Aplicação com um Robô Real usando Visão Artificial

Departamento de Engenharia Mecânica
Mestrado em Robótica e Sistemas Inteligentes
Robótica Industrial - Vitor Santos
Universidade de Aveiro

Filipe André Seabra Gonçalves - 98083
Bruno Alexandre Ferreira Duarte - 118326

Janeiro de 2024

1 Introdução

Este relatório tem por objetivo explicar a abordagem e o código desenvolvido para o desenvolvimento das duas partes que completam o projeto *Aplicação com um Robô Real usando Visão Artificial*, usando o robô ***Fanuc LR Mate 200 iD***.

Na primeira parte era preciso obter uma aplicação de consola do robô que usa instruções na linguagem *TP* e que ilustra diversas funcionalidades e movimentos do robô, dos quais tinha de ter cinco movimentos em juntas, cinco movimentos lineares, dois movimentos circulares e quatro atuações do *gripper*, com um limite mínimo de cinquenta linhas. O tema escolhido para esta primeira parte foi a montagem de uma pirâmide de taças de plástico.

Em relação à segunda parte, era preciso criar uma aplicação em *Matlab* que se ligasse ao robô via *TCP/IP* e que usando a câmara ao nosso dispor, identificavamos a localização de objetos ou pontos importantes e usando o *gripper* movimentavamos esses objetos para um ponto importante.

2 Primeira Parte

Na primeira parte deste trabalho desenvolvemos um programa na consola, utilizando a linguagem *TP* do robô, que retira taças de plástico de uma pilha e constrói uma pirâmide com as peças.

O programa da consola começa sempre com uma desativação do *gripper* para manter sempre o seu estado inicial consistente, quer o *gripper* tenha estado ativo ou não, e por dois movimentos em juntas para colocar o robô na posição inicial e, seguidamente, na posição exatamente acima da torre.

Para cada peça da torre, o robô começa sempre numa posição extamanente acima da torre, e é feito um movimento em junta sobre o seu z global seguido de uma atuação do *gripper* para colocar o *gripper* em cima da taça e a puder apanhar. É feito depois um movimento em junta para retirar a peça da torre, aumentando apenas o seu z global.

Com a taça sugada pelo robô, é feito um movimento circular para colocar a taça numa posição acima do plano onde irá ser colocada na pirâmide, seguida por dois movimentos de juntas para colocar a taça no plano, e por fim a deativação do *gripper* e um movimento de junta aumentando apenas o seu z global de modo a não estar com o *gripper* colado à taça.

Finalmente, para colocar o robô numa posição extamente em cima da torre de taças é feito um movimento linear.

O robô repete estes passos todos para as cinco taças existentes na torre. No entanto uma pirâmide de base três precisa de 6 objetos e por isso o robô quando finaliza de colocar todas as taças da torre nas suas posições respetivas na pirâmide, "encontra" um copo vermelho e usando dois movimentos de junta coloca-se exatamente em cima deste. Ativando o *gripper* e depois de fazer um movimento de junta aumnetando apenas o z global, o robô utiliza um movimento linear para derrobar a pirâmide anteriormente criada.

Por último, o robô utiliza um movimento de junta e uma atuação do *gripper* para libertar o copo vermelho, e termina o programa com um último movimento de junta para a posição inicial.

A Figura 1 mostra em diagrama de blocos todos os passos desta parte do trabalho.

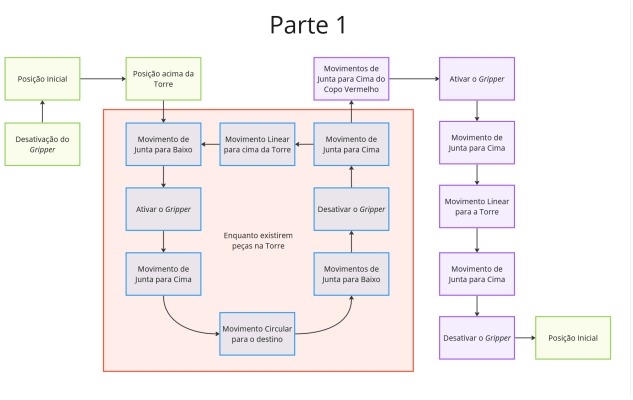

Imagem 1: Diagrama de Blocos - Parte 1

Para puder desenvolver esta parte do trabalho foi utilizado um procedimento manual, onde primeiramente deslocou-se o robô manualmente para as posições desejadas usando a consola do robô e por fim gravou-se a posição do robô no programa de consola.

