça ainda está em movimento) não seja transmitida para o leitor RFID. Desta forma, o critério de decisão permite passar à comunicação ou continuar a análise de imagem de acordo com o número de vezes que a peça já foi detetada pela câmara aumentando assim a resiliência do sistema a erros sem diminuir excessivamente a velocidade de leitura do mesmo (no capítulo 4.2.5. serão apresentados *benchmarks* relativos à velocidade de execução do algoritmo).
- **Comunicação** – É nesta parte que se executa o código responsável por configurar a *string* para a comunicação. Lembrando que a *string* tem de ser configurada de modo a garantir a compatibilidade com uma linguagem de programação distinta, neste caso de *python* para *arduino*. Também é nesta parte que todas os contadores das peças são reiniciados.
- **Controlo do Tapete** – Parte responsável por controlar se o tapete deve parar ou estar em movimento. A paragem do tapete advém da necessidade de garantir que a peça

fica imóvel em dois momentos distintos, sendo o primeiro aquando da sua leitura pela câmara, desta forma é garantida à camara o tempo necessário para uma leitura correta e imune a erros, já a segunda paragem é requerida pelo braço robótico de modo a garantir que a peça está corretamente posicionada para ser removido do tapete.

4.2.4. Implementação

A implementação das funções para o ***setup*** exigiram a utilização de bibliotecas disponibilizadas pelo *OpenMV Cam*, que, como já tinha sido mencionado no capítulo 4.2.1., a sua disponibilização representou uma vantagem na utilização desta plataforma.

Foi usada a biblioteca ***pyb***, que dispunha de um conjunto de funções que permitiam manipular a placa do *OpenMV*. Neste caso, foi usada a função ***pyb.SPI()*** para configurar os pinos relativamente à comunicação SPI com a polaridade e fase desejadas. Foi ainda usada a função ***pyb.ExtInt()*** para configurar o pino responsável por chamar a função ***nss_callback()***, função responsável polo envio dos dados, assim que requerido pelo *master* (Arduino). Usando a função ***pyb.LED()*** (por exemplo, *led1 = pyb.LED(1)*), seguido de ***led1.on()*** foram ligados os *leds* presente na placa do *OpenMV Cam,* de modo a obter um *feedback* relativamente ao sucesso da execução do *setup* e ao correto funcionamento da câmara.

Devido ao facto de as plataformas utilizarem linguagens de programação distintas (*python* no caso do *OpenMV*, *Arduino* no caso do RFID) é necessário configurar a *string* de envio de modo a garantir a compatibilidade com o Arduino. Foi utilização a biblioteca ***ustruct***, que dispunha da função ***ustruct.pack()*** que fazia exatamente o pretendido.

Uma vez que, ocupando o *OpenMV* a função de *Slave,* é necessário que esteja sempre disponível para enviar dados. Incluindo o tempo em que ainda não foi detetado uma peça, deste modo, e por padrão, a *string* que estará disponível para enviar será “Hello World”, de modo que, quando o leitor RFID receba esta *string* saiba que a comunicação está operacional, mas ainda nenhuma informação útil está disponível.

Por fim, foi utilizada a biblioteca ***sensor*** para que, através das funções ***sensor.set_pixformat()*** e ***sensor.set_framesize()*** fosse configurado o sensor da câmara para o formato RGB (*sensor.set_pixformat(sensor.RGB565)*) e resolução 160x120 (*sensor.set_framesize(sensor.QQVGA)*). A escolha do formato de cor RGB deveu-se à necessidade de distinguir a cor das peças, já a resolução foi escolhida de modo a maximizar a velocidade de processamento da imagem sem comprometer a qualidade da mesma. A função ***sensor.reset()*** foi usada, antes das configurações mencionadas para inicializar o sensor da camara e a função ***sensor.skip_frames()*** imediatamente após a configuração, por uma questão de boas práticas evitando assim bugs na configuração (como recomendado na documentação do *OpenMV Cam*) [1].

A implementação do algoritmo responsável pela **análise de imagem** seguiu a seguinte lógica. Inicialmente, foi usada a função ***sensor.snapshot()*** (da biblioteca ***sensor*** já usada no *setup*) para guardar em memória uma fotografia sobre a forma de um objeto do tipo *image* sob a qual é processado o algoritmo.

Começando por procurar um *blob* (isto é, uma mancha de cor) de uma das cores identificáveis (vermelho e verde, às quais é atribuído um valor numérico) para tal usou-se a função ***find_blobs()*** (os parâmetros fornecidos à função foram *fing_blobs([thresholdsVermelho, thresholdsVerde], área_threshold=2500, merge = True)*), onde o primeiro corresponde aos *thresholds* das cores Vermelho e Verde, respetivamente. Estes *thresholds* foram definidos com recurso a uma ferramenta disponível no *IDE* do *OpenMV*.

