

Lab-03 Report

st122246

June 28, 2022

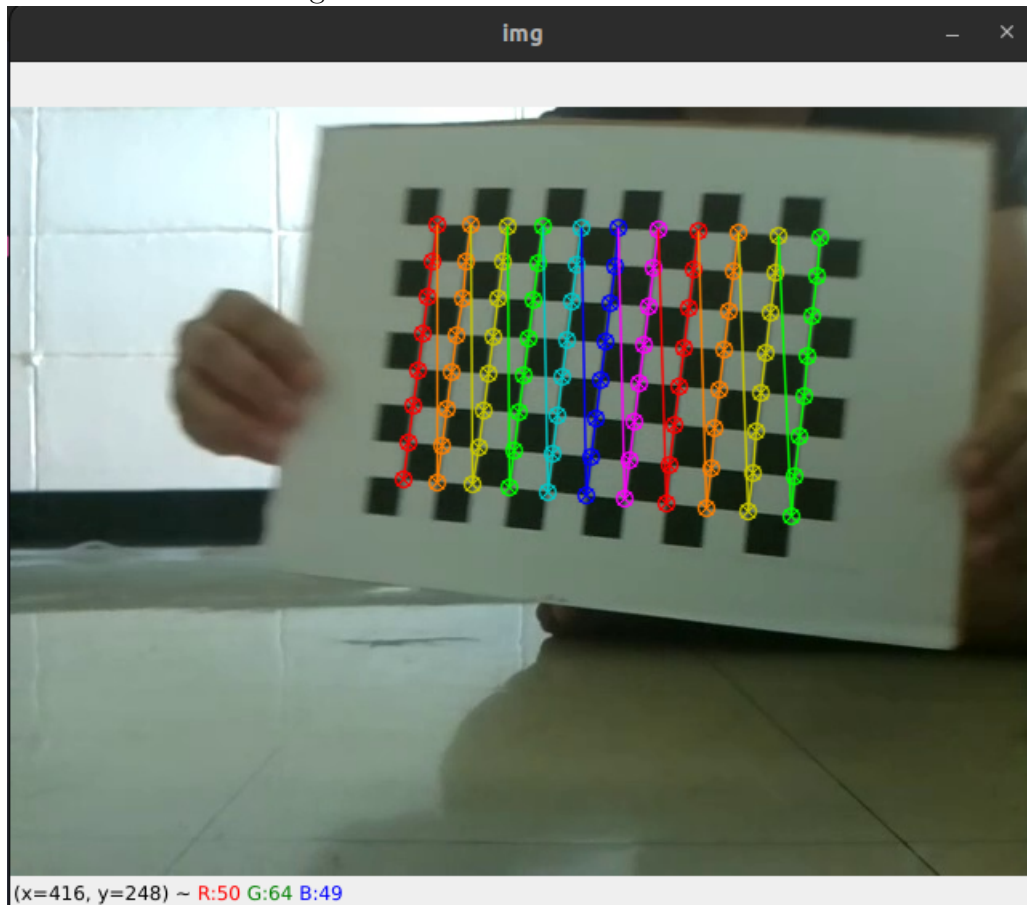
1 Chessboard calibration

1.1 Save the image of chessboard for use in camera

main.py

```
1 import cv2 as cv
2
3 video_capture = cv.VideoCapture()
4 video_capture.open(0, cv.CAP_ANY)
5 if not video_capture.isOpened():
6     print("ERROR! Unable to open camera\n")
7     exit()
8
9 print("Start grabbing")
10 print("Press s to save images and q to terminate")
11
12 frame_add = 0
13 while True:
14     _, frame = video_capture.read()
15     if frame is None:
16         print("ERROR! Blank frame grabbed\n")
17         exit()
18
19     cv.imshow("Live", frame)
20
21     iKey = cv.waitKey(5)
22     if iKey == ord('s') or iKey == ord('S'):
23         cv.imwrite("../../Data/Lab03/images/frame" +
                    str(frame_add) + ".jpg", frame)
```

Figure 1: 1.2 Checkerboard Corners



```
24         frame_add += 1
25         print("Frame: ", frame_add, " has been saved.")
26     elif iKey == ord('q') or iKey == ord('Q'):
27         break
28
```

1.2 Camera Calibration

main.py

```
1 import glob
2 import cv2 as cv
3 import numpy as np
4
5
```

Figure 2: 1.2 Camera Matrix and Distortion Coefficient

```
un: main x
/usr/bin/python3.10 /mnt/ntfs/Data/code/CV/computer_vision/Labs/Lab03-C
Camera matrix:

[[544.06254343    0.        321.767787   ]
 [  0.          548.01458   271.29350075]
 [  0.           0.         1.         ]]

Distortion Coefficient:

[[ 0.14004592 -0.61955377  0.02033056  0.01136857  0.45179258]]

Rotation vectors:

(array([[ -0.16156531,
          0.21668922,
         -1.50205925]], array([[ -0.19953453,
          0.25160957,
         -1.48578156]], array([[ -0.2375458 ],
          0.28581421,
         -1.46865635]], array([[ -0.28377355,
          0.31181889,
         -1.44963124]], array([[ -0.32923122,
          0.34512525,
         -1.43408361]], array([[ 0.47354846,
          0.4985352 ],
         -1.54783722]], array([[ 0.59123356,
          0.62990243,
```

