

# **Universidade Federal do ABC**

## **Relatório - Relatório n.3**

**Nome: Rafael Costa Fernandes**

**RA: 21201920754**

### **Resumo**

Neste relatório serão discutidos métodos de calibração de câmeras, correção de distorção, determinação de posição e atitude de objetos e técnicas de visão estereoscópica, os resultados obtidos para a calibração, correção de distorção e determinação de posição e atitude foram satisfatórios, enquanto os resultados de visão estereoscópica não foram ideais.

# Introdução

Os assuntos discutidos neste relatório são extensos, e serão divididos em tópicos. Nestes tópicos serão abordados alguns usos da câmera como um sensor de posição, atitude e profundidade e para todas estas tarefas é necessário que a camera esteja corretamente calibrada.

## Calibração de Câmeras

Segundo [1] a calibração de uma câmera é um passo necessário na visão computacional em três dimensões, a fim de extrair informações métricas de imagens de duas dimensões. O processo de calibração é o ato de determinar as características ópticas e geométricas de uma camera. Estes parâmetros são chamados de parâmetros intrínsecos. A partir desta calibração também é possível determinar a posição e orientação da câmera em um ambiente de três dimensões, relativos a um sistema de coordenadas, que são chamados de parâmetros extrínsecos [2].

A técnica de calibração de câmeras utilizada neste trabalho é baseada no princípio que uma câmera têm os seguintes parâmetros: Uma matriz de parâmetros intrínsecos e cinco parâmetros de distorção. A matriz de parâmetros intrínsecos pode ser escrita como

$$F = [[f_x, 0, c_x], [0, f_y, c_y], [0, 0, 1]]$$

Enquanto os parâmetros de distorção podem ser escritos como:

$$dis\_coeff = [k_1, k_2, p_1, p_2, k_3]$$

Os parâmetros  $f_x$  e  $f_y$  são as distâncias focais das lentes, enquanto  $c_x$  e  $c_y$  são os centros focais expressados em coordenadas de pixels. Os parâmetros  $k_1$ ,  $k_2$  e  $k_3$  são coeficientes de distorção radial das lentes, enquanto os coeficientes  $p_1$  e  $p_2$  são os coeficientes de distorção tangencial das lentes [3].

## Determinação de Posição e Atitude

Com uma camera calibrada podemos determinar a posição e atitude de um objeto resolvendo um problema de ponto e perspectiva, que pode ser exemplificado com a figura [1].



Figura [1] - Problema de Ponto e Perspectiva (PnP)

Segundo [4], dado uma projeção em perspectiva de três pontos de um triângulo conhecido em um espaço com três dimensões, é possível determinar a posição de cada um destes vértices. Este problema é importante na área de visão computacional, visto que têm uma variedade de aplicações, como a determinação de localidades no espaço, dado um conjunto de paisagens artificiais.

## Técnicas de visão estereoscópica

Visão estéreo é uma necessidade para problemas em que a profundidade é uma informação importante. Um exemplo é [5], onde a visão estéreo é utilizada para evitar colisões em sistemas autônomos quadrirotóres.



### Figura [2] - Exemplo de um sistema de visão estereoscópica [5]

Um sistema de sensoriamento por visão estereoscópica tem por princípio gerar mapas de profundidade, informando a profundidade dos objetos na cena em relação a câmera. Existem vários algoritmos de determinação de profundidade, neste trabalho será abordado o algoritmo SGBM, abreviação de Stereo Processing by Semi Global Matching (em português: Processamento Estéreo por Correspondências Semi

## Metodologia

Os resultados apresentados foram obtidos utilizando as webcams *Logitech C270* e *Logitech C920*. Os parâmetros de hardware da webcam podem ser observados na tabela abaixo.

	<b>C270</b>	<b>C920</b>
Resolução Máxima	720p	1080p
Distância Focal	3.6 mm	3.6 mm
Tamanho de Sensor	3.6mm x 2mm	4.8mm x 3.6mm

O computador utilizado tem um processador Intel i5-4460 e 16 Gb de memória RAM. Em todos os casos e em ambas câmeras a resolução utilizada foi de 640x480 pixels.

## Resultados

Os resultados obtidos dos exercícios propostos podem ser observados abaixo. Todos os códigos estão devidamente comentados para uma melhor compreensão. Os exercícios propostos foram:

1. Realize a calibração de sua própria câmera. Para isso, faça um programa que capture com sua câmera 15 imagens do tabuleiro padrão. Obtenha e salve a matriz da câmera e os coeficientes de distorção. Apresente lado a lado uma imagem original e a imagem correspondente após a correção de distorção.
2. Realize o desenho dos eixos coordenados X, Y e Z em suas próprias imagens do tabuleiro de calibração. Considere a matriz de caracterização da sua câmera bem como os coeficientes de distorção obtidos no exercício anterior. Apresente os resultados obtidos.
3. Obtenha a Matriz Fundamental (F) e a Matriz Essencial (E) para um par de suas câmeras próprias, e realize o desenho das epilinhas num par de imagens destas câmeras.
4. Obtenha o mapa de disparidade para o par de imagens próprias obtidas no item 3.

## Exercício 1

Para este exercício primeiramente foi escrito um algoritmo de captura de imagens pela webcam, chamado de camera\_capture. Este algoritmo mantem uma janela com visualização em tempo real da imagem da câmera. O usuário pode apertar a tecla 's' para a captura de uma foto e a tecla 'q' para sair do algoritmo. O algoritmo finaliza automaticamente quando o usuário captura quinze imagens com as duas câmeras conectadas.

Podemos observar as imagens obtidas no algoritmo de detecção de vértices do tabuleiro de xadrez, para ambas as câmeras utilizadas no experimento. Ambos os resultados foram satisfatórios, detectando os vértices corretamente e também na ordem correta. A taxa de detecção do tabuleiro foi de 87% para a C270 e 93% para a C920.

As matrizes intrínseca e de distorção das câmeras podem ser observados na saída do algoritmo. Os valores de calibração obtidos são salvos para utilização em outros algoritmos.

In [24]:

```

import numpy as np
import cv2 as cv
import glob
from matplotlib import pyplot as plt
"""

INF209B - TÓPICOS ESPECIAIS EM PROCESSAMENTO DE SINAIS:

```

*VISÃO COMPUTACIONAL*

*PRÁTICA 04*

*RA: 21201920754*

*NOME: RAFAEL COSTA FERNANDES*

*E-MAIL: COSTA.FERNANDES@UFABC.EDU.BR*

*Descrição:*

*Exercício n.1*

*Algoritmo de calibração de câmera baseado em detecção de padrão xadrez.*

*O padrão utilizado tem 9 x 6 vértices internos (totalizando 10x7 "quadrados").*

*Para que a calibração seja boa, tirar a maior quantidade de fotos possível (pelo menos 10), com diversas poses do padrão xadrez.*

*Utilizar o padrão em uma superfície plana e rígida.*

*Fonte parcial do código: [https://docs.opencv.org/master/dc/dbb/tutorial\\_py\\_calibration.html](https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html).*

"""

```

plt.close('all')
# critério de finalização do algoritmo de subpixel
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 120, 0.0001)

# preparação dos pontos do objeto (tabuleiro de xadrez)
x_col = 9
x_lin = 6
objp = np.zeros((x_col*x_lin,3), np.float32)
objp[:, :, 2] = np.mgrid[0:x_col, 0:x_lin].T.reshape(-1, 2)

# Inicialização das listas
objpoints = [] # Pontos do objeto no mundo 3D
imgpoints = [] # Pontos do objeto no plano da imagem

cameras = ['c270', 'c920']
for camera in cameras:
    caminho = './imagens/' + camera + '/*.jpg'
    propriedades = './camera_properties/' + camera + '.npz'
    caminho_salva = './imagens/' + camera + '/resultado_calibr.png'
    images = glob.glob(caminho)
    for i, fname in enumerate(images):
        img = cv.imread(fname)
        gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
        # Acha os vértices do tabuleiro de xadrez
        ret, corners = cv.findChessboardCorners(gray, (x_col, x_lin), None)
        # If found, add object points, image points (after refining them)
        if ret == True:
            objpoints.append(objp)
            corners2 = cv.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)
            imgpoints.append(corners)

```

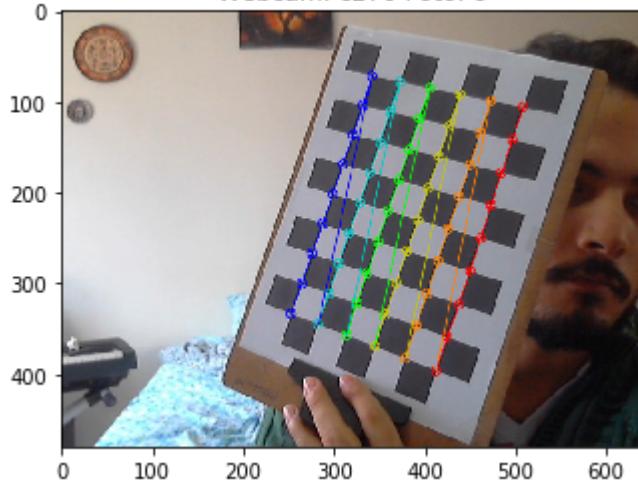
```
# Mostra o resultado obtido
cv.drawChessboardCorners(img, (x_col,x_lin), corners2, ret)
plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
plt.title('Webcam: '+camera+' Foto: '+fname[15:-4])
plt.show()
plt.pause(1)

# Determinação das matrizes da camera (matriz intrínseca e matriz de distorção)
ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, gray.shape
[:::-1], None, None)
img = cv.imread(images[1])
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
h, w = img.shape[:2]
newcameramtx, roi = cv.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1, (w,h))
np.savez(propriedades, mtx=newcameramtx, dist=dist)
print("Webcam " + camera + ":\nMatriz Intrínseca: ")
print(mtx)
print("Coeficientes de Distorção: ")
print(dist)
# tira a distorção da imagem
dst = cv.undistort(img, mtx, dist, None, newcameramtx)
# corta as partes da imagem sem informação (por causa do processo de retirada de distorção)
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]
cv.imwrite(caminho_salva, cv.cvtColor(dst, cv.COLOR_RGB2BGR))

plt.figure(figsize=(18, 16))
plt.title(camera)
ax1 = plt.subplot(121)
ax1.imshow(img)
ax2 = plt.subplot(122)
ax2.imshow(dst)
ax1.set_title('Original')
ax2.set_title('Sem Distorção')
plt.show()

print('Imagen planificada salva em: ' + caminho_salva)
mean_error = 0
for i in range(len(objpoints)):
    imgpoints2, _ = cv.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)
    error = cv.norm(imgpoints[i], imgpoints2, cv.NORM_L2)/len(imgpoints2)
    mean_error += error
print('Webcam: ' + camera + " calibrada!\nErro Total: {}".format(mean_error/len(objpoints)))
```