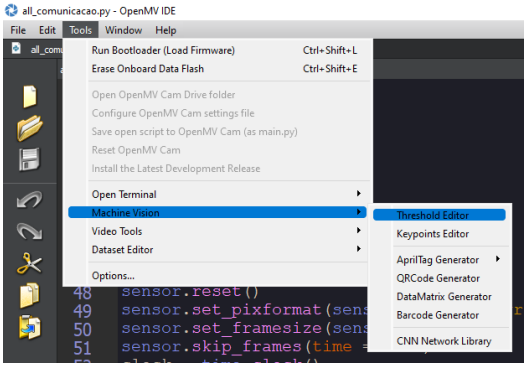

Figura 9- Threshold Editor -IDE do OpenMV

Caso nenhuma mancha relativa às cores identificáveis seja encontrada o algoritmo em nada altera as variáveis relativas aos contadores (como referido na arquitetura, 4.2.3., a cada peça é associado um contador que guarda o número de vezes que a mesma foi detetada pela câmara), caso alguma mancha seja encontrada tem de identificar se essa mancha corresponde a alguma das formas identificáveis (retângulos, triângulos, círculos ou estrelas).

Para identificar **retângulos** foi utilizada a função ***find_rects()*** (disponível através da biblioteca ***image*****,** e com o *threshold* devidamente ajustado), caso um retângulo seja encontrado pela função, é comparada a cor do pixel central do mesmo com a cor da mancha em questão (esta comparação permite aumentar a resiliência do código a erros pois evita que, por exemplo, uma mancha vermelha na mesma imagem que um retângulo azul seja entendida como um retângulo vermelho). Para executar tal operação são obtidas as coordenadas dos vértices do retângulo com a função ***corners()*** a partir dos quais são calculadas as coordenadas do ponto central (dividindo a soma de todas as componentes x e y por 4, obtendo um vetor da forma [(x1+x2+x3+x4)/4 ; (y1+y2+y3+y4)/4].

Através da função ***get_pixel()*** obteve-se o código RGB do ponto central que pode ser comparado com a cor do *blob* identificado. No caso de serem iguais o contador relativo à forma e cor em questão é incrementado (tal não é feito caso as cores entre o ponto central e o *blob* sejam distintas).

Para identificar **círculos** foi utilizada a função ***find_circles()*** (disponível através da biblioteca ***image*****,** e com os *thresholds* devidamente ajustados), é também obtido o código

RGB de um pixel do interior do circulo para comparar a sua cor com a do *blob* encontrado e caso coincidam o contador respetivo é incrementado.

Finalmente, para identificar **triângulos** e **estrelas** e, visto que para tais formas não existia nenhuma função na biblioteca que os identificasse diretamente como no caso das formas anteriores. Assim, foi criado um algoritmo que as identifica através da análise do número de linhas retas na imagem e ângulos relativos entre as mesmas. Isto sabendo que, um triangulo é um conjunto de 3 segmentos de reta com diferença de 60º entre os mesmos. Por sua vez, uma estrela pode ser identificada por 5 retas com diferença de 36º entre as mesmas.

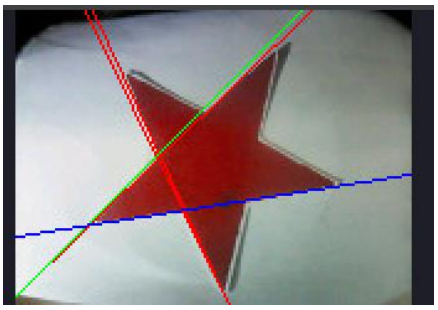

Figura 10 – Identificação das linhas de uma estrela no OpenMV Cam

Em ambos os casos são dados intervalos de erro visto que a leitura na camara é suscetível a erros. As linhas retas na imagem foram obtidas com recurso à função ***find_lines()***. Esta função retorna o início, o fim e o ângulo da reta relativo ao eixo vertical das linhas retas detetadas na imagem. Foi então criado e ordenado por ordem crescente um vetor com os valores dos ângulos das retas encontradas.

Posteriormente, foi verificado se o número de linhas retas obtidas é suficiente para representar um triangulo ou uma estrela (3 e 5, respetivamente). Por fim, e se se verificar o número de retas necessárias, são calculadas as diferenças entre ângulos consecutivos (previamente organizadas por ordem crescente num vetor).

Para o caso do triangulo, foram aceites ângulos com erro absoluto de 10º, isto é, no intervalo de 50º a 70º. Já para a estrela foram considerados ângulos no intervalo de 30º a 45º como válidos. Caso a diferença entre ângulos consecutivos cumpra os intervalos definidos o contador respetivo à peça é incrementado.

