SERVIÇO PÚBLICO FEDERAL
UNIVERSIDADE FEDERAL DE SANTA CATARINA
***CAMPUS* BLUMENAU**
Rua João Pessoa, 2750, Velha, Blumenau – SC.
www.blumenau.ufsc.br/blumenau@contato.ufsc.br

VISÃO COMPUTACIONAL

Relatório A3

Gabriel Alves Silvestre

Ronaldo William Baggio de Oliveira

Blumenau

Junho, 2018

1 INTRODUÇÃO

O presente trabalho trata da descrição técnica do terceiro trabalho apresentado para a disciplina de Visão Computacional, neste documento será demonstrada parte do desenvolvimento teórico do trabalho.

O objetivo deste trabalho é elaborar um algoritmo capaz de estimar a posição de um tabuleiro de xadrez e com base nesta posição projetar a imagem de um cubo sobre este mesmo tabuleiro. Tal algoritmo deve calcular os parâmetros intrínsecos, calcular os parâmetros extrínsecos e encontrar os pontos

O trabalho está segmentado em cinco tópicos ao longo dos quais serão apresentadas uma breve contextualização, em seguida uma abordagem das metodologias utilizas bem como a lógica de funcionamento, por fim são apresentados os resultados dos testes e o fechamento é realizado com a conclusão.

2 CONTEXTUALIZAÇÃO

Diversas aplicações na visão computacional exigem conhecer e relacionar posição de uma imagem em relação a câmera na qual foi capturada, e para tanto é necessário encontrar o modelo completo da câmera, através da qual é possível descrever a relação entre a imagem capturada e a posição da câmera.

O modelo completo da câmera é composto por duas partes principais, a primeira parte descreve os parâmetros intrínsecos da câmera (parâmetros próprios da construção da câmera), obtidos através da calibração da câmera, a segunda parte descreve os parâmetros extrínsecos (parâmetros de posicionamento), obtidos através da posição de pontos conhecidos no objeto de interesse.

É importante estudar e analisar os parâmetros da câmera pois através dele é possível remover distorções, mensurar tamanho de objetos e até mesmo reconstruir cenários, mais do que isso, por meio deste processo é possível obter aplicações na robótica, para auxiliar na localização de robôs e a até mesmo realidade aumentada, para construir projeções e simulações em imagens do mundo real.

3 METODOLOGIAS UTILIZADAS

Ao analisar o objetivo do algoritmo, estimar a posição do tabuleiro de xadrez e construir projeções com base nesta posição a primeira a necessidade que fica evidente é encontrar os parâmetros da câmera, isto é, a matrizes que transformam pontos reais em pontos na imagem.

Os parâmetros da câmera podem ser divididos em intrínsecos e extrínsecos, o primeiro tipo tem relação com características construtivas da própria câmera como centro óptico e distância focal, enquanto o segundo tipo tem relação com a localização da câmera. Em síntese, os pontos do mundo são transformados em coordenadas da câmera por meio dos parâmetros extrínsecos, já as coordenadas da câmera são mapeadas no plano da imagem por meio dos parâmetros intrínsecos, a Fig.1 ilustra este processo:

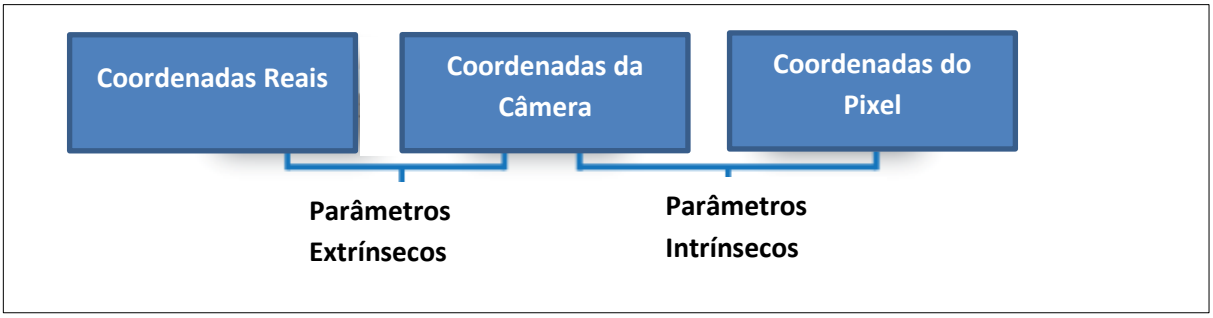

Fig.1 – Relação dos parâmetros da câmera e das coordenadas

Para estimar tais parâmetros, é preciso obter pontos reais 3-D e seus respectivos pontos na imagem 2-D, um dos métodos para obter essas correspondências é utilizar várias imagens com um padrão de calibração, neste caso um tabuleiro de xadrez. Sendo assim, para obter estes parâmetros é necessário:

i. Aplicar uma técnica para encontrar os parâmetros intrínsecos (internos)
ii. Aplicar uma técnica para encontrar os parâmetros extrínsecos (externos)
iii. Calcular a posição do tabuleiro com base nos parâmetros anteriores
iv. Desenhar Cubo em determinada posição do tabuleiro.

Atender o primeiro requisito (i) é possível através da função de calibração do Matlab, tal função recebe como entrada um conjunto de imagens (conjunto de calibração) do tabuleiro e o tamanho de cada quadrado do tabuleiro, após isto é capaz de extrair características intrínsecas da câmera como distância focal, por exemplo.

Esta função de calibração busca pelo padrão correspondente a quina do tabuleiro de xadrez, isto é, o ponto de contato entre os quadrados brancos e pretos, e com base neste padrão

cria marcadores. Após encontrar os marcadores em todas as imagens o algoritmo é capaz de calcular os parâmetros da câmera.

Atingir o segundo requisito (ii) é possível através da função *extrinsics* do Matlab, tal função recebe como parâmetros as coordenadas das quinas na imagem de interesse (imagem na qual será projetado o cubo), as coordenadas globais das quinas correspondentes as quinas na imagem e os parâmetros intrínsecos da câmera.

O terceiro requisito (iii) pode ser atendido através da função *cameraMatrix*, que calcula o modelo completo da câmera, para tanto esta função recebe como entradas os parâmetros extrínsecos e intrínsecos, e retorna a matriz capaz de transformar um ponto do mundo em ponto no plano da imagem.

Com a matriz completa da câmera é preciso definir as coordenadas dos pontos através dos quais o cubo será desenhado, no Matlab é possível desenhar um cubo através dos vértices, isto é, com os valores das posições dos oito vértices pode-se definir completamente um cubo. A Fig.2 apresenta os vértices de um cubo:

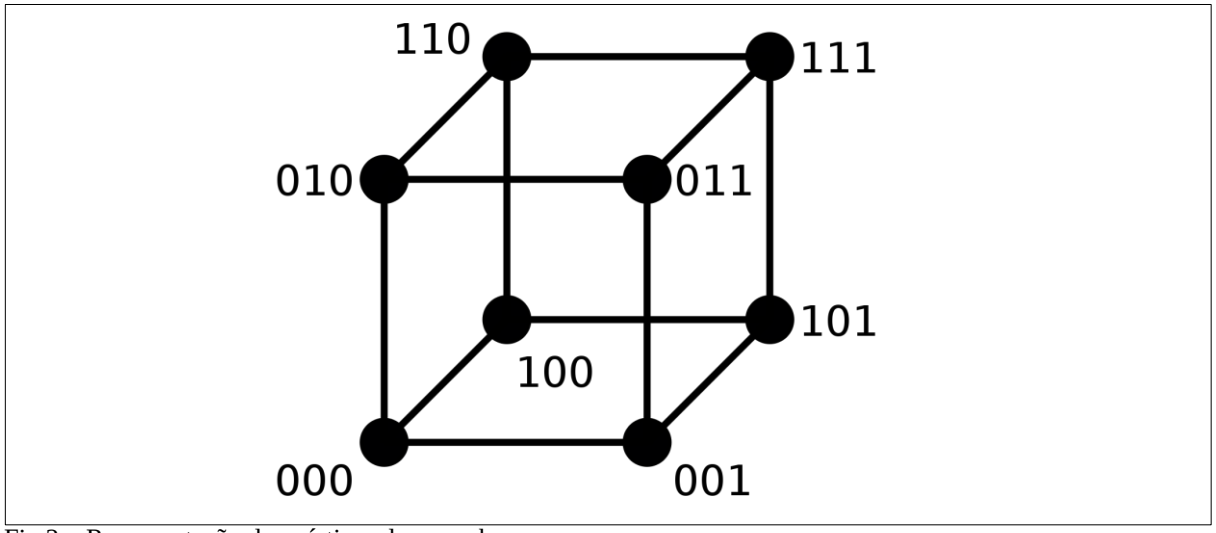

Fig.2 – Representação dos vértices de um cubo.

Com base na fig.2 fica evidente que ao definir um dos vértices é possível encontrar todos os demais definindo o tamanho da aresta do cubo, isto é, a distância entre um vértice e outro, e consequentemente esta distância (aresta) irá definir a tamanho do cubo. Sendo assim, de acordo com a Fig.2 o vértice de referência será o de número 110, e a partir dele serão definidas as posições dos demais.

Dentre as todas as funções do Matlab para projetar o cubo sobre o tabuleiro duas destacam-se pela facilidade ao utilizar: *plot* e *fill* , a função plot é mesma utilizada representar gráficos de funções 2D comuns, neste caso a função a ser representada são as arestas do cubo,

já a função *fill* tem por funcionalidade desenhar polígonos 2D, neste caso os polígonos serão as faces do cubo. A grande diferença entre tais funções é o *plot* fornece uma projeção apenas das arestas do cubo (cubo vazio), enquanto a função *fill* fornece as arestas e as faces do cubo (cubo preenchido).

3 LÓGICA DE FUNCIONAMENTO

Por meio das funções descritas anteriormente é possível elaborar o algoritmo para projetar um cubo sobre o tabuleiro de xadrez, tal algoritmo segue os seguintes passos:

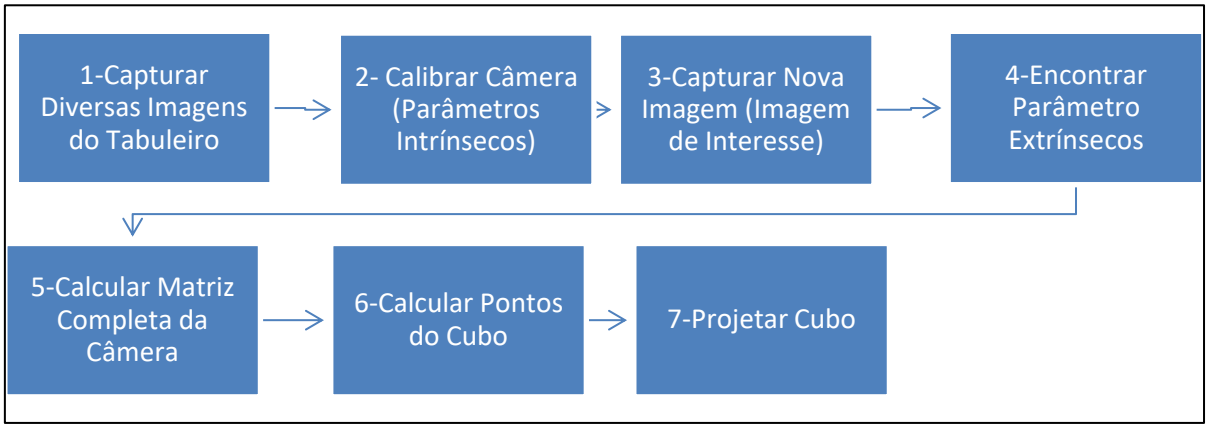

Fig.3 – Fluxograma do funcionamento do algoritmo.

Em primeiro lugar a função *detectCheckerboardPoints* deve ser aplicada para encontrar pontos de interesse nas imagens do conjunto de calibração (quinas), tal função retorna uma matriz as coordenadas de cada uma das quinas encontradas (*imagePoints*) e uma matriz com o número de quadrados (casas) em cada direção do tabuleiro. Na fig.2 é possível ver os pontos detectados, para tal imagem foram detectadas 54 quinas e 7 quadrados na vertical e 10 quadrados na horizontal (quadrados que aparecem apenas em parte também são computados).

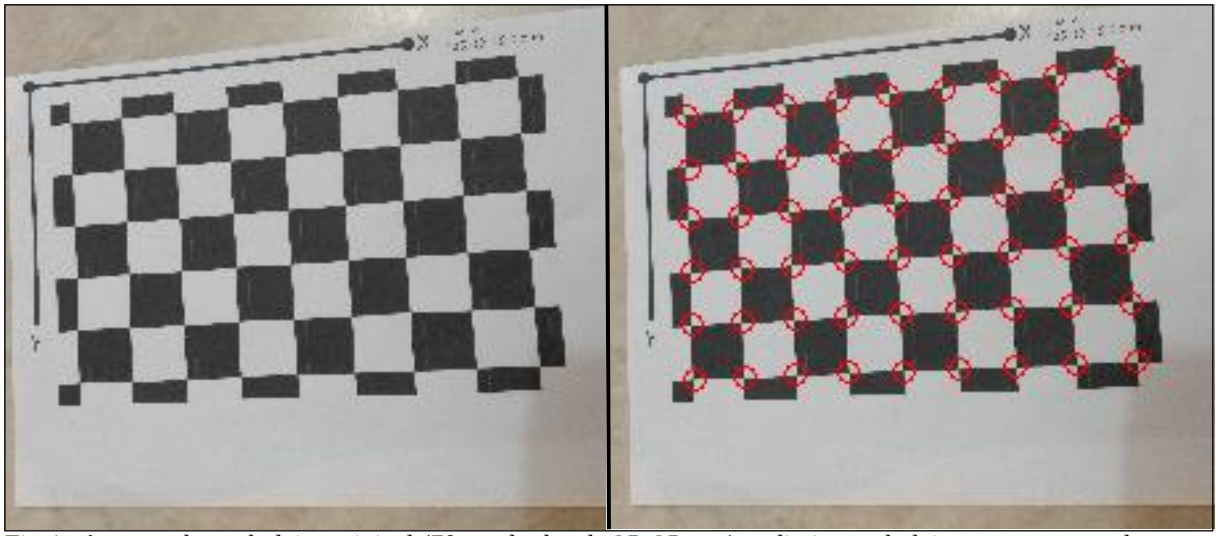

Fig.4 –A esquerda o tabuleiro original (70 quadrados de 25x25 mm), a direita o tabuleiro com os marcadores em vermelho.