Figure 3: 1.2 Result

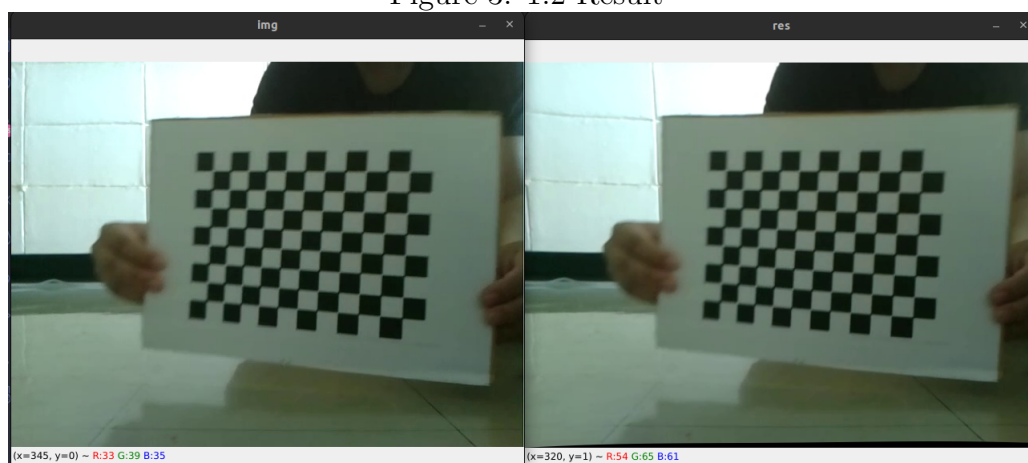


Figure 4: 1.2 Re-projection Error

```
Total error: 0.10270851690199258
```

```
Process finished with exit code 0
```

```
6 # Define the dimensions of checkerboard
7 CHECKERBOARD = (8, 11)
8 criteria = (cv.TERM_CRITERIA_EPS + cv.
    TERM_CRITERIA_MAX_ITER, 30, 0.001)
9
10 # Create vector to store vectors of 3D points for each
    checkerboard image
11 obj_points = []
12 # Create vector to store vectors of 2D points for each
    checkerboard image
13 img_points = []
14
15 # Define the world coordinates for 3D points
16 obj_point = np.zeros((1, CHECKERBOARD[0] * CHECKERBOARD
    [1], 3), dtype=np.float32)
17 obj_point[0, :, :2] = np.mgrid[0:CHECKERBOARD[0], 0:
    CHECKERBOARD[1]].T.reshape(-1, 2)
18 prev_img_shape = None
19
20 # Extracting path of individual image stored in a given
    directory
21 images = glob.glob("../../Data/Lab03/images/*.jpg")
22 img = None
23 gray = None
24 for f_name in images:
25
26     img = cv.imread(f_name)
27     gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
28
29     # Find the chessboard corners
30     # If desired number of corners are found in the
    image then ret = true
31     ret, corners = cv.findChessboardCorners(gray,
    CHECKERBOARD, cv.CALIB_CB_ADAPTIVE_THRESH + cv.
```

```

CALIB_CB_FAST_CHECK
32                                                                 + cv.
CALIB_CB_NORMALIZE_IMAGE)
33 """
34     If desired number of corner are detected,
    refine the pixel coordinates and display them
35     on the images of checkerboard
36 """
37     if ret:
38         obj_points.append(obj_point)
39
40     # Refining pixel coordinates for given 2D
    points.
41     corners2 = cv.cornerSubPix(gray, corners, (11,
    11), (-1, -1), criteria)
42
43     img_points.append(corners2)
44
45     # Draw and display the corners
46     img = cv.drawChessboardCorners(img,
    CHECKERBOARD, corners2, ret)
47     cv.imshow('img', img)
48     cv.waitKey(0)
49
50 cv.destroyAllWindows()
51
52 h, w = img.shape[:2]
53 """
54     Performing camera calibration by passing the value
    of known 3D points (obj_points) and
55     corresponding pixel coordinates of the detected
    corners (img_points)
56 """
57 ret, mtx, dist, r_vectors, t_vectors = cv.
    calibrateCamera(obj_points, img_points, gray.shape
    [:-1], None, None)
58
59 print("Camera matrix: \n")
60 print(mtx)
61 print("Distortion Coefficient: \n")
62 print(dist)

```

```

63 print("Rotation vectors: \n")
64 print(r_vectors)
65 print("Translation vectors: \n")
66 print(t_vectors)
67
68 # Show images of undistorted
69 for f_name in images:
70     img = cv.imread(f_name)
71     res = cv.undistort(img, mtx, dist)
72     cv.imshow('img', img)
73     cv.imshow('res', res)
74     cv.waitKey(0)
75
76 # Use ROI obtained above to crop the result
77 print("Undistorted using ROI")
78 for f_name in images:
79     print(f_name)
80
81     img = cv.imread(f_name)
82
83     h, w = img.shape[:2]
84     new_camera_mtx, roi = cv.getOptimalNewCameraMatrix(
85         mtx, dist, (w, h), 1, (w, h))
86     res = cv.undistort(img, mtx, dist)
87
88     # crop the image
89     x, y, w, h = roi
90     res = res[y:y+h, x:x+w]
91
92     cv.imshow('img', img)
93     cv.imshow('res', res)
94     cv.waitKey(0)
95
96 # Find a mapping function from the distorted image to
97     undistorted image.
98 # Then use the remap function.
99 print("Find mapping from distorted to undistorted image
100     .\nThen use the remap function.")
101 for f_name in images:
102     print(f_name)

```

```

101
102     img = cv.imread(f_name)
103     h, w = img.shape[:2]
104     new_camera_mtx, roi = cv.getOptimalNewCameraMatrix(
mtx, dist, (w, h), 1, (w, h))
105
106     map_x, map_y = cv.initUndistortRectifyMap(mtx, dist
, None, new_camera_mtx, (w, h), 5)
107     res = cv.remap(img, map_x, map_y, cv.INTER_LINEAR)
108
109     # Crop the image
110     x, y, w, h = roi
111     res = res[y:y+h, x:x+w]
112
113     cv.imshow('img', img)
114     cv.imshow('res', res)
115     cv.waitKey(0)
116
117 mean_error = 0
118 for i in range(len(obj_points)):
119     img_points2, _ = cv.projectPoints(obj_points[i],
r_vectors[i], t_vectors[i], mtx, dist)
120     error = cv.norm(img_points[i], img_points2, cv.
NORML2) / len(img_points2)
121     mean_error += error
122 print(f"Total error: {mean_error / len(obj_points)}")
123
124

```

2 Exercises

2.1 Read and write calibration file

```

1 # main.py
2 from calibrate_camera import CameraCalibration
3
4 camera_calibration = CameraCalibration()
5
6 camera_calibration.calibrate("../../../Data/Lab03/
images/")