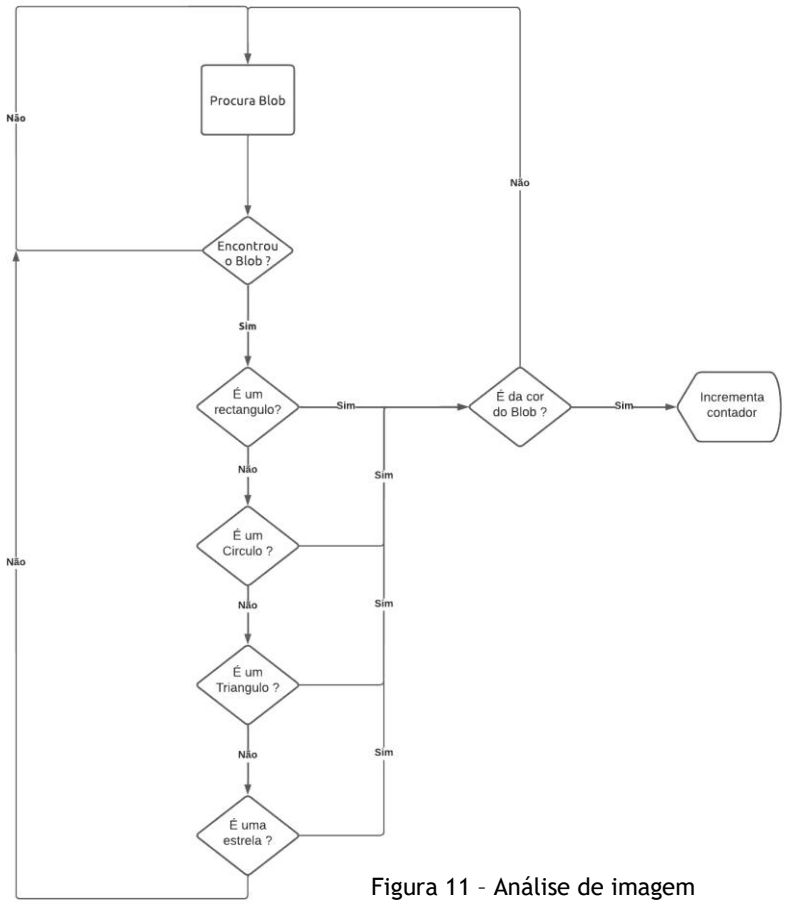

Figura 11 – Análise de imagem

A figura 11, mostra o fluxograma do código descrito.

A implementação do **critério de decisão** baseia-se no uso de uma ***flag booleana***. Em cada ciclo, todos os contadores (onde ficam registadas as vezes que a determinada peça foi detetada) são verificados, se algum dos mesmos estiver chegado ao valor pré-definido a *flag* é passada a 1, o que permite executar o código relativo à comunicação.

Também é preenchida a *string* que será usada na comunicação com a informação relativa à peça detetado. O número de vezes que uma peça tem de ser detetado para poder ser transmitido é de 5 para estrela e 10 para os restantes, independente da sua cor.

Ambos os valores foram obtidos fazendo vários testes no tapete e garantindo um compromisso entre resiliência a erros e velocidade de execução, isto porque, como é evidente, a exigência de mais ciclos para validar a leitura aumenta o tempo necessário para a mesma ser finalizada.

É para garantir este compromisso que o número de vezes que é preciso detetar uma estrela para considerar a leitura válida é menor relativamente às restantes formas, dado que a sua leitura é mais demorada.

A implementação do código relativo a à **comunicação** encontra-se dentro dum *if statement* para que só seja executado quando a *flag* estiver a 1 conforme a lógica explicada no "**Critério de Decisão**". Uma vez que já os pinos e configurações da comunicação já estão configurados (tal é feito no *setup*) a implementação da comunicação exige apenas configurar a *string* para envio de informação útil para o leitor RFID, deste modo, é usada a biblioteca *ustruct* da mesma maneira que no *setup* mas, desta vez, em vez de ser configurada a *string* padrão "Hello World" é configurada a *string* com informação acerca da forma e cor da peça que foi identificado (por exemplo, "Estrela Vermelha"). Para respeitar o formato em que a informação será escrita no RFID (blocos de 16 bytes) a *string* é configurada utilizando os primeiros 16 *bytes* para a forma e os 16 *bytes* seguintes para a cor, ou seja, no caso do exemplo anterior o que efetivamente é enviado é "ESTRELA

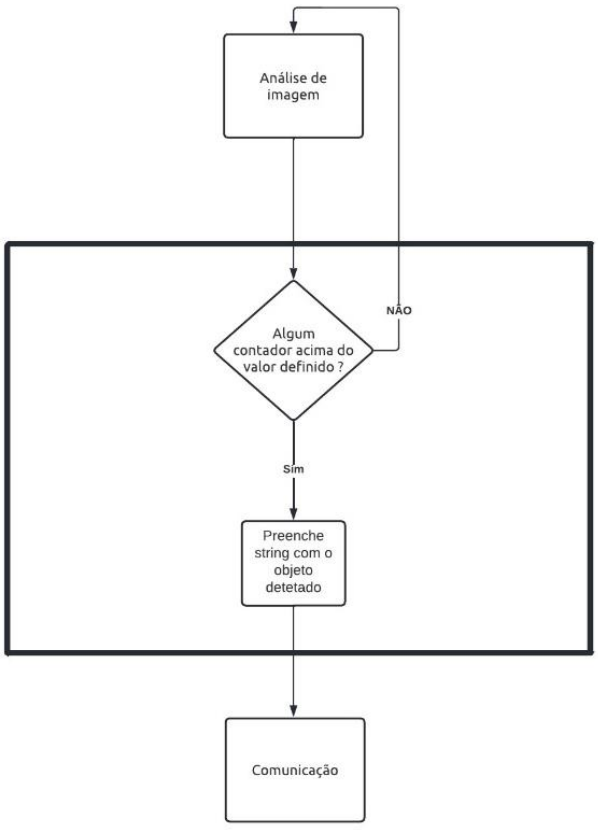

Figura 12- Critério de decisão

\0VERMELHO \0". Esta configuração facilita a divisão de uma única *string* em duas distintas e correta escrita nos blocos da *Tag* RFID.