Além de detectar os pontos na imagem, a função *detectCheckerboardPoints* organiza os *imagePoints* de acordo com um sistema de referência, onde o crescimento ocorre até a última quina do lado de menor tamanho, para em seguida passar para o lado de maior tamanho. Neste sistema de referência quina de número um equivale a coordenada (0,0), e a próxima coordenada irá depender do tamanho do quadrado do tabuleiro, neste caso como o tamanho de cada quadrado é de 25 mm a coordenada de número sete, por exemplo, estará na coordenada (25,25).

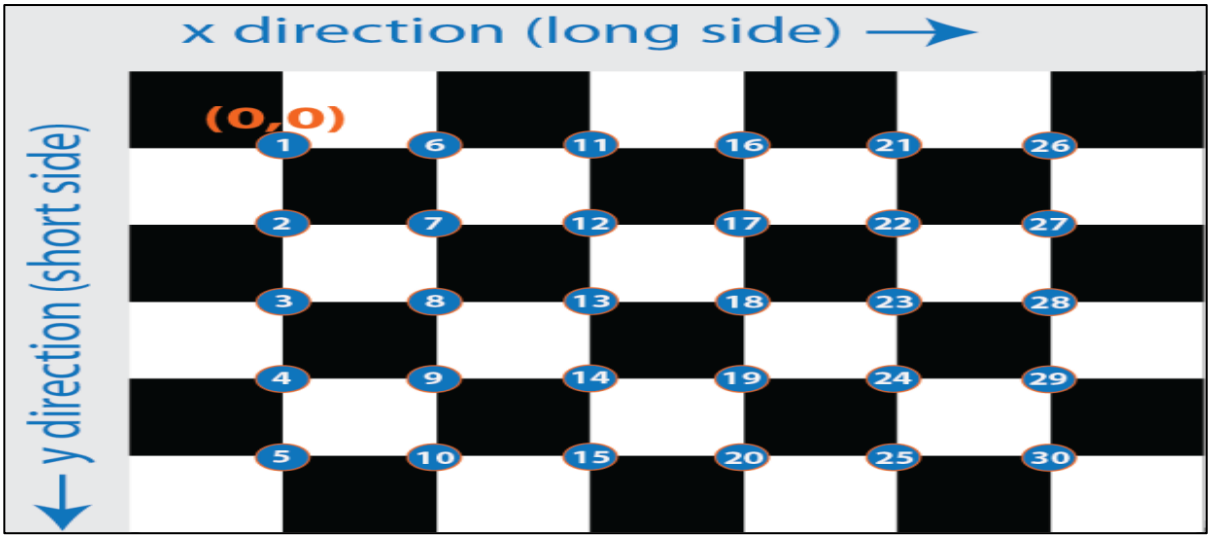

Fig.5 – Sistema de referência.

Em seguida, com a função *generateCheckerboardPoints* é criada uma matriz com as coordenadas reais das quinas do tabuleiro de xadrez (*worldPoints*), esta matriz tem o número de linhas equivalente ao número de quinas, enquanto que o número de colunas é igual a dois, onde cada coluna representa a posição (x, y) no eixo de coordenadas.

Com a matriz de coordenadas das quinas na imagem (*imagePoints*) e com a matriz de coordenadas reais (*worldPoints*) é possível estimar os parâmetros intrínsecos da câmera por meio da função *estimateCameraParameters*, que retorna um objeto do tipo *CamereParamenters* que armazena diversos parâmetros da câmera, entre eles os parâmetros intrínsecos.

As figuras 6 e 7 apresentam as várias posições nas quais foram obtidas imagens do tabuleiro, a função *estimateCameraParameters* utiliza estes diversos posicionamentos e as marcações das quinas para calcular os parâmetros internos:

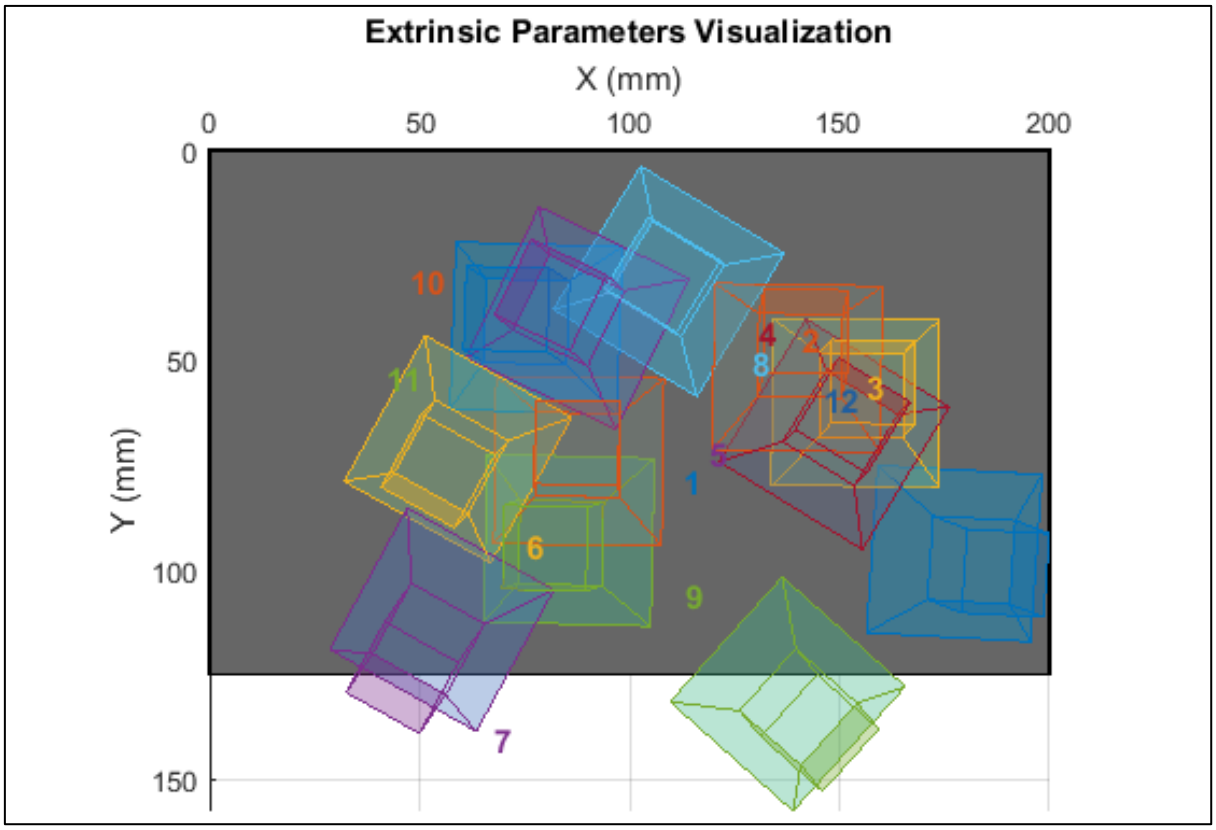

Fig. 6 – Posições da câmera em relação ao tabuleiro no conjunto de imagens de calibração.