```

Figure 5: 2.1 Camera Calibration File

```

1 # yaml:1.0
2
3 -MTX: !!opencv-matrix
4   rows: 3
5   cols: 3
6   dt: d
7   data: [ 6.4068675610525258e+02, 0., 3.2050317507210139e+02, 0.,
8           6.3853013415598036e+02, 2.3299583869329734e+02, 0., 0., 1. ]
9
10 -DIST: !!opencv-matrix
11   rows: 1
12   cols: 5
13   dt: d
14   data: [ -2.1940627009703402e-01, 7.0234268457071114e-01,
15           5.0569914964168853e-01, -8.3425777996263280e-03,
16           -6.4554018596636731e-01 ]
17
18 -R0: !!opencv-matrix
19   rows: 3
20   cols: 1
21   dt: d
22   data: [ -5.1954201346201094e-02, -7.1244090249529407e-02,
23           -1.5796116892110386e+00 ]
24
25 -T0: !!opencv-matrix
26   rows: 3
27   cols: 1
28   dt: d

```

```

7 print("Camera matrix: ", camera_calibration.camera_mtx)
8 print("Distortion coefficient: ", camera_calibration.
    distortion_coefficient)
9 print("Rotation vectors: ", camera_calibration.
    rotation_vectors)
10 print("Translation vectors: ", camera_calibration.
    translation_vectors)
11
12 camera_calibration.write_calibration_file("
    checkerboard_calibration_file.yml")
13 print()
14
15 camera_calibration.read_calibration_file("
    checkerboard_calibration_file.yml")
16 print("Camera matrix: ", camera_calibration.camera_mtx)
17 print("Distortion coefficient: ", camera_calibration.
    distortion_coefficient)
18 print("Rotation vectors: ", camera_calibration.
    rotation_vectors)
19 print("Translation vectors: ", camera_calibration.
    translation_vectors)
20
21 # calibrate_camera.py
22 import glob
23 import cv2 as cv
24 import numpy as np
25

```



```

6
7 class CameraCalibration:
8     camera_mtx = None
9     distortion_coefficient = None
10    rotation_vectors = None
11    translation_vectors = None
12
13    # Define the dimensions of checkerboard
14    _CHECKERBOARD = (8, 11)
15    __criteria = (cv.TERM_CRITERIA_EPS + cv.
16    TERM_CRITERIA_MAX_ITER, 30, 0.001)
17
18    # Create vector to store vectors of 3D points for
19    each checkerboard image
20    __obj_points = []
21
22    # Create vector to store vectors of 2D points for
23    each checkerboard image
24    __img_points = []
25
26    # Define the world coordinates for 3D points
27    __obj_point = np.zeros((1, _CHECKERBOARD[0] *
28    _CHECKERBOARD[1], 3), dtype=np.float32)
29    __obj_point[0, :, :2] = np.mgrid[0:_CHECKERBOARD
30    [0], 0:_CHECKERBOARD[1]].T.reshape(-1, 2)
31    __prev_img_shape = None
32
33    __img = None
34    __gray = None
35
36    def __init__(self, file_name=None):
37        if file_name is not None:
38            self.read_calibration_file(file_name)
39
40    def calibrate(self, file_path):
41        images = glob.glob(file_path + "/*.jpg")
42
43        for f_name in images:
44            self.__img = cv.imread(f_name)
45            self.__gray = cv.cvtColor(self.__img, cv.
46    COLOR_BGR2GRAY)

```

```

41
42         # Find the checkerboard corners
43         ret, corners = cv.findChessboardCorners(
self.__gray, self._CHECKERBOARD, cv.
CALIB_CB_ADAPTIVE_THRESH
44                                     +
cv.CALIB_CB_FAST_CHECK + cv.CALIB_CB_NORMALIZE_IMAGE
)
45
46         # If the desired number of corner are
detected
47         if ret:
48             self.__obj_points.append(self.
__obj_point)
49
50             # Refining pixel coordinates for given
2D points.
51             corners2 = cv.cornerSubPix(self.__gray,
corners, (11, 11), (-1, -1), self.__criteria)
52
53             self.__img_points.append(corners2)
54
55             # Draw and display the corners
56             self.__img = cv.drawChessboardCorners(
self.__img, self._CHECKERBOARD, corners2, True)
57
58             cv.imshow('img', self.__img)
59             cv.waitKey(0)
60             cv.destroyAllWindows()
61
62         # Calibrate the camera
63         ret, self.camera_mtx, self.
distortion_coefficient, self.rotation_vectors, self.
translation_vectors = \
64             cv.calibrateCamera(self.__obj_points, self.
__img_points, self.__gray.shape[:,-1], None, None)
65
66         def write_calibration_file(self, file_name):
67             file_storage = cv.FileStorage(file_name, cv.
FILE_STORAGE_WRITE)
68

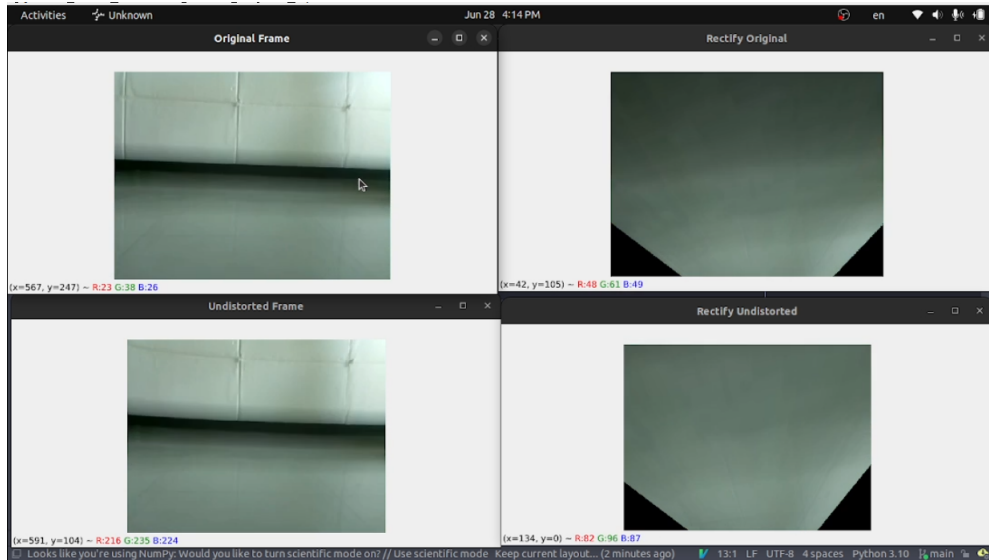
```

```

69         if not file_storage.isOpened():
70             return False
71
72         file_storage.write("MIX", self.camera_mtx)
73         file_storage.write("DIST", self.
distortion_coefficient)
74         for i in range(10):
75             file_storage.write("R" + str(i), self.
rotation_vectors[i])
76             file_storage.write("T" + str(i), self.
translation_vectors[i])
77             file_storage.release()
78
79         return True
80
81     def read_calibration_file(self, file_name):
82         file_storage = cv.FileStorage(file_name, cv.
FILE_STORAGE_READ)
83         if not file_storage.isOpened():
84             return False
85
86         self.camera_mtx = file_storage.getNode("MIX").
mat()
87         self.distortion_coefficient = file_storage.
getNode("DIST").mat()
88         r_vectors = tuple()
89         t_vectors = tuple()
90         for i in range(10):
91             r_vectors += (file_storage.getNode("R" +
str(i)).mat(),)
92             t_vectors += (file_storage.getNode("T" +
str(i)).mat(),)
93         self.rotation_vectors = r_vectors
94         self.translation_vectors = t_vectors
95         file_storage.release()
96
97         return True
98