Para a implementação do **controlo do tapete** foram analisadas as caraterísticas e especificações do material fornecido, descritas de seguida, e implementados os circuitos necessários à sua manipulação.

Figura 13 – Caraterísticas e especificações do material fornecido

Foi, então, projetado o seguinte circuito:

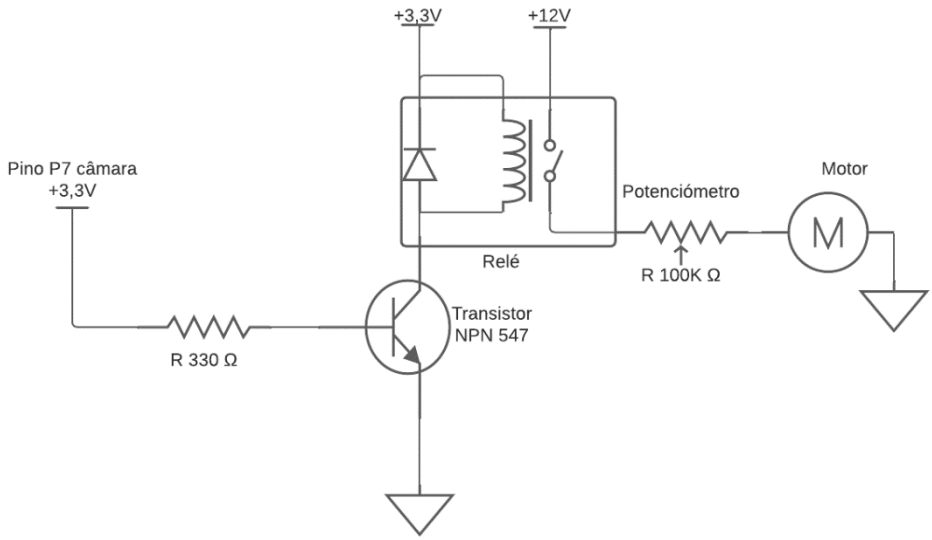

Figura 14- Circuito de controlo do tapete

De facto, **o circuito apresentado pode ser dividido em dois.** O primeiro circuito é responsável pela interação indireta com o motor do tapete, já o segundo é responsável pela interação direta com o motor. O controlo do tapete é efetuado através da interação entre estes dois circuitos, recorrendo ao componente elétrico "Relé" representado na figura a cima pela caixa devidamente identificada como tal.

Figura 15- Relé

Relativamente ao circuito que interage indiretamente com o motor através do relé. Este é constituido por um transistor bipolar NPN 547. A base do transistor é alimentada pelo pino 7 da câmara a +3,3V quando o pino se encontra no modo "High" podendo-se cortar a alimentação para parar o movimento do tapete comutando o pino para modo "Low". Foi colocada, na base do transistor, uma resistência de 330Ω para garantir uma corrente de aproximadamente 10mA na mesma de modo a que o transistor opere em saturação. O emissor ligou-se diretamente ao GND, enquanto que o coletor se ligou a um diodo em paralelo com uma bobina, sendo estes os componentes do relé, alimentados a +3,3V pelo microcontrolador do *OpenMV Cam*.

A bobina associada ao relé quando percorrida por corrente elétrica gera um campo magnético que força a movimentação fisica do interruptor presente no outro ciruito (fecho do interruptor). Quando é necessário a paragem do tapete, o pino 7 deixa de alimentar o circuito, forçando a transitor a sair de saturação e assim parando a alimentaçao do relé, acabando com o campo magnético gerado e deste modo o interruptor volta ao seu estado inicial (abertura do interruptor).

O diodo está presente no relé por motivos de segurança. Quando o relé deixa de ser alimentado, o diodo impede que a corrente siga no caminho inverso, obrigando-a a passar por si fazendo com que a corrente siga no sentido pretendido até não existir mais corrente no circuito.

O transistor tem uma função semelhante à do diodo, caso este falhe, o transistor impede que a corrente faça o caminho inverso, prevenindo a corrente de voltar para o microcontrolador evitando a sua danificação.
Como foi dito, o interruptor do relé que interage diretamente com o motor do tapete pode estar aberto ou fechado conforme for necessário. Consoante o estado em que este se encontra o motor é ou não é alimentado, fazendo o tapete movimentar-se ou estar parado respetivamente. Quando o motor é alimentado interage com um potenciometro permitindo regular a velocidade com que o tapete se vai mover.