Este procedimento é útil para tarefas em que o robô faça movimentos sistemáticas e saiba extamente as posições onde tem de estar - solução determinística. Para problemas mais complexos, em que a posição ou orientação de objetos não é sempre a mesma, seria necessário usar visão artificial, como iremos falar na segunda parte do trabalho.

Os requisitos concluídos e necessários para esta parte do trabalho pode ser vista na Tabela 1.

Descrição do Movimento	Nº Movimentos/Nº Movimentos Necessários
Movimento de Junta	32/5
Movimento Linear	5/5
Movimento Circular	5/2
Atuação do Gripper	13/4
Movimentos Totais	55/50

Table 1: Núumero de movimentos por tipo

3 Segunda Parte

3.1 Conceito

A segunda parte do projeto será sobre usar visão para identificar objetos de cores diferentes, áreas de trabalho de cores diferentes e movimentar os objetos para as àres de trabalho com a sua cor respetiva.

3.2 Calibração da Câmara

Antes de começarmos e nos conectarmos ao robô, o primeiro passo para qualquer tipo de projeto com visão artifical é a calibração da câmara. Assim, a aplicação terá uma compreensão mais precisa da posição e orientação do robô e dos objetos envolventes. Para tal, foi usado o script gerado pelo add-on *Camera Calibrator* do *Matlab*, com algumas adições.

Primeiramente, antes de qualquer tipo de calibração, o programa tenta identificar qualquer ficheiro de calibração guardado no diretório do projeto, e em caso afirmativo não é necessária nova calibração. Em caso contrário o programa faz a calibração normal da câmara e retorna os parâmetros da câmara calibrada.

A calibração da câmara é feita ao, primeiramente, obter N imagens, neste caso quinze, da *webcam*, posicionada acima do robô, conseguindo ver tanto o tabuleiro de xadrez de calibração como o robô. As imagens são depois guardadas no computador pelo nome *image_calibration_X*, onde X é o número da imagem, de um a quinze, neste caso, e por fim, obtemos os parâmetros intrínsecos e extrínsecos da camâra, para depois eliminarmos todas as imagens do diretório.

Para fazer a calibração da câmara foi criada uma função **calibrateCamera** que recebe como parâmetros de entrada, o tipo de sistema operativo, *0* para *Linux* e *1* para *Windows*, e qual o index da câmara a utilizar (no caso de ter um portátil com uma câmara incluída, a *webcam* do computador conta como **index 1**, e por isso a *webcam* do manipulador será o **index 2**).

3.3 Ligação TCP/IP

Para conectar o robô ao nosso programa *Matlab*, seguimos os passos auxiliares fornecidos pelo professor, apresentados a seguir:

- Ligar o controlador do robô em modo manual, T1 $<$ 200mm/s, e com o *TeachPendant* em *ON*
- Ativar a licensa pelo *ROBCOMM2* na consola
- Ativar o *ROBCOMM* pelo *ROBCOMM* na consola
- Premir *F1* para iniciar o *ROBCOMM*

Depois de ligado o *ROBCOMM* apenas é necessário criar uma conexão *TCP/IP* no *Matlab* com o *IP Address* a *192.168.0.229* e o *Port* a *4900*. Assim, temos a comunicação bem configurada e pronta para ser utilizada.

3.4 Reconhecimento da Área de Trabalho e dos Objetos

Para puder movimentar o robô, depois de todos os passos que fizemos, temos também de calcular o caminho da ponta. Para tal, é preciso detetar os objetos que queremos que o nosso *gripper* agarre e a área de trabalho para onde os tem de levar.

