# Instituto Superior de Engenharia de Lisboa

Visão Artificial e Realidade Mista

Engenharia Informática e Multimédia

## Realidade Aumentada por Marcadores

Engº Pedro Jorge

MM2N

Arman Freitas nº45414

10 de junho de 2022

# Conteúdo

# Lista de Figuras

# 1 Introdução e Enquadramento

O presente projeto visa o desenvolvimento de uma aplicação de visão computacional, que implementa realidade aumentada por marcadores, tendo objetos virtuais alinhados aos marcadores do mundo real.

A realização da aplicação terá como auxílio a biblioteca *OpenCV* do *Python* que contém imensas funções que ajudam no processamento de imagens.

**Realidade aumentada baseada em marcadores:** Uma aplicação de realidade aumentada baseada em marcadores utiliza uma imagem alvo (marcadores), de modo a colocar objetos num determinado espaço. É necessário detetar os marcadores e,através destes determinar onde será posicionado o objeto virtual dentro do campo de visão do utilizador.

É de notar que os marcadores são um padrão físico, num ambiente do mundo real que conecta o mundo real com o virtual. Para o projeto utilizou-se a ferramenta *ArUco* encontrada dentro do *OpenCV*, de modo a facilitar a criação e deteção dos marcadores.

O desenvolvimento terá como base os seguintes segmentos:

- Calibração da câmara
- Deteção da pose
- Realidade aumentada
- Oclusão

Ao longo do projeto será possível verificar todo o processo, desde a calibração da câmara para a futura deteção de marcadores, posicionamento de objetos e, adicionar oclusão a pontos escondidos por detrás de outros.

# 2 Desenvolvimento

## 2.1 Calibração da Câmara

A calibração da câmara é o método de estimar os parâmetros da câmara.

Com todos os parâmetros e coeficientes da câmara, é possível determinar a relação entre um ponto 3D no mundo real e, a respetiva projeção 2D na imagem.

Para tal, é necessário obter os seguintes parâmetros:

- Parâmetros intrínsecos - Distância focal, centro ótico e os coeficientes de distorção da lente.
- Parâmetros extrínsecos - Vetores de rotação e translação da câmara de acordo com o sistema de coordenadas do mundo.
- Coeficientes de distorção - Coeficientes que definem a distorção presente na imagem.

**Parâmetros Intrínsecos:** A distância focal é $(f_x, f_y)$ e, o centro ótico $(c_x, c_y)$ presentes na matriz da câmara (parâmetros intrínsecos) com o seguinte aspeto:

$$Matriz = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 0 \end{bmatrix}$$

**Parâmetros Extrínsecos:** Já a matriz de parâmetros extrínsecos é composta pela matriz 3x3 de rotação $(R)$ e, a matriz 3x1 de translação $(t)$. Esta matriz apresenta-se da seguinte forma: $[R|t]$.

**Coeficientes de Distorção** É também possível remover alguma distorção presente na mesma. Podem existir dois tipos de distorção:

- Distorção Radial - Causa linhas retas aparecerem como curvas. Esta distorção aumenta à medida que os pontos se afastam do centro.

- Distorção Tangencial - Ocorre devido à lente não estar perfeitamente alinhada com o plano da imagem. Assim, algumas áreas da imagem podem parecer mais perto do que o esperado.

A distorção radial segue as seguintes equações, para o $x$ e o $y$, respetivamente:

$$x_{distorcido} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{distorcido} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Já a distorção tangencial segue os seguintes cálculos:

$$x_{distorcido} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{distorcido} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

Assim sendo, são necessários ao todo os seguintes coeficientes de distorção:

$$\text{Coeficientes de distorção} = (k_1, k_2, p_1, p_2, k_3)$$

Com estes coeficientes, é possível corrigir a distorção presente como demonstra a figura 1, como exemplo.