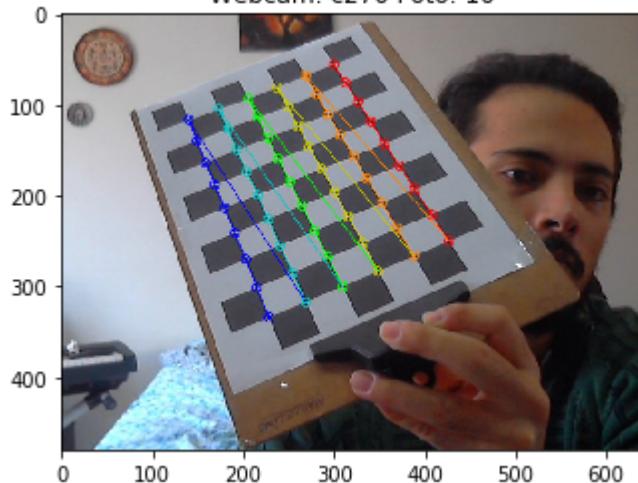
Webcam: c270 Foto: 0



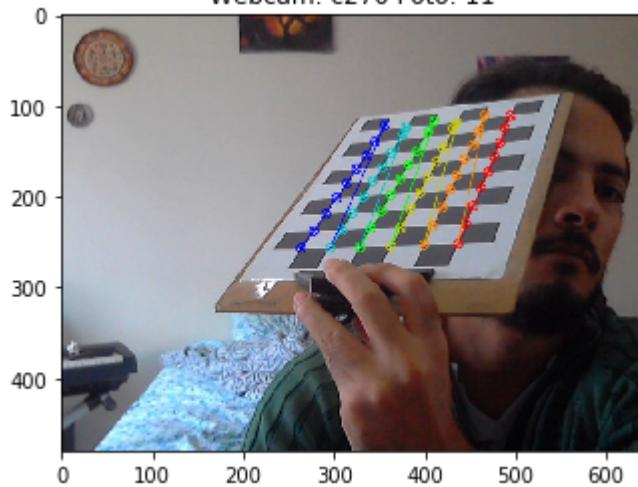
Webcam: c270 Foto: 1



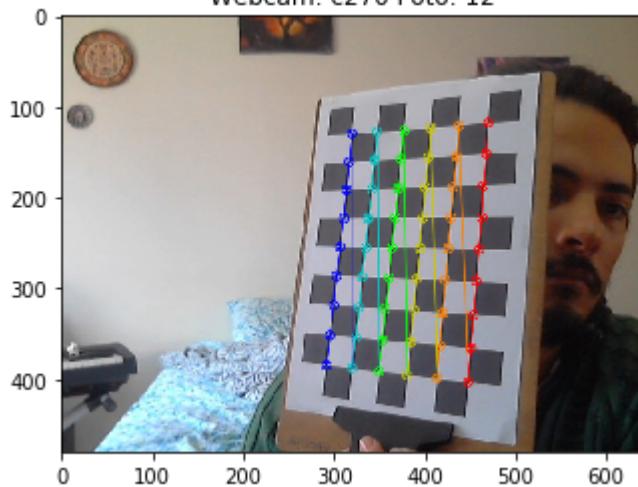
Webcam: c270 Foto: 10



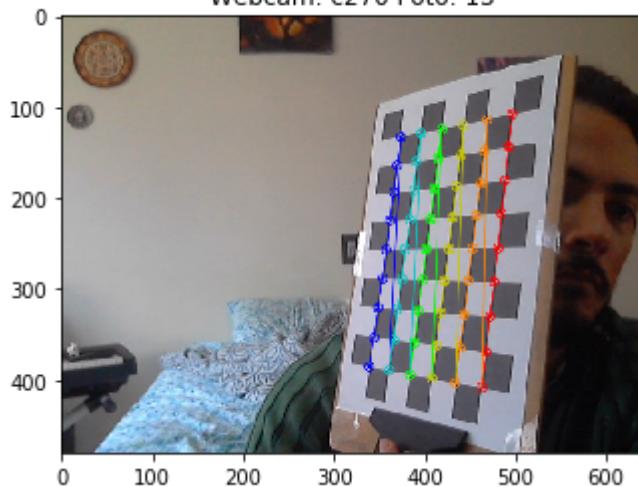
Webcam: c270 Foto: 11



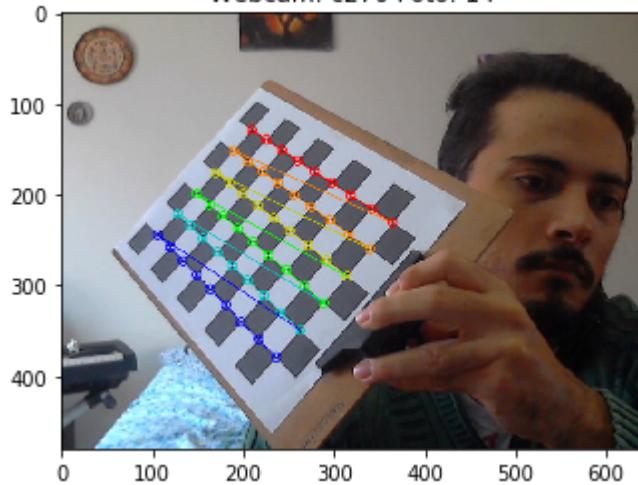
Webcam: c270 Foto: 12



Webcam: c270 Foto: 13



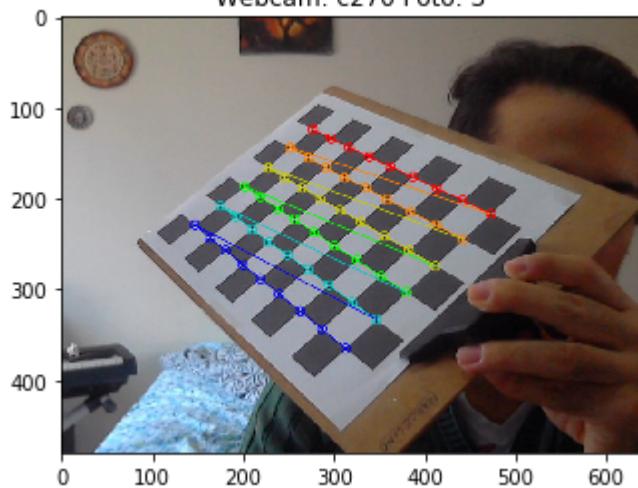
Webcam: c270 Foto: 14



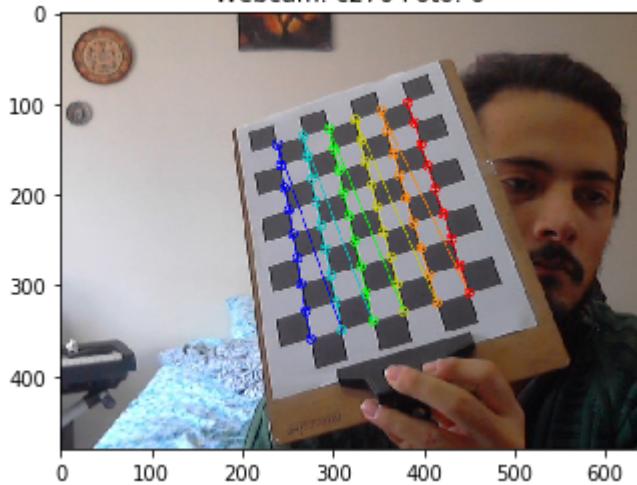
Webcam: c270 Foto: 2



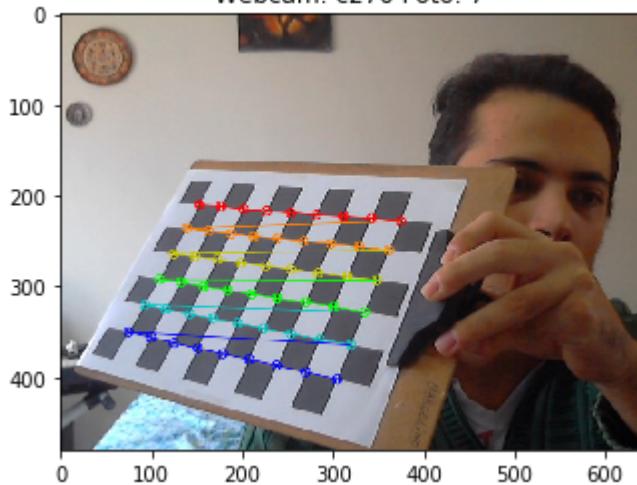
Webcam: c270 Foto: 3



Webcam: c270 Foto: 6



Webcam: c270 Foto: 7



Webcam: c270 Foto: 8



Webcam: c270 Foto: 9



Webcam c270:

Matriz Intrínseca:

```
[[848.0287819    0.        253.34826259]
 [ 0.        847.69817616 161.76229271]
 [ 0.        0.        1.        ]]
```

Coeficientes de Distorção:

```
[[ 0.05116295 -0.38504809 -0.02828233 -0.01844376  1.93482039]]
```

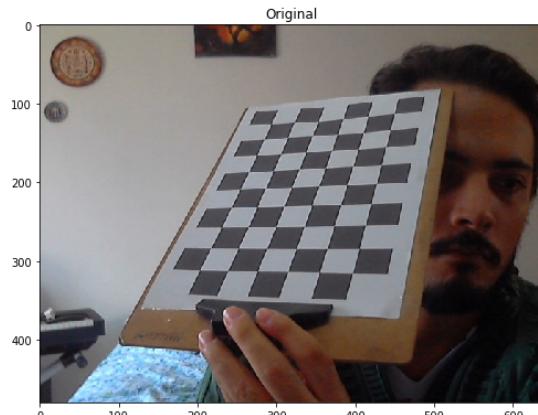
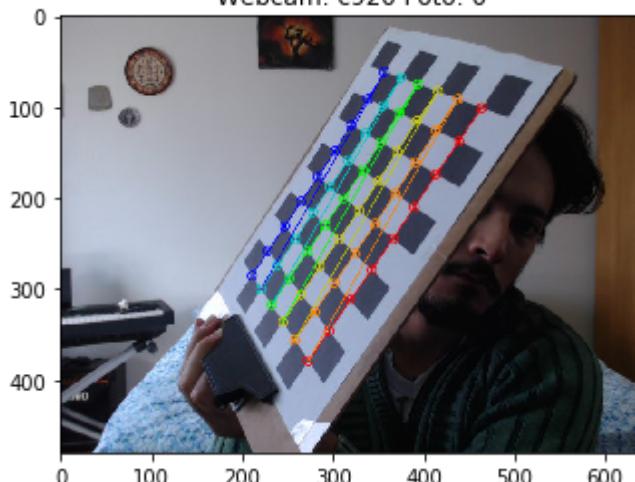


Imagen planificada salva em: ./imagens/c270/resultado\_calibr.png

Webcam: c270 calibrada!