Antes de fazermos qualquer tipo de reconhecimento, movimentamos o robô para uma posição base inicial ($P = [400\ 0\ 220]$) onde a câmara consegue ver a grande maioria do universo espacial, e desativamos o ar do *gripper* para cautela.

Para detetar os objetos, precisamos de reconhecer as suas cores, que neste caso são o verde e o vermelho. Com esse intuíto, obtemos uma imagem da câmara e transformamos o *frame* dado para *HSV* (*Hue*, *Saturation* e *Value*), ao invés do normal *RGB* (*Red*, *Green* e *Blue*), para nos ser mais fácil ajustar as máscaras de deteção dessas mesmas cores.

Para detetarmos objetos vermelhos, usamos o filtro $Hue < 0.1$, para nos filtrar as cores vermelhas das restantes, e $Saturation > 0.4$ e $V > 0.3$, para filtrarmos os vermelhos mais fracos, e deixar a máscara com menos ruído.

Para detetarmos objetos verdes, usamos o filtro $0.2 < Hue < 0.5$, $Saturation > 0.4$ e $Value > 0.3$. Temos um maior alcance no valores, pois os objetos verdes têm cores ligeiramente diferentes uns dos outros, além de serem mais afetados pela ilimunação do local.

Estes valores foram obtidos com a luz existente no suporte da câmara acesa.

Depois de criada a máscara, detetamos os centros das maiores áreas (descontando os planos verde e vermelho porque como são a maior área sabemos que são os planos de destino dos objetos) e guardamos os seus pixeis numa matriz.

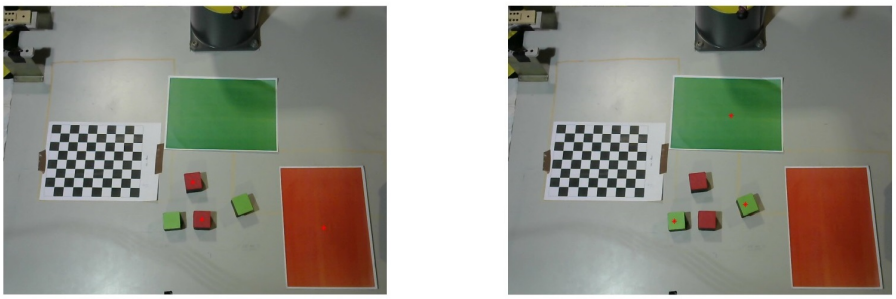

Imagem 2: Deteção dos objetos e dos planos de destino

Como podemos ver na Figura 2, a deteção dos objetos é feita corretamente, identificando dois grupos de cores, os dois objetos vermelhos e o seu plano de destino, e os dois objetos verdes e o seu plano de destino.

Para a deteção de objetos foram criadas duas variáveis, uma para cada cor, para identificar corretamente e apenas só esse número de objetos de cada cor. Aumentando ou diminuindo o valor, fará com que o programa identifique mais ou menos objetos dessa cor específica.

Posteriormente, para cada par de pixeis, depois é necessário converter para pontos do mundo real usando os parâmetros extrínsecos da câmara.

Por fim, temos de multiplicar os pontos pela transformação geométrica ${}^{R}T_{W}$ para converter os pontos do mundo real W para pontos que o robô consiga entender, no seu referencial R. Para calcular esta matriz, tivemos que perceber quais são as rotações que transcrevem as transformações geométricas do referencial do manipulador R para o mundol real W, e identificar o origem do referencial W com a ponta do robô.

A matriz ${}^{R}T_{W}$ ficou assim:

$$\begin{pmatrix} 0 & 1 & 0 & O_x \\ 1 & 0 & 0 & O_y \\ 0 & 0 & -1 & O_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Com O_x, O_y e O_z sendo o ponto de origem do referencial do xadrez visto pelo manipulador.