Relativamente aos pontos de paragem, como já foi mencionado, são dois os locais onde o tapete deve parar de modo a garantir que o sistema funcione corretamente, sendo estes a zona de identifação, onde a camâra analisa a imagem captada e identifica o tipo de peça conforme a sua cor e forma, e a zona de aramzenamento, onde a peça aguarda ser recolhida pelo braço robótico para a mesma ser devidamente armazenada.
A paragem na zona de identificação da peça é efetuada quando a câmara deteta a presença de uma das cores de peças identificaveis (vermelho e verde) interrompendo a alimentação do circuito. Tal identificação é realizada com recurso à função ***find_blobs()***.

Após a devida identificação da peça acrescentou-se um *delay* de 5 segundos para garantir que a informação detetada pela câmara é devidamente transmitida para a *tag* do RFID. Ultrapassado o *delay* volta a alimentar o circuito de controlo do tapete.

A segunda paragem é determinada pelo tempo que a peça necessita para chegar da zona de identificação à zona de armazenamento, para tal foi dimensionado um *delay*. À velocidade em que foi testado o projeto, o *delay* escolhido foi de 5 segundos, contudo, o mesmo teria de ser reajustado caso se pretendesse alterar a velocidade do tapete ou a distância entre as zonas de paragem. Mais uma vez, aquando da chegada à zona de paragem é acrescentado um *delay* (neste caso de 3 segundos) antes de retornar à movimentação para garantir o tempo necessário ao armazenamento da peça.

A implementação descrita apresenta algumas limitações uma vez que não permite a presença de mais de uma peça ao mesmo tempo no tapete e necessidade de constantes ajustes nos valores dimensionados para o *delay*. De facto, foi, inicialmente, tentada uma solução alternativa, com recurso a células fotoelétricas, nomeadamente um foto-emissor (STAL6100) e um foto-transistor (SFH213). Tal solução não foi implementada com sucesso no demonstrador final uma vez que, apesar de funcionar como esperado quando atuava isoladamente, a sua integração no sistema não foi bem-sucedida. Perante a aproximação ao final do projeto a solução acima descrita permitiu, com as respetivas limitações, cumprir os requisitos propostos.

Figura 16- STAL6100

Figura 17- SFH213

O **circuito projetado** para controlar o motor do tapete com as células fotoelétricas foi o seguinte:

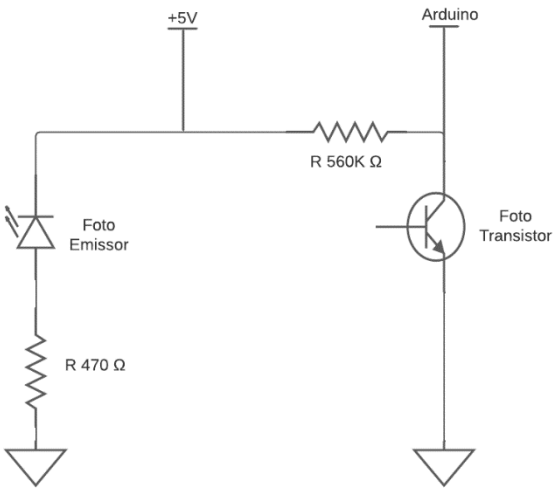

Figura 18- Circuito para controlar o motor do tapete

Alimentando este circuito com +5V do Arduino, de um lado (esquerdo) colocou-se o foto-emissor, representado por um LED no circuito, que irá enviar infravermelhos para o foto-transistor, representado por um transístor no circuito (visto que o seu comportamento é muito semelhante), e colocou-se uma resistência de 470Ω entre o GND e o foto-emissor de modo a controlar a corrente que iria passar pelo LED prevenindo que este não se danificasse.

Do outro lado (direito) colocou-se uma resistência de $560K\Omega$ para tornar estável a receção dos sinais por parte do foto-transistor, e em paralelo um pino do Arduino que iria receber o sinal do foto-transistor.

A partir do Arduino conseguiu-se ler o sinal recebido usando a função ***analogRead()*** que converte para bits o valor da tensão recebido, sendo os +5V, convertidos no número 1023, e os 0V convertidos no número 0. Ou seja, quando não for recebida luz por parte do foto-transistor, o Arduíno irá receber próximo de 1023 (devido à interferência do meio ambiente, o valor recebido nunca será exatamente 1023), já quando recebe luz o valor recebido será substancialmente menor.

Pensa-se que o motivo para que esta implementação não funcione quando integrada no sistema completo seja a interferência um de componente noutro circuito que faz o foto-transistor receber sempre a mesma sequência de valores, apesar de estar presente entre as duas células uma peça.

4.2.5. Resultados

No final, o Sistema de Visão Industrial cumpriu os requisitos propostos sendo capaz de distinguir entre as diferentes peças pretendidas, comunicar tal informação ao Sistema de Classificação das Peças e controlar devidamente as paragens do tapete.