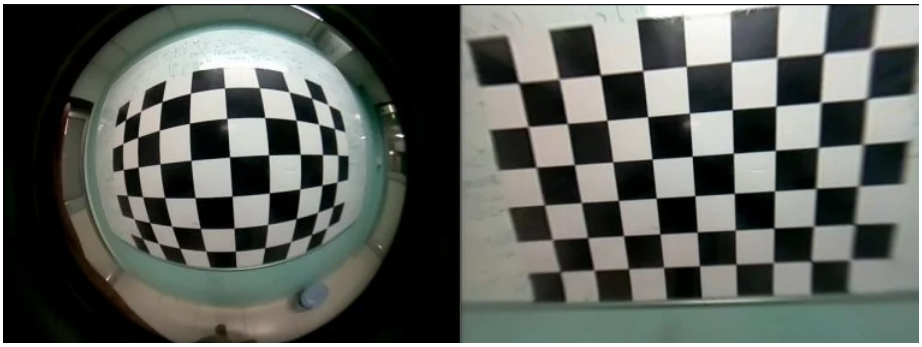

Figura 1: Efeito da correção numa imagem com distorção

No presente projeto foi utilizado o tabuleiro de damas para a calibração. A calibração requer o cálculo dos parâmetros da câmara por pontos 3D já conhecidos $(X_w, Y_w, Z_w)$ e os pixeis correspondentes $(u, v)$ na imagem.

Todos os pontos presentes no tabuleiro situam-se no mesmo plano, pelo que se pode dizer que $Z_w = 0$ em todos os pontos. Como os pontos estão todos espaçados igualmente, as coordenadas $(X_w, Y_w)$ são facilmente definidas pelo ponto (0, 0), sendo todos os pontos relativos a este.

Para a deteção dos cantos do tabuleiro foi utilizada a função *findChessboardCorners* do *OpenCV* e, para a detetar as coordenadas 2D dos pontos foi utilizada a função *cornerSubPix* também presente no *OpenCV*.

Foram tiradas várias fotografias do tabuleiro das quais a seguinte é uma delas:

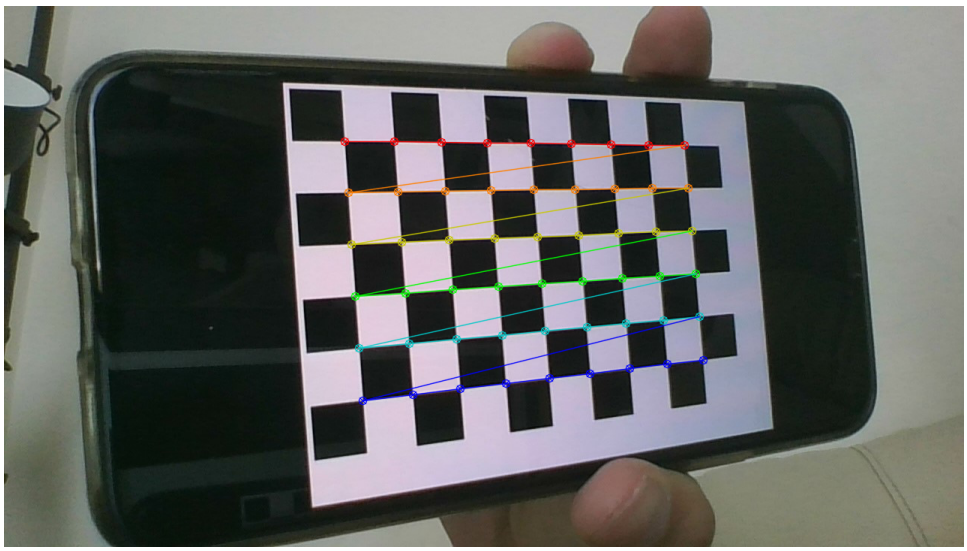

Figura 2: Exemplo de tabuleiro utilizado

Por fim, com os pontos guardados, apenas se utiliza a função *calibrateCamera* do *OpenCV* e são retornados tanto os coeficientes de distorção como os parâmetros intrínsecos e extrísecos da câmara, que serão guardados num ficheiro para serem utilizados nos próximos segmentos.

## 2.2 Deteção da Pose

Como explicado anteriormente, o projeto faz uso de marcadores para a realidade aumentada, mais precisamente *Aruco Markers*. Antes de prosseguir para o segmento de realidade aumentada, é necessário calcular onde é que os marcadores se situam na imagem.

A cada marcador pertence um id, pelo que é possível detetar onde se encontra o mesmo e qual o seu id. O processo de deteção consiste primariamente nos dois passos a seguir:

1. Deteção de possíveis marcadores. Aqui, é analisada a imagem toda em busca de formas tipo quadrados que sejam possíveis marcadores. Os contornos são extraídos e, os que não se assemelham a um quadrado são descartados. São também aplicados filtros extra que podem remover contornos muito pequenos/grandes ou se encontrem bastante juntos uns dos outros.