Erro Total: 0.05558763328268981

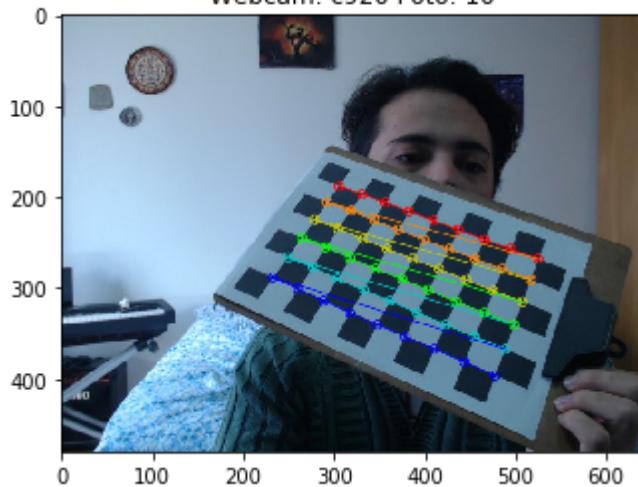
Webcam: c920 Foto: 0



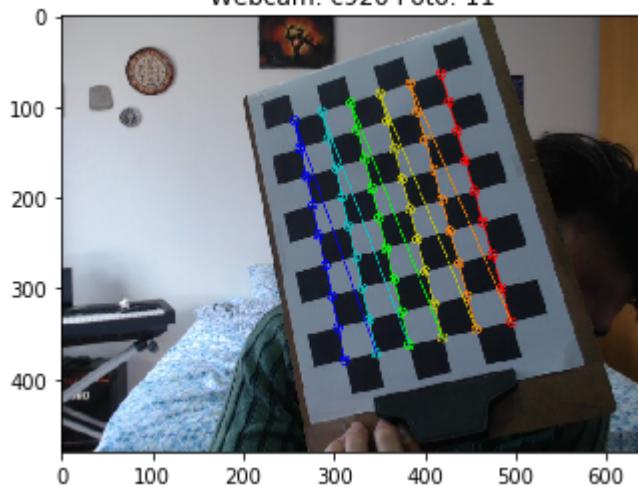
Webcam: c920 Foto: 1



Webcam: c920 Foto: 10



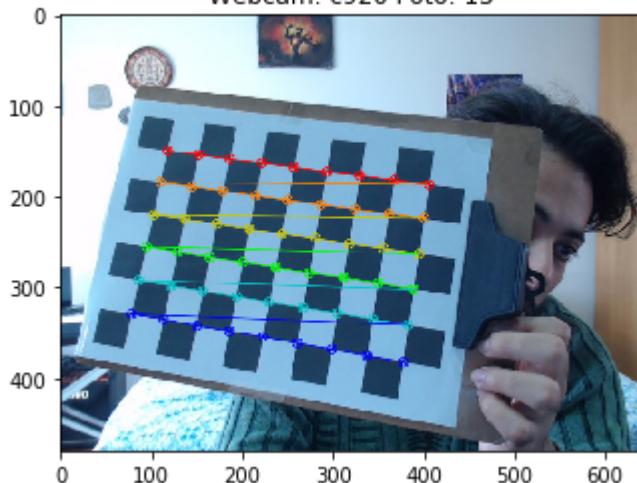
Webcam: c920 Foto: 11



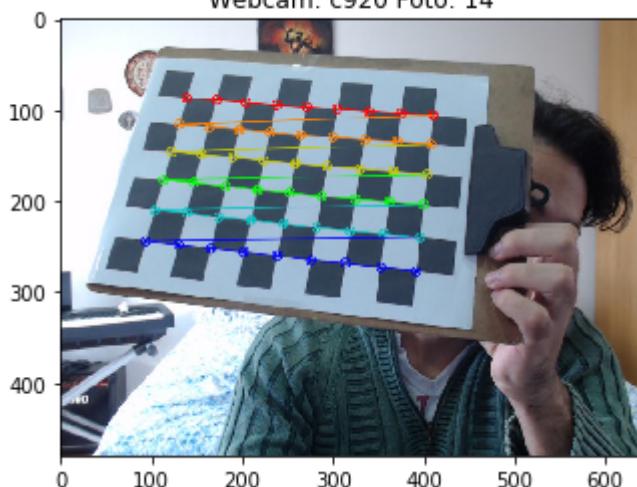
Webcam: c920 Foto: 12



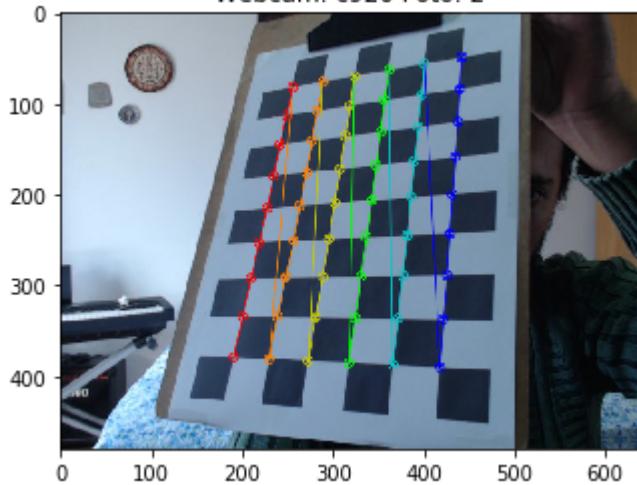
Webcam: c920 Foto: 13



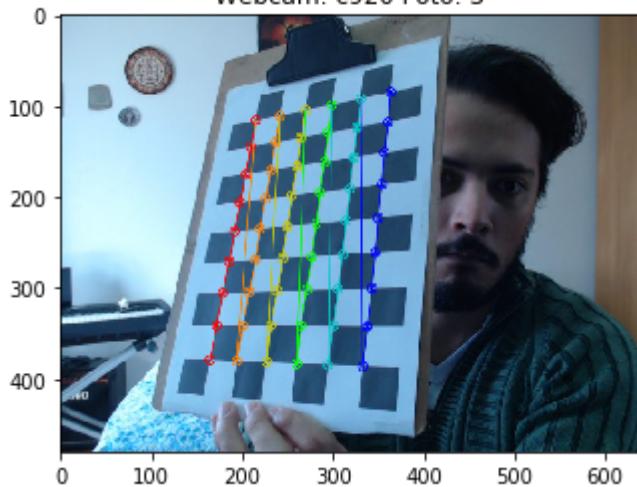
Webcam: c920 Foto: 14



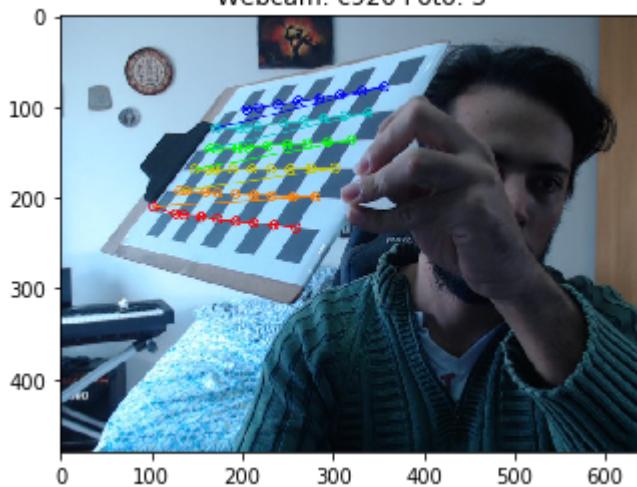
Webcam: c920 Foto: 2



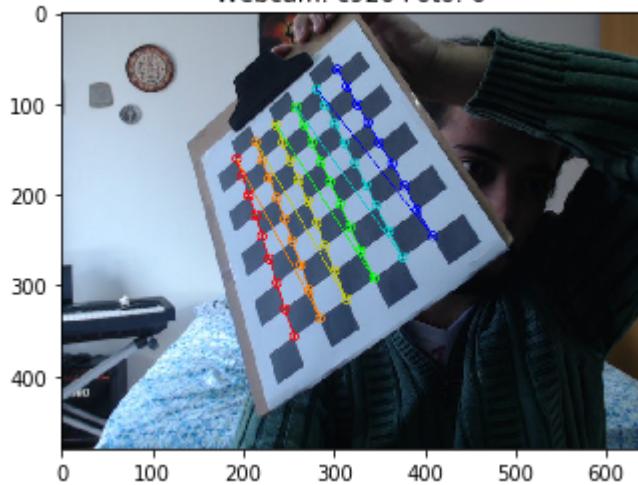
Webcam: c920 Foto: 3



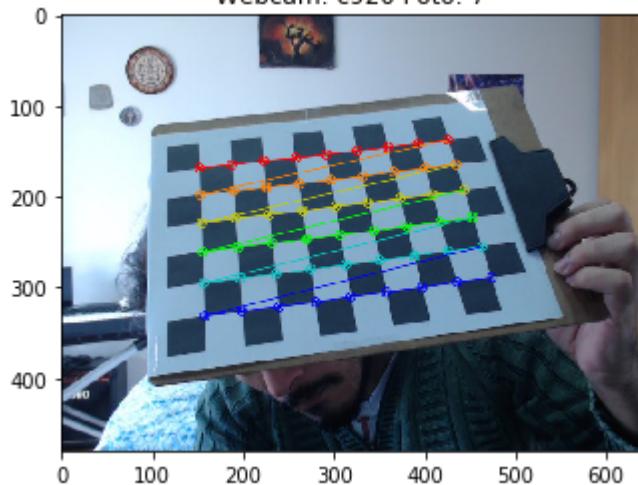
Webcam: c920 Foto: 5



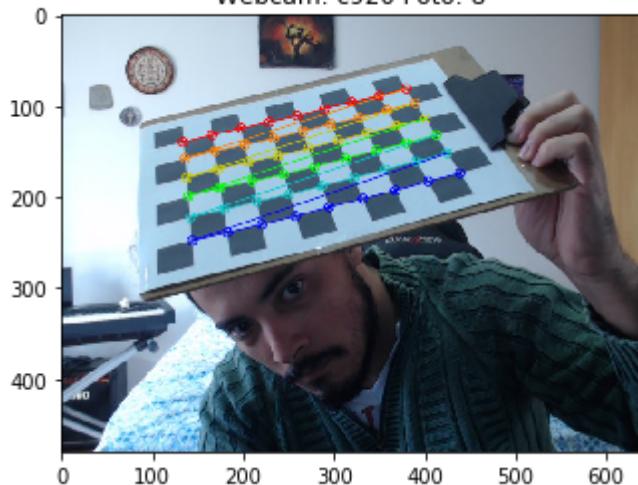
Webcam: c920 Foto: 6



Webcam: c920 Foto: 7



Webcam: c920 Foto: 8



Webcam: c920 Foto: 9



Webcam c920:

Matriz Intrínseca:

```
[[755.99355655  0.          249.98374757]
 [ 0.          756.56258605 182.26444592]
 [ 0.          0.          1.          ]]
```

Coeficientes de Distorção:

```
[[ 0.06335206 -0.33545357 -0.02322706 -0.02547997  1.50460019]]
```



Imagen planificada salva em: ./imagens/c920/resultado\_calibr.png

Webcam: c920 calibrada!