```

Figure 6: 2.2 Results on own video



2.2 Calibrate camera, re-calculate homography from own video

```

1 # capture_video.py
2 import cv2 as cv
3
4
5 def capture_video(file_name: str):
6     video_capture = cv.VideoCapture(0)
7
8     fourcc = cv.VideoWriter_fourcc(*"mp4v")
9     out = cv.VideoWriter(file_name, fourcc, 20.0, (640,
10     480))
11
12     while True:
13         ret, frame = video_capture.read()
14
15         out.write(frame)
16
17         cv.imshow("Web Camera", frame)
18
19         if cv.waitKey(30) & 0xFF == ord('q'):
20             break

```

```

20
21     cv.destroyAllWindows()
22     video_capture.release()
23     out.release()
24
25
26 def convert_video_to_images(file_name: str, path: str):
27     video_capture = cv.VideoCapture(path + file_name)
28
29     count = 0
30     while True:
31         ret, frame = video_capture.read()
32         if frame is None:
33             break
34         cv.imwrite(path + "images/frame%d.jpg" % count,
35                    frame)
36         count += 1
37
38         cv.waitKey(30)
39
40
41 1 # calibrate_camera.py
42 2 import glob
43 3 import cv2 as cv
44 4 import numpy as np
45 5
46 6
47 7 class CameraCalibration:
48 8     camera_mtx = None
49 9     distortion_coefficient = None
50 10    rotation_vectors = None
51 11    translation_vectors = None
52 12
53 13    # Define the dimensions of checkerboard
54 14    _CHECKERBOARD = (8, 11)
55 15    __criteria = (cv.TERM_CRITERIA_EPS + cv.
56 16    TERM_CRITERIA_MAX_ITER, 30, 0.001)
57 17
58 17    # Create vector to store vectors of 3D points for

```

```

each checkerboard image
18     __obj_points = []
19
20     # Create vector to store vectors of 2D points for
each checkerboard image
21     __img_points = []
22
23     # Define the world coordinates for 3D points
24     __obj_point = np.zeros((1, _CHECKERBOARD[0] *
_CHECKERBOARD[1], 3), dtype=np.float32)
25     __obj_point[0, :, :2] = np.mgrid[0:_CHECKERBOARD
[0], 0:_CHECKERBOARD[1]].T.reshape(-1, 2)
26     __prev_img_shape = None
27
28     __img = None
29     __gray = None
30
31     def __init__(self, file_name=None):
32         if file_name is not None:
33             self.read_calibration_file(file_name)
34
35     def calibrate(self, file_path):
36         images = glob.glob(file_path + "*.jpg")
37
38         for f_name in images:
39             self.__img = cv.imread(f_name)
40             self.__gray = cv.cvtColor(self.__img, cv.
COLOR_BGR2GRAY)
41
42             # Find the checkerboard corners
43             ret, corners = cv.findChessboardCorners(
self.__gray, self._CHECKERBOARD, cv.
CALIB_CB_ADAPTIVE_THRESH
44                                     +
cv.CALIB_CB_FAST_CHECK + cv.CALIB_CB_NORMALIZE_IMAGE
)
45
46             # If the desired number of corner are
detected
47             if ret:
48                 self.__obj_points.append(self.

```

```

49     __obj_point)
50
51         # Refining pixel coordinates for given
2D points.
52         corners2 = cv.cornerSubPix(self.__gray,
53         corners, (11, 11), (-1, -1), self.__criteria)
54
55         self.__img_points.append(corners2)
56
57         # Draw and display the corners
58         self.__img = cv.drawChessboardCorners(
self.__img, self.__CHECKERBOARD, corners2, True)
59
60         cv.imshow('img', self.__img)
61         cv.waitKey(30)
62         cv.destroyAllWindows()
63
64         # Calibrate the camera
65         ret, self.camera_mtx, self.
distortion_coefficient, self.rotation_vectors, self.
translation_vectors = \
66         cv.calibrateCamera(self.__obj_points, self.
__img_points, self.__gray.shape[::-1], None, None)
67
68         def write_calibration_file(self, file_name):
69             file_storage = cv.FileStorage(file_name, cv.
FILE_STORAGE_WRITE)
70
71             if not file_storage.isOpened():
72                 return False
73
74             file_storage.write("MIX", self.camera_mtx)
75             file_storage.write("DIST", self.
distortion_coefficient)
76             for i in range(10):
77                 file_storage.write("R" + str(i), self.
rotation_vectors[i])
78                 file_storage.write("T" + str(i), self.
translation_vectors[i])
79                 file_storage.release()

```

```

79         return True
80
81     def read_calibration_file(self, file_name):
82         file_storage = cv.FileStorage(file_name, cv.
FILESTORAGE_READ)
83         if not file_storage.isOpened():
84             return False
85
86         self.camera_mtx = file_storage.getNode("MIX").
mat()
87         self.distortion_coefficient = file_storage.
getNode("DIST").mat()
88         r_vectors = tuple()
89         t_vectors = tuple()
90         for i in range(10):
91             r_vectors += (file_storage.getNode("R" +
str(i)).mat(),)
92             t_vectors += (file_storage.getNode("T" +
str(i)).mat(),)
93         self.rotation_vectors = r_vectors
94         self.translation_vectors = t_vectors
95         file_storage.release()
96
97         return True
98
99
1 # homography.py
2 import cv2 as cv
3 import numpy as np
4
5
6 class Homography:
7     mat_h = np.zeros((3, 3))
8     width_out: int
9     height_out: int
10    c_points: int
11    a_points: list = []
12
13    __point = (-1, -1)
14    __pts = []
15    __var = 0