2. Após o passo anterior, são detetados os *bits* de cada marcador (quadrados pretos e brancos que constituem o padrão da imagem). Para tal, é necessária uma transformação da perspetiva e, divide-se a imagem em diferentes células, de acordo com o tamanho do marcador. Em cada célula são calculados o número de pixeis brancos e pretos de forma a determinar se a mesma é branca ou preta. Por fim, as células são analisadas para determinar o id a que o marcador pertence.

A figura 3 representa um marcador com as suas células assinaladas:

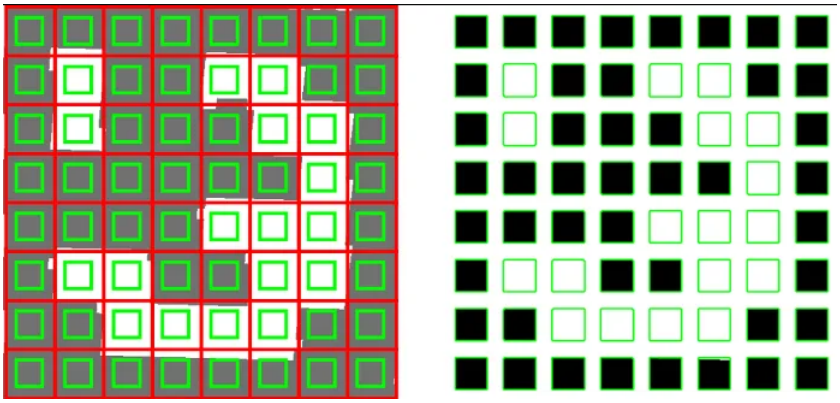

Figura 3: Marcador *Aruco*

## 2.3 Realidade Aumentada

### 2.3.1 Projeção de pontos na imagem

Para encontrar a projeção de uma coordenada 3D do mundo num plano 2D (imagem), é necessário aplicar transformações ao ponto para estar relativo ao sistema de coordenadas da câmara utilizando os parâmetros extrínsecos.

Após calculados os parâmetros, é finalmente possível projetar um ponto 3D na imagem. Para tal, obtém-se o seguinte vetor:

$$\begin{bmatrix} u' \\ v' \\ z' \end{bmatrix} = P \begin{bmatrix} X_w \\ Y'_w \\ Z'_w \\ 1 \end{bmatrix}$$

$X_w, Y_w, Z_w$ são as coordenadas do ponto a ser projetado e, P, a multiplicação da matriz de parâmetros intrínsecos (câmara) com a matriz de parâmetros intrínsecos.

Por fim, com o vetor acima calculado, é possível calcular a projeção $(u, v)$ na imagem, pelas seguintes equações: $u = \frac{u'}{w'}$ e $v = \frac{v'}{w'}$.

### 2.3.2 Adição de objetos virtuais nos planos

Com a possibilidade de projetar pontos 3D do mundo na imagem, é então possível projetar conjuntos de pontos que constituam um plano e, colocar objetos projetados nesses mesmos planos.

Para esta projeção, é primeiramente necessário do cálculo da matriz de homografia ($H$). Esta, consiste numa matriz 3x3 que consegue relacionar dois pontos de planos diferentes.

$$H = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}$$

A relação entre dois pontos de planos diferentes é a seguinte:

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = H \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

Para a obtenção da matriz de transformação de perspetiva ($H$), é utilizado o método *findHomography* presente no *OpenCV*. Este apenas recebe dois conjuntos de pontos para os dois planos a realizar a tranformação.

De seguida, é necessário aplicar o objeto virtual à imagem. Isto é realizado através da função *warpPerspective*, também do *OpenCV*, que muda a perspetiva da imagem para pertencer ao novo plano.

Por fim, apenas se soma a imagem modificada à imagem original e a imagem encontra-se no plano pretendido. É de notar que, para que a soma da imagem original com a imagem com o objeto resulte, é necessário colocar os pixeis do plano onde objeto se vai situar na imagem original a 0. Isto pois é

realizada a soma das duas imagens. Visto que se possuí os pontos pertencentes ao plano apenas se utiliza a função *fillConvexPoly* para preencher a área em questão com zeros.