Erro Total: 0.11759308885490016

## Exercício 2

Neste exercício foram utilizadas as técnicas de perspectiva e ponto para desenhar um sistema de eixos (ou um cubo) nas mesmas imagens do exercício anterior. Podemos obserbar que o algoritmo de solução *PnP* é preciso, determinando a posição e atitude do tabuleiro de xadrez no ambiente 3D em todos os casos em que o tabuleiro de xadrez foi detectado. O tempo computacional de detecção do tabuleiro foi em média de 27.8 milissegundos, enquanto o tempo de solução do problema de perspectiva e ponto foi de 0.592 milissegundos.

In [21]:

```
import numpy as np
import cv2 as cv
import glob
import time
from matplotlib import pyplot as plt

"""

INF209B - TÓPICOS ESPECIAIS EM PROCESSAMENTO DE SINAIS:
```

*VISÃO COMPUTACIONAL**PRÁTICA 04**RA: 21201920754**NOME: RAFAEL COSTA FERNANDES**E-MAIL: COSTA.FERNANDES@UFABC.EDU.BR**Descrição:**Exercício n.2**Algoritmo de detecção de pose de um padrão xadrez.**Utiliza técnica de PnP para a determinação da pose por vetores de rotação rvecs e translação através de um vetor tvecs.**Utiliza um randomizador para a figura que será desenhada na imagem (sistema de eixos ou cubo)**Fonte parcial do código: [https://docs.opencv.org/master/d7/d53/tutorial\\_py\\_pose.html](https://docs.opencv.org/master/d7/d53/tutorial_py_pose.html)*

```
"""

plt.close('all')
cameras = ['c270', 'c920']
t_chess = []
t_pnp = []
for camera in cameras:
    # Carrega as propriedades da camera
    caminho_propriedades = './camera_properties/' + camera + '.npz'
    with np.load(caminho_propriedades) as X:
        mtx, dist = X[i] for i in ('mtx', 'dist')

    # Define as funções de desenho (Sistema de eixos ou Cubo)
    def draw(img, corners, imgpts):
        corner = tuple(corners[0].ravel())
        img = cv.line(img, corner, tuple(imgpts[0].ravel()), (255,0,0), 5)
        img = cv.line(img, corner, tuple(imgpts[1].ravel()), (0,255,0), 5)
        img = cv.line(img, corner, tuple(imgpts[2].ravel()), (0,0,255), 5)
        return img

    axis = np.float32([[3,0,0], [0,3,0], [0,0,-3]]).reshape(-1,3)

    def cube(img, corners, imgpts):
        imgpts = np.int32(imgpts).reshape(-1,2)
        # draw ground floor in green
        img = cv.drawContours(img, [imgpts[:4]], -1, (0,255,0), -3)
        # draw pillars in blue color
        for i,j in zip(range(4),range(4,8)):
            img = cv.line(img, tuple(imgpts[i]), tuple(imgpts[j]), (255), 3)
        # draw top layer in red color
        img = cv.drawContours(img, [imgpts[4:]], -1, (0,0,255), 3)
        return img

t_chess.append(draw(img, corners, imgpts))
t_pnp.append(cube(img, corners, imgpts))

plt.imshow(t_chess[-1])
plt.show()
```

```

axis_cube = np.float32([[0,0,0], [0,3,0], [3,3,0], [3,0,0],
[0,0,-3],[0,3,-3],[3,3,-3],[3,0,-3] ])

#Critério de determinação de vértices em subpixel
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)

#Pontos do tabuleiro de xadrez
objp = np.zeros((6*9,3), np.float32)
objp[:, :2] = np.mgrid[0:9,0:6].T.reshape(-1,2)
caminho_fotos = './imagens/' + camera + '/*.jpg'
for i, fname in enumerate(glob.glob(caminho_fotos)):
    # Lê uma imagem no caminho das imagens
    img = cv.imread(fname)

    #Converte para escala de cinzas
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    #Tempo inicial do algoritmo de busca do padrão xadrez
    t_i = time.time()
    #Algoritmo de busca do padrão xadrez
    ret, corners = cv.findChessboardCorners(gray, (9,6), None)
    t_chess.append(time.time() - t_i)

    #Se achar o padrão xadrez
    if ret == True:
        corners2 = cv.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)

        # Tempo inicial do algoritmo PnP
        t_i = time.time()

        # Determinação da atitude e posição (rvecs e tvecs)
        ret, rvecs, tvecs = cv.solvePnP(objp, corners2, mtx, dist)
        t_pnp.append(time.time() - t_i)

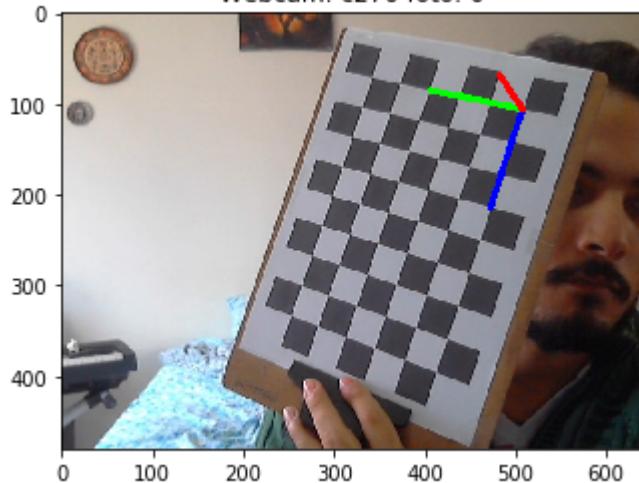
        # Projeta os pontos 3D na imagem
        if np.random.random() < 0.5:
            imgpts, jac = cv.projectPoints(axis, rvecs, tvecs, mtx, dist)
            img = draw(img, corners2, imgpts)
        else:
            imgpts, jac = cv.projectPoints(axis_cube, rvecs, tvecs, mtx, dist)
            img = cube(img, corners2, imgpts)

        # Mostra a imagem
        plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
        plt.title('Webcam: ' + camera + ' foto: ' + fname[15:-4])
        plt.show()
        plt.pause(0.5)

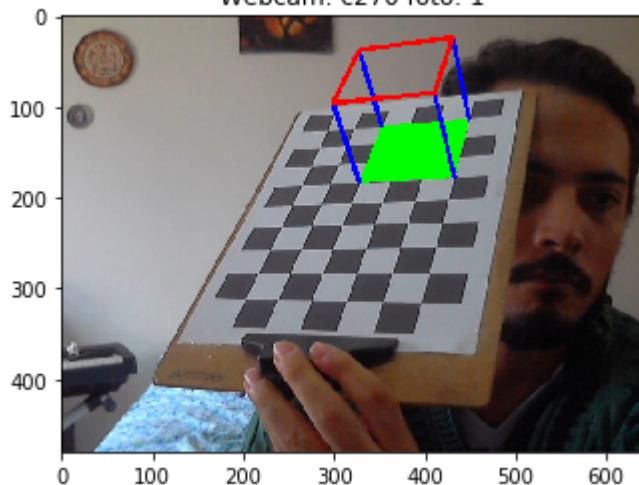
        # Salva a imagem
        cv.imwrite(fname[0:-4] + '_vec.png', img)
print('Imagens salvas em: ' + caminho_fotos)
print(f'Tempo médio Detecção Tabuleiro {np.mean(t_chess)}')
print(f'Tempo médio Solução PnP {np.mean(t_pnp)}')