```



```

16     __drag = 0
17     __mat_final = np.array ([])
18     __mat_result = np.array ([])
19
20     def __init__(self , homography_file=None):
21         self.c_points = 0
22         if homography_file is not None:
23             self.read(homography_file)
24
25     def read(self , homography_file: str):
26         file_storage = cv.FileStorage(homography_file ,
cv.FILE_STORAGE_READ)
27         if not file_storage.isOpened():
28             return False
29
30         self.c_points = 0
31         for i in range(4):
32             point = file_storage.getNode("aPoints" +
str(i))
33             self.a_points.append(point.mat())
34             self.c_points += 1
35
36         self.mat_h = file_storage.getNode("matH").mat()
37         self.width_out = int(file_storage.getNode("
widthOut").real())
38         self.height_out = int(file_storage.getNode("
heightOut").real())
39         file_storage.release()
40         return True
41
42     def write(self , homography_file):
43         file_storage = cv.FileStorage(homography_file ,
cv.FILE_STORAGE_WRITE)
44         if not file_storage.isOpened():
45             return False
46
47         for i in range(4):
48             file_storage.write("aPoints" + str(i) , self
.a_points[i])
49
50         file_storage.write("matH" , self.mat_h)

```

```

51         file_storage.write("widthOut", self.width_out)
52         file_storage.write("heightOut", self.height_out
53     )
54         file_storage.release()
55
56     return True
57
58     def __draw_circle_and_line(self, x, y):
59         self.__mat_result = self.__mat_final.copy()
60         self.__point = (x, y)
61
62         if self.__var >= 1:
63             cv.line(self.__mat_result, self.__pts[self.
64 __var - 1], self.__point, (0, 255, 0, 255), 2)
65             cv.circle(self.__mat_result, self.__point, 2,
66 (0, 255, 0), -1, 8, 0)
67             cv.imshow("Source", self.__mat_result)
68
69     def __mouse_handler(self, event, x, y, flags, param
70 ):
71         if self.__var >= 4:
72             return
73
74         if event == cv.EVENT_LBUTTONDOWN:
75             self.__drag = 1
76             self.__draw_circle_and_line(x, y)
77
78         if event == cv.EVENT_LBUTTONUP and self.__drag:
79             self.__drag = 0
80             self.__pts.append(self.__point)
81             self.__var += 1
82             self.__mat_final = self.__mat_result.copy()
83
84             if self.__var >= 4:
85                 cv.line(self.__mat_final, self.__pts
86 [0], self.__pts[3], (0, 255, 0, 255), 2)
87                 cv.fillConvexPoly(self.__mat_final, np.
88 array(self.__pts, dtype=np.int32), (0, 120, 0, 20))
89                 cv.imshow("Source", self.__mat_final)
90
91         if self.__drag:

```

```

86         self.__draw_circle_and_line(x, y)
87
88     def calculate(self, file_name):
89         mat_pause_screen = np.array([])
90         mat_frame_capture = np.array([])
91         key = -1
92
93         # ----- [STEP 1: Make video
capture from file] -----
94         # Open video file
95         video_capture = cv.VideoCapture(file_name)
96         if not video_capture.isOpened():
97             print("ERROR! Unable to open input video
file ", file_name)
98             return False
99
100         width = video_capture.get(cv.
CAP_PROP_FRAME_WIDTH)
101         height = video_capture.get(cv.
CAP_PROP_FRAME_HEIGHT)
102         ratio = 640.0 / width
103         dim = (int(width * ratio), int(height * ratio))
104
105         while key < 0:
106             # Get the next frame
107             _, mat_frame_capture = video_capture.read()
108             if mat_frame_capture is None:
109                 break
110
111             mat_frame_display = cv.resize(
mat_frame_capture, dim)
112
113             cv.imshow("Original", mat_frame_display)
114             key = cv.waitKey(30)
115
116         # ----- [STEP 2: pause the
screen and show an image] -----
117         if key >= 0:
118             mat_pause_screen = mat_frame_capture
119             self.__mat_final = mat_pause_screen.
copy()

```

```

120
121         cv.destroyAllWindows()
122
123         # ----- [STEP 3: use mouse
handler to select 4 points] -----
124         if mat_frame_capture is not None:
125             self.__var = 0
126             self.__pts.clear()
127             cv.namedWindow("Source", cv.
WINDOW_GULNORMAL)
128             cv.setMouseCallback("Source", self.
__mouse_handler)
129             cv.imshow("Source", mat_pause_screen)
130             cv.waitKey(0)
131             cv.destroyWindow("Source")
132
133             if len(self.__pts) == 4:
134                 src = np.array(self.__pts).astype(np.
float32)
135
136                 reals = np.array([
137                     (200, 200),
138                     (429, 200),
139                     (429, 429),
140                     (200, 429)
141                 ], dtype=np.float32)
142
143                 # ----- [STEP 4: Calculate
Homography] -----
144                 homography_matrix = cv.
getPerspectiveTransform(src, reals)
145
146                 # ----- [STEP 4: Calculate
Homography] -----
147                 self.mat_h = homography_matrix
148                 self.c_points = 0
149                 for i in range(4):
150                     self.a_points.append(src[i])
151                     self.c_points += 1
152                 self.width_out = int(width)
153                 self.height_out = int(height)

```

```

154
155             return True
156         else:
157             return False
158
159
1
2 # main.py
3 import sys
4 import cv2 as cv
5 # from capture_video import capture_video,
6     convert_video_to_images
7 from calibrate_camera import CameraCalibration
8 from homography import Homography
9
10 PATH = "../..../Data/Lab03/checkerboard/"
11 CALIBRATION_FILE = "checkerboard.mp4"
12 VIDEO_FILE = "car.mp4"
13
14 # Capture the checkerboard video
15 # capture_video(PATH + CALIBRATION_FILE)
16
17 # Convert the checkerboard video to images
18 # convert_video_to_images(CALIBRATION_FILE, PATH)
19
20 # Calibrate the camera
21 camera_calibration = CameraCalibration()
22 camera_calibration.calibrate(PATH + "images/")
23 camera_calibration.write_calibration_file("
24     camera_calibration.yml")
25
26 print("Camera matrix: ", camera_calibration.camera_mtx)
27 print("Distortion coefficient: ", camera_calibration.
28     distortion_coefficient)
29 print("Rotation vectors: ", camera_calibration.
30     rotation_vectors)
31 print("Translation vectors: ", camera_calibration.
32     translation_vectors)
33
34 # Calculate homography
35 homography_data = Homography()