De modo a quantificar a performance do *OpenMV* durante a execução do programa realizaram-se *Benchmarks* para todas as formas, calculando quanto tempo cada uma demorava a ser identificada através da diferença entre o tempo que é feita a primeira deteção e o tempo em que a informação é enviada para o leitor RFID. Na seguinte tabela são apresentados os valores obtidos em quatro testes para cada forma, todos os testes foram feitos com as peças na mesma cor (vermelha) uma vez que a cor não influencia significativamente o tempo necessário para a deteção.

Tabela 1- Benchmarks relativos à velocidade de execução do programa

Forma	Teste 1	Teste 2	Teste 3	Teste 4	Média
Retângulo	2,10 s	5,14 s	2,00 s	2,50 s	2,94 s
Círculo	1,20 s	1,00s	1,50 s	2,00 s	1,35 s
Triangulo	6,12 s	4,10 s	5,10 s	4,66 s	4,88 s
Estrela	14,25 s	16,60 s	7,85 s	8,23 s	11,24 s

4.3. Sistema de Armazenamento de Peças

4.3.1. Contextualização e Princípio de Funcionamento

Perante a necessidade de retirar as peças do tapete, foi necessário a conceção do Sistema de Armazenamento de Peças que para além de as retirar do tapete, também as armazena. Deste modo foi-nos sugerido o uso do robot Braccio que é controlado por um Arduíno e tem à disposição uma biblioteca fornecida pela empresa Arduíno. Esta biblioteca permite usar comandos simples para movimentar cada eixo do braço mecânico, facilitando o controlo do mesmo. Foram também fornecidas pinças para fixar o robot a uma superfície, de forma a obter mais estabilidade e precisão nos movimentos do braço robótico. O robot é adequado para o projeto uma vez que, para além das características referidas anteriormente, dispõe de um conjunto de interfaces I/O fundamentais para a comunicação entre dispositivos.

Figura 19- Braccio

Para além disto, este sistema inclui o módulo RFID, ou seja, uma plataforma computacional, que neste caso é um Arduino UNO, ligada a um leitor de RFID.

4.3.2. Requisitos

Relativamente aos requisitos necessários para o sucesso deste sistema, definimos como essencial a remoção da peça do tapete de forma segura e sistemática. Para além disto, também considerámos importante armazenar as peças tendo em conta a informação guardada nas suas *tags*, tendo como auxílio o módulo RFID.

4.3.3. Arquitetura

Este sistema está separado em 3 partes principais:

- **Leitura de *tags***: O leitor RFID é posicionado bastante próximo do tapete, de forma a estar à distância adequada a cumprir a sua função, sendo esta a leitura das *tags* associadas às peças.

- **Processamento de informação**: A informação recolhida é interpretada e filtrada. Posteriormente esta, resultante do processamento, é simplificada para um código de forma a facilitar a transmissão de informação para o braço robótico.

- **Manipulação da peça**: Remoção da peça do tapete e armazenamento da mesma tendo em conta as suas características.

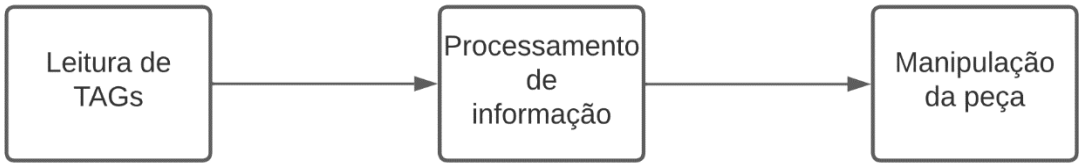

Figura 20- Arquitetura do Sistema de Armazenamento de Peças

4.3.4. Implementação

Inicialmente, tentámos usar o robot Mycobot, explorando o método de conexão do software VNC viewer com o raspberry PI para o tentar controlar através de programas como RoboFlow e Blockly. Contudo, este tinha um erro de comunicação entre o software de origem e o hardware.

Devido a este erro, decidiu-se utilizar o robot Braccio. Começamos por tentar efetuar a comunicação entre o robot e o RFID por SPI, mas os pinos do Arduino do Robot dedicados a esta comunicação não estavam disponíveis, uma vez que são dedicados ao controlo dos motores do robot. Decidiu-se então utilizar outro Arduino, para ser possível estabelecer comunicação entre todos os componentes.

Para se ler a *tag* associada à peça, utilizou-se um leitor RFID cujo código de implementação foi baseado no código previamente desenvolvido no Sistema de Classificação de Peças (4.1). Através da utilização das bibliotecas MFRC522 e SPI, e definindo o pino do Arduíno referente ao Slave Select associado ao RFID, foi possível estabelecer a comunicação entre o RFID e o Arduíno com sucesso.

No Arduíno, desenvolvemos código para interpretar e filtrar a informação lida e comunicada pelo leitor RFID. As *tags* têm informação relativa ao seu número forma e cor, sendo apenas necessário para a decisão de armazenamento a sua forma e cor. Deste modo, apenas se envia estas duas componentes de informação para o microcontrolador sob a forma de duas strings. No código do Arduíno, declarou-se uma variável numérica, na qual atribuímos um número de dois dígitos, conforme as strings presentes na *tag*, como mostra a tabela.