## 2.4 Oclusão (*Z-Buffer*)

Foi também desenvolvido um modo de oclusão de pontos atrás de outros pontos desenhados. Esta técnica consiste na construção de um *z-buffer* que, como o próprio nome sugere, tem como propósito guardar as coordenadas $z$ de cada ponto, apenas pintando pontos onde o $z$ guardado no *buffer* seja menor que o do ponto a desenhar.

O *buffer* consiste numa estrutura de dados idêntica à de uma imagem vinda da câmara, mas onde cada pixel apenas tem um valor, ao contrário de 3 que uma imagem teria (RGB).

Os índices de cada ponto do *buffer* correspondem aos pixeis na imagem. A transformação de pontos do mundo para coordenadas na imagem foi determinada no segmento 2.3. No entanto, é necessário também converter coordenadas 3D do mundo para o referencial da câmara.

Esta mudança do referencial é crítica, devido a se estar a calcular a distância de cada ponto à câmara.

Para converter uma coordenada do mundo para a câmara, são utilizados apenas os parâmetros extrínsecos da câmara, ignorando a matriz da mesma. Assim, a conversão é dada por:

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = [R|t] \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$[R|t]$ é a matriz com os parâmetros intrínsecos da câmara (matriz de rotação e de translação), $(x, y, z)$ são as coordenadas do ponto a mudar de referencial e, $(x_1, y_1, z_1)$ são as coordenadas dos pontos no referencial da câmara.

# 3 Resultados

As seguintes figuras representam os resultados da aplicação a diferentes marcadores. Foram colocados 4 tipos diferentes de objetos virtuais, sendo na figura 4 (marcador de id 62) representando planos (através de pontos) com diferentes alturas de forma a validar o funcionamento do *z-buffer*.

A figura 6 e 7 represenam outros objetos construídos para diferentes id's, os marcadores de id 203 e 23, respetivamente. Já a figura 5 preenche todos os outros id's com uma pintura.

Por fim, na figura 8 é possível ver um conjunto de marcadores todos assinalados com o seu devido objeto virtual.