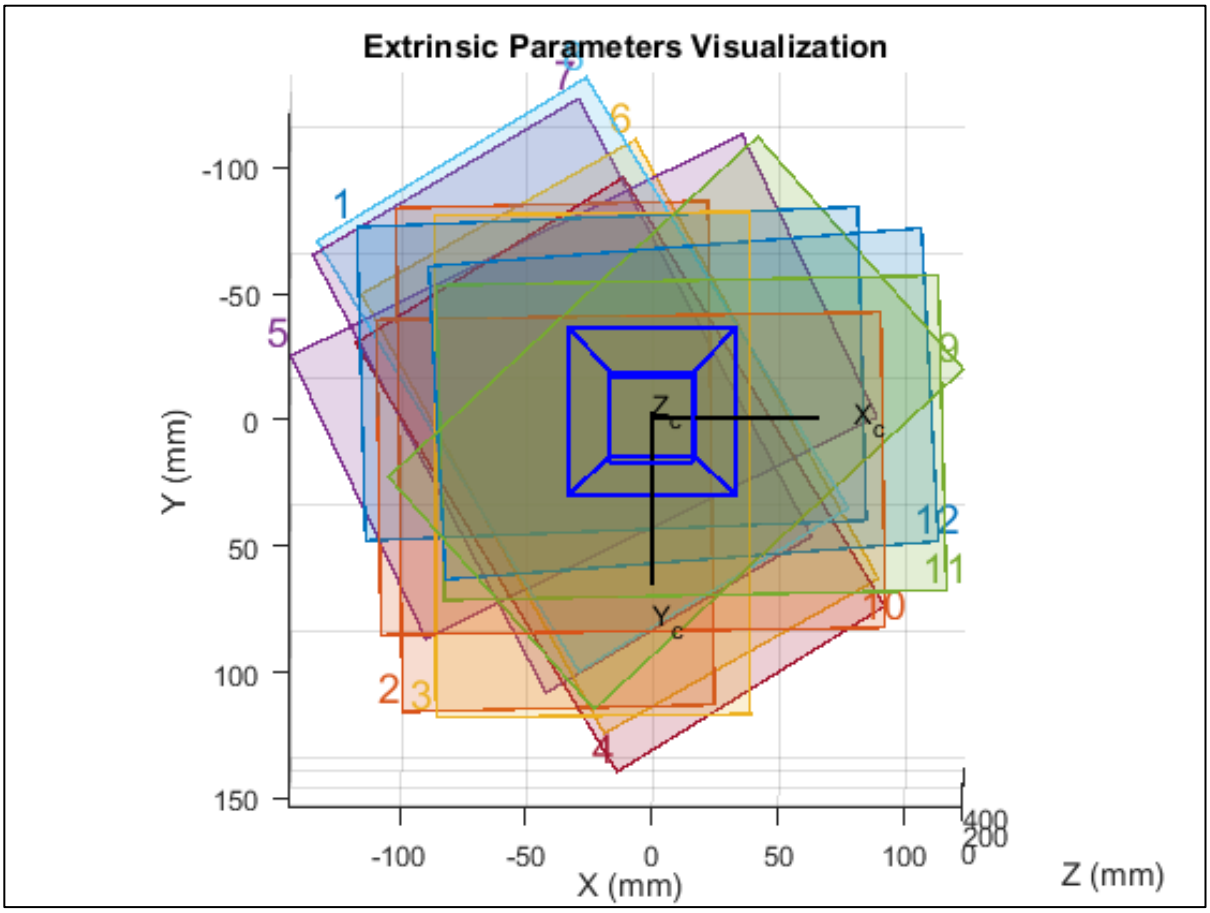

Fig.7 – Posições do tabuleiro em relação a câmera no conjunto de imagens de calibração.

Tendo conhecimento dos parâmetros intrínsecos (parâmetros de construção) da câmera é o momento de calcular os parâmetros extrínsecos, os parâmetros de construção serão os mesmos durante toda a execução do algoritmo, isto porque a câmera utilizada será a mesma, o que não é verdade para os parâmetros externos já que estes dependem da posição do objeto na imagem de interesse (imagem na qual será projetado o cubo) e esta posição pode ser alterada. Sendo assim, os parâmetros extrínsecos serão calculados toda vez que o algoritmo capturar uma nova imagem.

Antes de iniciar o cálculo dos parâmetros externos, deve-se aplicar a função *undistortImage* para corrigir a imagem das distorções causadas pela câmera, isto é possível pois através do processo anterior de calibração foram estimadas as distorções características da câmera utilizada.

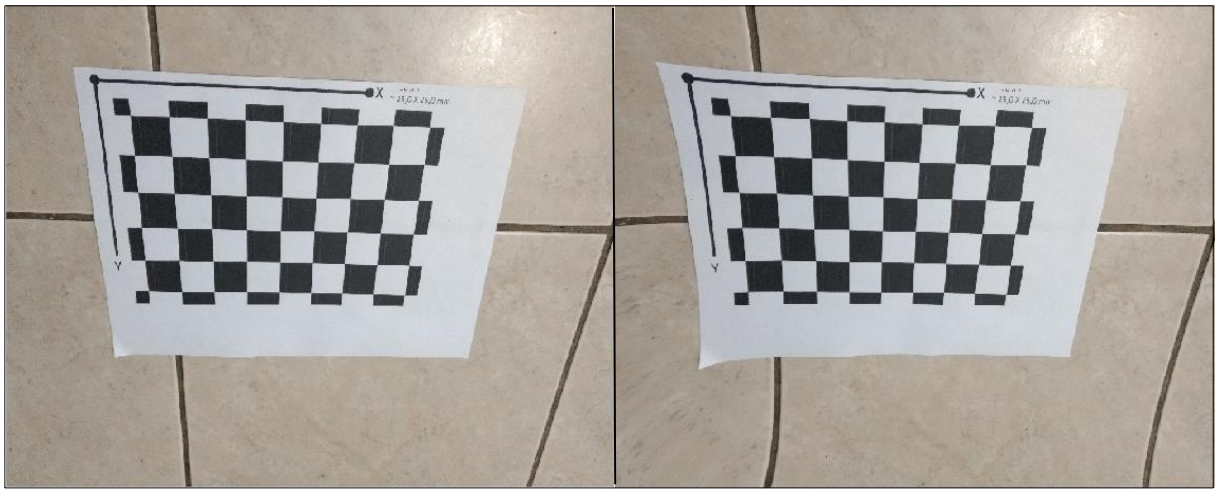

Fig.8 – A esquerda o tabuleiro antes da correção, a direita o tabuleiro após a correção.

O próximo passo para calcular os parâmetros externos é encontrar na imagem de interesse as coordenadas das quinas, *imagePoints*, e as coordenadas reais, *worldPoints* (que serão os mesmo que das imagens de calibração, já que o tabuleiro utilizado é o mesmo). Então com os *imagePoints*, os *worldPoints* e os parâmetros da câmera a função *extrinsics* pode ser aplicada, tal função retorna duas matrizes: a matriz de rotação e a matriz de translação.

Em seguida, para obter a matriz completa da câmera, capaz de transformar pontos da coordenada real para a coordenada do pixel, é necessário utilizar a função *cameraMatrix* que recebe como entrada os parâmetros intrínsecos (objeto do tipo *CameraParameters*) e extrínsecos (matrizes de rotação e translação encontradas anteriormente) da câmera, e como saída fornece uma matriz 4x3 capaz de transformar pontos 3D reais em coordenadas homogêneas na imagem.