```

```

30 homography_data.calculate(PATH + VIDEO_FILE)
31 homography_data.write("homography.yml")
32
33 print("Estimated Homography matrix: \n",
        homography_data.mat_h)
34
35 # Show undistorted and rectify images
36 video_capture = cv.VideoCapture(PATH + VIDEO_FILE)
37 if not video_capture.isOpened():
38     print("ERROR! Unable to open video file ",
        VIDEO_FILE)
39     sys.exit()
40 width = video_capture.get(cv.CAP_PROP_FRAME_WIDTH)
41 height = video_capture.get(cv.CAP_PROP_FRAME_HEIGHT)
42 ratio = 640.0 / width
43 dim = (int(width * ratio), int(height * ratio))
44
45 duration = 0
46 while True:
47     _, frame = video_capture.read()
48     if frame is None:
49         break
50
51     # Undistorted and Cropped the frame
52     undistorted_frame = cv.undistort(frame,
        camera_calibration.camera_mtx,
53                                     camera_calibration
        .distortion_coefficient)
54     h, w = frame.shape[:2]
55     new_camera_mtx, roi = cv.getOptimalNewCameraMatrix(
        camera_calibration.camera_mtx,
56
        camera_calibration.distortion_coefficient,
57
        (w, h), 1, (w, h))
58     x, y, w, h = roi
59     cropped_undistorted_frame = undistorted_frame[y:y+h
        , x:x+w]
60
61     # Show the original and undistorted frames
62     cv.namedWindow("Original Frame", cv.WINDOW_NORMAL |

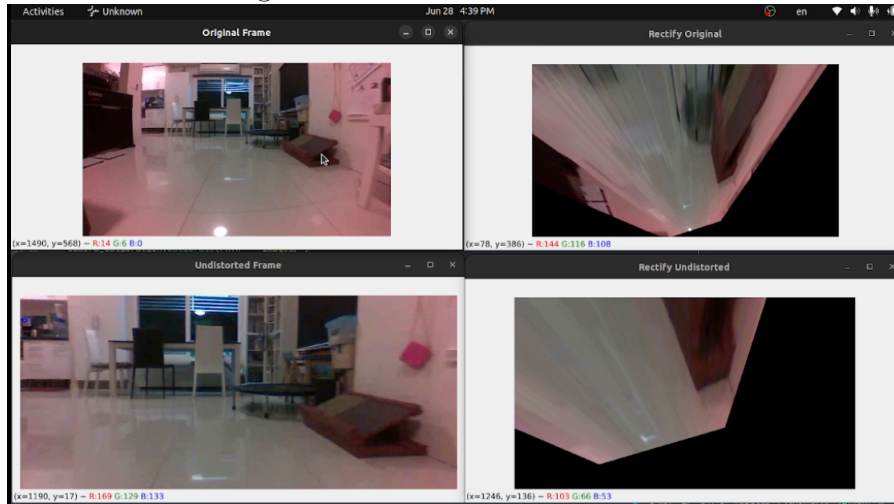
```

```

        cv.WINDOW_KEEPRATIO | cv.WINDOW_GUEXPANDED)
63     cv.imshow(" Original Frame", frame)
64     cv.namedWindow(" Undistorted Frame", cv.
WINDOW_NORMAL | cv.WINDOW_KEEPRATIO | cv.
WINDOW_GUEXPANDED)
65     cv.imshow(" Undistorted Frame",
cropped_undistorted_frame)
66
67     # Show the rectified original and undistorted
frames
68     result_original = cv.warpPerspective(frame,
homography_data.mat_h,
69                                     (int(
homography_data.width_out), int(homography_data.
height_out)),
70                                     cv.
INTER_LINEAR)
71     result_undistorted = cv.warpPerspective(
cropped_undistorted_frame, homography_data.mat_h,
72                                     (int(
homography_data.width_out), int(homography_data.
height_out)),
73                                     cv.
INTER_LINEAR)
74     cv.namedWindow(" Rectify Original", cv.WINDOW_NORMAL
| cv.WINDOW_KEEPRATIO | cv.WINDOW_GUEXPANDED)
75     cv.imshow(" Rectify Original", result_original)
76     cv.namedWindow(" Rectify Undistorted", cv.
WINDOW_NORMAL | cv.WINDOW_KEEPRATIO | cv.
WINDOW_GUEXPANDED)
77     cv.imshow(" Rectify Undistorted", result_undistorted
)
78
79     cv.waitKey(duration)
80     if duration == 0:
81         duration = 30
82 cv.destroyAllWindows()
83

```

Figure 7: 2.3 Results on Matt's robot video



2.3 Calibrate, Undistort the Matt's robot video

```

1 # calibrate_camera.py
2 import glob
3 import cv2 as cv
4 import numpy as np
5
6
7 class CameraCalibration:
8     camera_mtx = None
9     distortion_coefficient = None
10    rotation_vectors = None
11    translation_vectors = None
12
13    # Define the dimensions of checkerboard
14    _CHECKERBOARD = (6, 9)
15    __criteria = (cv.TERM_CRITERIA_EPS + cv.
16    TERM_CRITERIA_MAX_ITER, 30, 0.001)
17
18    # Create vector to store vectors of 3D points for
19    # each checkerboard image
20    __obj_points = []
21
22    # Create vector to store vectors of 2D points for
23    # each checkerboard image