É de realçar que no caso do xadrez de calibração mudar de posição no referencial global, é necessário, para além de fazer uma nova calibração da câmara, identificar a origem do xadrez no referencial do robô e trocar as suas coordenadas pelas respetivas na matriz ${}^{R}T_{W}$.

3.5 Movimentação do Robô

Movimentar o robô é a parte mais fácil, porque usamos as *tools* do *functools* dadas pelo docente, para fazermos qualquer tipo de interação com o manipulador.

Seguem-se os principais comandos usados:

- *tcpclient(IP, PORT)* - Para criar uma conexão *TCP/IP* entre o programa e o manipulador, a função recebe o *IP* e o *PORT* do servidor do robô
- *Gripper_Open(robCOMM, robot)* - Para desativamos o ar do *gripper*, a função recebe o *handler* criado do robô e o tipo de robô (1 para o robô grande e 2 para o robô pequeno)
- *Gripper_Close(robCOMM, robot)* - Para ativarmos o ar do *gripper*, a função recebe o *handler* criado do robô e o tipo de robô (1 para o robô grande e 2 para o robô pequeno)
- *Get_Cart_Abs(robCOMM)* - Para sabermos a posição atual do robô, a função recebe o *handler* criado do robô
- *Mov_Cart_Inc(robCOMM,x,y,z,w,p,r,md)* - Para mover o manupulador com valores incrementais de x, y e z, e com rotações incrementais de w, p e r

Obtendo as coordenadas absolutas do manipulador com a função *Get_Cart_Abs(robCOMM)*, e com a posição dos objetos, apenas precisamos de saber o quanto a ponta precisa de "caminhar" até chegar ao seu destino.

Para puder agarrar os objetos e os largar chamamos, respetivamente, as funções *Gripper_Close(robCOMM, robot)* e *Gripper_Open(robCOMM, robot)*.

Depois de termos o objeto agarrado, levantamos o objeto, aumentando um pouco o z do manipulador e, usando a mesma lógica que para o caminho da garra até ao objeto, levamos o objeto até ao plano.

Este processo repete-se até não existirem mais objetos.

A Figura 3 mostra em diagrama de blocos todos os passos desta parte do trabalho.

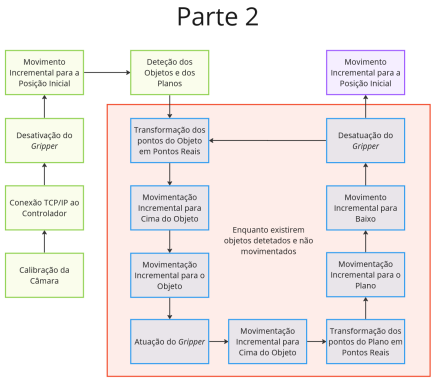

Imagem 3: Diagrama de Blocos - Parte 2

4 Conclusão

Concluindo, foram cumpridos todos os requisitos da primeira parte do projeto, bem como da segunda parte com exceção do requisito A de um plano de destino com diferença de cota de pelo menos 5cm.

Também é importante detalhar que o código criado em *Matlab* faz a deteção de um número N de objetos verdes e vermelhos, a sua correta conversão para pontos no mundo real e chamada da movimentação ao manipulador, no entanto o vídeo demonstrativo apenas tem a deteção e movimentação de dois objetos, um de cada cor, visto que o manipulador mais vezes do que era esperado dava erro de *"Out of Reach!"* mesmo estando dentro dos limites do manipulador. Em certos casos sem movimentar o robô pela consola e reiniciando o controlador, o manipulador conseguia fazer a movimentação que antes retornava *"Out of Reach!"*, sendo um problema que não está ao nosso alcance de resolver.

Por fim, segue-se o vídeo demonstrativo de ambas as partes do projeto: `https://youtu.be/22uPStxwBQY`.

Qualquer problema que tenha com a submissão de código, tem o link para o github aqui: `https://github.com/FlipGoncalves/ArtificialVision-Robot`.