Após isto basta definir a tamanho da aresta do cubo e a posição do vértice de referência, em seguida com as posições dos vértices basta multiplica-los pela matriz da câmera, a equação 1 apresenta a obtenção do vértice de referência:

$$Vértice\ 000\ (referência) = C * [x\ ;\ y;\ 0;\ 1] \quad (1)$$

Para obter o próximo vértice para somar o tamanho da aresta a uma das coordenadas, e realizar o mesmo procedimento para todos os demais vértices alternando a coordenada:

$$Vértice\ 001\ = C * [x + lado;\ y;\ 0;\ 1] \quad (2)$$

$$Vértice\ 101\ = C * [x + lado;\ y + lado;\ 0;\ 1] \quad (3)$$

$$Vértice\ 100\ = C * [x;\ y + lado;\ 0;\ 1] \quad (4)$$

$$Vértice\ 010\ = C * [x;\ y;\ lado;\ 1] \quad (5)$$

$$Vértice\ 011\ = C * [x + lado;\ y;\ lado;\ 1] \quad (6)$$

$$Vértice\ 111\ = C * [x + lado;\ y + lado;\ lado;\ 1] \quad (7)$$

$$Vértice\ 110\ = C * [x;\ y + lado;\ lado;\ 1] \quad (8)$$

Assim todos os vértices obtidos terão três valores em razão do espaço homogêneo, para converter os pontos da coordenada homogênea $(\check{x}, \check{y}, \check{z})$ em pontos na coordenada cartesiano (x, y) basta dividir as coordenadas dos eixos $\check{x}$ e $\check{y}$ pelo valor da coordenada no eixo $\check{z}$:

$$x = \frac{\check{x}}{\check{z}} \quad (9)$$

$$y = \frac{\check{y}}{\check{z}} \quad (10)$$

Por fim, basta utilizar as funções gráficas *plot* e *fill* para desenhar o cubo com as respectivas coordenadas x e y encontradas.

3 TESTES E RESULTADOS

Para avaliar o funcionamento do algoritmo foram realizados dois testes, o primeiro com a câmera de um celular e o segundo com a câmera do computador.

3.1 Câmera Celular

No primeiro conjunto as imagens foram obtidas através da câmera de um celular Moto G4 Plus, a seguir as imagens utilizadas para realizar a calibração são apresentadas:

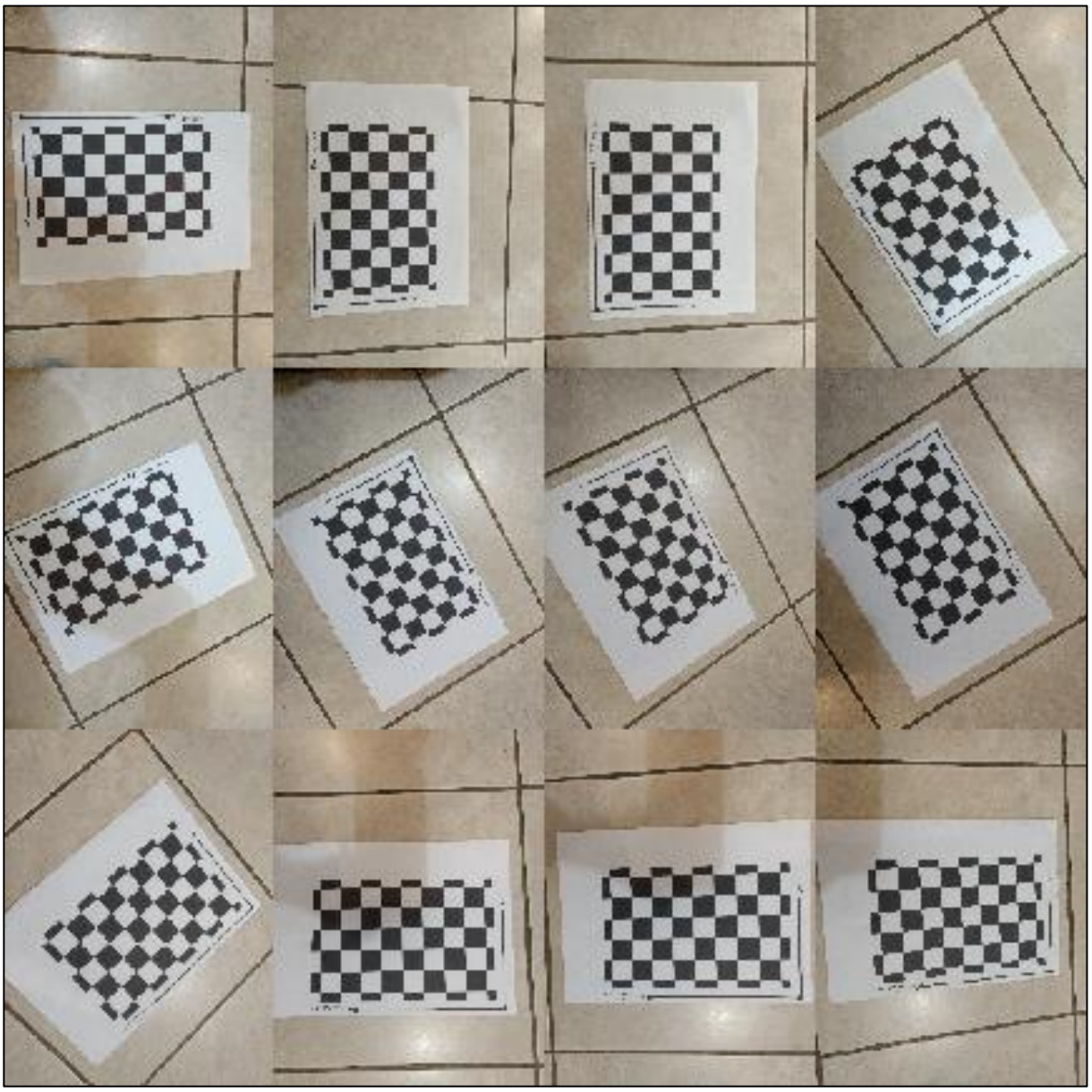

Fig.9 – Conjunto de imagens para calibração.

Após o processo de calibração com o conjunto de imagens anterior são obtidos parâmetros intrínsecos, a tabela I apresenta alguns deles:

Tabela I – Parâmetros Intrínsecos Câmera Celular

Parâmetro	***Valor***
Distância Focal	$[3.7050\text{x}10^3\ 3.7075\text{x}10^3]$
Coordenada do Ponto Principal	$[1.745\text{x}10^3\ 2.3115\text{x}10^3]$
Distorção Radial	$[0.1456\ -0.5117]$

O erro médio por imagem após o processo de calibração de aproximadamente 1.07 pixels, isto significa que a distância entre uma quina detectada e ponto projetado na mesma imagem é de 1.07 pixels. Este erro pode ser atenuado ao aumentar o número de imagens do conjunto de calibração e ao retirar do conjunto as imagens que possuem o maior erro.