STRINGs	Código	STRINGs	Código
RETANGULO VERMELHO	43	RETANGULO VERDE	42
TRIANGULO VERMELHO	33	TRIANGULO VERDE	32
CIRCULO VERMELHO	23	CIRCULO VERDE	22
ESTRELA VERMELHO	13	ESTRELA VERDE	12

Tabela 2- Códigos relativos a strings

No caso de ocorrência de algum erro (objeto não identificado, erro na leitura, etc) é atribuído um código de erro:14.

De forma a filtrar o excesso de leituras do módulo RFID, declarou-se uma variável que garantia que a mesma peça só seria lida uma vez consecutivamente. Assim, o excesso de leituras é descartado e permite uma melhor fluidez de comunicação.

Como mencionado anteriormente, os pinos de SPI do robot estavam ocupados, portanto, para este comunicar com o Arduino, optou-se pela comunicação por UART. No protocolo UART (Universal Asynchronous Receiver/Transmitter) a comunicação é assíncrona, ou seja, os bits são enviados/recebidos a uma dada cadência denominada *baudrate* (medida em bits por segundo). Para efetuar a ligação UART, conectam-se o pino TX (transmissão) de uma das plataformas computacionais ao pino RX (receção) da outra e vice-versa. Após a atribuição de um valor à variável numérica, como referido anteriormente, transmite-se o valor desta variável pela porta série (através do comando ***Serial.Write()***), e após a receção o robot guarda esse valor numa variável local (através do comando ***Serial.readByte()***).

De forma a mostrar as diferentes funcionalidades do robot e os diferentes critérios de decisão implementados, decidiu-se desenvolver a funcionalidade de um botão, que sempre que é pressionado, alterna o valor de uma FLAG binária criada. Portanto, se o valor da FLAG for 1 e o botão for pressionado este valor passa para 0 e vice-versa. Se o valor desta FLAG for 1, o robot armazena as peças nas caixas tendo em conta a sua cor, e se a FLAG for 0, o robot armazena as peças nas caixas tendo em conta a sua forma.

Manipulando os ângulos dos eixos do robot, é possível deslocar a garra do braço para onde for pretendido, assim como abrir/fechar a pinça.

Foram posicionadas 7 caixas, destinadas a cada uma das características: duas relativas ao armazenamento tendo em conta as cores, quatro tendo em conta as formas e uma em situação de erro, que pode derivar de *tags* com informação corrompida, leituras mal-sucedidas ou outras situações não esperadas.

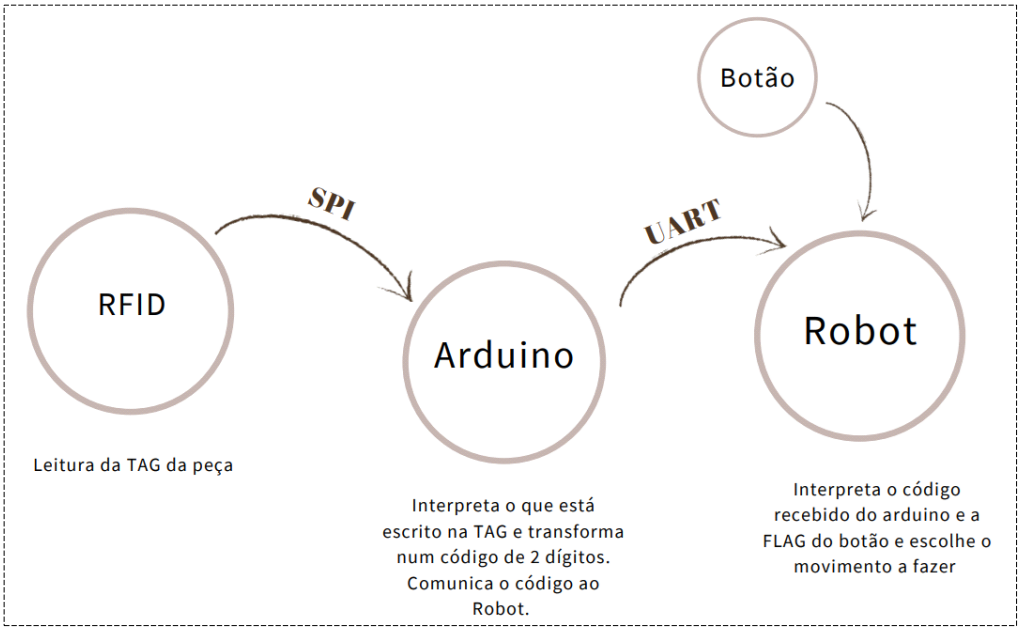

Figura 21- Diagrama de interação entre as diferentes componentes do subsistema

O robot encontra-se inicialmente numa posição de repouso, por cima do tapete, à espera de uma leitura do módulo RFID. Após a leitura, o braço robótico recebe o código relativo à peça e deteta a sua presença. Consequentemente, desce a garra, e considerando as características da peça, fecha-a de forma a prendê-la. De notar que o robot ajusta a forma como remove e transporta a peça tendo em conta a sua identificação, devido à largura, altura, forma e peso distintos entre todas as peças.