```



```

21     __img_points = []
22
23     # Define the world coordinates for 3D points
24     __obj_point = np.zeros((1, _CHECKERBOARD[0] *
25 _CHECKERBOARD[1], 3), dtype=np.float32)
26     __obj_point[0, :, :2] = np.mgrid[0:_CHECKERBOARD
27 [0], 0:_CHECKERBOARD[1]].T.reshape(-1, 2)
28     __prev_img_shape = None
29
30     __img = None
31     __gray = None
32
33     def __init__(self, file_name=None):
34         if file_name is not None:
35             self.read_calibration_file(file_name)
36
37     def calibrate(self, file_path):
38         images = glob.glob(file_path + "*.jpg")
39
40         for f_name in images:
41             self.__img = cv.imread(f_name)
42             self.__gray = cv.cvtColor(self.__img, cv.
43 COLOR_BGR2GRAY)
44
45             # Find the checkerboard corners
46             ret, corners = cv.findChessboardCorners(
47 self.__gray, self._CHECKERBOARD, cv.
48 CALIB_CB_ADAPTIVE_THRESH
49 +
50 cv.CALIB_CB_FAST_CHECK + cv.CALIB_CB_NORMALIZE_IMAGE
51 )
52
53             # If the desired number of corner are
54 detected
55             if ret:
56                 self.__obj_points.append(self.
57 __obj_point)
58
59             # Refining pixel coordinates for given
60 2D points.
61             corners2 = cv.cornerSubPix(self.__gray,

```

```

    corners, (9, 9), (-1, -1), self.__criteria)
52
53         self.__img_points.append(corners2)
54
55         # Draw and display the corners
56         self.__img = cv.drawChessboardCorners(
self.__img, self._CHECKERBOARD, corners2, True)
57
58         cv.imshow('img', self.__img)
59         cv.waitKey(30)
60         cv.destroyAllWindows()
61
62         # Calibrate the camera
63         ret, self.camera_mtx, self.
distortion_coefficient, self.rotation_vectors, self.
translation_vectors = \
64         cv.calibrateCamera(self.__obj_points, self.
__img_points, self.__gray.shape[:,-1], None, None)
65
66         def write_calibration_file(self, file_name):
67             file_storage = cv.FileStorage(file_name, cv.
FILE_STORAGE_WRITE)
68
69             if not file_storage.isOpened():
70                 return False
71
72             file_storage.write("MIX", self.camera_mtx)
73             file_storage.write("DIST", self.
distortion_coefficient)
74             for i in range(10):
75                 file_storage.write("R" + str(i), self.
rotation_vectors[i])
76                 file_storage.write("T" + str(i), self.
translation_vectors[i])
77             file_storage.release()
78
79             return True
80
81         def read_calibration_file(self, file_name):
82             file_storage = cv.FileStorage(file_name, cv.
FILE_STORAGE_READ)

```

```

83         if not file_storage.isOpened():
84             return False
85
86         self.camera_mtx = file_storage.getNode("MIX").
mat()
87         self.distortion_coefficient = file_storage.
getNode("DIST").mat()
88         r_vectors = tuple()
89         t_vectors = tuple()
90         for i in range(10):
91             r_vectors += (file_storage.getNode("R" +
str(i)).mat(),)
92             t_vectors += (file_storage.getNode("T" +
str(i)).mat(),)
93         self.rotation_vectors = r_vectors
94         self.translation_vectors = t_vectors
95         file_storage.release()
96
97         return True
98
1 # homography.py
2 import cv2 as cv
3 import numpy as np
4
5
6 class Homography:
7     mat_h = np.zeros((3, 3))
8     width_out: int
9     height_out: int
10    c_points: int
11    a_points: list = []
12
13    __point = (-1, -1)
14    __pts = []
15    __var = 0
16    __drag = 0
17    __mat_final = np.array([])
18    __mat_result = np.array([])
19
20    def __init__(self, homography_file=None):

```

```

21         self.c_points = 0
22         if homography_file is not None:
23             self.read(homography_file)
24
25     def read(self, homography_file: str):
26         file_storage = cv.FileStorage(homography_file,
cv.FILE_STORAGE_READ)
27         if not file_storage.isOpened():
28             return False
29
30         self.c_points = 0
31         for i in range(4):
32             point = file_storage.getNode("aPoints" +
str(i))
33             self.a_points.append(point.mat())
34             self.c_points += 1
35
36         self.mat_h = file_storage.getNode("matH").mat()
37         self.width_out = int(file_storage.getNode("
widthOut").real())
38         self.height_out = int(file_storage.getNode("
heightOut").real())
39         file_storage.release()
40         return True
41
42     def write(self, homography_file):
43         file_storage = cv.FileStorage(homography_file,
cv.FILE_STORAGE_WRITE)
44         if not file_storage.isOpened():
45             return False
46
47         for i in range(4):
48             file_storage.write("aPoints" + str(i), self
.a_points[i])
49
50         file_storage.write("matH", self.mat_h)
51         file_storage.write("widthOut", self.width_out)
52         file_storage.write("heightOut", self.height_out
)
53         file_storage.release()
54

```

```

55         return True
56
57     def __draw_circle_and_line(self, x, y):
58         self.__mat_result = self.__mat_final.copy()
59         self.__point = (x, y)
60
61         if self.__var >= 1:
62             cv.line(self.__mat_result, self.__pts[self.
__var - 1], self.__point, (0, 255, 0, 255), 2)
63             cv.circle(self.__mat_result, self.__point, 2,
(0, 255, 0), -1, 8, 0)
64             cv.imshow("Source", self.__mat_result)
65
66     def __mouse_handler(self, event, x, y, flags, param
):
67         if self.__var >= 4:
68             return
69
70         if event == cv.EVENT_LBUTTONDOWN:
71             self.__drag = 1
72             self.__draw_circle_and_line(x, y)
73
74         if event == cv.EVENT_LBUTTONUP and self.__drag:
75             self.__drag = 0
76             self.__pts.append(self.__point)
77             self.__var += 1
78             self.__mat_final = self.__mat_result.copy()
79
80             if self.__var >= 4:
81                 cv.line(self.__mat_final, self.__pts
[0], self.__pts[3], (0, 255, 0, 255), 2)
82                 cv.fillConvexPoly(self.__mat_final, np.
array(self.__pts, dtype=np.int32), (0, 120, 0, 20))
83                 cv.imshow("Source", self.__mat_final)
84
85             if self.__drag:
86                 self.__draw_circle_and_line(x, y)
87
88     def calculate(self, file_name):
89         mat_pause_screen = np.array([])
90         mat_frame_capture = np.array([])