Depois do processo de calibração para encontrar os parâmetros internos é o momento de encontrar os parâmetros externos, para tanto uma nova imagem do tabuleiro foi obtida (imagem que diferente das imagens do conjunto de calibração). A esta nova imagem o processo foi aplicado o processo de correção de distorção e em seguida foram detectadas as quinas:

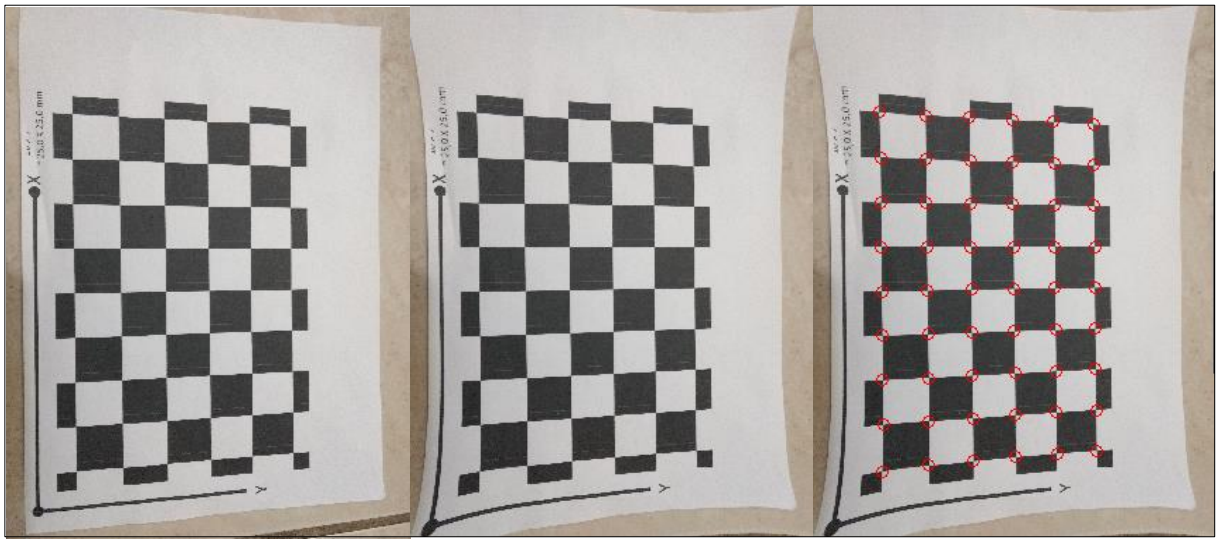

Fig.10 – A esquerda a imagem original, no centro a imagem após a correção e a direita a imagem corrigida com as quinas detectadas.

Com as quinas detectadas e os parâmetros intrínsecos é possível obter as matrizes de rotação e translação que representam os parâmetros extrínsecos:

$$Matriz\ de\ Rotação = \begin{bmatrix} -0.0101 & -0.9999 & -0.0033 \\ 0.9832 & -0.0106 & 0.1821 \\ -0.1821 & -0.0014 & 0.9833 \end{bmatrix}$$

$$Matriz\ de\ Translação = [-73.6903\ 110.903\ 241.91884]$$

Em seguida, com os parâmetros intrínsecos e extrínsecos obtêm-se a matriz completa da câmera, a seguir tal matriz (transposta) é apresentada:

$$Matriz\ da\ Câmera = 1x10^5 \begin{bmatrix} -0.0004 & 0.0396 & 0.0104 & 1.4919 \\ -0.0371 & 0.0038 & 0.0227 & 9.6921 \\ -0.0000 & 0.0000 & 0.0000 & 0.0024 \end{bmatrix}$$

Tendo conhecimento da matriz da câmera basta escolher a dimensão dos lados do cubo (aresta) e a posição do vértice de referência, com estas duas variáveis definidas e as equações dos vértices (equação 2 a 8) é possível definir cada um dos pontos. Ao definir posição do vértice de referência como zero nas coordenadas x e y, e o tamanho da aresta igual a 50mm:

$$p_1 = \begin{matrix} 0 \\ 0 \\ 0 \\ 1 \end{matrix} \quad p_2 = \begin{matrix} 50 \\ 0 \\ 0 \\ 1 \end{matrix} \quad p_3 = \begin{matrix} 50 \\ 50 \\ 0 \\ 1 \end{matrix} \quad p_4 = \begin{matrix} 0 \\ 50 \\ 0 \\ 1 \end{matrix} \quad p_5 = \begin{matrix} 0 \\ 0 \\ 50 \\ 1 \end{matrix} \quad p_6 = \begin{matrix} 50 \\ 0 \\ 50 \\ 1 \end{matrix} \quad p_7 = \begin{matrix} 50 \\ 50 \\ 50 \\ 1 \end{matrix} \quad p_8 = \begin{matrix} 0 \\ 0 \\ 50 \\ 1 \end{matrix}$$

Para encontrar os pontos na imagem basta multiplicar a matriz de cada ponto pela matriz da câmera:

$$p_1 = 1x10^5 \begin{bmatrix} 1.4919 \\ 9.6921 \\ 0.0024 \end{bmatrix} \quad p_2 = 1x10^5 \begin{bmatrix} 1.4702 \\ 7.8347 \\ 0.0024 \end{bmatrix} \quad p_3 = 1x10^5 \begin{bmatrix} 3.4506 \\ 8.0256 \\ 0.0025 \end{bmatrix} \quad p_4 = 1x10^5 \begin{bmatrix} 3.4722 \\ 9.8830 \\ 0.0025 \end{bmatrix}$$

$$p_5 = 1x10^5 \begin{bmatrix} 0.2013 \\ 1.0826 \\ 0.0003 \end{bmatrix} \quad p_6 = 1x10^5 \begin{bmatrix} 1.9909 \\ 8.9686 \\ 0.0029 \end{bmatrix} \quad p_7 = 1x10^5 \begin{bmatrix} 3.9712 \\ 9.195 \\ 0.0030 \end{bmatrix} \quad p_8 = 1x10^5 \begin{bmatrix} 0.3993 \\ 1.1017 \\ 0.0003 \end{bmatrix}$$

Então com os valores dos pontos e a funções de *plot* e *fill* o cubo é projetado sobre o tabuleiro:

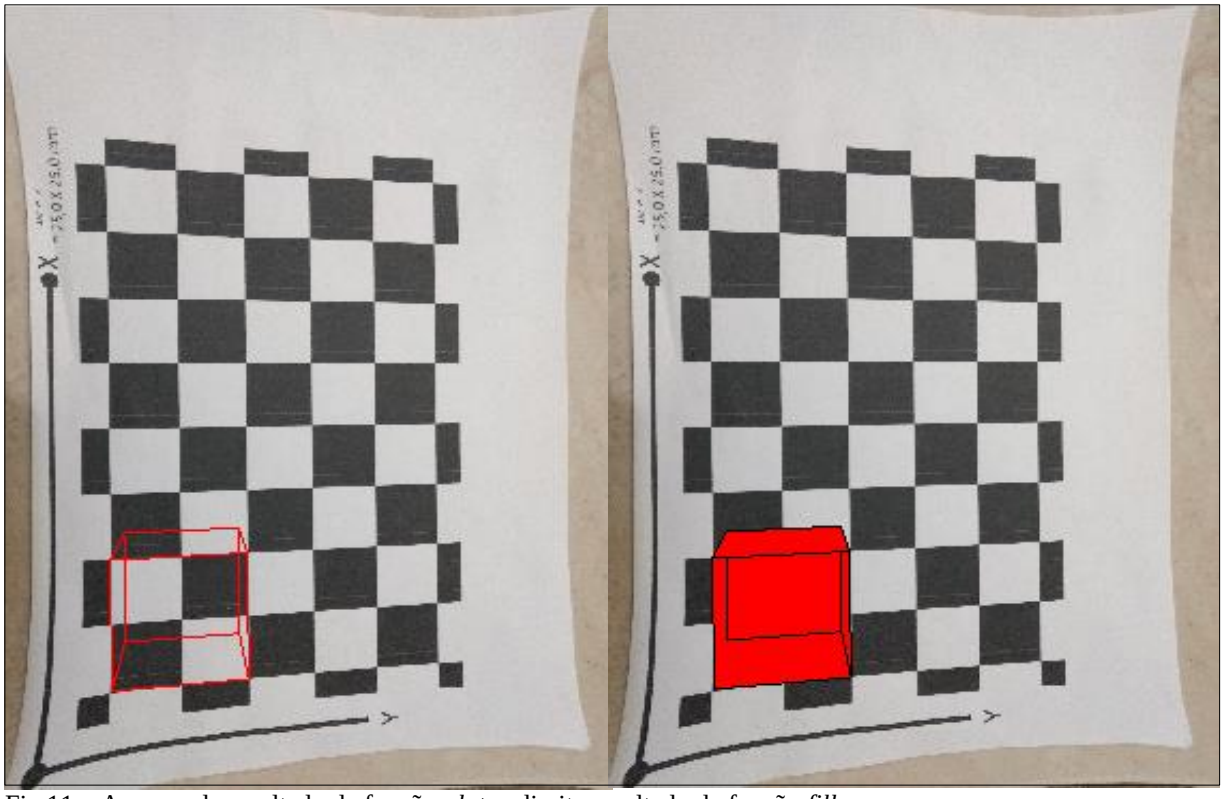

Fig.11 – A esquerda resultado da função *plot*, a direita resultado da função *fill*.