```

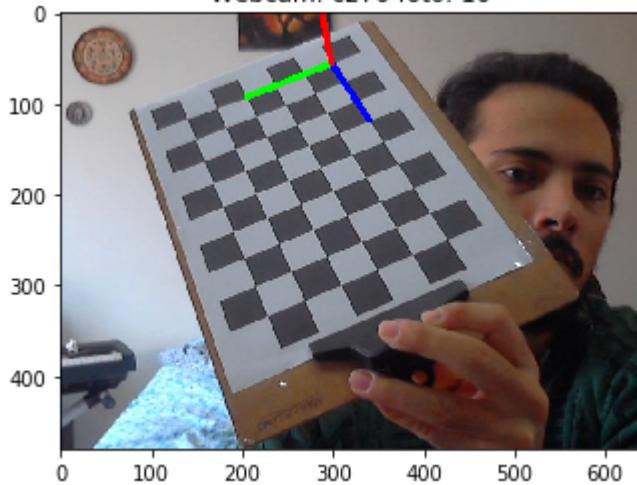
Webcam: c270 foto: 0



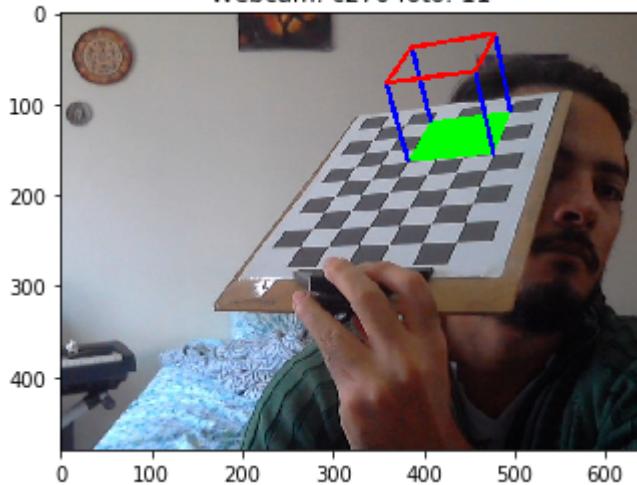
Webcam: c270 foto: 1



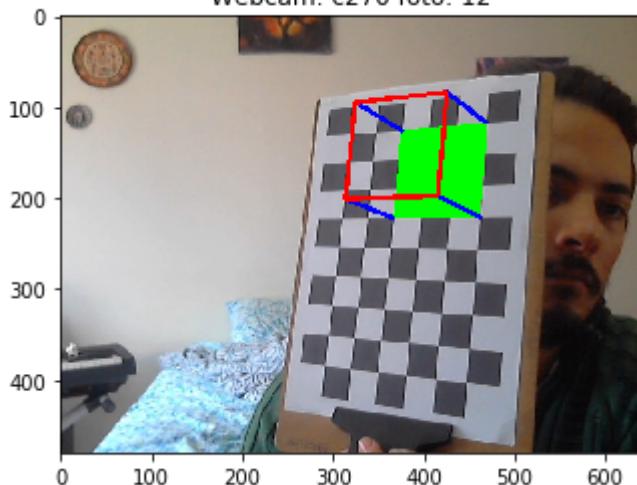
Webcam: c270 foto: 10



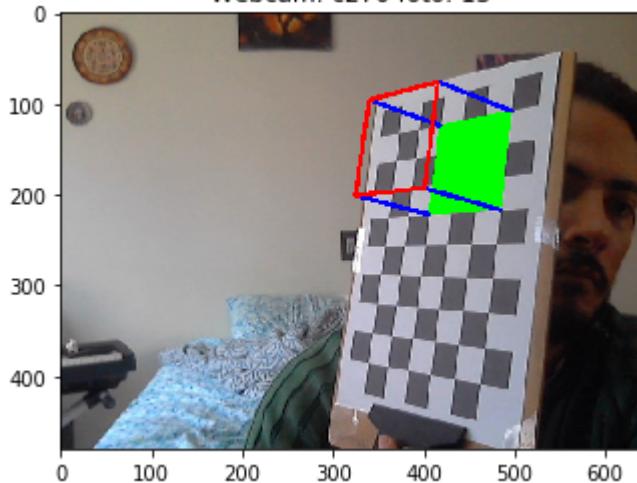
Webcam: c270 foto: 11



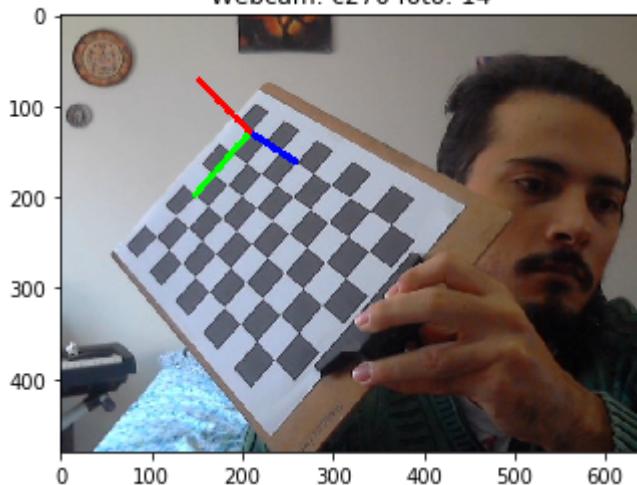
Webcam: c270 foto: 12



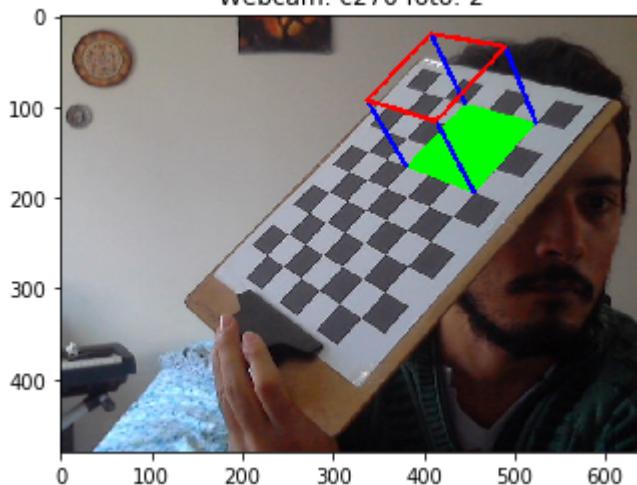
Webcam: c270 foto: 13



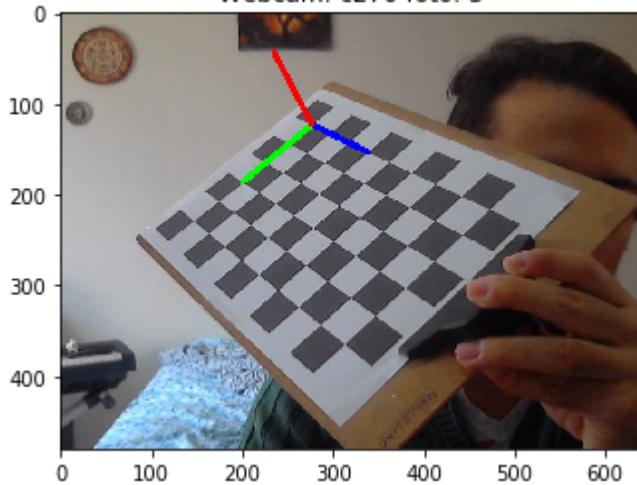
Webcam: c270 foto: 14



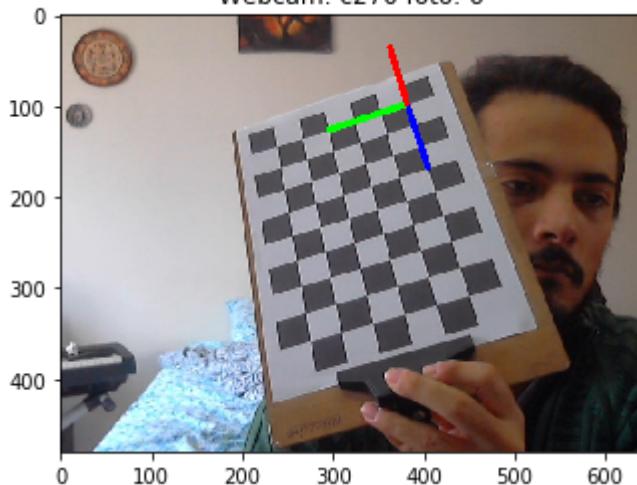
Webcam: c270 foto: 2



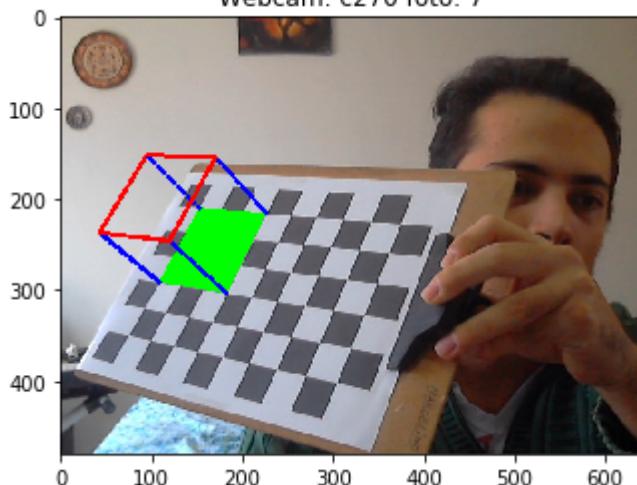
Webcam: c270 foto: 3



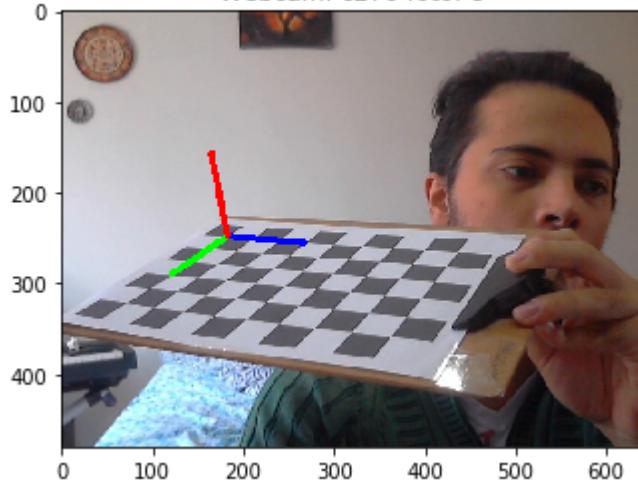
Webcam: c270 foto: 6



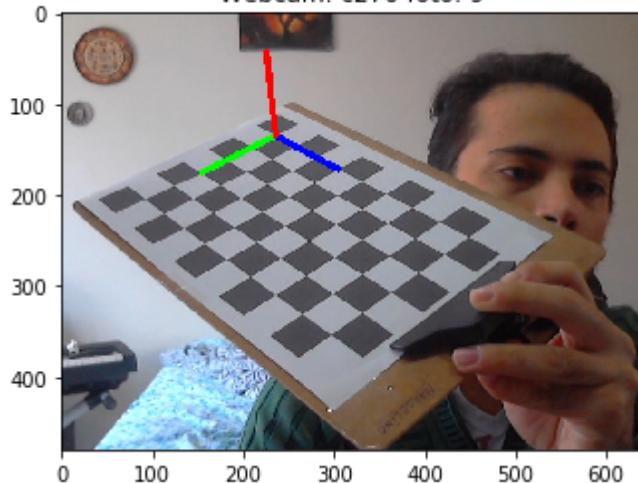
Webcam: c270 foto: 7



Webcam: c270 foto: 8

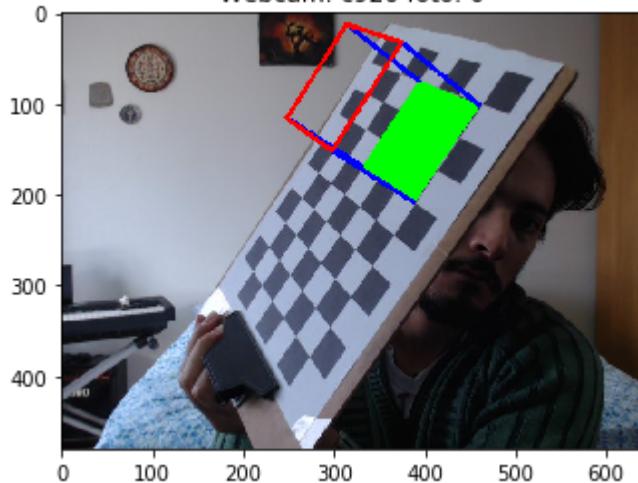


Webcam: c270 foto: 9

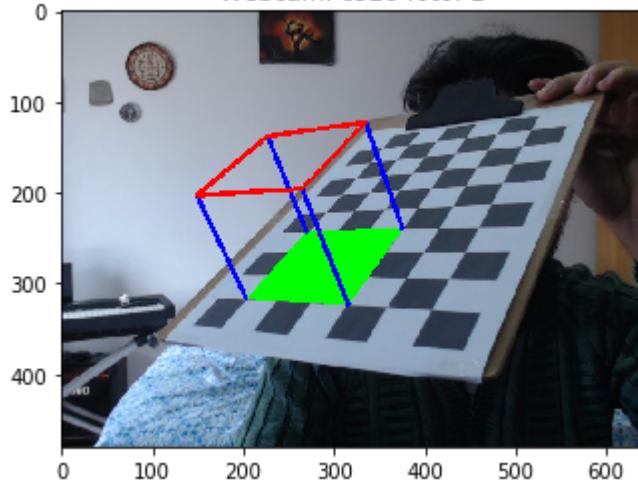


Imagens salvas em: ./imagens/c270/\*.jpg

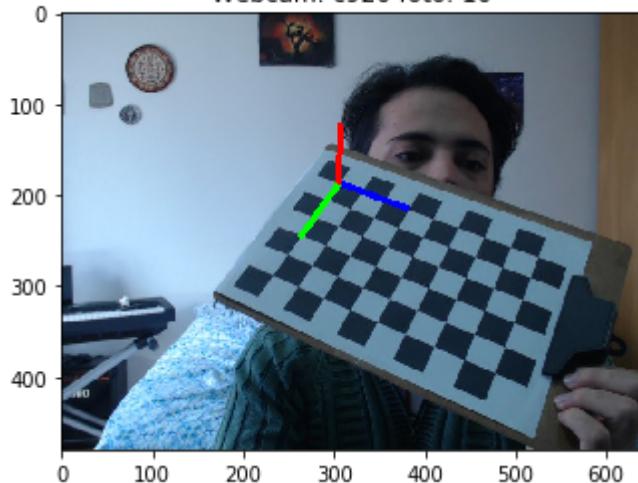
Webcam: c920 foto: 0



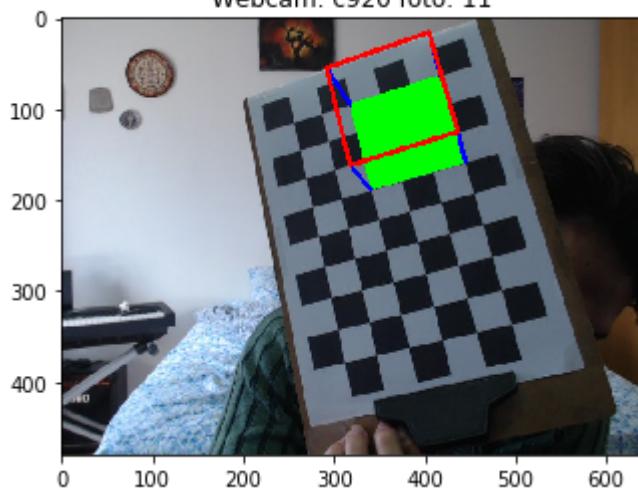
Webcam: c920 foto: 1



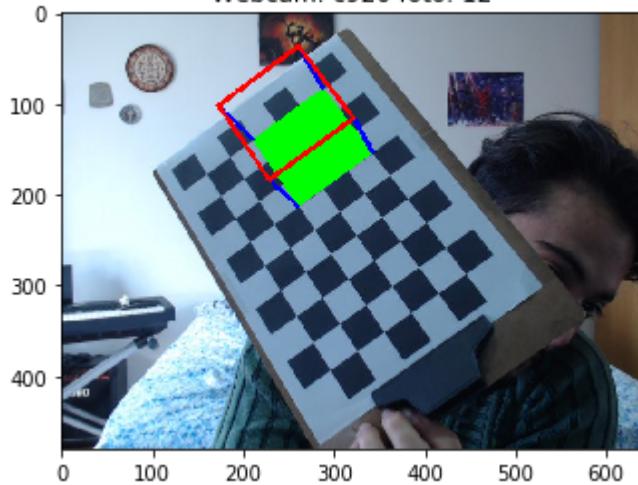
Webcam: c920 foto: 10



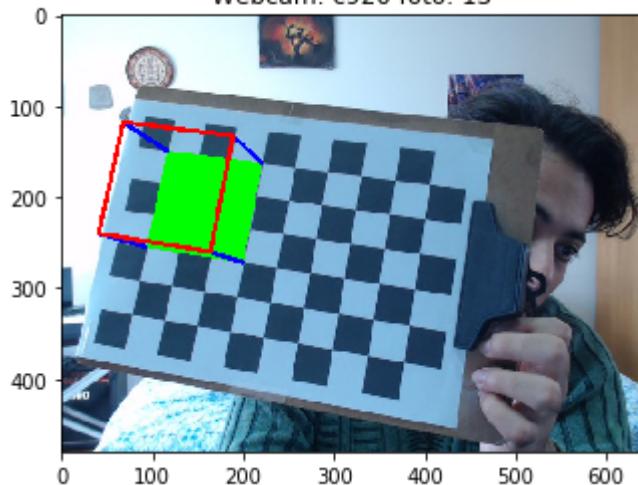
Webcam: c920 foto: 11



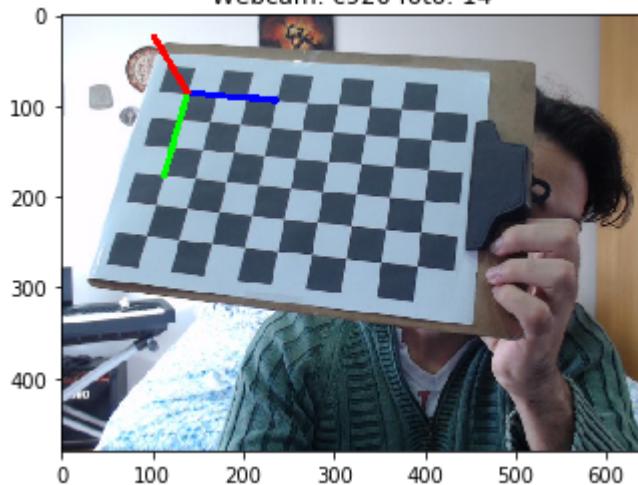
Webcam: c920 foto: 12



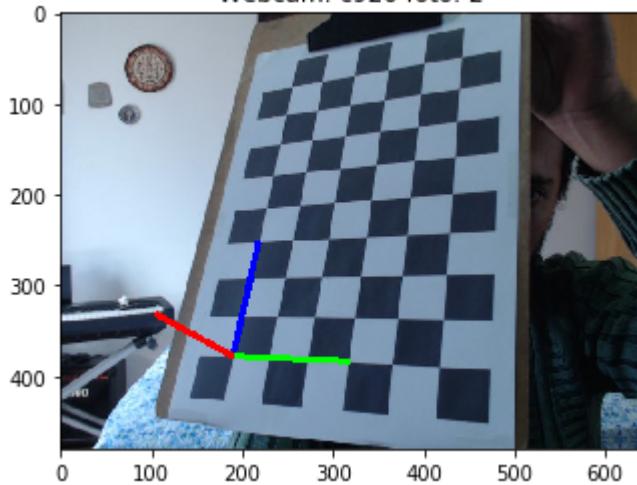
Webcam: c920 foto: 13



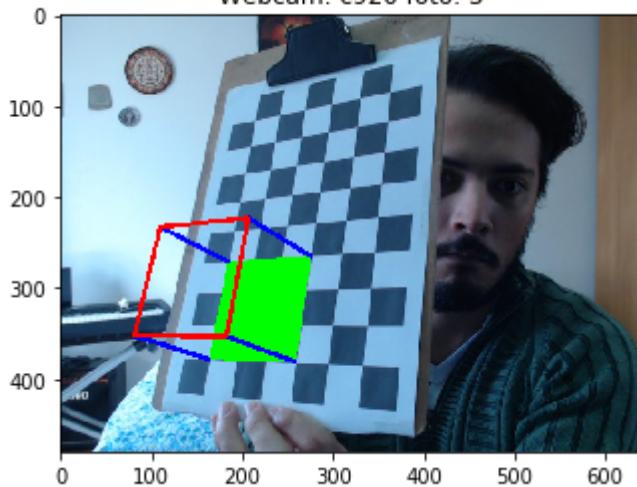
Webcam: c920 foto: 14



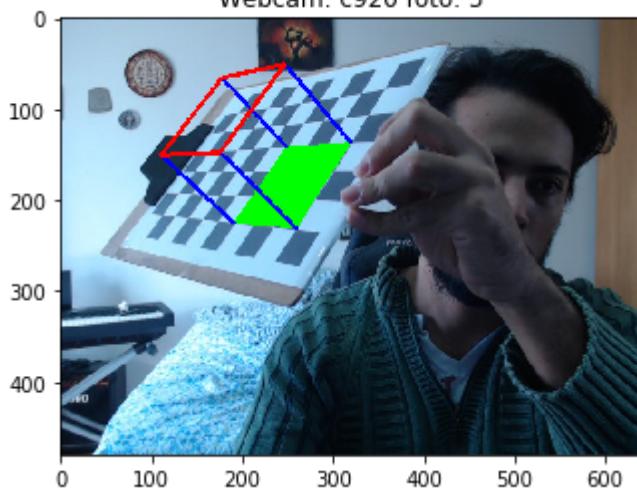
Webcam: c920 foto: 2



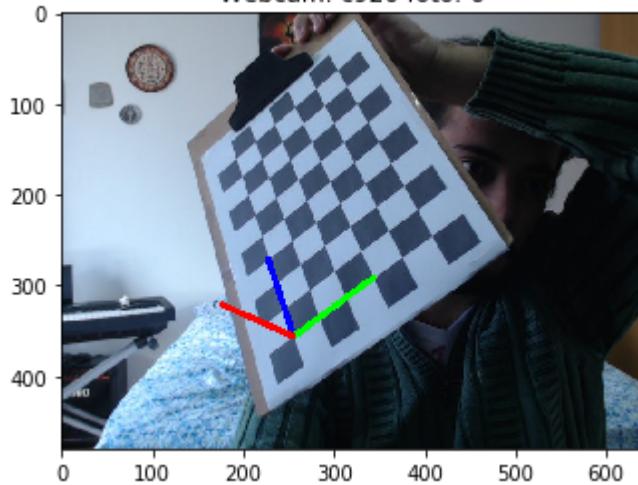
Webcam: c920 foto: 3



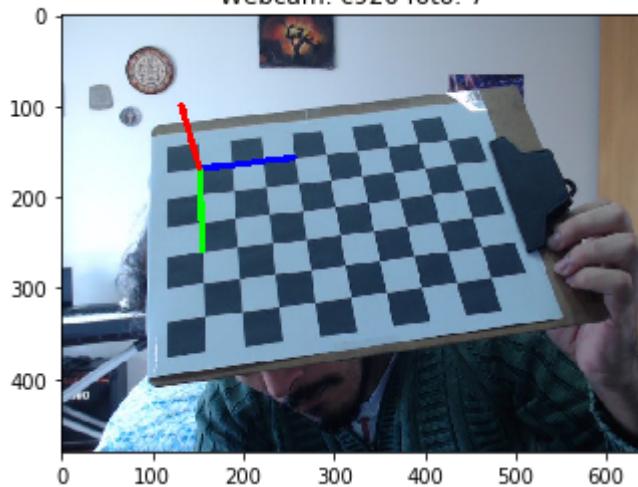
Webcam: c920 foto: 5



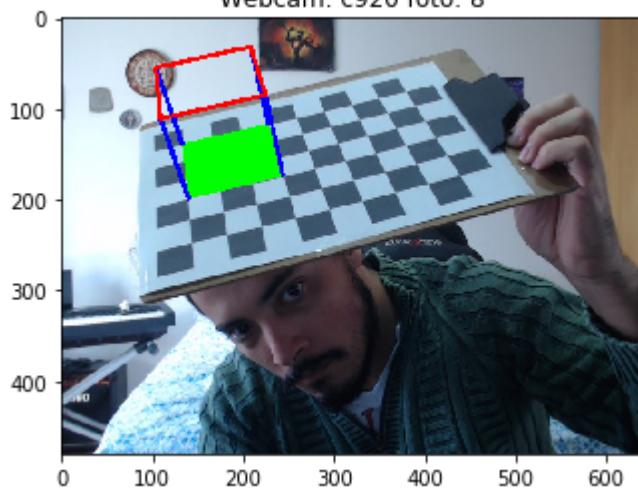
Webcam: c920 foto: 6



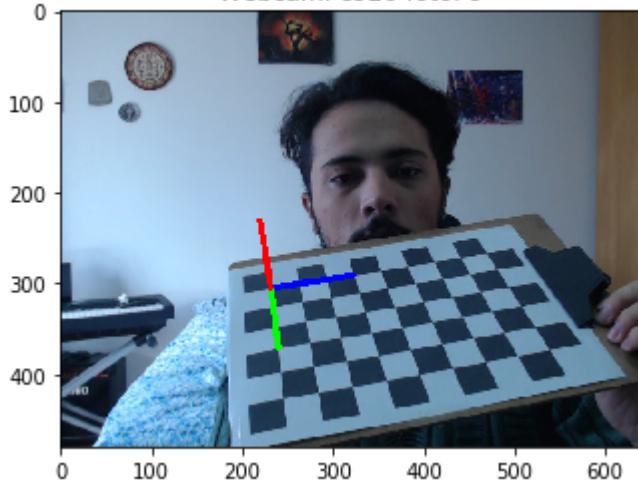
Webcam: c920 foto: 7



Webcam: c920 foto: 8



Webcam: c920 foto: 9



Imagens salvas em: ./imagens/c920/\*.jpg

Tempo médio Detecção Tabuleiro 0.026928373745509555

Tempo médio Solução PnP 0.00040711296929253475

## Exercício 3

Neste exercício são obtidos as epilinhas de um plano epipolar de um par de imagens estéreo. O resultado obtido teve coerência, mas não é exato. Como podemos observar nas imagens obtidas, as epilinhas não estão perfeitamente paralelas ao plano epipolar da imagem, este plano que é coincidente com o tabuleiro de xadrez. A matriz fundamental F e as matrizes essenciais E de cada câmera são apresentadas na saída do algoritmo.

A exatidão dos resultados obtidos pode ser explicada em alguns aspectos. As câmeras devem estar no mesmo plano de altura e, preferencialmente, paralelas. O ajuste de altura e paralelismo foi feito visualmente, provavelmente este tipo de ajuste não é o suficiente para uma boa precisão do plano epipolar.