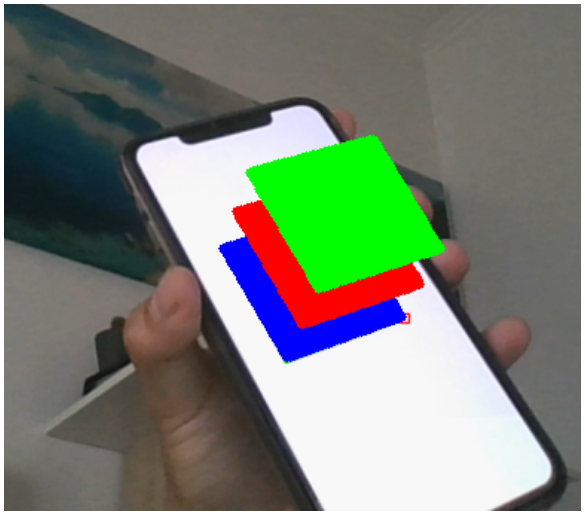

Figura 4: id = 62

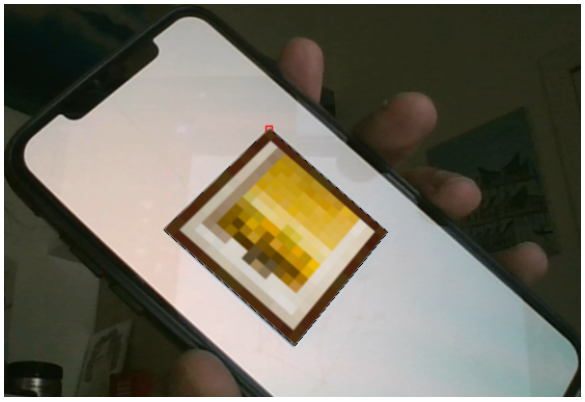

Figura 5: Todos os outros id's

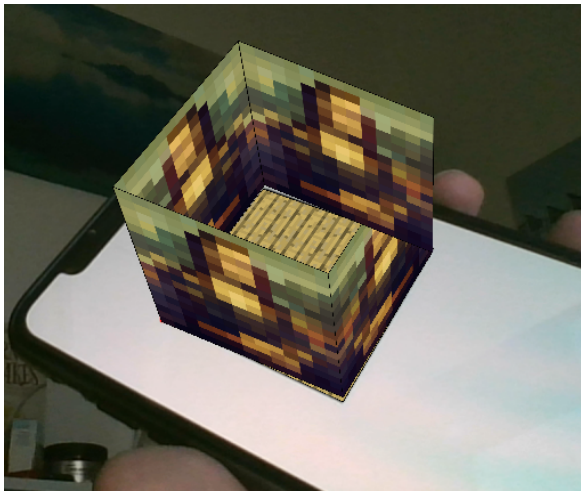

Figura 6: id = 203

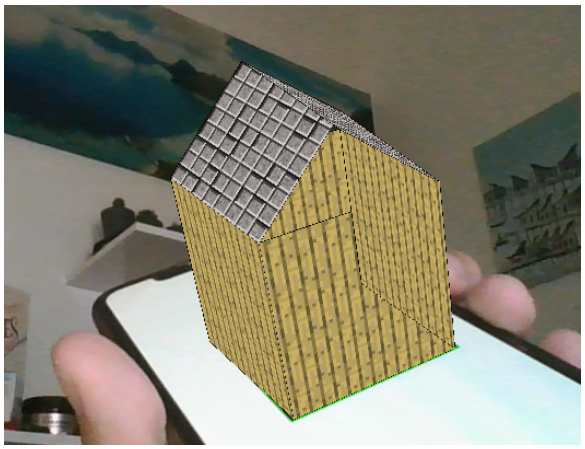

Figura 7: id = 23

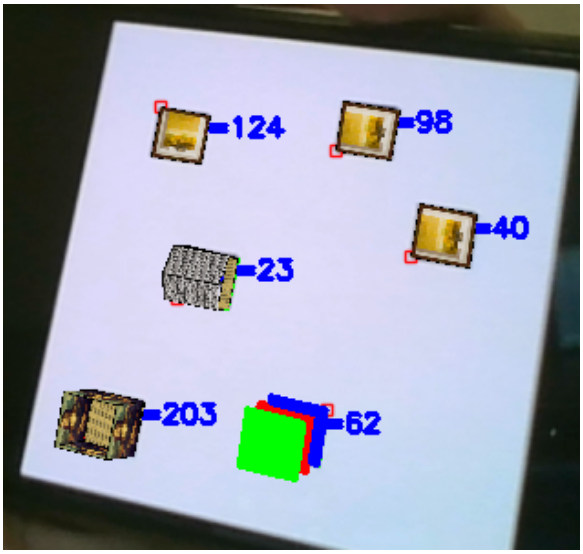


Figura 8: Conjunto de marcadores

# 4 Conclusões

Com a realização do projeto foi possível compreender o potencial deste ramo presente na realidade aumentada e, deveras despertar mais a curiosidade sobre esta área.

Um dos obstáculos que ocorreu durante a realização do *z-buffer* (implementação da oclusão) foi o facto dos pontos atrás de outros já estarem a ser escondidos. No entanto, o que estaria a acontecer era que, os pontos mais próximos da câmara estavam a ser pintados depois dos que estavam atrás (em todos os *frames*), sendo sempre mostrados por cima sempre. Ou seja, os últimos pontos a ser desenhados são os que ficam por cima e, coincidentemente, neste caso os pontos a ser ocludidos foram desenhados primeiro. Caso se trocasse a ordem, eram renderizados de ordem diferentes.

Após a realização do *z-buffer* resolveu-se trocar a ordem em que se pinta os pontos para que o facto de os estar a desenhar antes ou depois não interferisse, dando um resultado correto com o aspeto da figura 4 presente nos Resultados (3)

Considera-se ter atingido todos os objetivos determinados no inicio do projeto. O desenvolvimento foi sem dúvida complexo mas mais uma vez a biblioteca *OpenCV* tratou de imensas tarefas que, de outra forma deveriam ser realizadas manualmente. Muitas das transformações de coordenadas, sejam de mundo para câmara, ou de mundo para a imagem foram muito mais fáceis de implementar com esta biblioteca.

# 5 Bibliografia


- *Slides* fornecidos pelo docente
- https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html
- https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html