Ao examinar os resultados obtidos para ambas as funções é possível ver que um dos vértices do cubo está localizada na quina de número que equivale as coordenadas zero do sistema de referência do tabuleiro, que é exatamente a posição que foi definido para o vértice de referência. Em outras palavras a matriz da câmera, obtida através dos parâmetros intrínsecos e extrínsecos, é capaz de calcular os pontos na posição correta.

3.2 Câmera Computador

No segundo conjunto as imagens foram obtidas através da câmera da câmera de um computador Semp Toshiba NI 1401, a seguir as imagens utilizadas para realizar a calibração são apresentadas:

Após o processo de calibração com o conjunto de imagens anterior são obtidos parâmetros intrínsecos, a tabela II apresenta alguns deles:

Tabela II – Parâmetros Intrínsecos Câmera Computador

Parâmetro	*Valor*
Distância Focal	[995.7937 995.7748]
Coordenada do Ponto Principal	[622.5543 417.1410]
Distorção Radial	[0.0354 −0.1508]

O erro médio por imagem após o processo de calibração de aproximadamente 1.2024pixels, isto significa que a distância entre uma quina detectada e ponto projetado na mesma imagem é de 1.2024pixels. Este erro pode ser atenuado ao aumentar o número de imagens do conjunto de calibração e ao retirar do conjunto as imagens que possuem o maior erro.

Depois do processo de calibração para encontrar os parâmetros internos é o momento de encontrar os parâmetros externos, para tanto uma nova imagem do tabuleiro foi obtida (imagem que diferente das imagens do conjunto de calibração). A esta nova imagem o processo foi aplicado o processo de correção de distorção e em seguida foram detectadas as quinas:

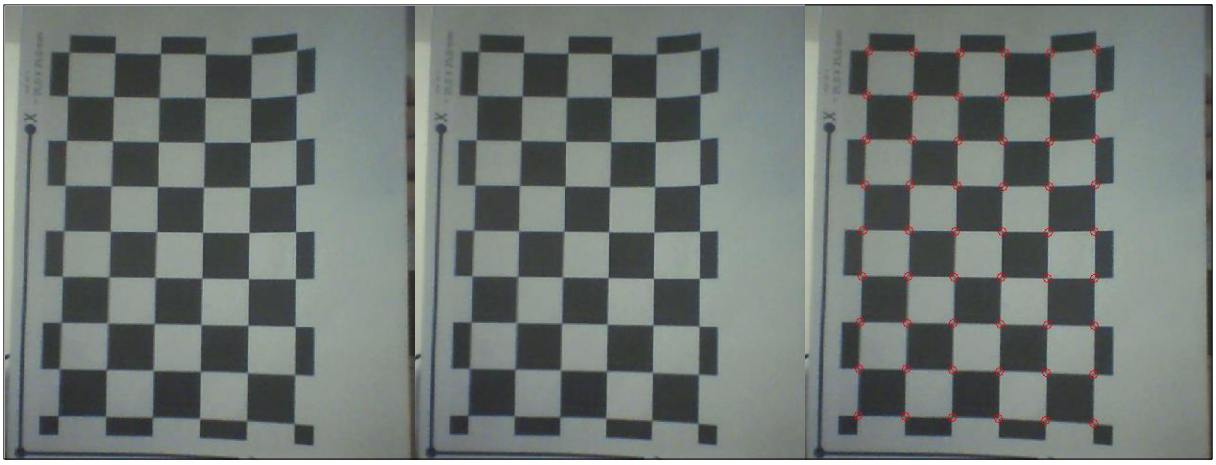

Fig.12 – A esquerda a imagem original, no centro a imagem após a correção e a direita a imagem corrigida com as quinas detectadas.

Com as quinas detectadas e os parâmetros intrínsecos é possível obter as matrizes de rotação e translação que representam os parâmetros extrínsecos:

$$Matriz\ de\ Rotação = \begin{bmatrix} 0.0147 & -0.9973 & 0.0714 \\ 0.9988 & 0.0113 & -0.0475 \\ 0.0466 & 0.0720 & 0.9963 \end{bmatrix}$$

$$Matriz\ de\ Translação = [-91.3815\ 79.0630\ 338.7952]$$

Em seguida, com os parâmetros intrínsecos e extrínsecos obtêm-se a matriz completa da câmera, a seguir tal matriz (transposta) é apresentada:

$$Matriz\ da\ Câmera = 1x10^5 \begin{bmatrix} 0.0006 & 0.0097 & 0.0067 & 1.1992 \\ -0.0096 & -0.0001 & 0.0049 & 2.200 \\ 0.0000 & 0.0000 & 0.0000 & 0.0034 \end{bmatrix}$$

Tendo conhecimento da matriz da câmera basta escolher a dimensão dos lados do cubo (aresta) e a posição do vértice de referência, com estas duas variáveis definidas e as equações dos vértices (equação 2 a 8) é possível definir cada um dos pontos. Ao definir posição do vértice de referência como 50 nas coordenadas x e y, e o tamanho da aresta igual a 50mm:

$$p_1 = \begin{matrix} 0 \\ 0 \\ 0 \\ 1 \end{matrix} \quad p_2 = \begin{matrix} 50 \\ 0 \\ 0 \\ 1 \end{matrix} \quad p_3 = \begin{matrix} 50 \\ 50 \\ 0 \\ 1 \end{matrix} \quad p_4 = \begin{matrix} 0 \\ 50 \\ 0 \\ 1 \end{matrix} \quad p_5 = \begin{matrix} 0 \\ 0 \\ 50 \\ 1 \end{matrix} \quad p_6 = \begin{matrix} 50 \\ 0 \\ 50 \\ 1 \end{matrix} \quad p_7 = \begin{matrix} 50 \\ 50 \\ 50 \\ 1 \end{matrix} \quad p_8 = \begin{matrix} 0 \\ 0 \\ 50 \\ 1 \end{matrix}$$