```

```

91         key = -1
92
93         # ----- [STEP 1: Make video
capture from file] -----
94         # Open video file
95         video_capture = cv.VideoCapture(file_name)
96         if not video_capture.isOpened():
97             print("ERROR! Unable to open input video
file ", file_name)
98             return False
99
100         width = video_capture.get(cv.
CAP_PROP_FRAME_WIDTH)
101         height = video_capture.get(cv.
CAP_PROP_FRAME_HEIGHT)
102         ratio = 640.0 / width
103         dim = (int(width * ratio), int(height * ratio))
104
105         while key < 0:
106             # Get the next frame
107             _, mat_frame_capture = video_capture.read()
108             if mat_frame_capture is None:
109                 break
110
111             mat_frame_display = cv.resize(
mat_frame_capture, dim)
112
113             cv.imshow("Original", mat_frame_display)
114             key = cv.waitKey(30)
115
116         # ----- [STEP 2: pause the
screen and show an image] -----
117         if key >= 0:
118             mat_pause_screen = mat_frame_capture
119             self._mat_final = mat_pause_screen.
copy()
120
121         cv.destroyAllWindows()
122
123         # ----- [STEP 3: use mouse
handler to select 4 points] -----

```

```

124         if mat_frame_capture is not None:
125             self.__var = 0
126             self.__pts.clear()
127             cv.namedWindow("Source", cv.
WINDOW_GULNORMAL)
128             cv.setMouseCallback("Source", self.
__mouse_handler)
129             cv.imshow("Source", mat_pause_screen)
130             cv.waitKey(0)
131             cv.destroyWindow("Source")
132
133             if len(self.__pts) == 4:
134                 src = np.array(self.__pts).astype(np.
float32)
135
136                 reals = np.array([
137                     (800, 800),
138                     (1000, 800),
139                     (1000, 1000),
140                     (800, 1000)
141                 ], dtype=np.float32)
142
143                 # ----- [STEP 4: Calculate
Homography] -----
144                 homography_matrix = cv.
getPerspectiveTransform(src, reals)
145
146                 # ----- [STEP 4: Calculate
Homography] -----
147                 self.mat_h = homography_matrix
148                 self.c_points = 0
149                 for i in range(4):
150                     self.a_points.append(src[i])
151                     self.c_points += 1
152                 self.width_out = int(width)
153                 self.height_out = int(height)
154
155                 return True
156             else:
157                 return False
158

```

```

1 # main.py
2 import sys
3 import cv2 as cv
4 from calibrate_camera import CameraCalibration
5 from homography import Homography
6
7 PATH = "../..../Data/Lab03/robot/"
8 VIDEO_FILE = "robot.mp4"
9
10 # Calibrate the camera
11 camera_calibration = CameraCalibration()
12 camera_calibration.calibrate(PATH + "images/")
13 camera_calibration.write_calibration_file("
    camera_calibration.yml")
14
15 print("Camera matrix: ", camera_calibration.camera_mtx)
16 print("Distortion coefficient: ", camera_calibration.
    distortion_coefficient)
17 print("Rotation vectors: ", camera_calibration.
    rotation_vectors)
18 print("Translation vectors: ", camera_calibration.
    translation_vectors)
19
20 # Calculate homography
21 homography_data = Homography()
22 homography_data.calculate(PATH + VIDEO_FILE)
23 homography_data.write("homography.yml")
24
25 print("Estimated Homography matrix: \n",
    homography_data.mat_h)
26
27 # Show undistorted and rectify images
28 video_capture = cv.VideoCapture(PATH + VIDEO_FILE)
29 if not video_capture.isOpened():
30     print("ERROR! Unable to open video file ",
        VIDEO_FILE)
31     sys.exit()
32 width = video_capture.get(cv.CAP_PROP_FRAME_WIDTH)
33 height = video_capture.get(cv.CAP_PROP_FRAME_HEIGHT)
34 ratio = 600.0 / width
35 dim = (int(width * ratio), int(height * ratio))

```



```

36
37 duration = 0
38 while True:
39     _, frame = video_capture.read()
40     if frame is None:
41         break
42
43     # Undistorted and Cropped the frame
44     h, w = frame.shape[:2]
45     new_camera_mtx, roi = cv.getOptimalNewCameraMatrix(
camera_calibration.camera_mtx,
46
camera_calibration.distortion_coefficient,
47
(w, h), 1, (w, h))
48     undistorted_frame = cv.undistort(frame,
camera_calibration.camera_mtx,
49
camera_calibration
.distortion_coefficient)
50     x, y, w, h = roi
51     cropped_undistorted_frame = undistorted_frame[y:y+h
, x:x+w]
52
53     # Show the original and undistorted frames
54     cv.namedWindow(" Original Frame", cv.WINDOW_NORMAL |
cv.WINDOW_KEEPRATIO | cv.WINDOW_GUEXPANDED)
55     cv.imshow(" Original Frame", frame)
56     cv.namedWindow(" Undistorted Frame", cv.
WINDOW_NORMAL | cv.WINDOW_KEEPRATIO | cv.
WINDOW_GUEXPANDED)
57     cv.imshow(" Undistorted Frame",
cropped_undistorted_frame)
58
59     # Show the rectified original and undistorted
frames
60     result_original = cv.warpPerspective(frame,
homography_data.mat_h,
61
(int(
homography_data.width_out), int(homography_data.
height_out)),
62
cv.

```

```

INTER_LINEAR)
63     result_undistorted = cv.warpPerspective(
        cropped_undistorted_frame , homography_data.mat_h ,
64                                     (int(
        homography_data.width_out) , int(homography_data.
        height_out)) ,
65                                     cv.
        INTER_LINEAR)
66     cv.namedWindow(" Rectify  Original", cv.WINDOW_NORMAL
        | cv.WINDOW_KEEPRATIO | cv.WINDOW_GUI_EXPANDED)
67     cv.imshow(" Rectify  Original", result_original)
68     cv.namedWindow(" Rectify  Undistorted", cv.
        WINDOW_NORMAL | cv.WINDOW_KEEPRATIO | cv.
        WINDOW_GUI_EXPANDED)
69     cv.imshow(" Rectify  Undistorted", result_undistorted
        )
70
71     cv.waitKey( duration)
72     if duration == 0:
73         duration = 30
74 cv.destroyAllWindows()
75

```

2.4 Do you get better result?

The better result is achieved by undistorting the image because distortion in the images especially radial distortion are removed from the images.