Também, idealmente, as câmeras devem ter o mesmo tamanho de sensor e distância focal, o que não é o caso. Foi necessário recortar a imagem da câmera C920 para aproximar a imagem da câmera C270 (diminuindo efetivamente o tamanho do sensor), mas este recorte altera a distância focal da imagem.

Ambas aproximações podem ter afetado o resultado final. Uma sugestão de melhoria é utilizar câmeras idênticas, montadas em um sistema de fixação que garante altura constante e paralelismo, garantindo as considerações teóricas da matemática do problema.

In [23]:

```

import numpy as np
import cv2 as cv
import glob
from matplotlib import pyplot as plt

"""

INF209B - TÓPICOS ESPECIAIS EM PROCESSAMENTO DE SINAIS:

VISÃO COMPUTACIONAL

PRÁTICA 04

RA: 21201920754
NOME: RAFAEL COSTA FERNANDES
E-MAIL: COSTA.FERNANDES@UFABC.EDU.BR

DESCRIÇÃO:
Exercício n.3

Plota as epilinhas, dado duas imagens de um par de câmeras em disposição estéreo.
Preferencialmente deve haver um plano bem definido na imagem.
Este plano será utilizado para a determinação do plano epipolar, e será a base para as
epilinhas.

Fonte parcial do código: https://docs.opencv.org/master/d4/de9/tutorial\_py\_epipolar\_geometry.html
A função SIFT não existe mais no opencv 4.2 por questões de Direitos Autorais
O opencv deve ser revertido para a versão 3.4.2 contrib para compatibilidade com a função.

Aparentemente a versão 3.4.2 roda todos os algoritmos utilizados na disciplina, então sem problemas.

"""

plt.close('all')
cam_name = glob.glob('.\\camera_properties\\*.npz')
mtx = []
dist = []
for cam in cam_name:
    cam_par = np.load(cam)
    mtx.append(cam_par['mtx'])
    dist.append(cam_par['dist'])

# Imagem da camera esquerda
img1 = cv.imread('.\\imagens\\epipolar\\left.jpg',0)

# Processo de igualar resolução e retirar as distorções
img1 = cv.resize(img1, (640, 480))
h, w = img1.shape[:2]
newcameramtx, roi = cv.getOptimalNewCameraMatrix(mtx[0], dist[0], (w,h), 1, (w,h))
dst = cv.undistort(img1, mtx[0], dist[0], None, newcameramtx)
x, y, w, h = roi
img1 = dst[y:y+h, x:x+w]

# Imagem da camera direita
img2 = cv.imread('.\\imagens\\epipolar\\right.jpg',0)
img2 = cv.resize(img2, (640, 480))

```

```

# Processo de igualar resolução e retirar as distorções
h, w = img2.shape[:2]
newcameramtx, roi = cv.getOptimalNewCameraMatrix(mtx[1], dist[1], (w,h), 1, (w,h))
dst = cv.undistort(img2, mtx[1], dist[1], None, newcameramtx)
x, y, w, h = roi
img2 = dst[y:y+h, x:x+w]

plt.figure('Original', figsize=(16, 18))