Para encontrar os pontos na imagem basta multiplicar a matriz de cada ponto pela matriz da câmera:

$$p_1 = 1x10^5 \begin{bmatrix} 1.7112 \\ 1.7146 \\ 0.0034 \end{bmatrix} \quad p_2 = 1x10^5 \begin{bmatrix} 1.7703 \\ 0.7512 \\ 0.0035 \end{bmatrix} \quad p_3 = 1x10^5 \begin{bmatrix} 2.7353 \\ 0.7426 \\ 0.0034 \end{bmatrix} \quad p_4 = 1x10^5 \begin{bmatrix} 2.6762 \\ 1.7060 \\ 0.0034 \end{bmatrix}$$

$$p_5 = 1x10^5 \begin{bmatrix} 2.3779 \\ 2.2019 \\ 0.0044 \end{bmatrix} \quad p_6 = 1x10^5 \begin{bmatrix} 2.4370 \\ 1.2385 \\ 0.0045 \end{bmatrix} \quad p_7 = 1x10^5 \begin{bmatrix} 3.4020 \\ 1.2299 \\ 0.0044 \end{bmatrix} \quad p_8 = 1x10^5 \begin{bmatrix} 3.3429 \\ 2.1933 \\ 0.0043 \end{bmatrix}$$

Então com os valores dos pontos e a funções de *plot* e *fill* o cubo é projetado sobre o tabuleiro:

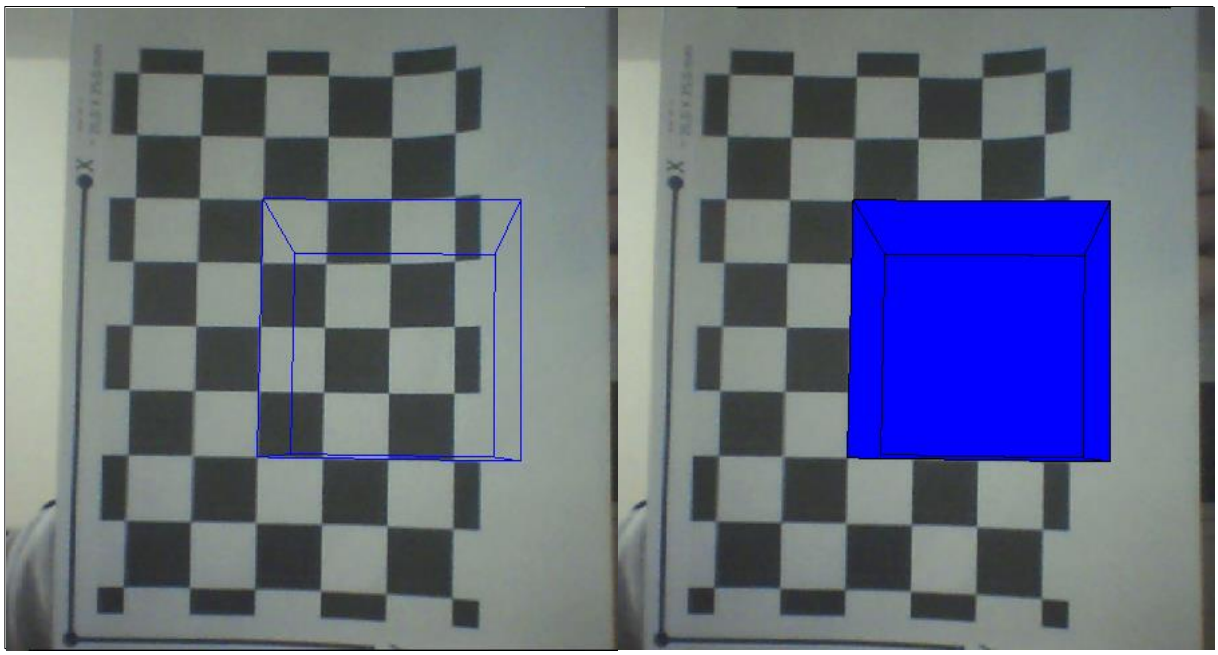

Fig.13 – A esquerda resultado da função *plot*, a direita resultado da função *fill*.

Ao examinar os resultados obtidos para ambas as funções é possível ver que um dos vértices do cubo está localizada na quina de número que equivale as coordenadas (50,50) do sistema de referência do tabuleiro, que é exatamente a posição que foi definido para o vértice de referência. Novamente a matriz da câmera, obtida através dos parâmetros intrínsecos e extrínsecos, é capaz de calcular os pontos na posição correta.

3.3 Algoritmo desenvolvido

O algoritmo desenvolvido bem como o conjunto de imagens e uma breve explicação podem ser encontrados no GitHub através do link:

- https://github.com/krausloko/TrabalhoA3

7 CONCLUSÃO

Depois de concluir todas as etapas desde a obtenção das imagens ao desenho do buco é possível relatar os pontos principais do trabalho. Ao avaliar diretamente o objetivo do trabalho, que era desenhar um cubo sobre o tabuleiro independente da posição da câmera, obteve-se um bom resultado.

O primeiro ponto a ser avaliado é a eficiência da calibração, ou seja, o quanto próximo da realidade estão os valores encontrados pelo processo de calibração, tanto nos testes com o celular quanto nos testes com o computador foi possível alcançar um erro de aproximadamente um pixel, valor completamente aceitável e que poderia ser melhorado retirando as imagens com maior erro ou inserindo mais imagens nos conjuntos.

Outro ponto a ser avaliado é a obtenção dos parâmetros externos e da consequente matriz completa da câmera, e ao perceber que o desenho do cubo foi projetado no lugar desejado fica claro que a tanto os parâmetros externos quanto a matriz completa da câmera representam com fidelidade a câmera.

Além disso, ao utilizar duas câmeras diferentes fica evidente a funcionalidade do processo de calibração, ou seja, é possível alterar o tipo de câmera utilizado se for realizado uma calibração correta.

O grande problema encontrado ao longo da elaboração do trabalho foram a limitação da função *detectCheckerboardPoints* para detectar as quinas, que em certas posições do tabuleiro não é capaz de encontrar todas as quinas, e os erros do processo de calibração, que dependendo do conjunto de imagem de calibração apresentar um erro elevado que irá prejudicar o cálculo correta da posição do cubo.

Por fim, ao reunir todas as características o algoritmo mostrou ser capaz de localizar o tabuleiro e projetar o cubo em uma posição determinada.

REFERÊNCIAS

MATHWORKS. **Camera calibrator**. Disponível em: <what is camera calibration?>. Acesso em: 21 jun. 2018.

MATHWORKS. **Cameraparameters**. Disponível em: <https://www.mathworks.com/help/vision/ref/cameraparameters.html>. Acesso em: 21 jun. 2018.

MATHWORKS. **Estimatecameraparameters**. Disponível em: <https://www.mathworks.com/help/vision/ref/estimatecameraparameters.html>. Acesso em: 21 jun. 2018.

MATHWORKS. **Evaluating the accuracy of single camera calibration**. Disponível em: <https://www.mathworks.com/help/vision/examples/evaluating-the-accuracy-of-single-camera-calibration.html>. Acesso em: 21 jun. 2018.

MATHWORKS. **Single camera calibrator app**. Disponível em: <https://www.mathworks.com/help/vision/ug/single-camera-calibrator-app.html>. Acesso em: 21 jun. 2018.

MATHWORKS. **What is camera calibration?**. Disponível em: <https://www.mathworks.com/help/vision/ug/camera-calibration.html>. Acesso em: 21 jun. 2018.

OPENCV. Basic concepts of the homography explained with code. Disponível em: <https://docs.opencv.org/3.4.1/d9/dab/tutorial_homography.html>. Acesso em: 11 abr. 2018.

PETER CORKE. Machine vision toolbox. Disponível em: <http://petercorke.com/wordpress/toolboxes/machine-vision-toolbox>. Acesso em: 09 abr. 2018.