Posteriormente, o braço robótico volta a subir e, dependendo do valor da FLAG e do código recebido, executa a rotação necessária para o destino decidido. Após o transporte, baixa a garra e solta-a para poder cair na caixa em questão. Seguido deste processo volta à posição inicial, aguardando por uma nova leitura. A execução de transporte e armazenamento da peça é baseada num sistema de coordenadas. Sabendo os limites de cada eixo do braço, fomos ajustando o posicionamento deste conforme o ângulo do destino pretendido.

4.3.5. Resultados

O Sistema de Armazenamento de Peças conseguiu cumprir os requisitos propostos, pois foi possível o armazenamento das peças tendo em conta a informação de identificação contida nas *tags* das mesmas, sempre garantindo a segurança nos movimentos de manipulação das peças conforme as diferentes características.

Como foi referido anteriormente, devido ao facto do robot ter os pinos necessários para a comunicação por SPI ocupados, utilizou-se um Arduino extra para poder estabelecer a comunicação entre todas as partes. Isto tornou o processamento de informação mais lenta e também criou dificuldades no *debugging*, uma vez que o processo de comunicação era mais complexo. Perante esta situação assumimos que se poderia recorrer a outra solução se o projeto continuasse a ser desenvolvido. No entanto, reforçamos a ideia de que o Braccio tem movimentos rigorosos e devido à calibração feita é efetuado um transporte bastante preciso e seguro, essenciais para o bom funcionamento do sistema.

5. Conclusão e futuros desenvolvimentos

5.1. Conclusão

No final do projeto, foi atingido um sistema completo capaz de identificar, através de Visão Industrial, diferentes tipos de peças de quarto formas e duas cores distintas, guardar a informação recolhida, através de *tags* RFID e armazenar, através do braço robótico, nos devidos locais separando-as segundo as suas caraterísticas. O sistema foi, também, capaz de funcionar como um todo garantido a coordenação e comunicação entre as diferentes partes que o constituem. Desta forma, conclui-se que cumpriu os requisitos propostos.

A divisão do sistema em diferentes subsistemas revelou-se a escolha acertada quer para a gestão de tempo e recursos do projeto quer como método para simplificar e localizar os problemas que foram surgindo. Todos os subsistemas cumpriram os requisitos que lhes foram individualmente atribuídos.

Apesar dos esforços para tornar o ambiente onde o sistema funciona o mais controlado possível, o mesmo, acabou por apresentar fragilidades relativas a fatores externos, das quais se destacam a luminosidade, a falta de uniformidade na cor do tapete e o curto alcance das *tags* RFID utilizadas. A luminosidade em conjunto com a falta de uniformidade na cor do tapete influenciava a maneira como o *OpenMV Cam* interpretava as cores o que, em certas condições, leva a que a paragem do objeto para respetiva identificação ficasse comprometida. Por sua vez, as *tags* RFID utilizadas neste projeto apresentavam um alcance muito reduzido, o que impedia a sua leitura pelo leitor RFID quando as peça não eram devidamente colocadas no tapete.

Relativamente às competências pessoais dos elementos do grupo, foram desenvolvidas capacidades técnicas em tecnologias com carater cada vez mais preponderante na automação industrial, nomeadamente visão industrial e identificação por radiofrequência, sendo este projeto uma oportunidade para ter o primeiro contacto com as mesmas. Além disso, foram também desenvolvidas competências não técnicas fundamentais para o sucesso do projeto como a gestão de tempo, divisão de tarefas e métodos de trabalho em grupo.

Foi realizado um vídeo de modo a demonstrar o funcionamento do completo projeto, tal pode ser consultado em https://www.youtube.com/watch?v=3IpZFLM8_l0 .

5.2. Futuros Desenvolvimentos

Numa eventual continuação deste projeto, seria interessante aumentar os seus mecanismos de resiliência a fator externos. Para tal, poderiam ser trocadas as *tags* RFID utilizadas por outras que garantissem um alcance superior evitando, desta forma, que peças colocadas ligeiramente mais afastadas do leitor falhassem a identificação. Os problemas relativos às variações de luminosidade podiam ser mitigados isolando a câmara dentro de uma caixa iluminada internamente, onde não entrasse luz exterior. Para além disso, também poderia ser mais explorado o uso de sensores para controlo do tapete, permitindo dessa forma ter mais que uma peça em simultâneo no tapete sem prejuízo para o correto funcionamento do sistema.

Finalmente, seria atrativo aumentar o leque de formas e cores identificadas permitindo aumentar, desta forma, a diversidade de peças usadas nas demonstrações do sistema.

Referências

[1] https://docs.openmv.io/library/omv.sensor.html