ax1=plt.subplot(121)
ax1.imshow(img1, cmap='gray')
ax2=plt.subplot(122)
ax2.imshow(img2, cmap='gray')
ax1.set_title('Original - C920')
ax2.set_title('Original - C270')
plt.show()

# Determinação dos pontos chave por algoritmo SIFT
sift = cv.xfeatures2d.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)

# Parâmetros FLANN
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)
flann = cv.FlannBasedMatcher(index_params,search_params)
matches = flann.knnMatch(des1,des2,k=2)
good = []
pts1 = []
pts2 = []

# Teste de Razão
for i,(m,n) in enumerate(matches):
    if m.distance < 0.8*n.distance:
        good.append(m)
        pts2.append(kp2[m.trainIdx].pt)
        pts1.append(kp1[m.queryIdx].pt)
pts1 = np.int32(pts1)
pts2 = np.int32(pts2)

# Matriz Fundamental
F, mask = cv.findFundamentalMat(pts1,pts2,cv.FM_LMEDS)
print('Matriz Fundamental: ')
print(F)

# Matriz Essencial
E = []
nomes = ['c920', 'c270']
for mtx_i, nome in zip(mtx, nomes):
    E_i = np.dot(mtx_i.T, np.dot(F, mtx_i))
    print('Matriz essencial webcam '+name)
    print(E_i)
    E.append(E_i)

# Selecionando apenas pontos internos
pts1 = pts1[mask.ravel()==1]
pts2 = pts2[mask.ravel()==1]

# Função para desenhar as epilines
def drawlines(img1,img2,lines,pts1,pts2):

```

```
''' img1 - image on which we draw the epilines for the points in img2
   Lines - corresponding epilines '''
r,c = img1.shape
img1 = cv.cvtColor(img1,cv.COLOR_GRAY2BGR)
img2 = cv.cvtColor(img2,cv.COLOR_GRAY2BGR)
for r,pt1,pt2 in zip(lines,pts1,pts2):
    color = tuple(np.random.randint(0,255,3).tolist())
    x0,y0 = map(int, [0, -r[2]/r[1] ])
    x1,y1 = map(int, [c, -(r[2]+r[0]*c)/r[1] ])
    img1 = cv.line(img1, (x0,y0), (x1,y1), color,1)
    img1 = cv.circle(img1,tuple(pt1),5,color,-1)
    img2 = cv.circle(img2,tuple(pt2),5,color,-1)
return img1,img2
```

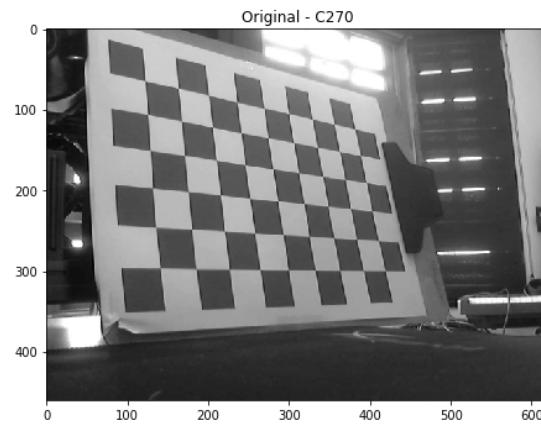
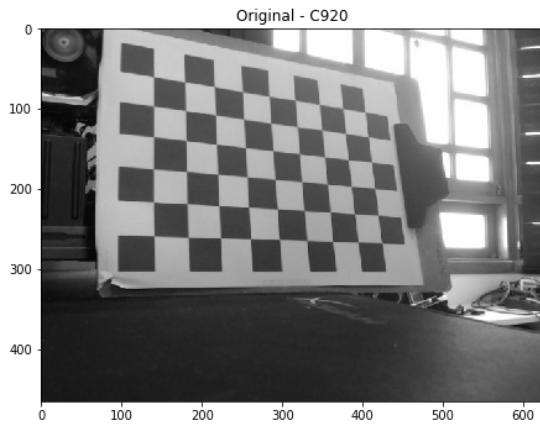
# Achando epilinhas da imagem direira e desenhando as linhas na imagem esquerda, vice versa.

```
lines1 = cv.computeCorrespondEpilines(pts2.reshape(-1,1,2), 2,F)
lines1 = lines1.reshape(-1,3)
img5,img6 = drawlines(img1,img2,lines1,pts1,pts2)
```

```
lines2 = cv.computeCorrespondEpilines(pts1.reshape(-1,1,2), 1,F)
lines2 = lines2.reshape(-1,3)
img3,img4 = drawlines(img2,img1,lines2,pts2,pts1)
```

#Plotando os resultados

```
plt.figure('Epilines', figsize=(16, 18))
ax1=plt.subplot(121)
ax1.imshow(img5)
ax2=plt.subplot(122)
ax2.imshow(img3)
ax1.set_title('Epilines - C920')
ax2.set_title('Epilines - C270')
plt.show()
```



Matriz Fundamental:

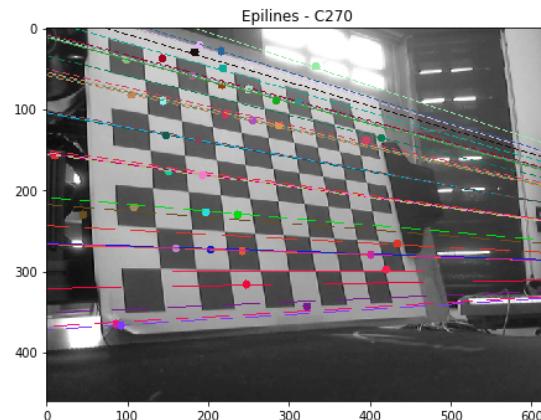
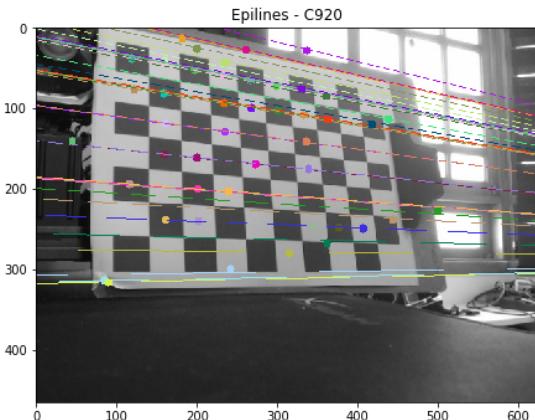
```
[[ -9.98848689e-06  5.18631211e-04 -1.36009811e-01]
 [ -3.21398458e-04 -1.08199169e-04  4.88258863e-01]
 [ 1.08713665e-01 -5.37084016e-01  1.00000000e+00]]
```

Matriz essencial webcam c270

```
[[ -7.08443057  363.58759274 -48.31845563]
 [-225.31712172 -74.97552088 325.80984582]
 [ 47.1267325 -353.79322149 -9.01804373]]
```

Matriz essencial webcam c270

```
[[ -5.68247654  292.10840343 -35.05985255]
 [-181.02109644 -60.33325845 291.88279678]
 [ 37.18262228 -321.16073242 -9.78265273]]
```



## Exercício 4

Neste exercício houve o mesmo problema do exercício anterior, em relação à configuração de paralelismo, altura, distância focal e tamanho de sensor das câmeras. Por este problema, não foi possível obter um mapa de disparidade que houvesse qualquer informação visível de disparidade, independente da configuração do algoritmo estéreo SGBM.

Foi decidido então utilizar uma biblioteca de imagens estéreo já calibradas, retiradas de

<http://vision.middlebury.edu/stereo/data/scenes2005/FullSize/Art/>

(<http://vision.middlebury.edu/stereo/data/scenes2005/FullSize/Art/>). O resultado obtido é comparável com o resultado real apresentado pela biblioteca, demonstrando que o problema está realmente na configuração das câmeras em modo estéreo, não no algoritmo utilizado.

In [25]:

```
import numpy as np
import cv2 as cv
import glob
from matplotlib import pyplot as plt

"""
INF209B - TÓPICOS ESPECIAIS EM PROCESSAMENTO DE SINAIS:
```

*VISÃO COMPUTACIONAL*

*PRÁTICA 04*

*RA: 21201920754*

*NOME: RAFAEL COSTA FERNANDES*

*E-MAIL: COSTA.FERNANDES@UFABC.EDU.BR*

*Descrição:*

*Exercício n.4*

*Algoritmo para obtenção de mapa de disparidade baseado em um par de imagens estereoscópicas.*

*Ajustar os parâmetros de setup do algoritmo SGBM, conforme necessidade.*

*Fonte parcial do código:*

*<https://medium.com/@omar.ps16/stereo-3d-reconstruction-with-opencv-using-an-iphone-camera-part-iii-95460d3eddf0>*

```
"""
plt.close('all')
```

*#Utilização de uma imagem estéreo já calibrada*

```
img1 = cv.imread('.\\imagens\\disparities\\view0.png',0)
```

```
img2 = cv.imread('.\\imagens\\disparities\\view1.png',0)
```

*#Equalização da imagem*

```
img2 = cv.equalizeHist(img2)
img1 = cv.equalizeHist(img1)
```

*#Plot da imagem original*

```
plt.close('all')
plt.figure('Original', figsize=(18,16))
ax1=plt.subplot(121)
ax1.imshow(img1, cmap='gray')
ax2=plt.subplot(122)
ax2.imshow(img2, cmap='gray')
plt.show()
```

*#Setup do algoritmo StereoSGBM*

```
win_size = 2
min_disp = -1
max_disp = 16*8-1
num_disp = max_disp - min_disp
```

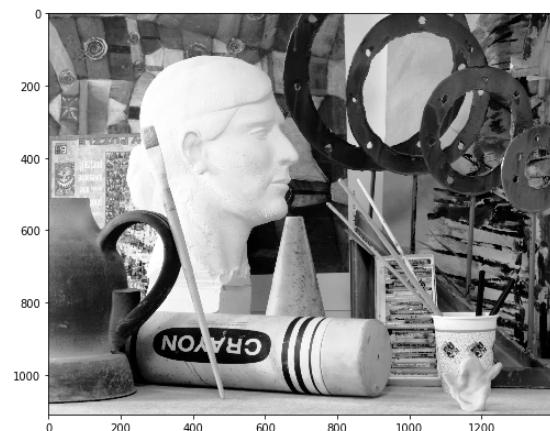
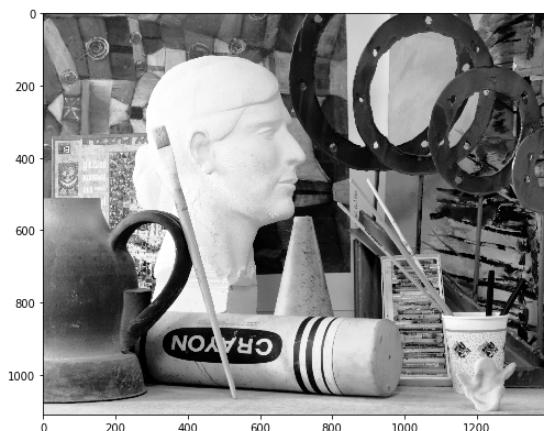
```
stereo = cv.StereoSGBM_create(minDisparity= min_disp,
```

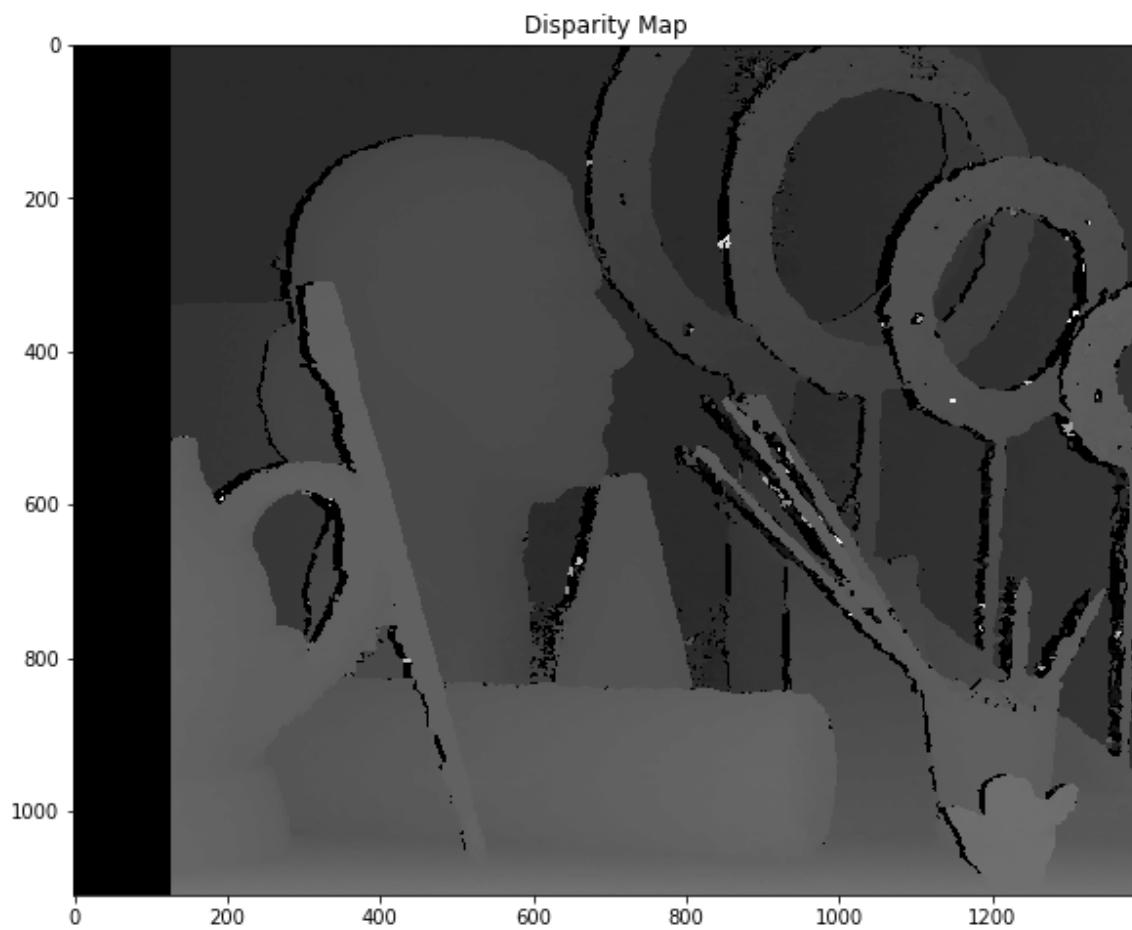
```
numDisparities = num_disp,
blockSize = 5,
uniquenessRatio = 5,
speckleWindowSize = 5,
speckleRange = 5,
disp12MaxDiff = 1,
P1 = 8*4*win_size**2,
P2 =32*4*win_size**2)

#Cálculo do mapa de disparidade
disparity = stereo.compute(img1, img2)
#Normalização do resultado entre 0 e 255 e uint8
disparity -= np.min(disparity)
disparity = np.multiply(disparity,255.0/np.max(disparity))
disparity = disparity.astype(np.uint8)

#Plot do resultado
plt.figure(figsize=(12,8))
string = f'Disparity Map'
plt.title(string)
plt.imshow(disparity, cmap='gray')
plt.draw()
plt.pause(0.5)

#Salva o mapa obtido
caminho_mapa = '.\\imagens\\disparities\\Disparidade_Calculada.png'
cv.imwrite(caminho_mapa, disparity)
print('Mapa de disparidade salvo em: '+caminho_mapa)
```





Mapa de disparidade salvo em: .\imagens\disparities\Disparidade\_Calculada.png

## Conclusões e Discussões

# Referências

- [1] Zhang, Zhengyou. "A flexible new technique for camera calibration." IEEE Transactions on pattern analysis and machine intelligence 22.11 (2000): 1330-1334.
- [2] Tsai, Roger. "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses." IEEE Journal on Robotics and Automation 3.4 (1987): 323-344.
- [3] Bradski, Gary. "The opencv library." Dr Dobb's J. Software Tools 25 (2000): 120-125.
- [4] Haralick, Bert M., et al. "Review and analysis of solutions of the three point perspective pose estimation problem." International journal of computer vision 13.3 (1994): 331-356.
- [5] Park, Jongho, and Youdan Kim. "Stereo vision based collision avoidance of quadrotor UAV." 2012 12th International Conference on Control, Automation and Systems. IEEE, 2012.